VU VRIJE UNIVERSITEIT AMSTERDAM

UNIVERSITY OF AMSTERDAM

MSc Computer Science

Master Thesis

Energy-efficient phonetic matching for spoken words of rare languages in low-resource environments

by

Auke Schuringa

2585488 (VU) 11023465 (UvA)

February 2025

Supervisor:
Dr. Anna Bon

Second Reader:
Prof. Dr. J.M. (Hans) Akkermans

W4RA

Submitted in partial fulfillment of the requirements for the VU/UvA joint degree of Master of Science in Computer Science.

Abstract

Indigenous rare languages are often spoken by people in low-resource environments, where the literacy rate is low in many cases as well. Most communication tends to happen using speech, instead of in writing. These factors reinforce each other, as spoken rare languages are more difficult to interpret in an automated way due to a lack of available data. This makes increasing productivity more difficult and prevents fast development. At the same time, development is important for building businesses and improving general living standard, which is correlated with an improvement in literacy. A lack of automated processing of communication with speech therefore prevents improvements in literacy rate and living conditions.

This thesis introduces a novel method of handling spoken rare languages, by introducing a method to automate matching of their spoken words, and therefore attempts to help solve a key problem in low-resource environments. The method matches spoken words with each other by (1) using a pre-trained trained model for a phonetically close popular language to create a phonetic transcription of a recording, (2) generalizing context-specific or unique language specific aspects out to create a broader phonetic description of the spoken segments, and (3) calculating similarities between phonetic descriptions to allow for matching.

The proposed method is tested on the Dagbani language, and shows promising results. The method opens up a path towards speech-based applications in rare languages without having to perform any expensive training or finetuning of a model. The presented solution is an energy-efficient one and very well suited for use in low-resource environments.

Acknowledgments

A special thank you goes out to dr. Anna Bon for always being ready to help with any questions around this thesis. Anna is a researcher in the field of Information and Communication Technologies for Development (ICT4D) and has many contacts in countries and communities in which the results of this thesis may be applied, which was very helpful in gathering data to experiment with. She was always ready to meet, have a talk about the thesis, or have a brainstorm session or discussion in general about related subjects. Anna has greatly contributed to this being an interesting, fun, and informative project.

Another thank you goes out to Francis Saa-Dittoh for his work on solutions and implementations in the fields. Francis provided the data set this thesis works with, and provided ideas and input on how the results of this thesis may be applied.

The project is created for and in cooperation with the Web alliance for Regreening in Africa (W4RA)[1] program.

Software

The implementation is written in two parts. The first part is implementation with the use of Flashlight [82] and Wav2Letter++ [127] written in C++ [77]. The second part is the implementation for the experiments and the results themselves, which is written in Python 3 [175] in an environment by Jupyter Notebook [84], with the use of FFmpeg [50], NumPy [61], SciPy [177], jellyfish [172], requests [134], and python-magic [71]. For plotting of results the package Matplotlib [70] was used. An example application was made for the Telegram API [166] using the pyTelegramBotAPI [47].

The entire implementation is packaged with Docker [107] to easily move between environments and use in Python [101].

The typesetting of this report was done with XƎLaTeX and BibLaTeX, with some figures created using PGF/Ti*k*Z.

Hardware

For the calculations done in this thesis, a server at Hetzner[2] was rented with an Intel® Core™ i7-7700 Processor[3] CPU, an NVIDIA GeForce® GTX 1080[4] GPU, and two times 32768 MB DDR4 RAM. Note that these are not the minimum specifications needed for running the implementation and experiments.

[1] *W4RA website*. [Online]. Available: `https://w4ra.org/` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20240905112020/https://w4ra.org/` on Sep. 5, 2024.

[2] *Hetzner website*. [Online]. Available: `https://www.hetzner.com/` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20240808133711/https://www.hetzner.com/` on Aug. 8, 2024.

[3] *Intel® Core™ i7-7700 Processor specifications page*. [Online]. Available: `https://www.intel.com/content/www/us/en/products/sku/97128/intel-core-i77700-processor-8m-cache-up-to-4-20-ghz/specifications.html` (visited on Aug. 2, 2024).

[4] *NVIDIA GeForce® GTX 1080 product page*. [Online]. Available: `https://www.nvidia.com/en-ph/geforce/products/10series/geforce-gtx-1080/` (visited on Aug. 2, 2024).

Contents

Chapter 1

Introduction

1.1 Rare languages

A language can be defined in different ways, one way to define it is as a description of the general abstract concept used to convey information, and another way is as a description of a specific specific language [103] and accent. Spoken languages may have existed since around 60 to 100 thousand years ago [9]. Around 1900 BC the first analyses were created that looked into the relations between and differences of languages, in that specific case between Sumerian and Akkadian. It was found that eventually all cultures that developed writing started performing these kind of analyses between languages [32].

Chomsky found in 1965 [34] that there is a generative theory of languages which allows them to be explained using a universal grammar. This would mean that the purpose of researching any individual language is only to find information about the rules that underlay the grammar of all languages. This set of underlying rules would be the same for every human and every language. A more recent theory [21] builds on top of this and proposes that the reason why humans are able to create this grammar to form languages with is a recent mutation in the human genome.

However, another theory around languages is that of cognitive linguistics, which does not view languages as abstract forms of their proposed universal grammar, but rather sees languages as abstract forms of expression that are based on underlying concepts users think with [38]. These concepts can be universal among humans, but can also differ depending on the path of cultural evolution groups of humans have taken over generations. This theory thus analysis languages more from a cultural perspective [43], [78], [150], [190].

The theories of languages of cognitive and cultural linguistics show the importance of being able to analyze languages and with that attempt to understand their underlying concepts [122]. As cultures change, languages change, and as cultures disappear, so do their languages. A language becomes a dead language first, when there are no more native speakers - there may still be non-native speakers of the language. After this the language can become extinct if there are also no non-native speakers anymore [39]. Since 50% of the world speaks the 20 most spoken language [13], often as their only spoken language, the number of dead languages quickly increases, and may soon reach percentages above 50% of the total number of currently spoken languages. Many languages are currently endangered [111], and many more will follow partially due to this increasing globalization [116], [171].

An important aspect for the analysis - perhaps the critical aspect - of rare languages is that there is not a lot of high quality audio material available [108]. There are efforts underway to make recordings of somewhat popular languages publicly available [11] through for example the Common Voice project [112], but also of rare languages [31], for example through the Endangered Languages Project [18]. Next to this, publications and other information on known languages, especially rare languages, is being recorded in the open source project Glottolog [59].

This thesis partially attempts to contribute to the awareness of language endangerment, by looking into ways of analyzing languages of which not a lot of clear spoken audio in combination with text is available.

1.1.1 Impact on communities

The rare language can also be seen as low-resource languages, often spoken in low-resource environments. The communities in these environments in many cases show a low literacy rate [25], [143], with an example of less than 35% in rural Mali.

In the urban areas and western countries, many technological solutions are available that increase productivity and contribute to economic growth. However, in low-resource areas, there is often a lack of infrastructure and knowledge that prevents the proper adoption of technological solutions [174]. The low literacy rate in these areas, together with the use of rare languages, also contributes to a low technological literacy rate [88]. These reasons together prevents development of communities or slows this down, which causes the issues to remain, for which solutions need to be found [4].

For these reasons, it is important to create methods that are energy-efficient to use, and are well suited to low-resource environments, to help development and build a better future for these low-resource communities. The method explained and experimented with in this thesis helps with this.

1.2 Speech recognition

The main goal behind this thesis is matching of speech through speech recognition. Traditionally speech recognition is the translation of speech into text that is interpretable by humans or computers. Research in this field goes back to the 1930s [80] with a proposal by Dudley, Riesz, and Watkins [44] for a system to perform speech analysis. From this moment on implementations were created, such as digital recognition limited to single word definition, which were based on 'classical' methods, for example hidden Markov models [27].

Later on 'modern' approaches were developed in the form of neural networks [187]. However, the performance of these methods was often less good than the performance from hidden Markov models, which caused methods with neural network to be used in combination with hidden Markov models, and usually as a preparational step towards the latter [68]. This is because early neural network were good at performing recognition at a short time scale, but not good at a continuous-time scale, where hidden Markov models did perform well.

After the introduction of neural networks in speech processing, further improvements were found through the introduction of techniques like the LSTM [66], deep recurrent neural networks [58], and transformers [100]. These methods still used hidden Markov models to produce final correct text from recognized speech. End-to-end speech recognition attempted to take out the step of the hidden Markov model and use neural networks to nearly directly produce a readable interpretation of speech. An example is the Connectionist Temporal Classification [57], which directly transcribes speech into characters. However, cleanup was still necessary afterwards to handle mistakes. Using very large sets of data, further iterations with increasingly better performance were introduced [8], [94]. Due to the very large size of the models and the data sets required for training, transfer learning was used more and more to adapt existing models to new domains [163].

Nowadays, several models are available to the public to transcribe speech. Examples of these are Whisper [130], which is very user friendly and usable through the command line or Python, or alternatives like Distil-Whisper [53] or WhisperX [17]. Further examples include Wav2Letter++ [127], Mozilla DeepSpeech [11], or NeMo ASR [86]. Solution are constantly created and improved, which causes the presented list to be likely out of date at the time of reading this.

1.3 This thesis

This thesis focuses on languages from which not a lot of recordings are available. This makes it difficult to train a modern neural network for the entire language and get meaningful results. The thesis proposes a method that combines both modern solutions with pre-trained artificial neural networks trained for popular languages and more ‘classical’ processing of textual representations of speech to get meaningful output in an energy-efficient way for matching of spoken words in a rare languages.

In more details, the method from this thesis focuses on matching single spoken words. The transcriptions of the spoken words are not necessarily interpretable by humans, but contain enough information to be interpretable by machines. The machine will be able to match two transcriptions of a spoken word together, and in that way literally match instances of recorded speech with each other, the result of which can then be presented in a human-interpretable form. This makes it possible to search a catalog of words with spoken definitions and find a correct match with a given speech recording. There are many applications to this that will later be discussed.

The concept of matching spoken words can be explained in several steps. Each of these steps can be described in an abstract way, after which different implementations of each step are possible. The steps are further explained and investigated in chapter 2.

Each step of chapter 2 is implemented in a way as described in chapters 3 and 4 and experimented with in chapter 5. Based on these experiments a conclusion is drawn in chapter 6, which is discussed in chapter 7. In this discussion, ideas for further research are presented in section 7.1, together with several ideas for applications. The implementation itself is available, and instructions on how to get and work with the implementation can be found in appendix B and attachment A.

The software resulting from this thesis has been packaged and released under the name “speechmatching” [148] version 1.0.0[5], which can be used with Python.

Archiving references

A major problem on the Internet is “link rot”, the decrease over time in fraction of links that are still accessible. Estimates at the time of the ‘early’ Internet, around the start of the 21st century, found that 40% to 50% of references are inaccessible after 4 years in a certain set of publications [154]. The ‘half-life’ of a link in articles in three journals published between 1997 and 2003 was found to be roughly 5 years [55]. More recently in 2015, 37% of URLs in theses defended between 2003 and 2010 were found to be not available [30]. This problem also exists in fields of law with 20% to 30% of links not being available in certain law journals and even documents published by the Supreme Court of the United States of America [192].

The way to prevent this is by using web archives to store digital data, and especially focusing on links that are at a higher risk of becoming unavailable [191]. Storing web pages is also know as web archiving, and several solutions exist that create, distribute and preserve these web archives [12]. There are some calls for the adoption of DOIs for digital resources to decrease the chances of link rot [23], [37]. This thesis uses the Wayback Machine[6] by the Internet Archive[7] to save web pages using the Save Page Now[8] tool. This is done for the web sources without DOI or ISBN identifier, both the archived version and date of archival are printed. Three resources in this thesis could only be found in online web archives, and have been quoted from there.

For those writing documents in LaTeX, BibLaTeX can make the process of mentioning an archived version easy by using a proposed solution[9] for implementing the `archiveurl{...}` and `archivedate{...}` fields for a BibTeX file.

By adopting this approach and explaining the ideas and reasons in this subsection, the author of this paper hopes to incentivize others to do the same in their reports and publications. Link rot is a problem that will never fully go away, but it can be greatly reduced, for which authors in the future will be thankful.

[5]A. Schuringa, *Speechmatching*, version 1.0.0, Jan. 2025. [Online]. Available: `https://pypi.org/project/speechmatching/1.0.0/` (visited on Feb. 22, 2025).

[6]*The Wayback Machine.* [Online]. Available: `https://web.archive.org/` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20240822085843/https://web.archive.org/` on Aug. 22, 2024.

[7]*The Internet Archive.* [Online]. Available: `https://archive.org/` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20240821003119/https://archive.org/` on Aug. 21, 2024.

[8]*Save Page Now.* [Online]. Available: `https://web.archive.org/save/` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20240925134018/https://web.archive.org/save/` on Sep. 25, 2024.

[9]moewe, *Biblatex: Add a second url where content is archived*, Jan. 2018. [Online]. Available: `https://tex.stackexchange.com/questions/302968/biblatex-add-a-second-url-where-content-is-archived/303178` (visited on Aug. 2, 2024), archived at `https://web.archive.org/web/20250110094903/https://tex.stackexchange.com/questions/302968/biblatex-add-a-second-url-where-content-is-archived/303178` on Jan. 10, 2025.

Chapter 2

Matching speech

Rare languages often do not have large amounts of recorded audio in combination with text available. This makes it difficult to train an artificial neural network to create a full model of the rare language and be able to fully interpret the language. This means that one needs to fall back to other methods of extracting information from recordings of the language, or reducing the expectations of this model.

2.1 Reducing details

One of the most spoken languages in the world is English. It is not very difficult to get a large data set of spoken English in combination with text to train a model on [35], [62], [128]. Because so much data is available, all details of the English language can be trained for, and very high scores can be achieved by artificial neural networks in tests for creating transcriptions [130] or even more in-depth understanding of the English language [170]. When less information is available, a choice between two options needs to be made. Either the scope or 'number' of supported details can be reduced, or the expectations of accuracy can be reduced

The reduced accuracy means that for example instead of a 95% accuracy over correctly recognizing certain sounds or words, an accuracy of for example 60% or less should be expected. Low accuracies are often unacceptable.

Reducing the scope of supported details can happen at the 'front' or the 'back' of the network. At the front of the network, the number of sounds or words that the network needs to recognize can be reduced. Instead of requiring the network to recognize the full language, the requirement could be to just recognize the local versions of the words 'yes' and 'no'. This means only recordings of 'yes' and 'no' are required, which reduces the overall numbers of samples greatly compared to a case in which tens of thousands of words need to be supported.

On the other hand, at the 'back' of a network one can reduce the scope of what should be supported by for example altering the form of the output. An example is that regular modern automatic speech recognition performs speech to text. This is done by directly mapping output of a neural network to probabilities for words, parts of words, or even parts of sentences, and in this way transcribe speech into a form readable for humans. This can require a lot of tokens, representing both the textual as well as the contextual aspects of speech. Reducing the amount of output may mean that certain details are not given as output by the network anymore. This in turn may mean that details required for easy interpretation of the output by humans are lacking, which makes it less likely that the output can be directly understood by humans. However, that does not mean that the output is meaningless. Output can still be interpretable for machines, instead of humans. If a machine is able to transliterate speech into a form of output that is static under the transliterations of the same word in different recordings of the same word, the machine is still able to work with the output.

Overall the complexity of the requirements for both the input, at the 'front' of the network, and that for the output, at the 'back' of the network, can be reduced at the same time. In this thesis

- the number of word supported at the same time is reduced,
- the information in the intermediate output is reduced and not well interpretable by humans, and
- finally the output is mapped to the predefined representations of supported words after which it is presented to humans.

2.2 Processing speech

The implementation in this thesis is initialized with a set of supported words. Examples are given of what these words sound like, which are translated by the implementation into some machine-interpretable transcription. Another speech recording is then given, which is also interpreted, and the machine matches the transcriptions of both pieces of speech with each other. These are compared between the various supported words and a decision is made on which word was most likely pronounced in the second recording.

2.3 Implementation steps

In the case of working with recorded speech there is a set of steps which are performed to process the data. Those steps are outlined here. To note again - these steps are designed to be used on a language from which not much data is available, which means limitations need to be introduced on the process as previously discussed. The idea is to use an existing model that is not specially made for the rare language, and alter the output of the model so that details very specific to the language the model was made for are generalized away. What is then left is a translation of speech to some generalized form that can be interpreted by a machine. The next subsections will go through the 3 steps.

2.3.1 Transcribe

The first step is extracting information from a recording of speech, for example of a word. Previously it was noted that at the 'front' of the model or algorithm, the number of supported words is reduced. This can be interpreted in two ways. The first is that the supported words are predefined and static and small in numbers, which reduces the required recordings to only those of the predefined words. The other way to interpret this is that the number of supported words is small, but not static. This means that any set of words can be chosen out of a large corpus, as long as the total set of chosen words is small.

In the first case of a static predefined set of words, a model can be trained on only these words [15], [159]. During training, the model is simply given input and output, and the artificial neural network will adjust itself to map the input as good as possible to the right output. This means that the calculations the model makes after training have been adjusted to work specifically with the given set of words. Any recording of a word not in this predefined set can be processed as well, but the result from the network will have no good relation to the recording, since the network was not explicitly trained on this word.

Training a network adjusts the (often) millions of calculations done to get to a certain output, to make this output as close to the correct result as possible. This means that the parameters of the calculations are not explicitly defined by a human, but implicitly defined through training in which a human only controls the input and output and general parameters to adjust the network with. It can be said that on a more abstract level, the artificial neural network implicitly creates criteria over the input for a certain output to be given. For a human it is therefore not always clear what these criteria are, or how the network exactly decides on an outcome. This is also described as the effect of a 'black box' with artificial neural networks [29], [153].

In the second case of a limited set of words that are not predefined, a neural network cannot be trained easily. If the set of words is not static, it may be different for each use case, and each use case would require a new model to be trained, if the method from the first case is adopted. For each model there need to be significant numbers of samples from each of the words that should be supported. This quickly becomes too expensive, forcing limits on the 'pool' of words from which combinations are chosen.

A solution would be to use classical methods that do not use neural networks, or use an existing neural network that closely aligns with a common aspect of the data that is being worked with. In the case of this thesis, an already trained network is used. This network is trained with data of a language with similar phonetic properties as the rare language, though there are differences.

2.3.2 Generalize

To handle the differences between the data the network was trained on and the data on which it is used in this thesis, the details in the output of the network that are very characteristic of the data it was trained for need to be removed.

As noted in section 1.1, languages usually have similar features. These similar features can be in the form of high level abstract concepts that are seen in general structures of languages, but also in less abstract concepts like the sounds themselves. Phonetic sounds are often shared or very similar across languages [85]. This means that in the case of two languages and their phonemes, a set can be found that contains the phonemes that are shared between both languages. This allows for a model trained on phonemes in one language to be used for another language, which shares phonemes with the first language. In practice, phonetic sounds are not exactly the same between languages but can have a strong overlap. This means that one language may have some similar phonemes that are perceived to have different meanings in that language, while they are perceived to have the same meaning for speakers of another languages. These kinds of similar sounding but different phonemes would need to be generalized over - mapping them to a single representation, so the similar phonemes are correctly interpreted as having the same meaning in the other language. This process shows how a 'baseline' in the form of phonetic sounds can be found that allows a phonetic model of one language to be used as the phonetic model of another language.

Abstract explanation of mapping phonetic sounds between languages

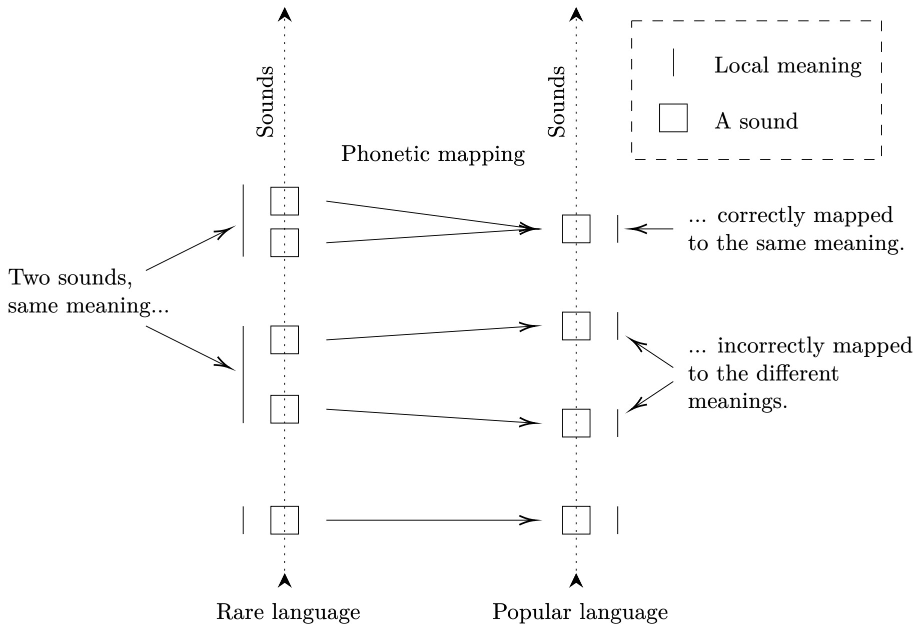

Sounds
Sounds
Local meaning
A sound
Phonetic mapping
Two sounds, same meaning...
... correctly mapped to the same meaning.
... incorrectly mapped to the different meanings.
Rare language
Popular language

Figure 2.1: An abstract drawing showing the acoustic sound 'lines' of a rare language on the left and a popular language to the right. The cubes are sounds on the abstract 'acoustic spectrum'. The solid lines next to the cubes are the local meanings. Two meanings are shown as each having two different sounds on the left line, possibly due to dialects or accents. One set of two sounds is correctly mapped on the right to a single meaning, while the other one is incorrectly mapped to two meanings.

Figure 2.1 shows an example of sounds and their meanings in different languages. The figure contains two examples of a 'meaning' in a rare language on the left being defined in two different sounds each. Differences in sounds of words or characters can happen when different accents are spoken, but those different pronunciations may still have the same meaning as is the case in this figure on the left line. Groups of people can develop accents on the main spoken language, which can create small but audible differences in the pronunciation of words [42]. These sounds should ideally be mapped in the other language to the same acoustic meaning. In figure 2.1, this goes well in the upper case, while the middle case shows the two sounds of the same meaning in the rare language being mapped to two different meanings in the popular language.

As details are generalized away, some information is lost. As a consequence, similar sounding but different words may be transcribed as the same word, which has the potential to decrease accuracy. Next to that, it will also be more difficult for humans to interpret the transcribed generalized details, as details that would make it interpretable in the original language of the model are taken out, while details that are left may not map easily to the letters or characters a word in the other language would be written with.

2.3.3 Match

After generalizing the strings into a set of characters that cover sounds that are close to each other, they can be matched with each other. The idea is that strings of recordings from which the spoken word is known are matched to the string of a recording from the which the spoken word is unknown. Then, by matching the strings with each other, the recording with unknown word can be matched to a known word. In that way the word that is spoken can be identified and further used.

Recordings of phonemes and words can have differences, even if they are spoken by the same person, meaning that the acoustics spoken by the person are the same between recordings. One difference can be the speed at which a word is spoken, and another may be the noise in the recording.

In the case of a different speed of pronouncing a word, the phonemes will be more stretched out in time, which means the characters that represent them are further away in time, or duplicated over several time steps. This needs to be handled when matching the strings representing the recordings with each other. In the other case of noise in the recording, this needs to be handled as well. The noise can introduce characters in the representation of the recording that do not belong to the word spoken in the recording. This requires the algorithms for matching strings to not statically check characters one after the other if they are matching, but require a more dynamic approach to handle any 'extra' unexpected characters in the strings.

Further, a combination between the two effects can be problematic. The impact of noise on longer pronunciations of words may cause additional characters to be introduced. As a first example, the sound for `A` is pronounced fast. If noise happens at a certain rate, then there is a chance that the noise happen right before the `A` or right after it. Representing the noise with `X`, this may then result in the representations `AX` or `XA`, in which case a single character is introduced by the noise. Now, take the case of a long pronounced `A`. If the noise still happens at the same rate as before, there is now an increased chance for the noise to appear in the middle of pronouncing the `A`, which may now also cause the representation `AXA` to be created. In this case, two characters are introduced due to the noise instead of one.

Various algorithms exist to handle matching of strings, which are introduced in the next section.

Chapter 3

Theory

The theory behind the three steps outlined in section 2.3 is described in this section. Because these steps can be implemented in different ways, the general explanation was separated out into the previous section, while the current section focuses on the theory behind the implementation choices in this thesis. While the implementation in this thesis gives reasonable results, different implementations of each of the steps may improve results, as will later be outlined in section 7.

3.1 Transcribe: English phonemes

As previously shortly explained, phonemes are often shared across languages. For that reason it is possible to use a model trained on a popular language for a language that is not popular. This is, as long as both languages have a reasonable amount of overlap in their phonemes. In the case of this thesis, the English language is used. Using Wav2Letter++ [127] recorded speech is transformed into letters corresponding to phonemes in the English language, also known as an English phonetic transcription. Wav2Letter++ is a further iteration on the original Wav2Letter [36].

The design of an artificial neural network is not the main objective of this thesis. An existing pre-trained neural network is used, and more 'classical' methods are used on its output. For this reason, no in-depth description and discussion will be written on either artificial neural networks in general or the specifics of the neural network used in this thesis. The very basics behind neural network will be introduced and each of the techniques used in the pre-trained model will be explained in short form. This is done so that a very basic understanding of neural networks is created, and the reader has some idea of how the pre-trained network works.

3.1.1 Artificial neural networks

Artificial neural networks perform calculations over connected nodes inspired by how the human brains work [137]. In the brain, neurons fire signals to each other, and in that way move data from input to output. Neurons can trigger different intensities of signals, and with billions of neurons a complex system arises. The artificial neural network works with roughly the same idea [14], although it does not fully approach it [145], the human brain is still being researched and discoveries are being made constantly [149], [186].

Basic structure

The basic structure of a neural network will now be explained, by building up a fully connected network from its building blocks.

A node in an artificial neural network represents a neuron, and performs a calculation

$$f\left(\vec{x}\right) = \vec{x} \cdot \vec{w} + b, \tag{3.1}$$

where $\vec{x}$ is the input in the form of a vector of values, $\vec{w}$ is the vector of the same size that multiplies its values with the values in $\vec{x}$ and adds them up, after which a bias b is added. This result then goes through an activation layer [151] which is modeled after the chance of a neuron to fire a signal depending on the input. If the input reaches a certain threshold, the neuron fires and else it does not. In the brain the neuron always fires at the same intensity, where the amount of energy that is passed on over a certain amount of time depends on how often a neuron is fired. For a greater intensity the neuron is fired more often than for a lower intensity.

As an example of an activation layer, a typical one is

$$f\left(x\right) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{else,} \end{cases} \tag{3.2}$$

which is the ReLU activation layer [3], or Rectified Linear Unit, which forwards input x if it is positive, meaning it 'fires', and else ignores the input. The threshold in this case is thus set at 0 to determine on firing or not. The output is equal to the input when positive, which is not biologically correct as the biological neuron only support forwarding a certain amount of 'energy' for each pulse.

Another example is the Sigmoid activation function [165] which calculates a value between 0 and 1 from any input value x with

$$\sigma\left(x\right) = \frac{1}{1 + \exp\left(-x\right)}. \tag{3.3}$$

This is different from the ReLU activation function in that it does not simply pass on the input value, but has a limit to how much 'energy' can be passed on. It requires a much higher input value to output a slightly higher value. Next to the ReLU and Sigmoid activation layers, many other activation layers exist. Which one to use depends on both the type of input and output data.

Taking the information on a linear layer and an activation function together, they can be chained and a network can be created with many layers, each of which passes on information to the next. An example of such a network is shown in figure 3.1. Which shows nodes as circles and connections between them as lines between the circles. Each node takes a number of connection, which is the vector $\vec{x}$ in formula 3.1 and outputs a vector that travels on to connected nodes for further calculations. The connections themselves pass through activation layers, in this case the ReLU activation function, with which the values on the

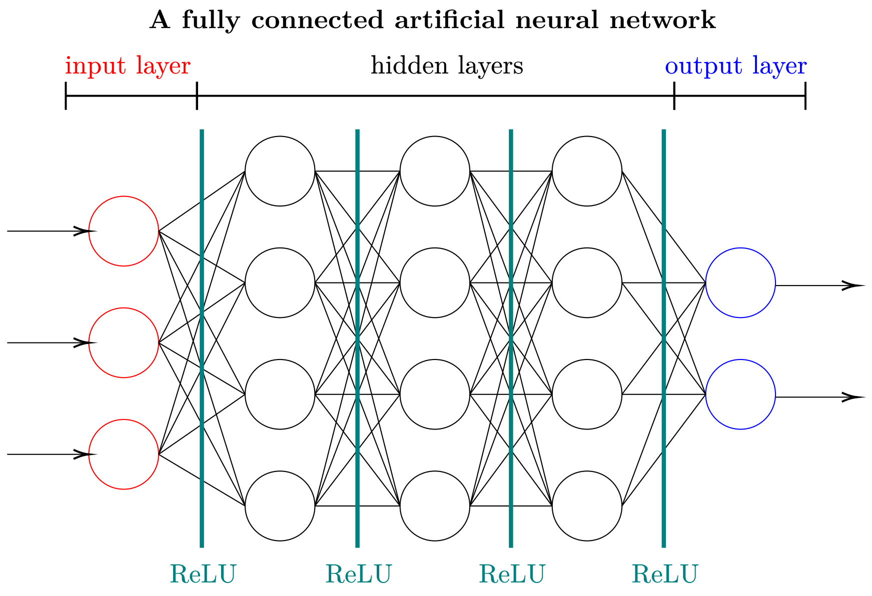


Figure 3.1: A simple example of an artificial neural network with fully connected layers, the ReLU activation function, and input and output nodes on the left and right sides respectively.

connections are changed. On the left of the network are the input nodes in red, and on the right the output nodes in blue.

In reality a network nearly never looks as plain as the network in figure 3.1. They can be very complex, with layers that branch off and join the network at a later stage [136], transformers [176], convolutional layers [89] that work on sets of neighboring nodes, or many more. This is an active fields of development. The important techniques for the model used in this thesis will be explained later.

Training

In order for an artificial neural network to be useful, it needs to have weights $\vec{w}$ and biases b from formula 3.1 with values that make the network perform calculations in such a way that it produces output that is meaningful in the context of the problem that is given to the network in the form of input data. This is done using training.

Training a neural network involves feeding data into the network, retrieving the output data, determining how far off the output data is from what is expected, and then adjusting the values in the network to give an outcome that is closer to the expected output. This means that during training both the input and output data need to be known, and only after training can input data be used for which the output is unknown. An untrained network starts out with predefined or random values.

The difference between output of the network and the expected output is used in the calculation of a loss $\mathcal{L}_\phi(x)$ [56], with ϕ the parameters of the network and x the data over which the loss is calculated. The negative slope of this loss shows the direction in which parameters of the network need to be adjusted to create a smaller loss on a next iteration. For each parameter a direction

$$\delta\phi_i = -\frac{\partial \mathcal{L}_\phi(x)}{\partial \phi_i}, \tag{3.4}$$

is calculated, with ϕ_i a parameter of the network. For ϕ_i, $\frac{\partial \mathcal{L}_\phi(x)}{\partial \phi_i}$ can be calculated using backpropagation [138], [182]. All calculations since ϕ_i leading up to the output are known. Taking the derivative of these calculations can be done using the chain rule [10], [91]

$$(f \circ g)' = (f' \circ g) \cdot g', \tag{3.5}$$

where $f(x)$ and $g(x)$ are for example those defined previously in formulas 3.1, 3.2, and/or 3.3. If the used formulas have simple derivatives, calculating the slope of the loss for each parameter can be done relatively cheap. The output of the network can be defined with previous definitions as

$$f_\phi(x),$$

where f is the network with parameters ϕ which receives input data x.

Knowing the slope of the loss at a parameter makes it possible to adjust this parameter with a learning rate γ [72], [95] as

$$\phi_i^* = \phi_i + \gamma \cdot \delta\phi_i, \tag{3.6}$$

where ϕ_i^* represents the adjusted parameter ϕ_i. The learning rate γ needs to be adjusted according to the shape of the loss landscape around the parameters. A loss landscape [93] is the loss around a parameter depending on the value of that parameter. At a certain change of the parameter the loss may become lower, but at a certain larger change of the parameter the loss may become higher again, as the loss can take on any smooth curves along the values of the parameter.

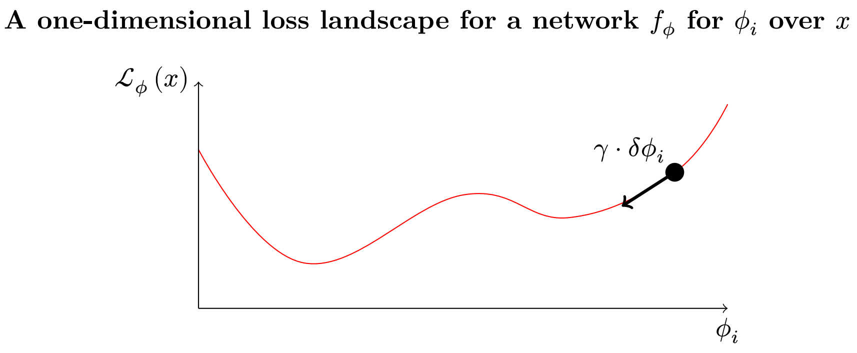


Figure 3.2: An image of a one-dimensional loss landscape over a single parameter ϕ_i in the set of parameters ϕ of the network f_ϕ when using data x.

A one-dimensional example is shown in in figure 3.2. This plot shows two aspects of the loss landscape. The first is that there is a local and a global minimum. As the large black point in the graph moves towards the closest minimum, it will find itself in a local minimum and cannot find the global minimum, unless the value of the learning rate γ is large enough to overcome the ‘bump’ in the loss between the two minima and reach the global minimum. Second is that there are clearly higher and lower regions in the loss landscape, which means that the learning rate γ can also not be too large, as the loss will then jump around some minimum, but never actually reach the minimum. The learning rate thus needs to be adjusted to some range of values that optimize the loss without over- or undershooting the changes in parameter values. This can be a time consuming task. In higher dimensions, the loss landscape is much more complex and is not as intuitive as figure 3.2 might suggest [26], [51], [179].

3.1.2 Feature extraction

Raw data is not often directly fed into a neural network because it either is not in the correct form to be given to the input nodes of the network, or because it is better for training if the network is ‘helped’ and the data is transformed into a form which helps the network focus on the important aspects of the data [56]. This transformation of the data is also known as the process of extracting features, which are then given to the network.

In the case of the network used in this thesis, the raw data is an audio recording. It is cut into pieces of 25 milliseconds, with a hop length of 10 milliseconds. This means the start of a piece is 10 milliseconds after the start of the previous piece, and both pieces have a length of 25 milliseconds. Mel-Frequency Spectral Coefficient (MFSC) features [36] are computed over each segment of audio. This is done by performing a set of steps for each segment similar to those performed for calculating Mel-Frequency Cepstrum Coefficient (MFCC) features [1], but for the calculation of MFSC features the last step is left out [87]:

1. Calculating the Fourier transform over the segments. This method calculates the frequencies in the sound wave and the amount that these frequencies are present in the sample. The properties of the frequencies can be inferred from the values in the audio signal, that create a wave like pattern.
2. The frequencies are transformed to the Mel spectrum. This spectrum changes the distribution of the frequencies from one in which every frequency is equally represented on the scale of frequencies, to one in which values are represented according to how much they are perceived by humans. This makes sure that the frequencies found in the audio that humans do not hear are taken out, while the frequencies more significant to humans become more visible.
3. The logarithms are taken of the frequencies. This is done so that very small amplitudes become more visible compared to the very large amplitudes for frequencies.

From the listed steps, there is a clear reason why the MFSC over each segment is calculated. This is done to make the information that is more significant to humans better visible to further calculations in the neural network, while making irrelevant information much less visible or invisible to the network. This allows the network to better focus on the aspects of the audio most important to its task, and it will not waste resources trying to optimize using irrelevant parts of the data.

3.1.3 Structure

After a very short introduction to artificial neural networks, the network used in this report is explained. This is again not done in great detailed, but up to a more abstract intuitive level. Please see the referenced papers for more details.

The network consists of various techniques layered on top of each other. Output from one layer is sent to the next layer. In order, the used layers are the following. The network is given as input the MFSC features over each segment of audio as explained in the previous section. In then performs the following actions.

1. **Layer Normalization** [16]: This technique normalizes the values given to the neural network before further processing it, similar to batch normalization [64]. An 'object' going into a network usually consists of multiple values representing in. Each of these values is a 'feature' of the represented object. Multiple objects can be given to the network at the same time in form of a 'batch', which can be seen as a list of the objects. In batch normalization, each feature of an object is normalized using a mean and standard deviation of the features in the same position of all objects in the batch. Layer normalization instead normalizes each item in the batch individually, with a mean and standard deviation calculated over all features in the batch together. This means that all features need to be of the same type, which they are in this case, since the features in this case are values from the image created using the MFSC transform as earlier explained.
2. **Convolutional layer** [90], [146]: This method is based on biological processes [69] - like much of artificial intelligence. In the case of an image of 100 by 100 pixels, there are a total of 10000 values. Performing calculations over all of these values in various combinations is expensive, and often unnecessary. From an image, the values from individual pixels are often not very important, but their values in the context of neighboring pixels form important patterns. These patterns are often more useful than individual pixel values. In the case of a two dimensional image, the network itself has various sets of smaller images, which are optimized during training. These smaller images are then taken across the pixels in the input image, and all values in the smaller images are multiplied at position with the values seen in the image, leading to output values. The network adjusts these smaller images using backpropagation, so that they represent a pattern. The output of a calculation between the pattern and a certain position in the input image then represent the relation between this pattern and the image at that location. This quantified relation is then further used in the network. An example is one of the first implementations of a convolutional layer, in the neocognitron [52], which extracts data from regions of an image, and optimizes the pattern with which this data is extracted through rounds of training.
3. **Gated Linear Unit** (**GLU**) [41]: The GLU is an activation function. It is based on the older Long Short Term Memory (LSTM) network [66],

which works on sequences of data. While processing this data, the LSTM keeps a matrix of information that is gradually adjusted as more data passes through the network. The information in this matrix functions as a form of ‘memory’ of aspects of the processed data. The information in this memory can then be used for calculations over next sets of data in the sequence. The GLU activation functions works in a related way. Internally it works with two linear layers, which both process the incoming data. One result is then multiplied with the with the other result taken through the Sigmoid function. When taking the definition from formula 3.1, this looks like

$$f(\vec{x}) = (\vec{x} \cdot \vec{v} + a) \cdot \frac{1}{1 + \exp\left(-\left(\vec{x} \cdot \vec{w} + b\right)\right)}.$$

Here, the two linear functions are clearly visible, which both use a different set of weights. Using the GLU, the network can learn to use the given information to determine which information to pass on to the next layer.

4. **Dropout layer** [65]: A layer that sets a value to 0 with a certain probability. In this case with a probability of 0.05. The idea, and effect, of this is that it reduces overfitting. The data a network is trained on is in most cases not unlimited, especially if the data is not purely simulated. During training, weights in the network are adjusted using backpropagation by using the difference between what the network gave as output and what the expected output is. The network attempts to make better predictions for the training data. However, the network is later used on data that was not seen during training, so the network should not only adapt to the training data, but find patterns that exists in both training and non-training data. With a limited amount of training data, the network may start to find patterns in the data that are specific for the training data alone, which is called ‘overfitting’. Overfitting will make the network produce good results for the training data, but not for data that was not included in the training data. Using dropout layers the presence of this problem in the final trained network is reduced [156].
5. 26 layers of **Transformers** [176]: The transformer architecture makes use of the attention mechanism introduced by Vaswani, Shazeer, Parmar, *et al.* [176]. The network itself consists of an encoder part and a decoder part. In this text only an abstract version of the transformer architecture is discussed, the network consists of more details with implications not discussed here. Overall, the transformer architecture works on sequences of data, for both input and output. An example is text. Text is split into pieces and fed to the encoder, which first creates embedding vectors from the input. These embedding vectors are an abstract representation of the input data, in a format that can be used in matrix calculations in the network.
   These embedding vectors are then processed using an encoder with multiple layers of self-attention and multi-layered linear networks. During self-attention, a calculation is performed using three matrices - two matrices are used to construct a matrix from which each vector decides how much ‘attention’ is given to each input vector. The result is an output matrix that goes through several linear layers for further processing. The output from these linear layers can be fed to another encoder performing

the same type of calculations, or it can be forwarded to the decoder part of the transformer.

A decoder takes as input (some of the) data previously calculated in the output sequence, and the output from the encoder. The output from the encoder represents the next part in the input sequence. The decoder performs self-attention on the previous vectors in the output sequence, after which cross-attention is performed using the output from the encoder. Cross-attention thus 'mixes' the previous output information and new input information together for the next chunk of output to be calculated. Cross-attention works in the same way as self-attention, only the two matrices that are used to construct the attention matrix come from the encoder, instead of its own input. The result from cross-attention is fed through several layers of fully connected linear layers, and the output is forwarded to another decoder block, or is used in the next data in the output sequence.

After one or more decoder blocks, the output from the last decoder block is used to construct output by un-embedding the output vectors into probabilities of certain output. Essentially the inverse operation is performed from the embedding that happened before input data was fed into the first encoder block.

6. **Linear layer**: This was discussed before around formula 3.1. In this case the layer has 382 input nodes and 29 output nodes, with each output node corresponding to a probability for a character in a time step.

3.1.4 Characters

The final fully connected linear layer in the network calculates 29 output values. These 29 values are related to the probabilities of a certain character. From values 1 to 29, the set $\mathcal{C}$ of characters are shown in table 3.1 from left to right and up to down, where the characters from the Latin alphabet are shown, together with characters `, | and #. Here, | represents a separation between words, and # a 'blank' space, meaning neither a character was identified nor a separation between words.

Table 3.1: The set $\mathcal{C}$ of characters the network calculates probabilities for, which is the set of Latin characters and three others.

'	|	A	B	C	D
E	F	G	H	I	J
K	L	M	N	O	P
Q	R	S	T	U	V
W	X	Y	Z	#	

For each time step, for each of the 29 characters a value is calculated. This value represents the unnormalized exponential of the probability of a character to exist here. To retrieve the normalized probabilities,

$$p_\phi\left(x\right)_{t,c} = \frac{\exp\left(f_\phi\left(x\right)_{t,c}\right)}{\sum_{c'\in\mathcal{C}}\exp\left(f_\phi\left(x\right)_{t,c'}\right)} \tag{3.7}$$

is calculated, where $f_\phi(x)_{t,c}$ is the output for the network with weights ϕ, and input x, and where t is the step in time in the recording, and c is a character of the supported characters $\mathcal{C}$ shown in table 3.1.

The probabilities from the network can now be used to calculate strings or texts that represent the recording. Further in the thesis, a predicted text will be written as $\vec{x}$.

3.2 Generalise: phonetic encoding

The model for the English language, a popular language, is used on a different rare language. The letters this model calculates over the audio are fitted for the English language, and not the target language. If both languages have sufficient overlap in phonetic characteristics this should not be a problem, however there are often still some sounds that are unique to a language, or have a slightly different sound in different languages or accents. For this reason the strings need to be normalized, to make the output more 'general', and details very specific to the English language or based on context need to be filtered out. In this section three algorithms are discussed that perform this kind of operation. In this thesis, the methods of generalization will be mathematically defined as

$$\mathrm{g}_n(\vec{x}), \tag{3.8}$$

where $\vec{x}$ is the 'vector' of characters as received from the method in section 3.1 and n is one of the following three discussed methods

$$\begin{aligned} n \in \mathcal{N} = \{&\text{Soundex}, \\ &\text{NYSIIS}, \\ &\text{Metaphone}\}. \end{aligned} \tag{3.9}$$

3.2.1 Soundex

Soundex [119] is an algorithm developed to transform words into a description of their phonetic form. A first version was created in 1918 [139]–[141]. The idea behind Soundex is to map similar sounds to an equal character. The author argues that the sounds of the English language are not represented well with letters. Instead, the author creates phonetic groups of letters which allowed for English words to be formed into their phonetic representations. This was initially used to translate human names into a form that would be the same across different spellings of the same name.

The algorithm

The rules of Soundex have slightly changed over time, but are currently defined as follows [114]. The final form has a length of four character as `LDDD`, with the first character `L` a letter, and the last three characters `D` digits. The rules are that

1. the first letter of the word is taken as the first character;
2. the letters `A`, `E`, `I`, `O`, `U`, `H`, `W`, and `Y` are removed from the word;
3. any consecutive double letters are removed;

4. a digit is added until 3 digits have been added or until no more characters are left, for each remaining character in the word, where the digit is
    - `1` in case of `B`, `F`, `P`, or `V`;
    - `2` in case of `C`, `G`, `J`, `K`, `Q`, `S`, `X`, or `Z`;
    - `3` in case of `D` or `T`;
    - `4` in case of `L`;
    - `5` in case of `M` or `N`;
    - `6` in case of `R`;

    and a digit is skipped if it was equal to the previous digit, unless the character from the equal digit was separated from the previous character in the original word by either `A`, `E`, `I`, `O`, or `U`;
5. if less than three digits were added, the digit `0` is added until 3 digits in total have been added.

Evaluation

As an example, the word `SPEECH` would be transformed to `S120`, which can be explained as

- `S`: The word starts with and `S`;
- `1`: After ignoring of `A`, `E`, `I`, `O`, `U`, `H`, `W`, and `Y`, the next letter is a `P` which is assigned number `1`;
- `2`: The next letter is the `C`, which is assigned number `2`; and
- `0`: There is no further letter, so the remaining places are filled with `0`.

This allows for similar sounding letters to be encoded to the same digit, for example `D` and `T`, while the first character of the word is always encoded directly into the final result since this is usually pronounced very distinctively.

The Soundex algorithm will cause the words `THREE` and `TRUE` to be encoded to the same representation `T600`. Whether this is correct is debatable, as the words 'three' and 'true' are pronounced differently, and are clearly separable. And, in the case of two words that do sound very similar, the algorithm does not always produce satisfactory results. An example is with the similar sounding words `WEIGHT` and `WAIT` are encoded into respectively the different forms `W230` and `W300`, while they are very similar in sound. It should be noted that the algorithm was created specifically for processing written names, and not words in general which would explain some of the unexpected results.

Two obvious downsides to the Soundex algorithm are that the output is not easily interpretable by humans, which in the case of this thesis is not a problem, and that Soundex only allows for up to 3 digits in its output, and any digits that would be added for other characters are skipped. This may cause the algorithm to skip large parts of the longer words.

Other languages

The algorithm is based on the English language, but has also been adopted for other languages. One example is the Daitch-Mokotoff Soundex algorithm [110] which is designed to work with Slavic and Germanic names. Instead of a form `LDDD`, this uses a form `DDDDDD` of six digits.

3.2.2 NYSIIS

An alternative to the Soundex algorithm is the New York State Identification and Intelligence System (NYSIIS) algorithm [164], which shows an improvements over the Soundex system [133]. Where Soundex creates a more abstract representation of a word by keeping only the first letter of the word and encoding the rest of the word into numbers, NYSIIS creates a more human interpretable version. Similar to Soundex, the algorithm was created for indexing names, and not words in general although it can be used for that.

The algorithm

The algorithm creates an output of a length depending on the length of the word the algorithm is run on. This means that unlike Soundex, there is no limit on the length of the encoded result. The algorithm works works by

1. replacing the character sets `MAC` by `MCC`, `KN` by `NN`, `K` by `C`, `PH` and `PF` by `FF`, and `SCH` by `SSS`;
2. replacing the last letters of the word if they are `EE` or `IE` by `Yƀ`[10] and if they are `DT`, `RT`, `RD`, `NT`, or `ND` by `Dƀ`;
3. taking the first letter as the first character in the NYSIIS code;
4. choosing and performing one of the following actions on the next character;
   - going to step `7` if no next letter follows in the word or if this character is `ƀ`;
   - changing the current two letters to `AF` if they are `EV`, and else to `A` if they are one of `E`, `I`, `O`, or `U`;
   - replacing the current letter `Q` by `G`, `Z` by `S`, or `M` by `N`;
   - replacing `K` by `N` if the character after it is `N`, and else by `C`;
   - replacing current letter `SCH` by `SSS`, or `PH` by `FF`;
   - replacing the current letter `H` by the previous letter if the previous or next letter is not one of `A`, `E`, `I`, `O`, or `U`;
   - replacing the current character `W` by the previous letter if the previous letter is one of `A`, `E`, `I`, `O`, or `U`; or
   - do nothing;
5. going to step 4 if the current letter is the same as the letter last added to the NYSIIS code;
6. taking the current character as the next character in the NYSIIS code, and go to step 4;[11]
7. removing the last letter of the NYSIIS code if it is the character `S`;
8. replacing `AY` by `Y` if the NYSIIS code ends with that; and
9. removing the last letter of the NYSIIS code if is the character `A`.

Evaluation

Using the NYSIIS algorithm, the words processed as example in section 3.2.1 can be evaluated again. The words `THREE` and `TRUE` were assigned the same string by Soundex. For NYSIIS, they are not assigned the same string, but rather

[10] Here the special character ƀ is used according to the original algorithm.

[11] Note that this creates a loop going back to step 4 until all characters in the original word are processed.

THREE is encoded as TRY, and TRUE is encoded as TR. This can be argued to be more correct output than what Soundex gave, as the two words are pronounced very differently.

The other example is with two words that sound very similar when pronounced, and can be claimed to indistinguishable without context. The NYSIIS algorithm encodes the string WEIGHT as WAGT, and the string WAIT as WAT. While the expectation would be that the two words are encoded to the same representation, they are not. However, they differ by only one character, with the algorithm replacing EI and AI by A and removing the H, which makes both representations very similar to each other, with only one character difference. It should again be noted that NYSIIS was designed for processing written names, similar to Soundex, which may explain the imperfect results.

3.2.3 Metaphone

The algorithms of Soundex and NYSIIS are made for American names. While they can be applied to words in general, and can output a meaningful representation, they may make mistakes due to this. An alternative to these algorithms is the Metaphone algorithm [22], [124]. The algorithm is more similar to NYSIIS than to Soundex, in the way that it creates a phonetic output of arbitrary length that is better interpretable by humans than the output of the Soundex algorithm. However, Metaphone preserves fewer character from the word than NYSIIS, which causes it to lose more information on the one hand, but on the other hand may create a more general phonetic version of a word than NYSIIS, which could lead to better results when further used in this thesis.

The algorithm

The algorithm can be simplified from the original more complex algorithm. This simplified algorithm is shown here. The algorithm encodes a word into a combination of 16 characters, namely B, X, S, K, J, T, F, H, L, M, N, P, R, 0, W, and Y. A word is transformed into a phonetic representation by [135]

1. reducing every double character to a single character, except for the character C;
2. dropping the first character of the word if it begins with KN, GN, PN, AE, or WR;
3. dropping the last character of the word if the two last characters are MB;
4. replacing C by X in every substring CIA or CH, or by S in every substring CI, CE or CY, and else by K;
5. replacing D by J in DGE, DGY, or DGI, or else by T;
6. removing G in GH;
7. removing G from GN or GNED if the word does not end with that and they are not followed by a vowel;
8. replacing G by J if not in GG and if followed by I, E, or Y, or else replace G by K;
9. removing H after a vowel if there is no vowel following after it;
10. removing K when it follows after C;
11. replacing P by F if in PH, and Q by K, V by F, S by X if in SH, SIO, or SIA;
12. replacing T by X if in TIA or TIO, or T by 0 if in TH, or removing it if in TCH;

13. removing `H` from substring `WH` if the word start with that, and removing `W` as well if no vowel follows;
14. replacing `X` by `S` if it is the first character of the word, or else replacing `X` by `KS`;
15. removing `Y` if it is not followed by a vowel;
16. replacing `Z` by `S`; and
17. removing any vowel that is not the first letter of the word.

Evaluation

For the strings `THREE` and `TRUE` that were also processed with Soundex and NYSIIS, the outcome of the Metaphone algorithm for them is respectively `OR` and `TR`, which is again different from each other similar to NYSIIS, which would be correct as these words sounds different to humans. However, in this case they are different due to different pronunciations of the `THR[...]` and `TR[...]` parts of the words, instead of the different pronunciations of parts `[...]REE` and `[...]RUE`.

On the other hand, both Soundex and NYSIIS encoded the strings `WEIGHT` and `WAIT` differently, while they sound similar. Metaphone in this case encodes them both as `WT`, which is a better result than the other algorithms as the words sound the same. It should be noted however on the other hand that this is also a shorter encoded version than either NYSIIS or Soundex gave, which means that less information is preserved.

Metaphone performing better on some strings likely is because it is, unlike the other two algorithms, not made with only human names in mind. However, it may be that information valuable for later steps would be available through the Soundex and NYSIIS algorithms, while it would not be available through the Metaphone algorithm, since the Metaphone algorithm calculated shorter encoded versions. This depends on the processing in the next step, in which string are matched with each other.

Newer versions

Next to the first Metaphone algorithm, a newer version was released 10 years later called the Double Metaphone [125]. The new algorithm attempts to better account for various ways in which words (and particularly names) can be pronounced. It calculates two encoded versions for words, hence the name Double Metaphone, and allows for both cases to be used in comparisons. If one case does not fit, the other may fit. This version also attempts to handle various languages other than English, and has a much more complex algorithm than the original Metaphone.

A third version of the Metaphone was released nearly 10 years after Double Metaphone, but was this time released commercially [126]. It is named Metaphone 3 and improves phonetic processing of words more, achieving better performance in English and other languages.

3.3 Matching: calculating similarity

After receiving the phonetic description of a recording, it needs to be compared to other phonetic transcriptions to see which other phonetic description most likely fits. This can be done with algorithms that calculate either a 'distance' between two strings, or those that calculate a 'similarity' score. In case of the calculation of a 'distance', an integer number is calculated which represents the distance between two strings. In the case of a 'similarity', a value is not calculated of how unequal two string are, but how equal they are. This is usually a score between 0 and 1, where 0 represents no similarities between the strings, and 1 represents completely equal strings.

While the definitions are different, and the algorithms for their calculation can be different too, they can be used in the same way to compare transcriptions of words. In the case of a distance measure, the description of the recording that has the lowest number in relation to the given unknown recording matches best. In the case of a similarity score, the recording with instead the highest value in relation to the given recording matches best.

Several algorithms will now be introduced and discussed, with the first one being the most simple algorithm, and the last two more complicated.

3.3.1 Hamming Distance

The Hamming distance [60] is one of the most basic algorithms to calculate a distance between two strings. The Hamming distance is defined as

$$\mathrm{d}_{\mathrm{Hamming}}\left(\vec{u}, \vec{v}\right) = \sharp\{i : u_i \neq v_i, i = i, \ldots, n\}, \tag{3.10}$$

where $\vec{u}$ and $\vec{v}$ are two vectors, usually words, and n is the number of values in these vectors to compare. Here $\sharp$ is used to denote the length of a set instead of the more popular $|\ldots|$, since $|\ldots|$ is later used to calculate an absolute value, so $\sharp$ is used to prevent confusion. Usually n is defined by the length of $\vec{u} = (u_1, \ldots, u_n)$ and $\vec{v} = (v_1, \ldots, v_n)$ with $n = \dim(\vec{u}) = \dim(\vec{v})$. This raises the issue of what happens when vectors $\vec{u}$ and $\vec{v}$ are not of the same length n. How to handle this depends on the implementation, but the usual choices are to

- refuse the calculation;
- set $n = \dim(\vec{u})$ if $\dim(\vec{u}) \leq \dim(\vec{v})$ or else $n = \dim(\vec{v})$ and add $a = |\dim(\vec{u}) - \dim(\vec{v})|$ to the calculated distance; or
- fill the smallest vector up with a string not present in the other vector, so $n = \dim(\vec{u}) = \dim(\vec{v})$.

In the case of this thesis, the third option will be used.

Evaluation

The Hamming distance was originally created to detect errors in signals. As an example, if on one end the string `WHAT` is sent and on the other end of some communication channel the string `WBAT` is received, then the Hamming distance as defined in formula 3.10 is

$$\mathrm{d}_{\mathrm{Hamming}}\left([\mathtt{W}, \mathtt{H}, \mathtt{A}, \mathtt{T}], [\mathtt{W}, \mathtt{B}, \mathtt{A}, \mathtt{T}]\right) = 1.$$

For use outside of the scope of error correction, the Hamming distance has mixed outcomes. For example, in the case of NYSIIS in section 3.2.2, the strings `WEIGHT` and `WAIT` were encoded as respectively `WAGT` and `WAT`. By comparison by eye, one can note there is one extra character in the `WAGT` compared to `WAT`, and the distance between the two words may be best set to 1. However, in this case the Hamming distance gives

$$\mathrm{d}_{\mathrm{Hamming}}\left([\mathtt{W},\mathtt{A},\mathtt{G},\mathtt{T}],[\mathtt{W},\mathtt{A},\mathtt{T}]\right)=2,$$

since it will find the first two character are equal to each other, but the third characters are not with $\mathtt{G} \neq \mathtt{T}$, and a fourth character only exists in the first vector of characters, which is counted as an added number to the distance.

3.3.2 Levenshtein Distance

A different algorithm that measures the distance between two words is the Levenshtein distance by Levenshtein *et al.* [92], from which the Damerau-Levenshtein distance is also defined. Both are discussed in this section.

Levenshtein Distance

Introduced in 1965 and named after the person introducing it [193], the Levenshtein distance calculates a distance between two strings. As opposed to the Hamming distance, it does not simply check one by one which characters are not equal, but works with the idea of counting the 'changes' that need to be made to one string to make it the same as the other string. These changes can be one of three in the Levenshtein algorithm, either

- insertion of a character in one of two compared strings,
- deletion of a character, or
- substitution of a character for another.

The Levenshtein distance can be written in the form of a number of rules. Given the two strings or vectors $\vec{u}$ and $\vec{v}$, where $\vec{u}_i$ and $\vec{v}_j$ represent respectively the characters at locations i and j in $\vec{u}$ and $\vec{v}$, and where $\vec{u}_0$ is the first characters in $\vec{u}$, a distance D can be calculated by an algorithm $\mathrm{d}_{\mathrm{Levenshtein}}\left(\vec{u},\vec{v}\right)=\mathrm{d}_{\mathrm{L}}\left(\vec{u},\vec{v}\right)$, which works by

1. initializing the distance $D=0$;
2. adding to the distance $D=D+\dim\left(\vec{u}\right)$ if $\dim\left(\vec{v}\right)=0$, or setting the score to $D=D+\dim\left(\vec{v}\right)$ if $\dim\left(\vec{u}\right)=0$, and going to step 5 if one of the conditions was found to be true;
3. if $\vec{u}_0=\vec{v}_0$, then removing the first characters from vectors $\vec{u}$ and $\vec{v}$ and going to step 2, or else going to the next step;
4. adding $D=D+1$, and adding to D the smallest Levenshtein distance of the following three options in which the Levenshtein distance is calculated as either
   - adding the Levenshtein distance to D of vectors $\vec{u}$ and $\vec{v}$ after removing the first character of $\vec{u}$,
   - adding the Levenshtein distance to D of both vectors after removing the first character of $\vec{v}$, or
   - again adding the Levenshtein distance to D of both vectors after removing the first characters from both vectors;

   and;
5. declaring the calculation as finished.

The steps for the Levenshtein distance show that the algorithm is iterative. In step 4 the Levenshtein distance is calculate for three combinations of the strings $\vec{u}$ and $\vec{v}$, and the smallest distance is added to the total distance. This algorithm can also be defined in a formula as

$$\mathrm{d_L}\left(\vec{u},\vec{v}\right)=\begin{cases}\max\left(\dim\left(\vec{u}\right),\dim\left(\vec{v}\right)\right), & \text{if } \dim\left(\vec{u}\right)=0 \\ & \quad\wedge\dim\left(\vec{v}\right)=0 \\ \mathrm{d_L}\left(\vec{u}_{1,\ldots,\dim(\vec{u})},\vec{v}_{1,\ldots,\dim(\vec{v})}\right), & \text{if } \vec{u}_0=\vec{u}_0 \\ 1+\min\left(\begin{array}{l}\mathrm{d_L}\left(\vec{u}_{1,\ldots,\dim(\vec{u})},\vec{v}\right),\\ \mathrm{d_L}\left(\vec{u},\vec{v}_{1,\ldots,\dim(\vec{v})}\right),\\ \mathrm{d_L}\left(\vec{u}_{1,\ldots,\dim(\vec{u})},\vec{v}_{1,\ldots,\dim(\vec{v})}\right)\end{array}\right), & \text{else}\end{cases} \tag{3.11}$$

where $\vec{u}_i$ means taking the character in $\vec{u}$ at location i, with the character positioned at $i=0$ as the first character, and where $\vec{u}_{1,\ldots,\dim(\vec{u})}$ represents a new string with all characters from $\vec{u}$ starting as position 1 up to $\dim\left(\vec{u}\right)$, which effectively means the first character in $\vec{u}$ is removed.

Damerau-Levenshtein Distance

Next to the three operations of inserting, deleting, or substituting a character in the Levenshtein algorithm, the Damerau-Levenshtein algorithm [40] also allows for transposition [28], which is switching of two neighboring characters. Where the Levenshtein algorithm is defined as $\mathrm{d_{Levenshtein}}\left(\vec{u},\vec{v}\right)=\mathrm{d_L}\left(\vec{u},\vec{v}\right)$, the Damerau-Levenshtein algorithm is similarly defined as $\mathrm{d_{Damerau\text{-}Levenshtein}}\left(\vec{u},\vec{v}\right)=\mathrm{d_{DL}}\left(\vec{u},\vec{v}\right)$.

Evaluation

Taking again the examples of `WAGT` and `WAT`, the Levenshtein algorithm supports adding or removing a character. This improves matching these strings compared to how the Hamming distance worked. The two given strings give

$$\mathrm{d_L}\left([\mathtt{W},\mathtt{A},\mathtt{G},\mathtt{T}],[\mathtt{W},\mathtt{A},\mathtt{T}]\right)=1,$$

and the Damerau-Levenshtein distance gives the same value.

In the case that some characters were switched, the outcomes between the Damerau-Levenstein algorithm and Levenshtein algorithm would be different with

$$\mathrm{d_{DL}}\left([\mathtt{W},\mathtt{A},\mathtt{G},\mathtt{T}],[\mathtt{W},\mathtt{A},\mathtt{T},\mathtt{G}]\right)=1,$$

as opposed to

$$\mathrm{d_L}\left([\mathtt{W},\mathtt{A},\mathtt{G},\mathtt{T}],[\mathtt{W},\mathtt{A},\mathtt{T},\mathtt{G}]\right)=2.$$

It is not immediately clear if this is important the case of this thesis, since the network identifying, not identifying or misidentifying a character is much more likely than switching characters. However, one case in which the Damerau-Levenshtein algorithm may perform better with output from the network is if the network calculates probabilities for a certain character over several time steps, and these probabilities are for example distributed over the time steps as a normal distribution. If another letter has a similar but time shifted distribution, then string representations calculated from the output of the network may have these characters in different order, depending on how strings are drawn from the character probabilities.

3.3.3 Jaro Similarity

Instead of a distance measure, the Jaro and Jaro-Winkler similarities calculate a score between 0 and 1, with 0 indicating that no similarities have been found between two given strings, and 1 indicating that the strings match perfectly. For this, the algorithm does not perform counting in a direct calculation of the outcome like the distance measurements, but uses the count in a formula to calculate a final similarity score.

Jaro Similarity

The Jaro similarity [79] calculates a score between two strings $\vec{u}$ and $\vec{v}$ by calculating

$$\mathrm{s}_{\text{Jaro}}\left(\vec{u},\vec{v}\right) = \begin{cases} 0, & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{\dim(\vec{u})} + \frac{m}{\dim(\vec{v})} + \frac{m-t}{m}\right), & \text{else} \end{cases} \tag{3.12}$$

where m is the number of 'matches' and t is the number of transpositions. The rules for the number of matches are not simply the number of characters that are in the same position, and the rules from the Levenshtein distance are also not followed. In this case, a value of

$$c_m = \left\lfloor \frac{\max\left(\dim\left(\vec{u}\right), \dim\left(\vec{v}\right)\right)}{2} \right\rfloor - 1$$

is calculated, where $\lfloor ... \rfloor$ represents rounding down to the nearest smaller integer, and any characters that are within a distance of c_m from each other in both strings are found to be matching. Characters cannot be matched to multiple characters. Once matched, two characters are taken out of the equation. The value for m is then the total number of characters that are matching within a distance of c_m from each other. The value for the transpositions t is the number of swaps of two characters that need to occur to get matching characters in the right order.

Jaro-Winkler Similarity

The Jaro-Winkler similarity [185] builds on the Jaro similarity by introducing a special importance for the first characters that match at the beginning of the string, up to four characters. This means the algorithm gives strings a higher similarity score when there are more matching characters near the start of the strings.

The algorithm in formula 3.12 is modified with an extra term, and is defined as

$$\mathrm{s}_{\text{Jaro-Winkler}}\left(\vec{u},\vec{v}\right) = \mathrm{s}_{\text{Jaro}}\left(\vec{u},\vec{v}\right) + 0.1 \cdot l \cdot \left(1 - \mathrm{s}_{\text{Jaro}}\left(\vec{u},\vec{v}\right)\right), \tag{3.13}$$

where l is the length of the matching number of characters at the beginning of the two strings with a maximum value of 4. The factor 0.1 is a default normalizing factor.

Evaluation

The Jaro similarity score is a number between 0 and 1, as opposed to the distance algorithms. An example can be using strings `SPEECH` and `SPEYHC`, which are two different words that - with some imagination - could represent the same meaning. Calculating the Jaro similarity gives

$$c_m = \left\lfloor \frac{\max(6,6)}{2} \right\rfloor - 1 = 2,$$

which means characters match when they within two steps from each other. The first three letters `S`, `P`, and `E` are in the same position. The second `E` is not found anywhere in the second word, and similarly is `Y` in the second word. The last characters `C` and `H` are switched, but are within c_m characters of each other. This means $m = 5$ and $t = 1$, leading to

$$\mathrm{s}_{\text{Jaro}}(\vec{u}, \vec{v}) = \frac{1}{3}\left(\frac{5}{6} + \frac{5}{6} + \frac{5-1}{5}\right) = 0.82,$$

with $\vec{u} = [\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}]$, and $\vec{v} = [\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{Y}, \mathtt{H}, \mathtt{C}]$, while the Jaro-Winkler similarity would in this case give

$$\mathrm{s}_{\text{Jaro-Winkler}}(\vec{u}, \vec{v}) = \mathrm{s}_{\text{Jaro}}(\vec{u}, \vec{v}) + 0.1 \cdot 3 \cdot (1 - \mathrm{s}_{\text{Jaro}}(\vec{u}, \vec{v})) = 0.88,$$

where $l = 3$, because the first three matching characters are in the same position, up to a maximum of four characters.

The Jaro-Winkler similarity gives importance to early characters that match and are in the same position. If there is noise in the recording, and in the phonetic transcript, this means the position of the noise can have a great impact on the Jaro-Winkler similarity scores. For example, take the strings `SPEECHX` and `XSPEECH`. In these strings, `X` represents noise. In the first string, the noise is at the end of the string, while in the second string the noise appears in the beginning. If there strings are to be matched with `SPEECH`, then for the Jaro similarity

$$\begin{aligned} &\mathrm{s}_{\text{Jaro}}([\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}], [\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}, \mathtt{X}]) \\ &= \mathrm{s}_{\text{Jaro}}([\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}], [\mathtt{X}, \mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}]) = 0.95, \end{aligned}$$

while the Jaro-Winkler similarity gives

$$\mathrm{s}_{\text{Jaro-Winkler}}([\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}], [\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}, \mathtt{X}]) = 0.97$$

$$\neq \mathrm{s}_{\text{Jaro-Winkler}}([\mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}], [\mathtt{X}, \mathtt{S}, \mathtt{P}, \mathtt{E}, \mathtt{E}, \mathtt{C}, \mathtt{H}]) = 0.95,$$

which is due to the extra term in formula 3.13 which gives importance to the up to four first characters that match and are in the same position at the beginning of the string. Having the noise `X` at the beginning of the string sets $l = 0$ and cancels this term out.

Chapter 4

Experiments

With the implementation of the three steps and the methods belonging to them explained in chapter 3, experiments can be conducted. In these experiments there are three aims, they are to

- compare the phonetic encoding models explained in section 3.2,
- compare the scoring models to compare phonetic encodings in section 3.3, and
- investigate the possibilities of the combinations of the methods for matching speech segments.

4.1 Language

The experiments are performed on a single language. This language is Dagbani, a language spoken in parts of Ghana. Among the people who spoke the language, the percentage being able to write the language was less than 20% in smaller settlements [20]. The language is one of six languages selected by the Ghanese government for regional radio and television broadcasts [6], and is currently not endangered.

Table 4.1: The currently defined set of characters for the written version of the Dagbani language [19].

a	b	ch	d	e	ɛ
f	g	gb	ɣ	h	i
j	k	kp	l	m	n
ny	ŋ	ŋm	o	ɔ	p
r	s	sh	t	u	w
y	z	ʒ	’		

The Dagbani language uses a total of 34 characters for their alphabet. This includes many characters of the Latin alphabet, with the addition of several others. The entire alphabet is shown in table 4.1 [19]. Of these characters, it can be shown that those with a phonetic overlap with characters from the Latin alphabet are the most used in the languages [33], [160], [173].

Data set

From the Dagbani language, ten words representing the letters one to ten are used in the experiments. The number zero is not used, because while this word is strictly speaking defined as ‘muɣim’, words like ‘babiɛlifu’, ‘baabi’, ‘fako’, ‘muɣimuɣi’, and ‘shɛli’ are used to mean ‘zero’ as well. Although these words do not exactly represent zero, they represent a form of ‘nothingness’, which is often interpreted as equal to the word ‘zero’. Therefore, ‘zero’ is taken as a not clearly defined word in Dagbani. The used Dagbani words [113], their romanized version [24], and their English translation are shown in table 4.2.

Table 4.2: The Dagbani words with romanized form [24], English translation, and number of samples in the used data set [142] for the words used in the experiments, as defined in a Dagbani dictionary [113].

Dagbani word (romanized form)	English translation	samples
iin	yes	720
ai, aayi, iiyi	no	710
yini, zaɣ’ yini[12] (yini, zagyini)	one	218
ayi	two	223
ata	three	220
anahi	four	223
anu	five	220
ayɔbu (ayobu)	six	219
ayɔpɔin, apɔin[13] (ayopoin, apoin)	seven	224
anii	eight	222
awɛi (awei)	nine	219
pia	ten	162

For the words in the experiments, recordings from a data set [142] have been used. This data set included the word zero in Dagbani, which was not used due to previously mentioned reasons. Table 4.2 also contains the number of samples of each word that have been used from the data set.

Dialects

This thesis attempts to use the phonetic aspects of the words in Dagbani to match recorded words with each other. For this, it is important to note the differences in which speakers of the Dagabani language may pronounce the words. There are three main dialects in Dagbani [2]. The first is Tomosili, which is

[12]The word zaɣ’ yini was present in the used data set [142] in romanized form, and is composed of yini and zaɣ’, described in entry “zaɣa$_2$” in the dictionary by Naden *et al.* [113].

[13]The word apɔin was found in the data set [142] in the romanized form, but not in accessible dictionaries. The word is used however in government publications [129], by organizations [83], and on various sites [184].

known as the western dialect and is spoken around the region of Tamale. The eastern dialect is Nayaɣili, which is spoken in Yendi, the capital of the Dagbaŋ Kingdom. Finally there is the dialect Nanunli[14], which is spoken by the Nanumba ethnic group. Both Nayaɣili and Tomosili are languages of the Dagbamba people within the Dagbaŋ Kingdom, while the Nanunli dialect of the Nanumba people comes from the Nanuŋ Kingdom.

The different dialects can be shown to change the phonology of words by deletion, compounding, lengthening and shortening of certain phonological characteristics [74], [75]. Lengthening and shortening refers to the longer or shorter pronunciation of vowels in certain combinations of letters. Deletion refers to when the pronunciation of a character is skipped, for example when the dialect adds another set of characters to a word which causes the 'deleted' character to not be pronounced. The different Dagbani dialects are not accounted for or annotated in the data set used in this thesis.

English

This thesis uses English as the language for the transliteration of recordings to a phonetic encoding in the form of the Latin alphabet. This means that acoustic information specific to Dagbani, and not present in the English language, may be lost in the process. The other way around, Dagbani has cases in which English words are used and transformed to a more 'Dagbani-like' pronunciation. For the adaptation of an English word, the characters in strings of consecutive consonants are changed. Usually vowels are added [76] between the consonant, which changes pronunciation.

4.2 Text

The network calculates probabilities over characters at each time step. One character at each time step is then selected, and all selected characters are joined into a single string, which represents the recording. This string will include non-alphanumerical characters from table 3.1 like `'`, `|`, and `#`. Since the methods defined in section 3.2 cannot deal with these characters, they are removed. Furthermore, the network may identify consecutive time steps to contain the same character, while these characters are not spoken multiple times in a row. To account for this, consecutive sets of equal characters are replaced by a single character.

4.2.1 Probabilities

Each identified string $\vec{s}$ has a probability $p\left(\vec{s}\right)$ which can be calculated using the probabilities from the network with in formula 3.7

$$p\left(\vec{s}\right) = \prod_{t=1,\ldots,T} f_{\phi}\left(x\right)_{t,\{c_t=\vec{s}_{t-1}\}}, \tag{4.1}$$

where T is the total number of time steps, $\vec{s}$ a string of length T, x is the data going into the network f_{ϕ} from formula 3.7, ϕ are the parameters of the network,

[14]Nanunli is sometimes written as Nanuni.

and $\vec{s}_{t-1} = c_t \in \mathcal{C}$ represent a character chosen from the set of characters $\mathcal{C}$ at a time step t.

For use in an implementation, a recording can be evaluated in various ways. The recording can be assigned the text that has the highest probability according to the model, by simply picking $c_t \in \mathcal{C}$ at each time step t with the highest probability $f_\phi(x)_{t,c_t}$, and concatenating the chosen characters into the final string $\vec{s}$.

A second method is by not picking the text with highest probability, but a set of texts with a minimal probability. For this, the minimal probability needs to be set. The minimal probability can be defined for the texts themselves, after which all texts with a probability higher than the minimal probability are selected. This raises a problem however, as the lengths of the recordings differ, while the minimal probability would be set the same for each recording.

For example, a recording may be split in 50 time steps. If at each time step the chosen character has a probability of 0.98 (or 98%), then the total probability $p(\vec{s}) = \prod_{t=1,\ldots,T=50} f_\phi(x)_{t,\vec{s}_{t-1}} = 0.98^{50} = 0.36$, while for a recording of $T = 100$ time steps, the total probability $p(\vec{s}) = 0.98^{100} = 0.13$. This shows that even though only characters of a high probability, 0.98, have been chosen, the probabilities of the identified strings themselves differ greatly due to length. This means the minimal probability for a string cannot be static among all recordings, but needs to be adjusted according to length.

Instead of choosing a minimal probability per string, an alternative is using only a probability per character. For example, one can define that the minimum probability for a character needs to be 0.2 before it is included in a string. At each time step, every previous string would be appended with every character that has a high enough probability. This means that if there is a time step at which several characters have a high enough probability, the number of candidate strings increases.

Having retrieved a set of texts $\vec{s_i}$ with probabilities, the probabilities can be normalized, representing the probabilities of retrieving each selected text given the minimal character probability with

$$\begin{aligned} p(\vec{s_1}|p_c > 0.2) &= \frac{p(\vec{s_1})}{p(p_c > 0.2)} \\ &= \frac{p(\vec{s_1})}{\sum_i p(p_c > 0.2|\vec{s_i}) \cdot p(\vec{s_i})} \\ &= \frac{p(\vec{s_1})}{\sum_i 1 \cdot p(\vec{s_i})} \\ &= \frac{p(\vec{s_1})}{\sum_i p(\vec{s_i})}, \end{aligned} \tag{4.2}$$

where $p(p_c > 0.2|\vec{s_i})$ represents the chance that every character in $\vec{s_i}$ has a probability high than 0.2, which is equal to 1 since the texts $\vec{s_i}$ were chosen based on this, so the formula becomes a simple case of normalization.

A case that can happen is that the probability is set to such a value that no character in a time step can be selected. In this case, the time step can be ignored and one may move on to the next time step, or the character with the highest probability is selected in any case. If the time step is skipped, it is possible that for no time step a character could be selected, in which case no textual representation of the recording can be found. In this case the text can be set to an empty string, or the most likely text can be selected.

4.3 In-word comparison

The in-word comparison means that strings representing phonetic encodings for the recordings from the same word are compared with each other. Each recording only needs to be translated to English phonemes with the details in section 3.1 once. After this, it is generalized with one or more of the methods of section 3.2, after which words can be matched with one or more of the methods described in section 3.3.

Each recording is thus translated to a textual form. After this the strings can be further processed and compared. Comparing happens in several steps. Since the used data set [142] is unfiltered, there is an expectation of noise, which means the data set may contain recordings that do not contain the supposedly spoken word at all, or recordings that contain the word, but contain significant background noise which makes the first step of translating the recording to English phonemes not easily possible. Therefore the first step is to match recordings of the same word with each other. This will create clusters of similarities, the recordings that match each other well will have low distances or high similarities assigned, while the opposite is true otherwise.

This comparison happens by taking all recordings of a word. Each recording is processed using the methods in sections 3.2 and 3.3. After each recording of a word is matched with all other recordings of the same word, the scores can be averaged. Recordings that are translated to the same phonetic transcription will then get the same average score compared to all recordings, and noisy recordings with different sets of assigned phonetic characters will get a lower score assigned. Assuming the majority of the recordings are not too noisy, they will cluster together as the group of recordings with the highest average similarity or lowest average distance.

This allows for the recordings to be analyzed and for data to be possibly cleaned. However, in this case the decision is made for data to not be cleaned, as in a real implementations used by real people similar noise as in the data set would be found.

Furthermore, comparing similarities and clusters withing the recordings of a single word allows for the different methods in sections 3.2 and 3.3 to be compared. More can then possibly be said about which method may be best be used for in-word comparison to handle noise.

4.3.1 Combining methods

An extension on the proposed in-word experiment is using several methods at the same time for a comparison. This gives several possibilities of combinations between the three methods of generalizing a phonetic translation of the word, and three methods of matching these generalizations.

Multiple methods could be combined by averaging their results. For this, the methods that provide a distance measure need to be changed to provide a similarity score instead, so they can be used in combination with other similarity scoring methods. This can for example be done by scaling a given distance using the length of the string and transforming it to a similarity score, for example by

$$\mathrm{s}_{\text{Hamming}}\left(\vec{u},\vec{v}\right) = 1 - \frac{\mathrm{d}_{\text{Hamming}}\left(\vec{u},\vec{v}\right)}{\max\left(\dim\left(\vec{u}\right),\dim\left(\vec{v}\right)\right)} \tag{4.3}$$

or similarly with for example $\mathrm{s}_{\text{Damerau-Levenshtein}}$ using $\mathrm{d}_{\text{Damerau-Levenshtein}}$. This allows for an averaged score

$$\mathrm{s}_{\mathscr{m}}\left(\vec{u},\vec{v}\right) = \frac{1}{\sharp\mathscr{m}}\sum_{m\in\mathscr{m}}\mathrm{s}_{m}\left(\vec{u},\vec{v}\right) \tag{4.4}$$

to be calculated, where subset

$$\begin{aligned}\mathscr{m} \in \mathcal{M} = \{&\text{Hamming},\\ &\text{Damerau},\\ &\text{Damerau-Levenshtein},\\ &\text{Jaro},\\ &\text{Jaro-Winkler}\}.\end{aligned} \tag{4.5}$$

The total score calculated in formula 4.4 can be extended with the three methods in section 3.2 as

$$\mathrm{s}_{\mathscr{n},\mathscr{m}}\left(\vec{u},\vec{v}\right) = \frac{1}{\sharp\mathscr{n}\cdot\sharp\mathscr{m}}\sum_{\substack{n\in\mathscr{n},\\ m\in\mathscr{m}}}\mathrm{s}_{m}\left(\mathrm{g}_{n}\left(\vec{u}\right),\mathrm{g}_{n}\left(\vec{v}\right)\right), \tag{4.6}$$

where subset $\mathscr{m}$ was previously defined, subset $\mathscr{n}$ is defined as in 3.9, and where $\vec{u}$ and $\vec{v}$ are defined as the two strings being compared. Note that formula 4.6 is the formula that can be generally used for any comparison, for example by using only the NYSIIS method and matching using only the Damerau-Levenshtein distance, the subsets $\mathscr{m} = \{\text{Damerau-Levenshtein}\} \in \mathcal{M}$ and $\mathscr{n} = \{\text{NYSIIS}\} \in \mathcal{N}$ can be used.

Next to formula 4.6, the algorithms can also be mixed using two mixing vectors $\dim\left(\vec{q}_{\text{norm.}}\right) = 3$ and $\dim\left(\vec{r}_{\text{match}}\right) = 5$, which are used as

$$s_{\vec{q}_{\text{norm.}},\vec{r}_{\text{match}},\mathscr{n},\mathscr{m}}\left(\vec{u},\vec{v}\right) = \frac{1}{3\cdot 5}\sum_{\substack{n\in\mathcal{N},\\ m\in\mathcal{M}}}\vec{r}_{\text{match},m}\cdot\vec{q}_{\text{norm.},n}\cdot\mathrm{s}_{m}\left(\mathrm{g}_{n}\left(\vec{u}\right),\mathrm{g}_{n}\left(\vec{v}\right)\right), \tag{4.7}$$

which is similar to formula 4.6, or similarly with a mixing matrix T of size 3×5 and formula

$$s_{T,\mathscr{n},\mathscr{m}}\left(\vec{u},\vec{v}\right) = \frac{1}{3\cdot 5}\sum_{\substack{n\in\mathcal{N},\\ m\in\mathcal{M}}}T_{n,m}\cdot\mathrm{s}_{m}\left(\mathrm{g}_{n}\left(\vec{u}\right),\mathrm{g}_{n}\left(\vec{v}\right)\right). \tag{4.8}$$

Formulas 4.7 and 4.8 are respectively $3+5=8$ and $3\cdot 5=15$ dimensional optimization problems, which can be solved by using a minimization algorithms, for example the Nelder-Mead algorithm [117] with constraints and bounds and a consistent, or differential evolution [161], [162] in the case of more stochastic results.

4.4 Cross-word comparison

Cross-word comparison is the comparison of recordings of different words between each other. The goal of this thesis is to explore how well the proposed methods can work with matching a single recording of an unknown word to a set of recordings of different known words. There are several ways in which this comparison can happen, which overlap with each other.

4.4.1 1-to-1 comparison

In the simplest interpretation of cross-word comparison, an unknown recording is compared to a known recording of each candidate word. One recording from each candidate word is processed using discussed methods, and is then added to the set of recordings of words to 'choose' from. An unknown recording is then taken, processed, and matched to each of the known recordings. The recording with the highest similarity score according to formula 4.6 is the best match, and the decision is then made that the word registered with that known recording is the word equal to the word spoken in the unknown recording.

By doing this many times over, average rates of correctly matching a word can be calculated. For this, the 'unknown' recording is randomly chosen from the pool of recordings from the word for which this rate needs to be calculated. When matching correctly, the chosen recording should match best with the recordings of the pool it was chosen from, and not with recordings from other words. The probabilities of correctly matching can be calculated for every word in this way, and these can be compared to see which word has recordings that match better with the word itself than with other words.

Correction

Next to calculating probabilities for every word to match a recording correctly, an overall statistic can be calculated over all words to give some sense of how well the used combination of normalization and matching algorithms works. For this comparison a mean and standard deviation trivially defined as

$$\mu\left(\vec{u}\right) = \frac{\sum_i u_i}{\dim\left(\vec{u}\right)} \text{ and } \sigma\left(\vec{u}\right) = \sqrt{\frac{\sum_i \left(u_i - \mu\left(\vec{u}\right)\right)^2}{\dim\left(\vec{u}\right)}}$$

can be calculated. Here, $\vec{u}$ are the probabilities for correctly matching the used words, with one value for each word. The mean $\mu\left(\vec{u}\right)$ then represent the mean probability of correctly matching one of the used words.

On the one hand, the mean should be as high as possible, since the words should be correctly matched as much as possible. On the other hand, the standard deviation should be as low as possible, since there should be some stability in how well words are matched. As a first example, if two words are considered, and a correct match with one word is found with a 100% probability, while the other word matches correctly with a 0% probability, then the mean probability will be $\mu = (1+0)/2 = 0.5$, or 50%, while the standard deviation will be $\sigma = \left(\sqrt{(1-\mu)^2 + (0-\mu)^2}\right)/2 = 0.5$. As a second example, if both words have a 50% probability of matching correctly, their mean $\mu = 0.5$ again, while their $\sigma = 0$. In these cases, the second example would be preferred, which can be identified using the low value for the σ.

When calculating an overall statistic for how well the algorithms perform, the mean should be corrected for by the variability, for which the standard deviation can be used. In this thesis, this correction is done using

$$\mu_{\text{corrected}} = \frac{\mu}{(a + \sigma)^c}, \tag{4.9}$$

which is based on the coefficient of variation [49] CV $= \frac{\sigma}{\mu}$, which calculates a variability of the data given the mean. In this case, the inverse coefficient of variation is used, to correct the mean with a factor based on the standard deviation, with an additional factor $a > 0$ to prevent a blow up of $\mu_{\text{corrected}} \to \infty$ as $\sigma \to 0$. Variable c in 4.9 defines how much the standard deviation should contribute to the correction. Here, this is set to $c = 1$, which mimics the inverse coefficient of variation. Variable $a > 0$ is set to 0.1, which is taken as a significant standard deviation, with the idea that this is the value above which which the standard deviation becomes noticeable to a user.

Indecision

One important aspect to the comparison of a recording with multiple different words is when the similarity score to two different recordings is the same. There are then two options. Either a random word can be picked from the words that matches the best, or indecision can happen - the refusal to decide on a best matching word.

The first option of picking a random word would not be great from a user and testing perspective. By introducing randomness, debugging becomes more difficult, and the user may become confused if too much information is shown to explain this randomness. From a design perspective it is often better to not show too much information to the user [104]. In general the idea of the algorithm is to find the best match among match scores, therefore the assumption is that a case of equal similarity scores happens only in few cases. If the user would be notified upon selecting a random match, the user may be confused since this is a rare event. The event may also be missed, or the final decision by the user might be to not pay too much attention to it, or interpreting this as correct output from the underlying algorithm. Therefore, action may not be taken upon being notified of a random outcome being chosen, which may lead to negative effects if the random result is trusted as being a correct result. These mentioned complications and complexities show that the option to choosing a random outcome from equally best matching results is not the best option with the end-user in mind.

The second option is then to refuse choosing a matching word. This means the a new recording needs to be given to retry finding a best matching word to this recording. This has one clear downside - there is the possibility of requiring multiple recorded samples before a match can be found, this would also negatively affect the user experience.

When performing an experiment many times for a single word, a rate of indecision can be calculated, which represents the number of retries that are needed before a match is found. For this rate, a mean and standard deviation can be calculated, and formula 4.9 can again be used to correct the mean indecision rate with the standard deviation. However, the variables a and c would take on different values in this case. When correcting the rate of a correct match, the probability of getting a correct match was lowered if the standard deviation was higher, as a higher standard deviation would have a negative effect on the user experience. In the case of the indecision rate however, a higher standard deviation should increase value for the corrected indecision rate, since the indecision rate is already perceived as negative when its mean value is high. To ensure this, $c < 0$ and in this case simply $c = -1$ is chosen. Variable $a = 1.1$, higher than the $a = 0.1$ used for the previous correction, since the effect of the standard deviation in the indecision rate is already on top of the negative user experience from a high indecision rate, and so the 'extra' negativity in the user experience from a higher standard deviation is a secondary effect and should have a smaller impact.

Combining statistic

With the above sections two separate statistics are calculated - that for the probability to match correctly and the rate of indecision. Generally, the probability to match should be as high as possible, while the indecision rate should be as low as possible. A single statistic can be calculated by combining these. For this, formula 4.9 is re-used, but with different variables as

$$s = \frac{p_{\text{match}}}{r^d_{\text{indecision}}}, \tag{4.10}$$

where d again represents how much the indecision rate should count in the combined statistic. In this case, a difference in low indecision rates may seem large when compared to each other, for example the rate of 0.1 is twice as large as the lower rate of 0.05, however from a user perspective this may not be very noticeable. On the other hand, a rate of 1 is twice as large as 0.5, but is very noticeable. Finally a rate of 10 - meaning 10 tries need to be done before a match is found - is twice as large as 5, but from the end users perspective these may both be considered unusable, and therefore not be very different in perception.

Overall, the indecision rates closer to 1 should affect the single statistic less than rates away from 1 - the either the 0 or ∞ sides. For this reason, $d < 1$ in formula 4.10, and $d = 0.5$ in this case.

Algorithm comparison

Using the combined statistic and the formulas in equation 4.6, 4.7, and 4.8, the combination of statistics can be compared to see which combination or set of mixing variables gives the best results.

4.4.2 1-to-many comparison

Next to the 1-to-1 comparison, there are other types of comparisons that can be done. The 1-to-1 comparison compared one recording to one sample recording of each of the candidate words. Another comparison is comparing one recording to multiple recordings of each candidate word. This is a 1-to-many comparison. Multiple scores are then calculated between the given unknown recording and the available recordings of each word. As with the 1-to-1 comparison, a decision then needs to be made with which word the given unknown recording matches best.

Highest similarity

The first obvious implementation of this, is by simply picking the recording with the highest similarity score to the given unknown recording, and taking the word attached to that recording as the word matching to the unknown recording.

Highest n similarities

Instead of choosing the single best similarity score, a different algorithm is choosing the candidate word with the best n similarity scores as the matching word. This can be interpreted in two ways.

The first is that the top n similarity scores of all sampled recordings to compare to need to be of the same word to decide on a match with that word. This has the upside that more samples need to match before a decision is made, which will likely give the decision a higher likelihood of being correct compared to the previous strategies. However, a downside is that an indecision may happen more frequently. Since the top n similarity scores all need to be of the same candidate word before a decision is made, the chance of indecision is higher than when only the top score is used to make a decision, as not a single other word can have a recording with a similarity score between the top n similarity scores. There would at the very least be a linear relation between the rate of indecision and the value of n in this scenario.

A second interpretation is that the top n similarity scores are taken, and the candidate word that is most represented in those top scores is chosen as match for the unknown word. This has the upside that the chance for indecision is not as high for increasing n.

When choosing the top n similarity scores, there can be a case in which similarity scores are the same. For example if the n-th highest similarity score occurs three times, and the top n similarity scores need to be taken, then a decision has to be made on which of the three recordings with equal similarity scores to include. In the case of this thesis, all three are included. This can cause the top n recordings with highest similarity scores to actually become a top $n + m$ set of recordings, where the extra m recordings are included because their similarity score is the same as the n-th highest similarity score.

4.4.3 Multiple probable texts

The previous methods described using only the most likely text per recording to match recordings with each other. However, each recording can have multiple probable texts calculated for it. When choosing only the most likely text, information in the form of another probable texts is thrown away. Keeping this information may improve probabilities of matching the right word, especially under a low number of recordings or high amounts of noise.

While not experimented further with in this thesis, it is worth mentioning this is an option to improve correct matching probabilities. Instead of using one text per recording, all texts with either a probability higher than a certain percentage, or a probability close to the highest probability can be used for matching. This means that when using multiple recordings to match against, the total number of texts for these recordings is at least as high, and likely higher, than the number of recordings themselves. The previous described method may then be used in the same way on this larger number of texts to find a match.

One important aspect to keep in mind is that the number of texts extracted from a recording my be different depending on the algorithm used to calculate these texts. When the number of texts for one recording is high, while low for another recording, the recording with more extracted texts is more represented in a set of texts to match against than the recording with a lower number of texts. To correct for this, some weight should be given to each text of a recording, so that the total of these weight adds up to 1 for each recording. In that way, recordings with different numbers of texts each will have an equal weight in the result.

4.5 Summary

The previous sections described various ideas and theories used in the experiments. The experiments aim to show the in-word and cross-word comparisons as described in those sections, as well comparisons between the different used algorithms. Each experiment consists of three steps. Phonetic transcriptions are created of words, then these transcriptions are generalized using one or more of the Soundex, NYSIIS, and Metaphone algorithms, and finally similarity scores are calculated between them using one of or more of the Hamming, Levenshtein, Damerau-Levenshtein, Jaro, and Jaro-Winkler similarity algorithms.

With formula 4.6 combinations of these algorithms can be used. With the list of candidate words and their recordings, generalizing algorithms and matching algorithms, a multidimensional problem is created which can be difficult to analyze thoroughly. Therefore, not all combinations of algorithms will be experimented with at all times, but between experiments decisions are made to continue with one or a selection based on previous results.

In the results, the Damerau-Levenshtein and Jaro-Winkler algorithms are represented with respectively the strings `damerau` and `winkler`, while the other algorithms are represented with their own name.

Chapter 5

Results

With the theory and ideas as described before, the results are presented in this chapter. The experiments were performed in a Jupyter-Notebook with the various packages mentioned in the acknowledgments section. The source code for the experiments is described in section B.2.

5.1 Model

The experiments start with a model to get a phonetic transcription in the English language. After converting a given audio file to a WAV file with a single channel and 16000 hertz, it is fed to the network. For this, it is split up into pieces of 25 milliseconds each, with a step size of 10 milliseconds, and probabilities are calculated for each character at each time step.

Figure 5.1 shows the probabilities at each time step for a recording of the word 'anii'[15]. As can be seen in this figure, character `#` has the highest probability in most time steps, which indicates no character was identified. Around time steps 50 to 70, high probabilities are seen for several of the alphanumerical characters, and at the final time step, character `'` is favored, which indicates a split between spoken words.

As was previously explained, the probabilities per character can be used to construct pieces of text that represent the English phonetic description of the recording with a certain probability. Table 5.1 shows all pieces of text constructed from characters with a probability of at least 0.2 per character.

The word in the recording that was used was 'anii', and listening to the recording itself, this word is clearly there, and understandable in English when pronouncing 'anii'. It is thus expected that the texts extracted from the probabilities in figure 5.1 correspond to this, and looking at table 5.1, this is indeed the case. The string with the highest probability `ANE` indeed comes the closest to what was heard in the recording, when pronouncing the `ii` and as an English `e`. The strings following after `ANE`, like `INE`, `ANI` are also clearly close to what is being said. As the probability of a string goes down, the correctness of the text as interpreted by a human compared to what said in the recording also goes down, although they keep a connection to 'anii'. The network favors the

[15]For this plot, recording "`Eight - Anii/recording-25ad2f6b-8b23-4107-a91d-75422febe027.3gp`" was used from the data set.

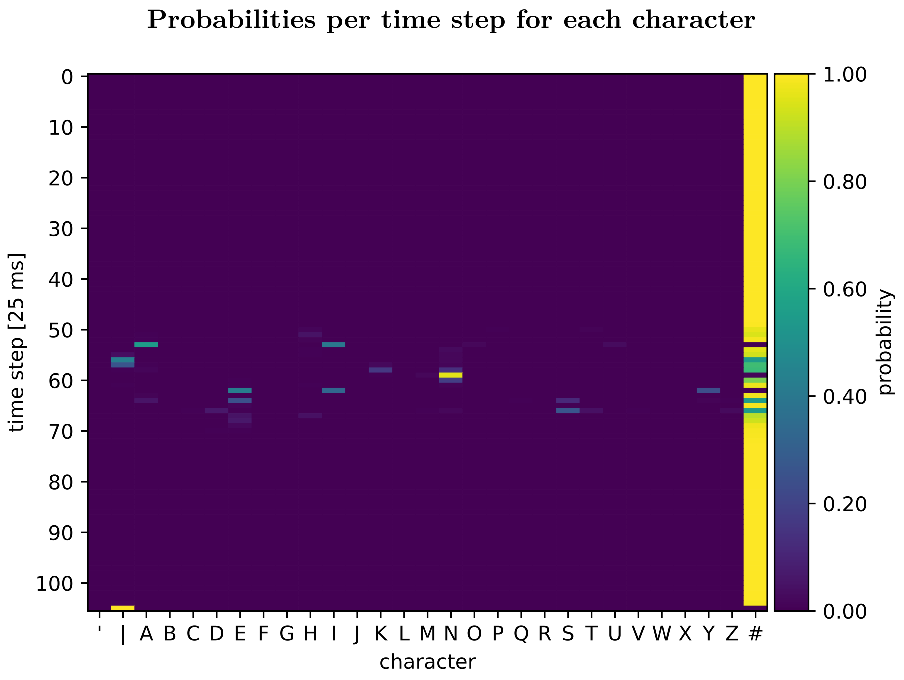


Figure 5.1: This plot shows the probabilities at each time step for every character supported by the model. Probabilities are normalised per time step. The used recording is of the word ‘anii’.

Table 5.1: This table shows the constructed strings from the characters in figure 5.1 for a recording of ‘anii’ with a probability of at least 0.2 per character. The strings have been cleaned, meaning non-alphanumerical characters have been removed and consecutive repeated characters have been replaced by a single character. The probabilities have been normalized with formula 4.2. The entire table is split in two.

probability	text
0.167	`ANE`
0.122	`INE`
0.089	`ANI`
0.080	`ANES`
0.065	`INI`
0.064	`ANY`
0.058	`INES`
0.047	`INY`
0.042	`ANIS`
0.040	`ANIE`
0.031	`INIS`
0.030	`ANYS`
0.029	`INIE`
0.028	`ANYE`
0.022	`INYS`
0.021	`INYE`
0.019	`ANIES`
0.014	`INIES`
0.013	`ANYES`
0.010	`INYES`

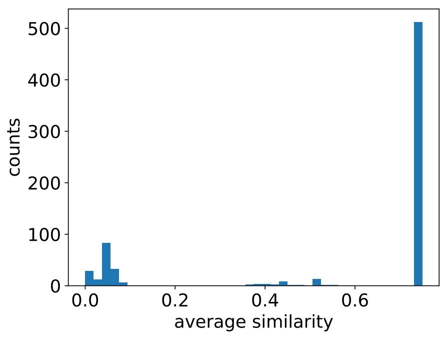
simplest representation, and goes into more complex and longer representations as the probability of a text goes down.

This thesis will now continue with simply using the phonetic transcription with the highest probability.

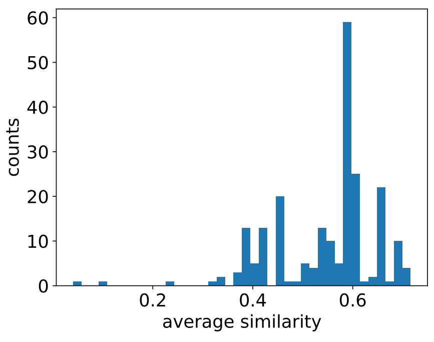
5.2 In-word comparison

An in-word comparison is now done. This compares similarity scores within a group of recordings of the same word and not between words. Figure 5.2 shows this for the words ‘aayi’ (‘no’ in English) and ‘anahi’ (‘four’ in English) in respectively figures 5.2a and 5.2b in the form of a histogram of the average similarity score for each recording compared to the other recordings of the same word. Here, the Metaphone and Jaro algorithms are used for respectively normalization and matching.

Histograms of average Metaphone in-word Jaro similarity scores



(a) aayi (no)



(b) anahi (four)

Figure 5.2: Histogram of the average similarity per recording of the word ‘aayi’ (‘no’) on the left and ‘anahi’ (‘four’) on the right compared to the other recordings of the same word, using phonetic string generalisation with the Metaphone algorithm and matching with the Jaro similarity.

In figure 5.2a two major densities are found, the highest one being around the average score of 0.75, and the other one at the average scores close to 0. Around 0.45 and 0.5 there are some smaller densities. To inspect these closer, the combinations of score, phonetic transcription, and normalized string are counted and shown in table 5.2a for all counts higher than 1. The normalised string with highest count is `I`, coming from `I`, `IH`, and `IE`, which shows that the ‘aa’ part in the word ‘aayi’ is not found well. The normalized string `I` is also very generic, which has the risk that it can be easily confused with normalized strings of recordings of other words which may also end up as `I`. The strings with a low similarity of less than 0.1, are more noisy, where strings like `MHM`, `UHUH`, or `AREYOU` are understood. It is reasonable and correct that these are assigned low scores, because they have a low similarity with other recordings.

Looking at the figure 5.2b for ‘anahi’ (‘four’), the concentrations of average similarity scores are more spread out between the values 0.35 and 0.7. The details are again shown in table 5.2b, which makes it clear that more details are extracted in the phonetic transcription then in table 5.2a. The best most often found string is for `IKNOW` with normalized string `IKN`. When pronouncing the word ‘anahi’, and string `IKNOW` as they would be by an English speaker,

Table 5.2: The average similarity scores per recording for a word with other recordings of the same word are shown in these tables for the word ‘aayi’ (‘no’) and ‘anahi’ (‘four’) respectively. The average similarity score, phonetic (phon.) transcription, and normalised (norm.) string are shown, together with the count this combination of three values was found in the histogram in figure 5.2.

(a) aayi (no)

score	count	phon.	norm.
0.749	500	I	I
0.749	4	IH	I
0.749	4	IE	I
0.516	2	IS	IS
0.447	2	ISIT	IST
0.081	2	AHET	AHT
0.058	24	MHM	MHM
0.058	2	UHIT	UHT
0.054	3	AN	AN
0.052	2	MH	MH
0.046	4	UHUH	UH
0.046	27	HI	H
0.045	6	AREYOU	ARY
0.044	2	AYE	AY
0.043	7	A	A
0.043	3	AH	A
0.008	3	YE	Y
0.0	7		

(b) anahi (four)

score	count	phon.	norm.
0.715	3	IKNOWT	IKNT
0.682	8	IMNOT	IMNT
0.660	9	INOT	INT
0.660	6	INIGHT	INT
0.660	3	INAT	INT
0.611	5	IKNOWIT	IKNWT
0.603	11	INOW	IN
0.603	4	INO	IN
0.585	58	IKNOW	IKN
0.537	5	ATNIGHT	ATNT
0.520	2	ANDTHEN	ANTON
0.456	8	ANIGHT	ANT
0.456	4	ANDI	ANT
0.456	3	AND	ANT
0.456	2	ANIT	ANT
0.420	2	UNIGHT	UNT
0.418	4	ONIGHT	ONT
0.415	2	IMA	IM
0.404	2	ILIKE	ILK
0.382	2	I	I
0.379	3	ANE	AN
0.379	2	AN	AN

clear similarities are found. It makes sense that IKNOW would be a phonetic transcription according to an English phonetic model. Opposed to the previous table, this table and the histogram show that there are various phonetic transcriptions that match better than the one that was found most. These are from noisy recordings compared to the ones for IKNOW, which still clearly include the target word, but for which other (noisy) details are also found. The reason that these words match with a higher similarity is because the Jaro similarity algorithm finds characters match when they are closely located to each other, which gives the strings which had extra details extracted due to noise next to the ‘main’ details more likely to match both the ‘main’ transcription and some of the noisy ones, giving them a higher average similarity score.

The histograms in figure 5.2 for ‘aayi’ and ‘anahi’ can be created for all twelve available words. Figure 5.3 shows these histograms with Metaphone and the Jaro similarity algorithms for all words, stacked on top of each other. The histograms for ‘aayi’, ‘ayi’, and ‘pia’ are very similar to each other, for which the explanation behind the histogram in figure 5.2a can be used. Words that are defined in two ways, meaning those for ‘ayɔpɔin’/‘apɔin’ and ‘yini’/‘zaɣ’ yini’

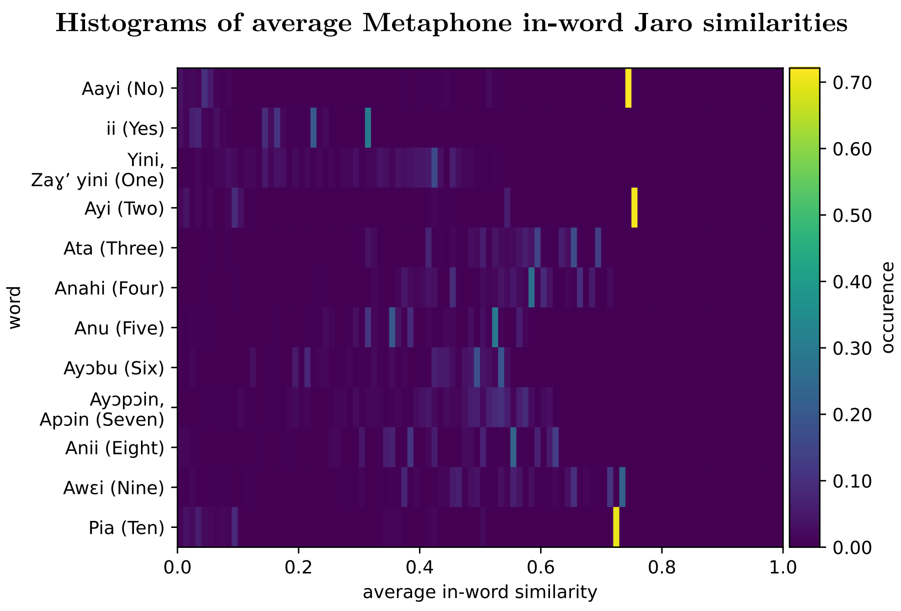


Figure 5.3: Image of histograms similar to that in figure 5.2, where the histograms have now been normalised and are stacked on top of each other. Each histogram is for a different word and the Metaphone and Jaro similarity algorithms have been used.

both show a clustering of average similarities near 0.5. This would make sense if the distribution of the two words is roughly equal in the data set, giving a 50% chance of matching one or the other. Finally, there are the recordings for which a single word is defined, but which do not show a clear density in average similarity score. These recordings are likely more difficult for the phonetic algorithm to handle, or are more easily affected by noise.

The previous figures and tables have been created with the Jaro similarity algorithm, but other algorithms are available as well. Using again the Metaphone algorithms and each of the five matching algorithms up to a combination of two, the plot in figure 5.4 shows a histogram for each combination stacked on top of each other, for the word 'anahi'.

Looking at the distributions for the single algorithms in figure 5.4, meaning the top five histograms, those for the Hamming, Levenshtein and Damerau-Levenshtein distance algorithms are nearly the same, with small differences. The distributions for the Levenshtein and Damerau-Levenshtein algorithms are the same, which would mean that the transposition ability in the Damerau-Levenshtein algorithm over the Levenshtein algorithm has no effect on the similarity scores in this specific case. On the other hand, the histograms for the Jaro and Jaro-Winkler similarities are different from the other three, with the Jaro algorithm showing some densities in the distribution at a lower similarity score than the Jaro-Winkler algorithm. This would be due to the extra term in the Jaro-Winkler algorithm creating a higher score for matching characters at the start of strings. The Jaro and Jaro-Winkler similarity algorithms both show data above the highest density of average scores, for which the explanation is

Histograms of average Metaphone in-word similarities for 'anahi'

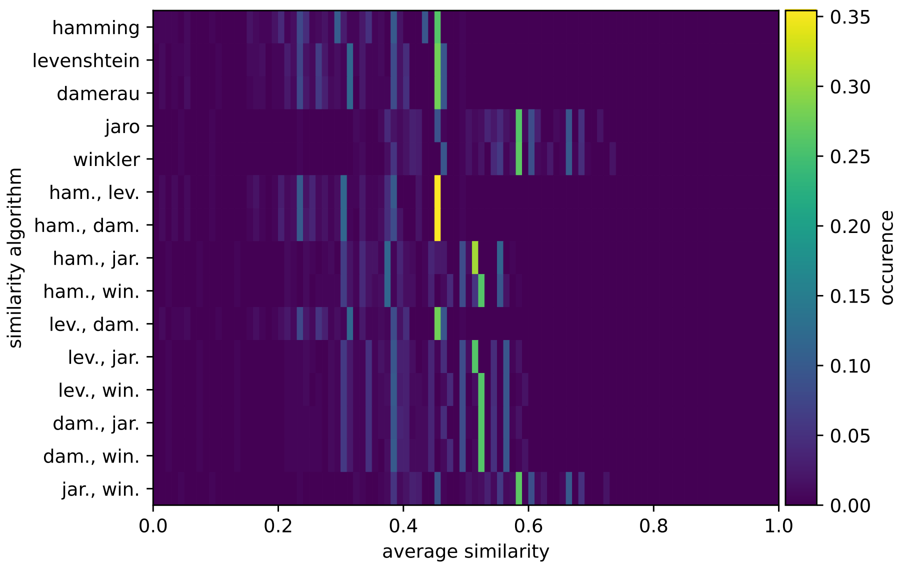


Figure 5.4: And image similar to that in figure 5.3. Each histogram is made for the word 'anahi' with the Metaphone algorithm and using the different matching algorithms up to a combination of two, as shown on the y-axis.

similar to the previous explanation of this phenomenon for figure 5.2b.

The figure also shows that the distributions of average scores stretch out over different ranges, which is an indication of how the algorithm calculates a score, but does not say anything about how well recordings may be matched between recordings of different words.

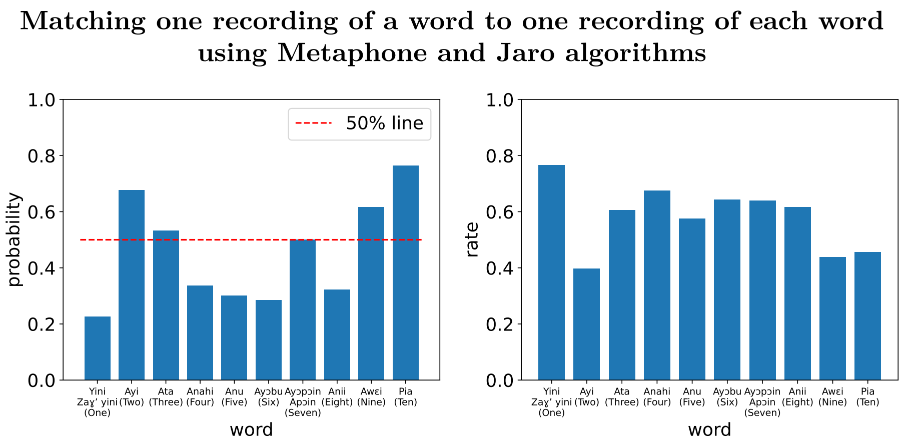
The combinations of algorithms in figure 5.4 are averaged over the single algorithms, which is clearly visible in the image. This averaging takes place during matching with formula 4.6.

5.3 Cross-word comparison

From the in-word comparison in the previous section, results for cross-word comparison are now discussed. Some of the following figures will have a seemingly too small font on the x-axis due to showing words which otherwise do not fit the axis well. In these cases, the words are always sorted in order from one to ten.

5.3.1 1-to-1 comparison

As a first simple example of a cross-word comparison, and continuing with the use of the Metaphone and Jaro algorithms from the previous section, figure 5.5 shows histograms of the comparison of one recording of a word to one recording of each candidate word, after which the best match is chosen. The histogram in figure 5.5a shows the probability of a correct match when the recording to be matched is sourced from the word on the x-axis, and the histogram in figure 5.5b shows the rate of indecision. To repeat, the rate of indecision is the number of times no decision can be made before a decision is made. The results in the figure have been retrieved using 10000 samples.



(a) Probability of correct match

(b) Rate of indecision

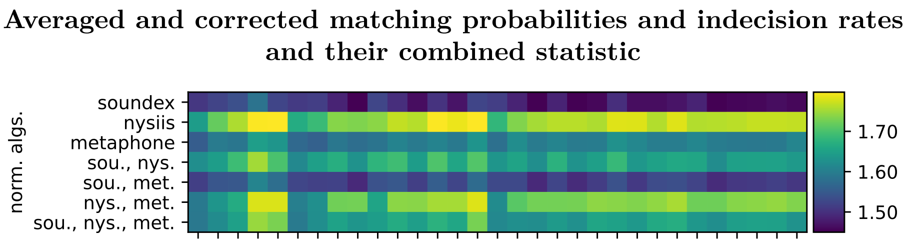
Figure 5.5: For matching one 'unknown' recording sampled from a word to one recording of each candidate word, and then deciding which recording and attached candidate word matches best with the given recording, the histogram of the probability of a correct match is shown in the left plot, while the right plot shows the rate of indecision for each word. Here the Metaphone and Jaro algorithms are again used.

According figure 5.5a five out of the ten words match the correct word with a chance on or above the 50%, which may be seen as low. The rate of indecision is lower than 1, which means it is more likely a match is found than that an indecision occurs.

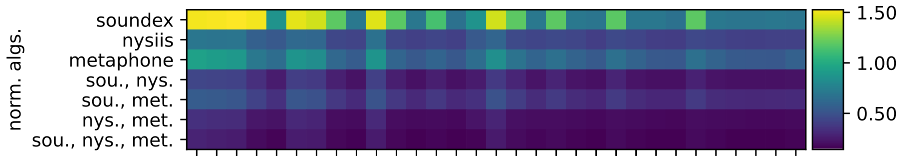
With the simplicity of the 1-to-1 matching, probabilities and indecision rates can be calculated for the various combinations of generalization and similarity algorithms, for each word. The matching probabilities and indecision rates for recordings of each word are then averaged over all words and the corrected using formula 4.9 to take into account both the importance of a matching probability and a low standard deviation of this probability. The corrected probabilities and indecision rates are then combined using formula 4.10.

Figure 5.6 shows the corrected average values for the correct matching probabilities and indecision rates, and their combined statistics, in respectively the three figures from up to down. In the plot for the correct matching probabilities, the color bar shows that the values are relatively close to each other. Of these values, the NYSIIS algorithms seems to be giving relatively high probabilities of matching correctly. The combination of NYSIIS and Metaphone also gives high probabilities of matching.

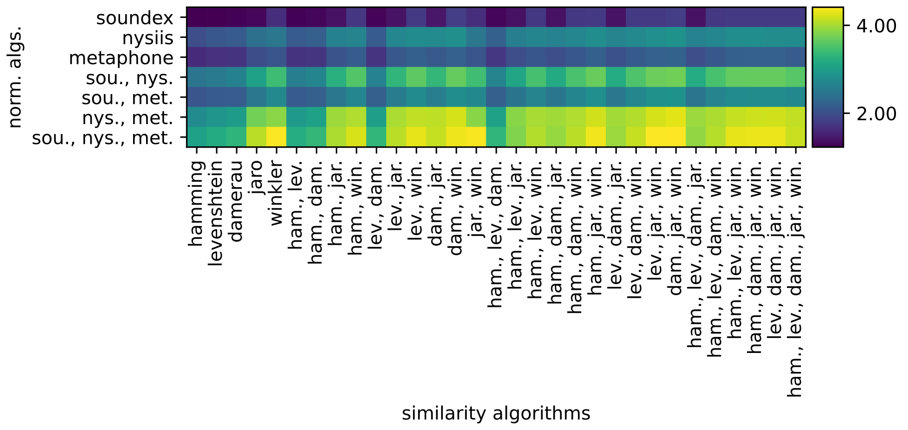
When looking at the plot for corrected indecision rates in figure 5.6b, Soundex shows the highest indecision rates for the single similarity algorithms. Overall the spread of the corrected indecision values are between 0.3 and 1.5, which is a relatively large spread. Those for NYSIIS and the combinations that include



(a) The corrected average correct matching probabilities.



(b) The corrected average indecision rates.



(c) The combined corrected average correct matching probabilities and average indecision rates.

Figure 5.6: This figure shows three plots from up to down for respectively the corrected average correct matching probabilities for the combinations of algorithms over the analysed words, the average indecision rates, and the combined statistics of the two. The x-axis is the same for each figure and only shown once.

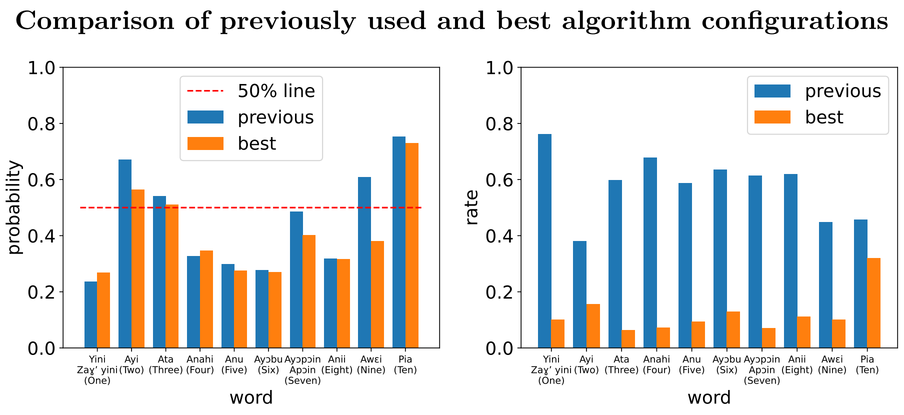
NYSIIS show relatively low corrected indecision values compared to the other combinations. However, the values for the combination of all three Soundex, NYSIIS, and Metaphone shows the lowest indecision values. Further, when taking into account the similarity algorithms, the Jaro-Winkler algorithm consistently gives lower corrected indecision rates compared to other similarity algorithms.

Finally looking at the statistic for the combination of the two types of algorithms in figure 5.6c, the combinations of normalizing algorithms including NYSIIS have high values. All three normalizing algorithms combined gives the best results. The top values and their associated algorithms are shown in table

5.3, which shows the combination of Soundex, NYSIIS, and Metaphone algorithms with Damerau-Levenshtein, Jaro, and Jaro-Winkler similarity algorithms has the best performance. Interestingly, the Jaro-Winkler similarity algorithm (shown as 'winkler') is the one constant algorithm among those combinations that have top ten scores.

Table 5.3: The top 10 values in the plots in figure 5.6c and the algorithms represented by them.

Score	Normalisation algorithms	Matching algorithms
4.412	soundex, nysiis, metaphone	damerau, jaro, winkler
4.390	soundex, nysiis, metaphone	levenshtein, jaro, winkler
4.389	soundex, nysiis, metaphone	winkler
4.377	soundex, nysiis, metaphone	jaro, winkler
4.338	soundex, nysiis, metaphone	hamming, jaro, winkler
4.326	soundex, nysiis, metaphone	damerau, winkler
4.316	soundex, nysiis, metaphone	levenshtein, damerau, jaro, winkler
4.314	soundex, nysiis, metaphone	hamming, damerau, jaro, winkler
4.297	soundex, nysiis, metaphone	levenshtein, winkler
4.294	soundex, nysiis, metaphone	hamming, levenshtein, jaro, winkler



(a) Probability of correct match

(b) Rate of indecision

Figure 5.7: The plot from figure 5.5 which used the Metaphone and Jaro algorithms is recalculated using the combination of algorithms that was found to perform best according to table 5.3.

By plotting and comparing the results in figure 5.5 again using the highest scoring algorithms in figure 5.7, it can be seen that the probability for a correct match goes down slightly with the best configuration from table compared to using only the Metaphone and Jaro algorithms, with the exceptions that the probability increases slightly for the words 'yini' and 'anahi'. The probability decreases significantly for the word 'awɛi'. Comparing the rates of indecision in figure 5.7b shows a very significant improvement. This is because the best combination of algorithms was determined by a statistic combining both the probability of a correct match and requiring a low rate of indecision. It can be argued that the decrease in rate of indecision compensates for the slight decrease in correct matching probability.

From now on, the experiments continue with the best configuration from table 5.3.

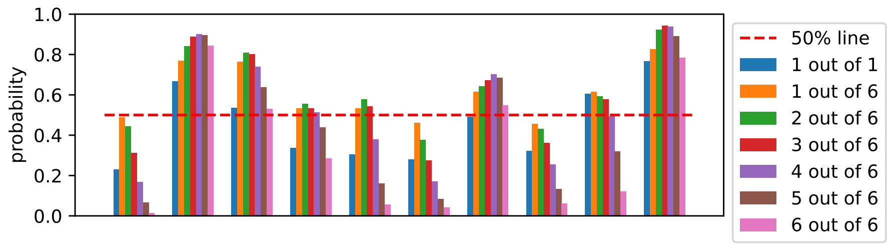
5.3.2 1-to-many comparison

Moving on from comparing with one recording of each candidate word, a comparison can be done with multiple recordings of candidate words. For example if n recordings are used from each word, and there are ten words, then a total of $n \cdot 10$ recordings are compared to an unknown recording. Here, a number of different ways of matching are possible as previously discussed.

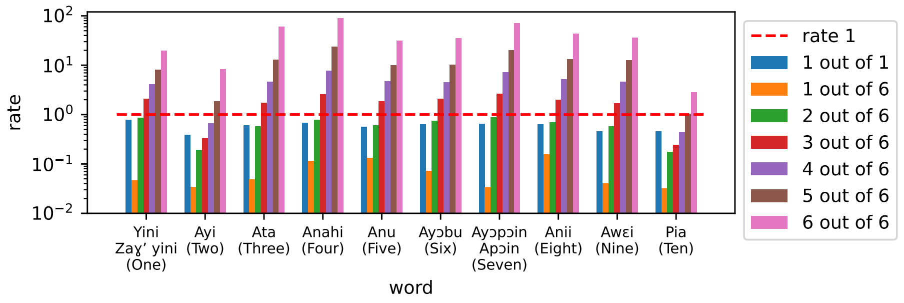
Highest n similarities

One method of matching is by requiring that a certain number of most similar recordings to the unknown recording are of the same word before a match is decided.

Histogram of probabilities and rates when requiring top n similarities to be of the same word



(a) Probability of correct match



(b) Rate of indecision, $\log_{10}$ scale on the y-axis

Figure 5.8: This figure contains the plots of the probability of finding the correct match and the rate of indecision when the most similar recordings are required to be of the same word before a match is decided, for the top $\{1, 2, 3, 4, 5, 6\}$ similarities out of 6 recordings from each candidate word. For a comparison, the case for using a single recording from each candidate is also included.

Figure 5.8a shows the probabilities when taking 6 recordings of each candidate word, and requiring the top 1, 2, 3, 4, 5, or 6 similarity scores with the unknown word to be of the same candidate word before deciding a match has been found. For comparison, the case of using only a single recording from each candidate word is added as well. The plot shows that there are different effects of increasing the number of top matching recordings that need to be of the same word. For all words, the probability to find a correct match initially goes up to well above the “1 out of 1” bar. As the required top samples increases towards 6 however, the probability of matching goes down again, in some cases below the “1 ouf of 1” bar.

An explanation for this is that there may be two effects at play here. The first effect is that as the number of included recordings from candidate words increases, the number of included variations increases. Since words are pronounced slightly differently by different people from for example different communities (with dialects for example), not all recordings are described with the same phonetic string. As the number of used recordings per candidate word increases, the chance of including different spoken versions increases as well. This increases the chance that the version of the spoken unknown word is included among one of the chosen recordings of each word.

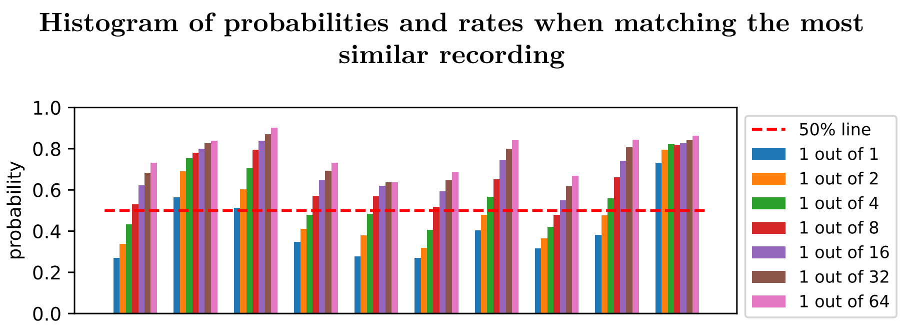
The second effect that comes into play is that as the minimum number increases of recordings with the highest similarity scores that need to be of the same word, it becomes more difficult for a match to be found - after all the top n similar recordings need to be of the same word. Going back to the example of the dialects, if two different dialects are included, and they both need to have the highest similarity to the unknown words before a match is found, then it becomes nearly impossible to find the correct match, as the given unknown recording follows only one of the two included dialects. This causes the chance to decrease of finding a match, and increases the chance for ‘noisy’ or bad recordings from other candidate recordings to be in between the top similarity scores match. This would also cause the indecision rate to increase.

Looking at the plot for the indecision rates in figure 5.8b in $\log_{10}$ scale on the y-axis, the second effect can be confirmed, as this plot clearly shows that it becomes exponentially more difficult to match an unknown recording with one of the candidate recordings, as the minimum number of required top similarities increases to 6. The indecision rates increase to such high numbers, towards 100 retries, that this would not be usable in applications.

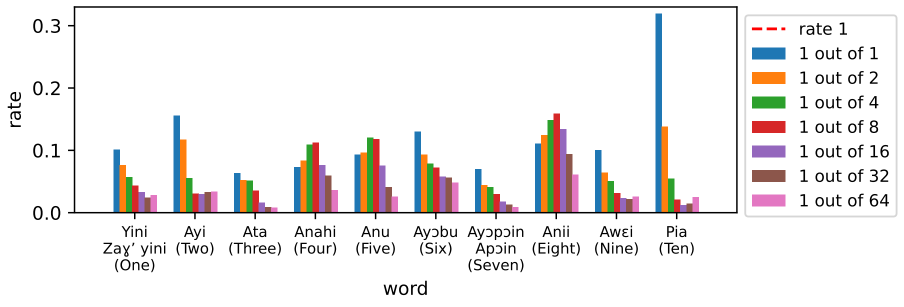
Highest similarities

The two described effects at play in the results in figure 5.8 represent a positive and a negative effect. To explore only the first effect, an increasing number of sample recordings can be taken of each candidate word, after which simply the single most similar recording is taken as the best matching recording, after which the word it represents is taken as the word representing the unknown word.

The plots in figure 5.9 show the results of this experiment, the single most similar recording to the given recording with unknown word is taken as the matching recording, and this is done for an increasing selection of unique recordings per candidate word. The probabilities in figure 5.9a show an increase for every word every time the number of used recordings is increased. For the words with an initially lower matching probability, the probability increases



(a) Probability of correct match



(b) Rate of indecision, y-axis cut off at just above 0.3.

Figure 5.9: This figure shows a plot similar to that in figure 5.8, but for using the most similar recording to decide which candidate word matches an 'unknown' recording when $\{1, 2, 4, 8, 16, 32, 64\}$ recordings are taken of each candidate word.

more strongly than for those that already have a 'high' probability. All probabilities seem to move towards slightly above 0.6 or slightly above 0.8, after which increases become minimal. This may be an effect of the noise, the data set is not filtered and by listening to random recordings, a small but significant fraction of them does not contain clearly spoken words.

Looking at the second plot for the rate of indecision in figure 5.9b, the rate goes down on average for all words, the rate for 'pia' start out at nearly 0.3, and moves down to below 0.03, nearly all rates move below 0.05 except for those for 'anii' and 'ayɔbu'. However, there are words for which an increase is initially visible, like 'anahi', 'anu' and 'anii'. This may again be a case in which multiple effects are at play.

The idea behind these effects is that for any recording, there will be many imperfectly matching recordings, and only few perfectly matching recordings. Or in other words, the chance is higher to get an imperfectly matching recording than a perfectly matching one. Since the words 'anahi', 'anu' and 'anii' are phonetically very close together, imperfectly matching recordings are more likely to be similar between the words. As the number of included words to choose between increases, the likelihood becomes higher that at least one imperfectly matching recording is included for a word. This then increases the chance of an indecision to happen. If a perfectly matching recording were to be included, a match would happen with this recording and no indecision would happen. As the number of included recordings increases, the chance for a perfectly matching recording to be included also goes up, but lacks behind the chance of inclusion of an imperfectly matching recording. However, at some point the chance is at such height that it starts outperforming the factor related to imperfectly matching recordings, and then with each increase in number of included recordings, the indecision rate goes down as it becomes more likely a perfectly matching recording is included.

Mathematically, the previous paragraph can be explained as follows. The chance of a recording of a certain type to be included is defined as r. The chance of this type of recording to not be included when n recording are chosen is $(1-r)^n$, and thus the chance of at least one recording of this type to be included given n chosen characters is $1-(1-r)^n$. Now take two types of recordings, for example the type of perfectly matching recordings, and that of imperfectly matching recordings, and define their chances of being included as respectively s and t. The ratio of how much more likely it is for one type to be included over another type is then defined as $\frac{1-(1-s)^n}{1-(1-t)^n}$. As n increases, the ratio becomes closer to 1, or also written as

$$\lim_{n\to\infty}\left(\frac{1-(1-s)^n}{1-(1-t)^n}\right)=1, \tag{5.1}$$

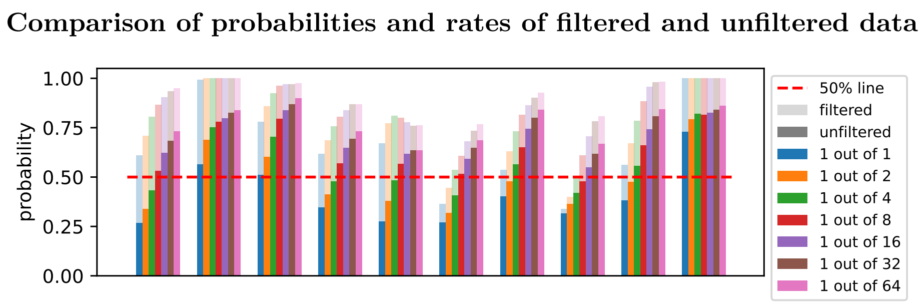
where it does not matter if $s>t$ or $s<t$, as long as $\{s,t\}\leq 1$. This means that as the number of included recordings n to choose from grows, the chances of either type of recording to be included becomes closer. In the case an imperfectly matching recording is included, this may cause an indecision, while an included perfectly matching recording will almost always prevent an indecision. This means that including a perfectly matching recording is more consequential than including an imperfectly matching recording, which creates a point at which n is high enough, so that the probabilities for including at least one of either types are close enough, for the effect of the perfectly matching recording to become more noticeable than that of the imperfectly matching recording. This effect can be seen in the plot in graph 5.9b.

Filtering noise

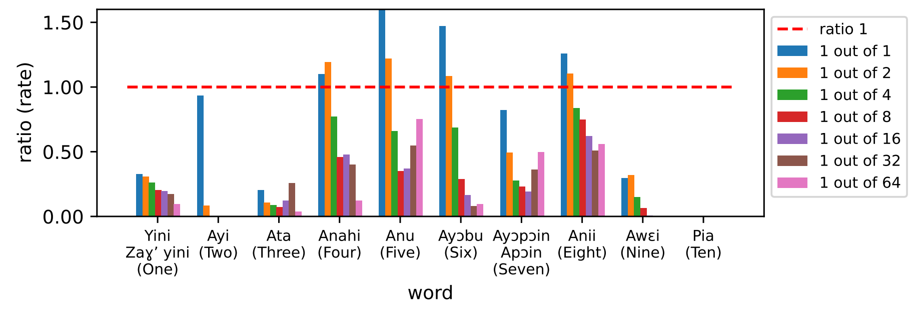
An explanation for the apparent ceiling in correct matching probability in figure 5.9a at around 0.8 is that noise prevents more correct matches from being found. This can be tested.

Some filtering of noise is possible by looking at the in-word similarities plotted in figure 5.3. If the average similarity of one recording to the other recordings of the same word is low, then that would mean that the recording has little overlap in phonetic structure to the other recordings, which could indicate it is a noisy recording. By selecting only the recordings that have a high average similarity to other recordings, the noisy recordings can technically be filtered out.

To do this in a very rough way, the upper 50% of the recordings with highest average similarity score can be taken, the algorithm used for the plots in figure 5.9 can be run on this selection of recordings. Note that this method of filtering noise is too rough to be used in applications, and is in this case only used to test the idea around the impact of noise.



(a) Probability of correct match



(b) Ratio of the rate of indecision for filtered data over unfiltered data

Figure 5.10: This figure shows the experiment from figure 5.9 for filtered data compared to unfiltered data. The plot for the probabilities shows the light colored probabilities for filtered data with those for unfiltered data more clearly on top. The second plot shows the ratio of the rates of indecision with the rate of filtered data over the rate of unfiltered data.

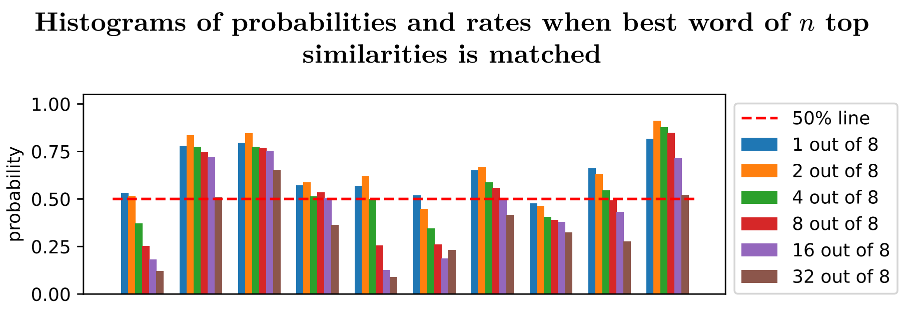
Figure 5.10 shows the results. In the top figure 5.10a, the probability of correctly matching are shown for the filtered and unfiltered data, and figure 5.10b shows the ratio of indecision rates, with the indecision rate of the filtered data divided by that of the unfiltered data. The top figure shows that probabilities of correctly matching become better with filtered data in all cases, with probabilities maxing out at nearly 100% for some words. Interestingly, for the word 'anu', the probability for a correct match eventually decreases slightly when more sampled recordings are included from each candidate words. It is not entirely clear why. It is clear however that overall, the rough filtering of data gives higher probabilities, which indicates the negative significance of noise on correctly matching, and shows that without noise the probabilities can go to nearly 100%.

The bottom figure 5.10b shows the rate of indecision for filtered data is lower for the cases that have a ratio below one, but the rate is higher when the ratio is shown as being above one. For the words 'anahi', 'anu', 'ayɔbu', and 'anii' the rate of indecision is higher at low numbers of included samples from candidate words. A reason for this may be the similarity between words. The words 'anahi', 'anu', and 'anii' all start with 'an[...]', which makes them similar, especially when the Jaro-Winkler similarity algorithm is used. The reason for the ratio between the indecision rates being above 1 is likely related to the same effects described in the text explaining figure 5.9b, but in a different way, with a ratio involving different probabilities for imperfectly matching and perfectly matching recordings.

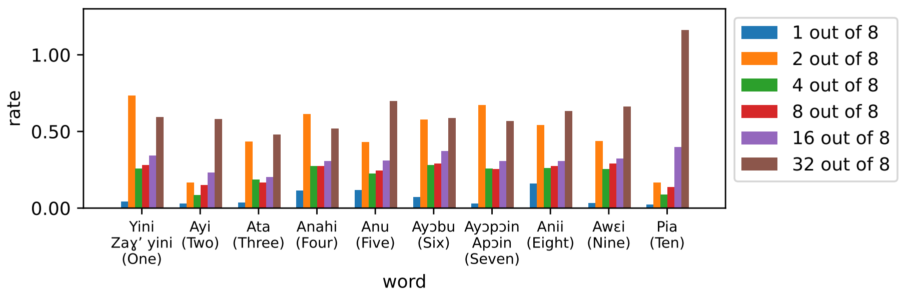
Overall there is a correct matching probability of 0.77 ± 0.09 for unfiltered data with an indecision rate of 0.030 ± 0.015, while for rough filtered data this is 0.90 ± 0.09 and 0.007 ± 0.011 respectively.

Most similarities in top n

The strategy behind the results in figure 5.8 is to only accept a match if the top similar recordings are all of the same word. This can be seen as very strict. To relax this, instead of requiring the top similarities to all be of the same candidate word, the most represented candidate word in a certain number of most similar recordings can be chosen as the word belonging with the unknown recording



(a) Probability of correct match



(b) Rate of indecision

Figure 5.11: This figure shows plots similar to those in figure 5.9, but for using the most similar $\{1, 2, 4, 8, 16, 32\}$ recordings to decide which candidate word matches an unknown recording when 8 recordings of each candidate word are chosen. The word that is most represented within the top similar recordings is the word that is chosen to match the given recording of an unknown word.

Results for this experiment for the correct matching probabilities and indecision rates are seen in figure 5.11. The possibly confusing legend require an explanation. For example "16 out of 8" means that from each candidate word 8 recordings were randomly picked, leading to a total of 80 recordings. Then from the most similar 16 recordings to the unknown recording, the candidate word from which the most recordings were in this selection of 16 was chosen as the word representing the given unknown recording.

The results show that this relaxed version of the results in figure 5.8 does not perform well either, with the probability of correctly matching going down as the number of used best similarities goes up. Similarly the rate of indecision goes up. Which means that both statistics get worse. Compared to the results in figure 5.8, the probabilities in this experiment go down more strongly, while the indecision rate goes up much less. The rate goes up less fast because the relaxation of the rule in figure 5.8 allows for some (noisy) recordings of other words to show up in the top n similarities without preventing a match from being found. At the same time, the effect that caused an initial increase in probability in figure 5.8 seems to be less strong here. Oddly, the indecision rates for “2 out of 8” are surprisingly high compared to other values. For this, similar effects may be at play as those in figure 5.8, but with each effect having a different impact.

Comparing single and n best similarities

Figure 5.8 showed the chance of correctly matching when the n recordings with the best similarity scores all need to be of the same candidate word. It showed that as n (out of 6) increased, the probability of a correct match increases, before going down, while the indecision rate increases exponentially in all cases of increasing n. The result in figure 5.9 showed a significantly improved situation. Combining the result from figures 5.8 and 5.9, there may be a case ‘in the middle’, which gives the best results.

To test this, an increasing number of recordings of each candidate word have been taken, with 1 to 5 of the top similarities having to be of the same word before a match is decided. This means that if 15 recordings of each candidate word are used, a total of 150 recordings are used for 10 words. A single recording is then taken, which is of course not in the 150 recordings, and the similarity with each of the 150 recordings is calculated. The best n similarities are then taken and if these are all of the same word, a match is found.

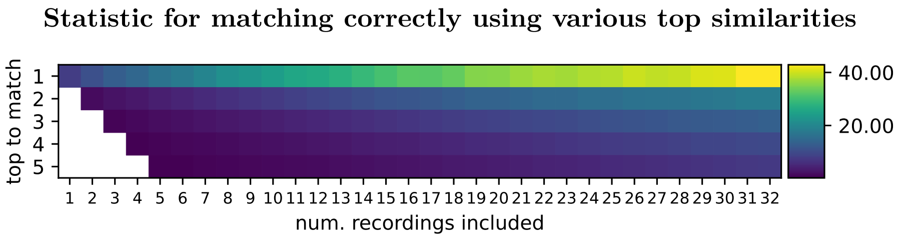


Figure 5.12: This figure shows the values of the statistic of formula 4.10 for corrected average correct matching probabilities and indecision rates for requiring the top 1 to 5 similarity scores to be for the same candidate word, out of 1 to 32 recordings of each candidate word, before a match is decided.

The plot in figure 5.12 is created for 1 to 32 recordings of each word, and requiring the top 1 to 5 similarities to be the same word. The numbers in the plot are calculated in the same way as those in figure 5.6c. The plot shows that using only the single most similar recording to decide on a match performs the best over all others for all numbers of included recordings per candidate word.

As the number of used recordings increases, the results become better, which is similar to what was seen in figure 5.9.

Plot of probabilities and rates using most similar recording as match

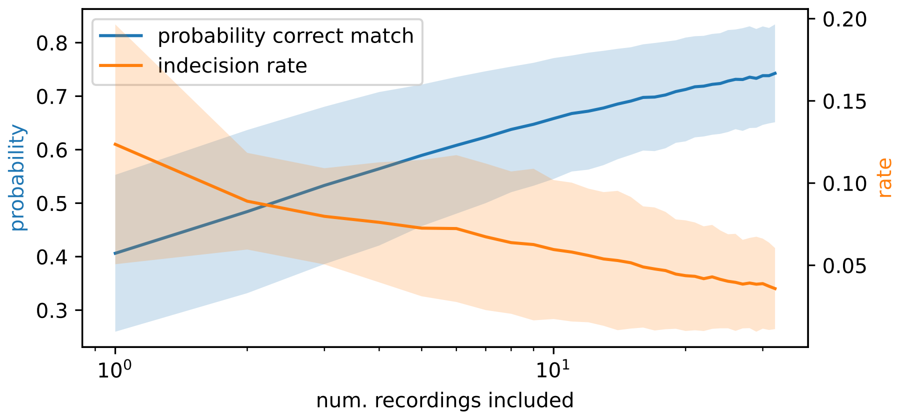


Figure 5.13: This figure shows the plot of correct matching probability on the left and indecision rate on the right for taking the most similar recording from the candidate words as the one matching the unknown recording. This is the top row in figure 5.12.

To confirm the relation between the probabilities and rates, and the number of recordings taken from each candidate, while using only the most similar recording to match an unknown word, the plots in figure 5.13 have been created. With the x-axis on a logarithmic scale, it can be seen that relation for both the rates and probabilities is roughly linear with the logarithm of the number of included recordings per word. This indicates that the more recordings are included in the comparison, the higher the correct matching probabilities and the lower the indecision rate, but that the positive effects of including more recordings goes down as more are included.

Chapter 6

Conclusion

This thesis attempted to contribute towards solving the problem with matching spoken words from rare languages from which not enough material is available to train a full neural network solution. It attempted to do this for words in Dagbani for one to ten, by using three steps:

1. creating a phonetic transcription of a recording,
2. generalizing this transcription using one or a combination of
   - Soundex,
   - NYSIIS, or
   - Metaphone, and
3. matching the generalized strings using calculated similarities with one or a combination of
   - the Hamming distance,
   - the Levenshtein distance,
   - the Damerau-Levenshtein distance,
   - the Jaro similarity, or
   - the Jaro-Winkler similarity.

These three steps could be combined in many ways, leading to a high-dimensional problem, which was explored by changing one variable at a time, and changing parameters for next experiment using results from previous experiments.

The combination of generalization algorithms Soundex, NYSIIS, and Metaphone with the similarity algorithms Damerau-Levenshtein, Jaro, and Jaro-Winkler gave the best results when it comes to a combination of correct matching probabilities and indecision rates.

The results were the best for simply taking the candidate word behind the recording most similar to the unknown recording, and using that word as the word spoken in the unknown recording. This was confirmed using 64 sampled recordings from each candidate word. For the used Dagbani words, the probability of a correct match is 0.77 ± 0.09 and the indecision rate is 0.030 ± 0.015 for unfiltered data, and respectively 0.90 ± 0.09 and 0.007 ± 0.011 for rough filtered data. The result in figure 5.13 show that the probabilities and rates move further into the right directions as more sample recordings are included from candidate words.

It is very likely that these results hold for combinations of other words as well, especially as those words are often more detailed and contrastive to each other than the words for the letters 1 to 10 used in the experiments, leading to greater differences in phonetic transcription and thus likely higher correct matching probabilities and lower indecision rates. Further, this would not only hold for the Dagbani language, but for any language that phonetically overlaps with the language of the acoustic model, in this case English.

These results contribute to performing automatic speech recognition on rare language. The methods are are suitable for use in low-resource environments due to their energy-efficient way of working with recordings of speech. With this, the thesis attempts to improve the adoption of technologies in these environments, as it opens up paths towards speech-based applications suitable for low-resource languages, which will help with development in the areas in which these languages are used.

Chapter 7

Discussion

The results from the experiments in this thesis are promising. Several aspects need to be kept in mind however when building further on these results.

There is a random component to these results. This means that in the situations in which numbers are very close to each other, like in table 5.3, this randomness might have a significant effect that could change results upon rerunning experiments. However to minimize this risk, when samples where taken that included randomness, a total of 10000 samples have been taken for each calculated result.

A neural network was used in these experiments. A major downside from neural networks is the lack of ability to explain results. Artificial neural networks consist of a very large number of nodes (million of even billions of nodes) which perform an equally or even larger number of calculations. When training, the weights and biases in the nodes are gradually adjusted to improve output towards what is expected. While this happens, the network implicitly finds patterns and discovers concepts that it uses during calculations. However, these discovered patterns and concepts are not clearly communicated in the end results. Therefore the network starts acting as a bit of a 'black box' [153], to which data is given, then "calculations happen", and output is given which usually comes close to what is expected. While the network used in this report seems to give reasonable output on average, there is a chance that the network gives very unexpected output in specific cases, which could lead to very negative effects in user experience. While useful, for these reasons, there can be trust issues around neural networks [46].

The words used in the experiment have been limited to the Dagbani words for one to ten, and the words for yes and no in some initial experiments. There are many more words in the language for which experiments have not been conducted. The reasonable expectation is that results would be equally good for other words in either Dagbani or other languages, and perhaps even better since other words are likely more contrastive than those in these experiment.

For the correction and combination of correct matching probabilities and indecision rates, formulas were created. While these formulas have reasoning behind them that explain the decisions made in their design, there may be alternatives that are better suited for the task, and may improve the results.

7.1 Further research and implementations

Next to the experiments done in this thesis, a lot more can be experimented with. Either closely related to the goal of this thesis, or to somewhat different goals. The theories are also very promising for future implementations. The following sections contain ideas and what may be possible with further research and implementations, however it should be noted that these ideas are untested, and may not work out as implied in these sections.

7.1.1 Personalization

Words from the same language are not always spoken the same, due to dialects and accents, as was also explained in section 4.1. This means that when matching a word spoken, there is a chance it will not match well due to this. This problem can be greatly reduced or even solved.

An idea is to perform personalized matching. This means matching words spoken by one person against words spoken by that same person. This ensures that any differences in how the person pronounces words compared to others are accounted for by their own personalized data set.

How this may work in practice is as follows. If a user communicates with some implementation, the implementation needs to be able to understand words initially. In the beginning there may be a relatively low but noticeable indecision rate or probability of matching a wrong word. An implementation would check this with the user, for example by repeating the word said and then pronouncing a version of the word that was understood. The user can then either speak the word again or agree that it was correctly interpreted.

If the word is correctly interpreted, it is added to a separate 'group'. This group lives next to the groups the implementation starts out with. For example, in the case of a word for 'house' and a word for 'tree', the implementation starts out with a two groups - one for each word. When the user correctly matches a word, a new group is created for that word, in which from then on the correct matches are stored. This means that after some use four groups exist. The two initial groups for 'house' and 'tree', and the two groups containing only correctly matched recordings for those same words 'house' and 'tree'.

Matching can then happen in three ways. The first way is that all four groups are kept in use by the implementation. Whenever a recording is given, it is matched to all four groups and the most likely group is chosen. In this case there are two groups of each word to choose from, where the personalized group is assumed to be matching better. Since the recordings will match with a higher similarity to the personalized groups, those two groups will be largely at play, while the initial other two groups can be seen as a 'fallback'.

A second way of using this is by deciding on a moment to switch over completely to the personalized groups, effectively discarding the two initial groups. This method has two upsides to the first method. The first upside is that any noise from the initial groups is not attempted to be matched with anymore, and can therefore not interfere with the results. Secondly, different dialects or ways in which words are pronounced are taken into account with the personalized group and can be handled, which can be especially interesting for people that have speech disabilities. A downside is that the matching is then optimized to the person it was personalized for, or people who speak in exactly the same way, so correctly matching with how other people speak the words may become more difficult. If someone else is to speak, the implementation could switch back to also using the two original groups, but this may be difficult to automate.

Finally, something in the middle can be used. That is to match first with the two personalized groups. If the similarity score is above a certain threshold for either of these groups, a decision for a match is made. If the similarity score is lower than the threshold, the recording is matched again with this time all four groups, and a decision is made on the best match. This has as upside that any noise from the two initial groups will not interfere with the personalized matches, and in the case of someone else speaking, a switch can be made to the original two groups.

To test these ideas, experiments need to be conducted in the field, which can best be done with all three discussed variants. It is also especially interesting to conduct this experiment with users of different dialect of the same language. Afterwards the personalized groups can be matched against each other, and the different dialects should be clearly visible through the similarity scores.

7.1.2 Clustering

The in-word similarity distributions in the plot in figure 5.3 and the discussion around it explained that there seem to be three major explanations for the distribution of the average similarity scores:

1. In the case of a single clear density of high average similarity scores, the word is clearly defined with a single pronunciation, causing the one clear density in the distribution.
2. In the case of roughly two high densities of average similarity scores around 0.5, there are likely two pronunciations the word at play, which both match themselves, and not the other, leading to an average similarity score of near 0.5. And, for example in the case of three high densities at around 0.3, this may be the case of three versions of the spoken word.
3. In the case of a more spread out distribution, there may be some spectrum of ways in which the word can be spoken, or a lot of noise is present in the recordings - more than in recordings from other words. A reason for this could be that the phonetic data extracted from the recordings is not detailed, and a lot of information is lost, leading to noise having a higher impact on the similarity scores.

These three explanations create the idea of clustering average similarities together. By doing this, it may be possible to separate out the different ways of pronouncing a word. This allows for two important uses. The first is from a perspective of linguistics. It may allow for a kind of automatic segmentation of dialects [48], [183], which could help with problems in speech recognition [45] or incentivize future research in this. The second use is in matching words. If the different pronunciations can be clearly separated from each other, they may be put in separate groups of recordings, possibly allowing for more accurate matching of a spoken word. It may also allow for a different type of personalization than discussed in section 7.1.1. When enough words are matched, the implementation may decide which cluster best fits the pronunciations of the user and continue matching against only this cluster for a certain word.

7.1.3 Searching sentences

The output from the network includes probabilities for the characters `#` and `|`. Here, `|` represents a break between words. Since this thesis focuses on matching words, these characters are simply removed. This means that if multiple words are spoken, they are concatenated together.

By not concatenating the words together, but rather splitting the results from the network into multiple words using the `|` character, technically multiple words can be recognized and matched in spoken sentences. For example, one might speak a sentence from which multiple words are identified. Then with another set of spoken words, this sentence can be searched. Using this, a search index can be created from larger spoken texts, which can then be searched with spoken words.

It is unclear at the moment how reliable predictions for the `|` characters are, but it is reasonable to assume that a high probability can be assigned up to the same level in which the probabilities for alphanumerical characters can be trusted. Further, the research done in this thesis was done on only a small set of words. Given enough contrast between different words, a very large set of words may be supported, but to which extend is not clear at the moment. This means that very long recordings of speech may run into the problem of having too many words, which could greatly decrease the accuracy of predictions.

7.1.4 Data filtering

In the thesis, a very rough approach was taken to filtering of the data set, done for a simple and rough comparison to test an idea. This filtering was done using the in-word matches, by simply taking the top 50% of recordings for each word that had the highest average similarity scores to the other recordings of the same word.

To reiterate, the idea is that valid recordings - meaning recordings that are not noisy - are very similar to each other. While recordings with a high amount of noise, or recordings that do not contain the word at all, do not have many recordings similar to them. With this in mind, the recordings with the highest similarity scores are likely those with the least noise - or the most 'correct' recordings. By selecting these recordings, the invalid recordings are effectively filtered out.

The approach taken here was to simply take the top 50%. However, this is too rough, and either keeps a lot of noisy recordings in, or takes too many valid recordings out. More research should be done on how to best filter recordings. An idea could be to use a clustering approach similar to that in section 7.1.2, but this time to find the clusters of recordings that are most likely valid, and filter the set of recordings based on that.

7.1.5 Port to PyTorch

For this thesis, the Flashlight library [82] is used for the model to create a phonetic transcription of recordings of speech. Flashlight is not very well known and may be difficult to get used to, especially as it is written in C++ instead of the more popular [157], [169] Python. The most popular library for Python to run and develop neural networks with is PyTorch. Porting the neural network to

PyTorch would allow for only Python to be used, and may increase participating in this project as more people would be able to understand and work with the network itself. Do note that the major part of the implementation is written in Python, only the first step is not, that of using the network.

7.1.6 Fine-tuning the network

A network was used that outputs probabilities for an English phonetic description of audio. A downside of this is that the network will likely only give good results for the phonetic characteristics of the rare language that overlap with the English language. This causes significant information that may get lost during the processing. Fine-tuning the network may help in these cases.

There are various ways in which artificial neural networks can be fine-tuned. The idea behind fine-tuning [105], [123] is that the network has already reached the ability to find patterns to match sounds with. With fine-tuning, this already learned information would be slightly altered or adapted for a new model to allow for better handling of slightly different input data. This can only be done if the data on which fine-tuning happens is sufficiently close in input features to the data the network was originally trained on, and the new expected output is close as well, so that the network is not required to learn entirely new concepts.

Three methods that can be looked into are mentioned here. First of all, the entire network can be trained from the state it currently has [99], [102]. The number of output nodes cannot be changed in this case, which means that there are only 29 output nodes available from the network to infer the probabilities for characters with. The characters represented by these 29 nodes may be changed to the characters that are most frequently used in the rare language. After fine-tuning, the 29 nodes should represent the majority of the English characters overlapping with the rare language, and the few characters in the rare language that are not found in the English language.

A second method of fine-tuning is by altering the network, while keeping large parts of the network unchanged [180], using for example REFT [188]. Another example is fine-tuning only the top layer in a network [7], [152], the last layer in the network used here is a linear network with 384 input nodes and 29 output nodes. This means the 384 input nodes contain all information needed to construct the values for the 29 output nodes. This layer can be removed and replaced by one or a few new layers, where the first layer also has 384 input nodes, but where the last layer has a number of output nodes equal to the number of characters in the other language. Retraining would then happen with only the last newly added layers, and the rest of the network would not be retrained. This has the upside that it is cheaper as there are fewer nodes to perform backpropagation for, and it has the upside that the number of output nodes can be changed. The downside however is the assumption is that all information required to construct probabilities for the rare language are encoded in the 384 nodes coming from the first part of the network, which may be not entirely the case, since there may be phonemes in the rare language that do not exist at all in the English language, and for that reason are not encoded in the 384 input nodes either.

A third method of fine-tuning is by constructing a second network next to the trained network using PEFT methods [98], [189] like LoRA [67]. The trained network forms the 'main' part of calculations, while the second network would be intertwined with the trained network, to ‘guide’ the calculations into certain directions so they are better suited for the rare languages. This method may be combined with the second method for better results.

7.1.7 Other acoustic models

The probabilities for the characters inferred by the acoustic models are normalized and matched using algorithms that are designed for the English language. Languages that are phonetically close to English are very well supported by this approach. However there are many languages that are not phonetically close to English.

Other languages can be looked into, like French. Due to the history of French on the African continent and other locations around the world, many languages are phonetically close to French [106]. Using English acoustic models for these languages would likely not yield good results. Using French acoustic models would, although this is not investigated in this thesis.

For further projects it would be interesting to look into acoustic models other than English, and see how this performs on matching between samples of various languages. This could be automated in some way - some experiments in this thesis can be repeated for each available acoustic model, and the acoustic model that matches best is taken for further use with this language.

This could be further automated in applications. Allowing people to speak a completely arbitrary language, then automatically identifying which acoustic model produces the best results with this language, and continue using this acoustic model from then on. Alternatively, multiple acoustic models could be used, in the same way that the use of multiple normalization and matching algorithms is done in this thesis. For example, if a language phonetically overlaps with both English and French, it may be interesting to use acoustic models for both languages, and use results from both to decide on a match.

7.1.8 Large Language Models

With the introduction of GPT-1 [131] and GPT-2 [132], the rise of the large language models was started [115]. Subsequent models showed emergent behaviors [181] to various degrees. Behaviors are said to be emergent in neural networks when they do not appear in smaller networks, but seem to somewhat ‘spontaneously’ appear in larger networks. Examples are the sudden appearance of the ability to unscramble words, or the appearance of the ability to perform various levels of arithmetic tasks [155].

In the theories around languages in section 1.1, the concept that each language is an abstract way of representing the same kind of information is discussed. As large language models grow and become multi-modal [5], [54], [120], [121], they are able to handle not only text, but also audio and video as input. These multimodal large language models may at some point show emergent behavior that shows they understand this underlying theory behind languages - the theory that binds different languages together. This may allow the network to very quickly adapt itself to understand languages from which only very little data is available, which may allow for rare languages to be understood up to the level in which popular languages can be understood.

Appendix A

Sample application

A simple application of the theory was implemented using pyTelegramBotAPI [47] for the Telegram API [166]. It allows for recordings to be registered under a certain string and for recordings to be matched to an earlier registered recordings. For using the source code of the application, please see the information in section B.1 and attachment A.

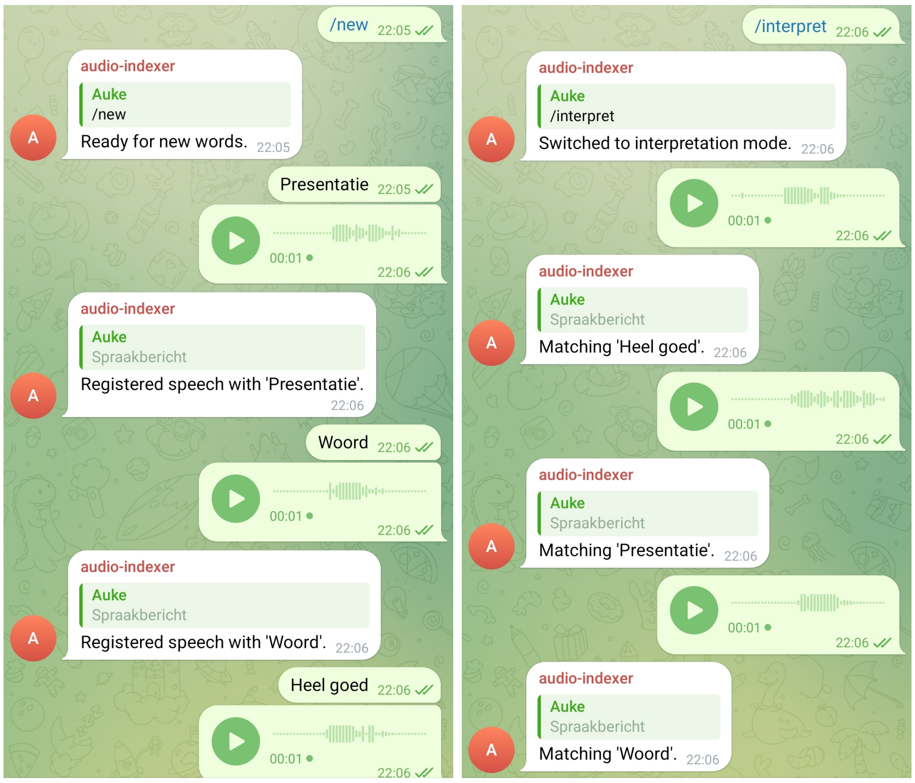
Recordings can be made and registered by typing `/new`, after which a string representing the incoming recordings should be typed and submitted, and then one or more recordings to register with that string. A confirmation then follows, after which another string and recording can be given.

To switch to the interpretation mode, `/interpret` needs to be typed, after which recordings can be submitted, and the bot replies with the string that matches the recording best. When command `/new` is submitted again, the application removes context and starts over.

Figure A.1 shows two screenshots of the bot. On the left is a screenshot of registering of strings and recordings with the `/new` command, and on the right is a screenshot of the interpretation mode with `/interpret`, in which recordings can be given to be matched. The words registered are in the Dutch language.

The bot is available under the name `audio-indexer`, and can be added to groups on Telegram. An effort will be made to keep the bot online for at least a year after finishing of this thesis. This is a very simple example, and is only meant as example to try out language matching. It does not allow for multiple recordings to be registered per word.

Screenshots of the sample application


/new 22:05

audio-indexer
Auke
/new
Ready for new words. 22:05

Presentatie 22:05

00:01 22:06

audio-indexer
Auke
Spraakbericht
Registered speech with 'Presentatie'. 22:06

Woord 22:06

00:01 22:06

audio-indexer
Auke
Spraakbericht
Registered speech with 'Woord'. 22:06

Heel goed 22:06

00:01 22:06

/interpret 22:06

audio-indexer
Auke
/interpret
Switched to interpretation mode. 22:06

00:01 22:06

audio-indexer
Auke
Spraakbericht
Matching 'Heel goed'. 22:06

00:01 22:06

audio-indexer
Auke
Spraakbericht
Matching 'Presentatie'. 22:06

00:01 22:06

audio-indexer
Auke
Spraakbericht
Matching 'Woord'. 22:06

Figure A.1: A simple application of the theory in the form of a Telegram bot using the Telegram API. The screenshot on the left shows registering new words after typing the command `/new`, and the screenshot on the right shows the interpretation phase after typing command `/interpret`.

Appendix B

Source code

Source code for the implementation, the experiments and the application in appendix A is made available and explained here.

B.1 Package

To make the ideas and results from this thesis usable in other applications, the source code is published[16] [148]. This code contains a Python package, C++ code, and a Dockerfile for building and running various parts.

B.1.1 Flashlight

For the experiments a trained neural network was used to infer probabilities for characters over time steps of audio, which is loaded using Flashlight [82] and compiled C++ code. The trained models come in two sizes and are available [97] with a size of 70 million parameters[17] and a size of 300 million parameters[18], where the model with 70 million parameters was used in this thesis, as the work is aimed at people and organisation with lower resources, and so less expensive models are more favorable to run. The currently actively developed Flashlight version is version 0.4.0 and has a problem with loading the models [81], so

[16]Repository on Github: `https://github.com/W4RA/speechmatching`

[17]Network with 70 million parameters at URL `https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_70Mparams.bin` (archived at URL `https://web.archive.org/web/20230603172217/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_70Mparams.bin`) with network structure defined at URL `https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_70Mparams.arch` (archived at URL `https://web.archive.org/web/20230603180553/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_70Mparams.arch`).

[18]Network with 300 million parameters at URL `https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_300Mparams.bin` (archived at URL `https://web.archive.org/web/20220729100332id_/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_300Mparams.bin`) with network strcture defined at URL `https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_300Mparams.arch` (archived at URL `https://web.archive.org/web/20230523044211/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_300Mparams.arch`.

version 0.3.1[19] is used instead. Next to the model, a list of tokens is required. This list is static for the pre-trained models and can be downloaded as well[20]

The required version of Flashlight is relatively difficult to compile, as it requires certain versions of packages. While these version could be compiled and stored in the `/opt` directory, the decision was made to have these in a Docker image. The repository contains a `Dockerfile`, which will install the required versions, build the C++ code to run the models using Flashlight, and install the Python package using `pip`. The compiled binary file stored is then available from `/opt/bin/acoustic`, and models are downloaded to the default location in the container `~/.cache/acoustic`. The Docker image is available online for pulling under name `aukesch/speechmatching`.

Please see the README in the documentation in attachment A for details on how to run this with Docker.

B.1.2 Running

As the software required to run this is compiled in a Docker image, this Docker image is involved in running the code. Two ways are available in which the code can be run. The first is using Docker alone, meaning that software written to use the `speechmatching` package is run in a Docker container as well.

The second method is by running the Python code locally, with the Docker container on the side, which the Python code will communicate with. For the second method to run well, it is important to make sure the Docker image with the `acoustic` binary is pulled from `aukesch/speechmatching` or built under name `speechmatching`, else it cannot be automatically found.

Please see the documentation in attachment A for more details, especially the first chapter and the examples.

B.2 Experiments and application

The experiments were in written in an IPython Notebook, and are included in the repository on Github as one of the examples. The application shown in appendix A is also available in the repository as an example. Instruction for how to run and use either of these are in the documentation in attachment A.

[19]Flashlight v0.3.1: `https://github.com/flashlight/flashlight/releases/tag/v0.3.1`.

[20]The `tokens.txt` file at URL `https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/tokens.txt` (archived at URL `https://web.archive.org/web/20220729100540/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/tokens.txt`).

Bibliography

[1] Z. K. Abdul and A. K. Al-Talabani, "Mel frequency cepstral coefficient and its applications: A review," *IEEE Access*, vol. 10, pp. 122 136–122 158, 2022, ISSN: 2169-3536. DOI: 10.1109/access.2022.3223444. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2022.3223444.

[2] H. Abubakari and S. A. Issah, "Nominal classification in mabia languages of west africa," *Language Sciences*, vol. 95, p. 101 514, Jan. 2023, ISSN: 0388-0001. DOI: 10.1016/j.langsci.2022.101514. [Online]. Available: http://dx.doi.org/10.1016/j.langsci.2022.101514.

[3] A. F. Agarap, *Deep learning using rectified linear units (relu)*, 2018. DOI: 10.48550/ARXIV.1803.08375. arXiv: 1803.08375 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1803.08375.

[4] A. Agbeyangi and H. Suleman, "Advances and challenges in low-resource-environment software systems: A survey," *Informatics*, vol. 11, no. 4, p. 90, Nov. 2024, ISSN: 2227-9709. DOI: 10.3390/informatics11040090. [Online]. Available: http://dx.doi.org/10.3390/informatics11040090.

[5] P. Agrawal, S. Antoniak, E. B. Hanna, *et al.*, *Pixtral 12b*, 2024. DOI: 10.48550/ARXIV.2410.07073. arXiv: 2410.07073 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2410.07073.

[6] J. N. Akpanglo-Nartey and R. A. Akpanglo-Nartey, "Some endangered languages of ghana," *American Journal of Linguistics*, vol. 1, no. 2, pp. 10–18, 2012, ISSN: 2326-0769. [Online]. Available: http://article.sapub.org/10.5923.j.linguistics.20120102.01.html, archived at https://web.archive.org/web/20240728125127/http://article.sapub.org/10.5923.j.linguistics.20120102.01.html on Jul. 28, 2024.

[7] P. D. Alfano, V. P. Pastore, L. Rosasco, and F. Odone, *Top-tuning: A study on transfer learning for an efficient alternative to fine tuning for image classification with fast kernel methods*, 2022. DOI: 10.48550/ARXIV.2209.07932. arXiv: 2209.07932 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.07932.

[8] D. Amodei, R. Anubhai, E. Battenberg, *et al.*, *Deep speech 2: End-to-end speech recognition in english and mandarin*, 2015. DOI: 10.48550/ARXIV.1512.02595. arXiv: 1512.02595 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1512.02595.

[9] S. Anderson, *Languages: A Very Short Introduction* (Very Short Introductions). Oxford University Press (OUP), 2012, ISBN: 9780199590599. [Online]. Available: https://global.oup.com/ukhe/product/languages-a-very-short-introduction-9780199590599.

[10] T. Apostol, *Mathematical Analysis* (Addison-Wesley series in mathematics). Addison-Wesley, 1974, ISBN: 9780201002881. [Online]. Available: https://openlibrary.org/works/OL2180423W/Mathematical_Analysis.

[11] R. Ardila, M. Branson, K. Davis, *et al.*, *Common voice: A massively-multilingual speech corpus*, 2019. DOI: 10.48550/ARXIV.1912.06670. arXiv: 1912.06670 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1912.06670.

[12] M. Aturban, M. L. Nelson, M. C. Weigle, M. Klein, and H. Van de Sompel, *Collecting 16k archived web pages from 17 public web archives*, 2019. DOI: 10.48550/ARXIV.1905.03836. arXiv: 1905.03836 [cs.DL]. [Online]. Available: https://arxiv.org/abs/1905.03836.

[13] P. Austin and J. Sallabank, *The Cambridge Handbook of Endangered Languages* (Cambridge Handbooks in Language and Linguistics). Cambridge University Press (CUP), 2011, ISBN: 9780521882156. [Online]. Available: https://www.cambridge.org/universitypress/subjects/languages-linguistics/sociolinguistics/cambridge-handbook-endangered-languages.

[14] F. Azam, "Biologically inspired modular neural networks," Ph.D. dissertation, Virginia Tech, 2000. [Online]. Available: https://vtechworks.lib.vt.edu/bitstreams/c6cd9bfa-d273-4fce-ae7a-7061d177e290/download (visited on Aug. 3, 2024), archived at http://web.archive.org/web/20240803171155/https://vtechworks.lib.vt.edu/bitstreams/c6cd9bfa-d273-4fce-ae7a-7061d177e290/download on Aug. 3, 2024.

[15] P. Azunre and N. D. Ibrahim, "Breaking the low-resource barrier for dagbani ASR: From data collection to modeling," in *4th Workshop on African Natural Language Processing*, 2023. [Online]. Available: https://openreview.net/forum?id=1je9lI9zV8 (visited on Jul. 20, 2024), archived at https://web.archive.org/web/20240728125045/https://openreview.net/forum?id=1je9lI9zV8 on Jul. 28, 2024.

[16] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. DOI: 10.48550/ARXIV.1607.06450. arXiv: 1607.06450 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1607.06450.

[17] M. Bain, J. Huh, T. Han, and A. Zisserman, *Whisperx: Time-accurate speech transcription of long-form audio*, 2023. DOI: 10.48550/ARXIV.2303.00747. arXiv: 2303.00747 [cs.SD]. [Online]. Available: https://arxiv.org/abs/2303.00747.

[18] F. Baisden, L. Campbell, C. Cornelius, *et al.* "About the endangered languages project." (Jun. 2012), [Online]. Available: https://www.endangeredlanguages.com/about/ (visited on Jun. 5, 2024), archived at https://web.archive.org/web/20240728121354/https://www.endangeredlanguages.com/about/ on Jul. 28, 2024.

[19] S. Baldi and M. Adam, *Dagbani Basic and Cultural Vocabulary* (Studi africanistici / Istituto universitario orientale, Dipartimento di studi e ricerche su Africa e paesi arabi). Università degli studi di Napoli: L'Orientale, Dipartimento di studi e ricerche su Africa e paesi arabi, 2005, ISBN: 9788895044071. [Online]. Available: `https://openlibrary.org/books/OL16295714M/Dagbani_basic_and_cultural_vocabulary`.

[20] J. A. Bawa and G. N.-N. Marley, "The interplay between literacy skills, literacy use and the personal lives: An assessment of women of rural dagbon," *Journal of Education and Practice*, vol. 3, no. 13, 2012, ISSN: 2222-288X. [Online]. Available: `https://www.iiste.org/Journals/index.php/JEP/article/view/3049`, archived at `https://web.archive.org/web/20240728125241/https://www.iiste.org/Journals/index.php/JEP/article/view/3049` on Jul. 28, 2024.

[21] R. Berwick and N. Chomsky, *Why Only Us: Language and Evolution* (The MIT Press). MIT Press, 2016, ISBN: 9780262034241. [Online]. Available: `https://mitpress.mit.edu/9780262034241/why-only-us/`.

[22] A. Binstock and J. Rex, "Metaphone: A modern soundex," in *Practical Algorithms for Programmers.* Addison-Wesley, 1995, pp. 160–169, ISBN: 9780201632088. [Online]. Available: `https://openlibrary.org/works/OL3746935W/Practical_algorithms_for_programmers`.

[23] R. Bird. "Dealing with link rot – are dois the cure?" (Apr. 2014), [Online]. Available: `https://www.slaw.ca/2014/04/11/dealing-with-link-rot-are-dois-the-cure/` (visited on Aug. 1, 2024), archived at `https://web.archive.org/web/20240801101945/https://www.slaw.ca/2014/04/11/dealing-with-link-rot-are-dois-the-cure/` on Aug. 1, 2024.

[24] R. Blench, H. Blair, Tamakloe, *et al.*, *Dagbani-english dictionary*, Online, Cambridge, United Kingdom, Dec. 2004. [Online]. Available: `https://www.rogerblench.info/Language/Niger-Congo/Gur/Dagbani%20dictionary%20CD.pdf`, archived at `https://web.archive.org/web/20240724051255/https://www.rogerblench.info/Language/Niger-Congo/Gur/Dagbani%20dictionary%20CD.pdf` on Jul. 24, 2024.

[25] A. Bon, H. Akkermans, and J. Gordijn, "Developing ict services in a low-resource development context," *Complex Systems Informatics and Modeling Quarterly*, no. 9, pp. 84–109, Dec. 2016, ISSN: 2255-9922. DOI: `10.7250/csimq.2016-9.05`. [Online]. Available: `http://dx.doi.org/10.7250/csimq.2016-9.05`.

[26] L. Böttcher and G. Wheeler, "Visualizing high-dimensional loss landscapes with hessian directions," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2024, no. 2, p. 023 401, Feb. 2024, ISSN: 1742-5468. DOI: `10.1088/1742-5468/ad13fc`. [Online]. Available: `https://dx.doi.org/10.1088/1742-5468/ad13fc`.

[27] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition.* Springer US, 1994, ISBN: 9781461532101. DOI: `10.1007/978-1-4615-3210-1`. [Online]. Available: `http://dx.doi.org/10.1007/978-1-4615-3210-1`.

[28] E. Brill and R. C. Moore, "An improved error model for noisy channel spelling correction," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics - ACL '00*, ser. ACL '00, Association for Computational Linguistics, 2000. DOI: `10.3115/1075218.1075255`. [Online]. Available: `http://dx.doi.org/10.3115/1075218.1075255`.

[29] M. Bunge, "A general black box theory," *Philosophy of Science*, vol. 30, no. 4, pp. 346–358, Oct. 1963, ISSN: 1539-767X. DOI: `10.1086/287954`. [Online]. Available: `http://dx.doi.org/10.1086/287954`.

[30] P. Burnhill, M. Mewissen, and R. Wincewicz, "Reference rot in scholarly statement: Threat and remedy," *Insights the UKSG journal*, vol. 28, no. 2, pp. 55–61, Jul. 2015, ISSN: 2048-7754. DOI: `10.1629/uksg.237`. [Online]. Available: `http://dx.doi.org/10.1629/uksg.237`.

[31] L. Campbell and A. Belew, *Cataloguing the World's Endangered Languages*. Routledge, 2020, ISBN: 9780367580902. [Online]. Available: `https://www.routledge.com/Cataloguing-the-Worlds-Endangered-Languages/Campbell-Belew/p/book/9780367580902`.

[32] L. Campbell, "The handbook of linguistics," in *The Handbook of Linguistics*. Blackwell Publishing Ltd, 2002, pp. 81–104, ISBN: 9781405102520. DOI: `10.1111/b.9781405102520.2002.00006.x`. [Online]. Available: `http://dx.doi.org/10.1111/b.9781405102520.2002.00006.x`.

[33] C. Chanard and R. L. Hartell, "Dagbani sound inventory (aa)," in *PHOIBLE 2.0*, S. Moran and D. McCloy, Eds. Jena: Max Planck Institute for the Science of Human History, 2019. [Online]. Available: `https://phoible.org/inventories/view/690`.

[34] N. Chomsky, *Aspects of the Theory of Syntax* (ACLS Humanities E-Book v. 10). MIT Press, 1965, ISBN: 9780262530071. [Online]. Available: `https://mitpress.mit.edu/9780262530071/aspects-of-the-theory-of-syntax/`.

[35] A. Clifton, S. Reddy, Y. Yu, *et al.*, "100,000 podcasts: A spoken english document corpus," in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain: International Committee on Computational Linguistics, Dec. 2020, pp. 5903–5917. DOI: `10.18653/v1/2020.coling-main.519`. [Online]. Available: `http://dx.doi.org/10.18653/v1/2020.coling-main.519`.

[36] R. Collobert, C. Puhrsch, and G. Synnaeve, *Wav2letter: An end-to-end convnet-based speech recognition system*, 2016. DOI: `10.48550/ARXIV.1609.03193`. arXiv: `1609.03193 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1609.03193`.

[37] V. Craigle, A. Retteen, and B. J. Keele, "Ending law review link rot: A plea for adopting doi," *Legal Reference Services Quarterly*, vol. 41, no. 2, pp. 93–97, Apr. 2022, ISSN: 1540-949X. DOI: `10.1080/0270319x.2022.2089810`. [Online]. Available: `http://dx.doi.org/10.1080/0270319X.2022.2089810`.

[38] W. Croft and D. Cruse, *Cognitive Linguistics* (Cambridge Textbooks in Linguistics). Cambridge University Press (CUP), 2004, ISBN: 9780521667708. [Online]. Available: `https://www.cambridge.org/universitypress/subjects/languages-linguistics/cognitive-linguistics/cognitive-linguistics`.

[39] D. Crystal, *Language Death* (Canto Refresh Your Series). Cambridge University Press (CUP), 2002, ISBN: 9780521012713. [Online]. Available: `https://www.cambridge.org/nl/universitypress/subjects/languages-linguistics/english-language-and-linguistics-general-interest/language-death-2`.

[40] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, Mar. 1964, ISSN: 1557-7317. DOI: `10.1145/363958.363994`. [Online]. Available: `http://dx.doi.org/10.1145/363958.363994`.

[41] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, *Language modeling with gated convolutional networks*, 2016. DOI: `10.48550/ARXIV.1612.08083`. arXiv: `1612.08083 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1612.08083`.

[42] M. Diallo, C. Fourati, and H. Haddad, *Bambara language dataset for sentiment analysis*, 2021. DOI: `10.48550/ARXIV.2108.02524`. arXiv: `2108.02524 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2108.02524`.

[43] B. Duda, R. Kiełtyka, and E. Konieczna, *Culture, Cognition, Discourse and Grammar: Cognitive Considerations on Formulaic Language* (G - Reference,Information and Interdisciplinary Subjects Series). Peter Lang, 2019, ISBN: 9783631770160. [Online]. Available: `https://www.peterlang.com/document/1055483`.

[44] H. Dudley, R. Riesz, and S. Watkins, "A synthetic speaker," *Journal of the Franklin Institute*, vol. 227, no. 6, pp. 739–764, Jun. 1939, ISSN: 0016-0032. DOI: `10.1016/s0016-0032(39)90816-1`. [Online]. Available: `http://dx.doi.org/10.1016/S0016-0032(39)90816-1`.

[45] M. G. Elfeky, P. Moreno, and V. Soto, "Multi-dialectical languages effect on speech recognition," *Procedia Computer Science*, vol. 128, pp. 1–8, 2018, ISSN: 1877-0509. DOI: `10.1016/j.procs.2018.03.001`. [Online]. Available: `http://dx.doi.org/10.1016/j.procs.2018.03.001`.

[46] W. J. von Eschenbach, "Transparency and the black box problem: Why we do not trust ai," *Philosophy & Technology*, vol. 34, no. 4, pp. 1607–1622, Sep. 2021, ISSN: 2210-5441. DOI: `10.1007/s13347-021-00477-0`. [Online]. Available: `http://dx.doi.org/10.1007/s13347-021-00477-0`.

[47] eternnoir, Badiboy, *et al.*, *Pytelegrambotapi*, version 4.21.0, Jul. 2024. [Online]. Available: `https://pypi.org/project/pyTelegramBotAPI/` (visited on Jul. 20, 2024), archived at `https://web.archive.org/web/20240725124344/https://pypi.org/project/pyTelegramBotAPI/` on Jul. 25, 2024.

[48] A. Etman and A. A. L. Beex, "Language and dialect identification: A survey," in *2015 SAI Intelligent Systems Conference (IntelliSys)*, Institute of Electrical and Electronics Engineers (IEEE), Nov. 2015, pp. 220–231. DOI: 10.1109/intellisys.2015.7361147. [Online]. Available: http://dx.doi.org/10.1109/IntelliSys.2015.7361147.

[49] B. Everitt, *Cambridge Dictionary of Statistics.* Cambridge University Press (CUP), 1998, ISBN: 9780521593465. [Online]. Available: https://openlibrary.org/works/OL1687731W/The_Cambridge_dictionary_of_statistics.

[50] FFmpeg Developers, *Ffmpeg tool*, version 7.0.1, May 2024. [Online]. Available: http://ffmpeg.org/ (visited on Jul. 28, 2024), archived at https://web.archive.org/web/20240728123902/https://ffmpeg.org// on Jul. 28, 2024.

[51] S. Fort and S. Jastrzebski, *Large scale structure of neural network loss landscapes*, 2019. DOI: 10.48550/ARXIV.1906.04724. arXiv: 1906.04724 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1906.04724.

[52] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119–130, Jan. 1988, ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90014-7. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(88)90014-7.

[53] S. Gandhi, P. von Platen, and A. M. Rush, *Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling*, 2023. DOI: 10.48550/ARXIV.2311.00430. arXiv: 2311.00430 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2311.00430.

[54] Gemini Team, R. Anil, S. Borgeaud, *et al.*, *Gemini: A family of highly capable multimodal models*, 2023. DOI: 10.48550/ARXIV.2312.11805. arXiv: 2312.11805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2312.11805.

[55] D. H.-L. Goh and P. K. Ng, "Link decay in leading information science journals," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 1, pp. 15–24, Nov. 2006, ISSN: 1532-2890. DOI: 10.1002/asi.20513. [Online]. Available: http://dx.doi.org/10.1002/asi.20513.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (Adaptive Computation and Machine Learning series). MIT Press, 2016, Online version URL https://www.deeplearningbook.org/, ISBN: 9780262035613. [Online]. Available: https://openlibrary.org/works/OL17801809W/Deep_Learning.

[57] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: Proceedings of Machine Learning Research PMLR, Jun. 2014, pp. 1764–1772. [Online]. Available: https://proceedings.mlr.press/v32/graves14.html, archived at https://web.archive.org/web/20250129182055/https://proceedings.mlr.press/v32/graves14.html on Jan. 29, 2025.

[58] A. Graves, A.-r. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, 2013. DOI: 10.48550/ARXIV.1303.5778. arXiv: 1303.5778 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1303.5778.

[59] H. Hammarström, R. Forkel, M. Haspelmath, and S. Bank, *Glottolog/glottolog: Glottolog database 5.0*, 2024. DOI: 10.5281/ZENODO.596479. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.596479.

[60] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, Apr. 1950, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1950.tb00463.x. [Online]. Available: http://dx.doi.org/10.1002/j.1538-7305.1950.tb00463.x.

[61] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://dx.doi.org/10.1038/s41586-020-2649-2.

[62] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève, "Ted-lium 3: Twice as much data and corpus repartition for experiments on speaker adaptation," in *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pp. 198–208, ISBN: 9783319995793. DOI: 10.1007/978-3-319-99579-3_21. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-99579-3_21.

[64] G. Hinton, L. Deng, D. Yu, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012, ISSN: 1053-5888. DOI: 10.1109/msp.2012.2205597. [Online]. Available: http://dx.doi.org/10.1109/MSP.2012.2205597.

[65] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012. DOI: 10.48550/ARXIV.1207.0580. arXiv: 1207.0580 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1207.0580.

[66] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[67] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021. DOI: 10.48550/ARXIV.2106.09685. arXiv: 2106.09685 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2106.09685.

[68] H. Hu and S. A. Zahorian, "Dimensionality reduction methods for hmm phonetic recognition," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, Institute of Electrical and Electronics Engineers (IEEE), 2010, pp. 4854–4857. DOI: 10.1109/icassp.2010.5495130. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2010.5495130.

[69] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *The Journal of Physiology*, vol. 148, no. 3, pp. 574–591, Oct. 1959, ISSN: 1469-7793. DOI: 10.1113/jphysiol.1959.sp006308. [Online]. Available: http://dx.doi.org/10.1113/jphysiol.1959.sp006308.

[70] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, ISSN: 1521-9615. DOI: 10.1109/mcse.2007.55. [Online]. Available: https://dx.doi.org/10.1109/MCSE.2007.55.

[71] A. Hupp *et al.*, *Python-magic*, version 0.4.27, Jun. 2022. [Online]. Available: https://pypi.org/project/python-magic/ (visited on Jul. 26, 2024), archived at https://web.archive.org/web/20240728124316/https://pypi.org/project/python-magic/ on Jul. 28, 2024.

[72] C. P. Igiri, O. U. Anyama, and A. I. Silas, “Effect of learning rate on artificial neural network in machine learning,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 4, no. 2, pp. 359–363, Feb. 2015, ISSN: 2278-0181. [Online]. Available: https://www.ijert.org/research/effect-of-learning-rate-on-artificial-neural-network-in-machine-learning-IJERTV4IS020460.pdf, archived at https://web.archive.org/web/20250129183008/https://www.ijert.org/research/effect-of-learning-rate-on-artificial-neural-network-in-machine-learning-IJERTV4IS020460.pdf on Jan. 29, 2025.

[74] A. Inusah, “Segmental phonology of dagbani dialects,” *International Journal Advances in Social Science and Humanities*, vol. 7, no. 1, pp. 15–30, 2019, The original URL is no longer available, the archived version was used., ISSN: 2347-7474. [Online]. Available: https://www.ijassh.com/index.php/IJASSH/article/download/304/310, archived at https://web.archive.org/web/20190801164627/https://www.ijassh.com/index.php/IJASSH/article/download/304/310 on Aug. 1, 2019.

[75] A.-R. Inusah, “Dialectal variation in dagbani phonology,” Master’s thesis, University of Ghana, Legon, Ghana, Jun. 2016. [Online]. Available: https://www.researchgate.net/publication/355007425_Phonological_Variation_in_Dagbani_Dialects, archived at https://archive.is/uOWKI on Jan. 29, 2025.

[76] A.-R. Inusah and E. S. Mahama, “The phonological structure of english borrowed words in dagbani,” *South African Journal of African Languages*, vol. 39, no. 3, pp. 281–290, Nov. 2019, ISSN: 2305-1159. DOI: 10.1080/02572117.2019.1672332. [Online]. Available: http://dx.doi.org/10.1080/02572117.2019.1672332.

[77] T. C. I. J. 1. 22, “ISO/IEC 14882:2020 Information technology — Programming languages — C++,” International Organization for Standardization (ISO), Geneva, Switzerland, Standard, Dec. 2020, p. 1853. [Online]. Available: https://www.iso.org/standard/79358.html, archived at https://web.archive.org/web/20240728120956/https://www.iso.org/standard/79358.html on Jul. 28, 2024.

[78] L. A. Janda, "From cognitive linguistics to cultural linguistics: How cognitive categories reflect culture," in *Ten Lectures on Cognitive Linguistics as an Empirical Science*. BRILL, Apr. 2018, pp. 1–31, ISBN: 9789004363502. DOI: 10.1163/9789004363519_002. [Online]. Available: http://dx.doi.org/10.1163/9789004363519_002.

[79] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, Jun. 1989, ISSN: 1537-274X. DOI: 10.1080/01621459.1989.10478785. [Online]. Available: http://dx.doi.org/10.1080/01621459.1989.10478785.

[80] B.-H. Juang and L. R. Rabiner, "Automatic speech recognition–a brief history of the technology development," *Georgia Institute of Technology, Atlanta Rutgers University and the University of California*, vol. 1, no. 67, p. 1, 2005. [Online]. Available: https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/354_LALI-ASRHistory-final-10-8.pdf, archived at https://web.archive.org/web/20240615232026/https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/354_LALI-ASRHistory-final-10-8.pdf on Jun. 15, 2024.

[81] J. Kahn. "Issue #916, Load pre-trained rasr model failed." (Jul. 24, 2022), [Online]. Available: https://github.com/flashlight/flashlight/issues/916 (visited on Aug. 1, 2024), archived at https://web.archive.org/web/20240801114943/https://github.com/flashlight/flashlight/issues/916 on Aug. 1, 2024.

[82] J. Kahn, V. Pratap, T. Likhomanenko, *et al.*, *Flashlight: Enabling innovation in tools for machine learning*, 2022. DOI: 10.48550/ARXIV.2201.12465. arXiv: 2201.12465 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2201.12465.

[83] Kinderrechte Afrika e. V. and Pan-African Organisation for Research and Protection of Violence on Women and Children, *Bihi hachinima mini biɛh'suŋ zaligu zaŋ tum tuma ghana zaŋ jɛndi dɔɣinda-bisa sabbu, bihi gbaai ti mini dooyili daŋ kundi ni bihi zubu*, Lahr, Germany, 2023. [Online]. Available: https://www.kinderrechte-afrika.org/app/download/34418538/2024_CR_Manual_Ghana_DAGBANI.pdf (visited on Jul. 25, 2024), archived at https://web.archive.org/web/20240718123640/https://www.kinderrechte-afrika.org/app/download/34418538/2024_CR_Manual_Ghana_DAGBANI.pdf on Jul. 18, 2024.

[84] T. Kluyver, B. Ragan-Kelley, F. Pérez, *et al.*, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds., IOS Press, 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87. [Online]. Available: https://dx.doi.org/10.3233/978-1-61499-649-1-87.

[85] J. Kohler, "Multi-lingual phoneme recognition exploiting acoustic-phonetic similarities of sounds," in *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*, ser. ICSLP-96, Institute of Electrical and Electronics Engineers (IEEE), Oct. 1996. DOI:

10.1109/icslp.1996.607240. [Online]. Available: http://dx.doi.org/10.1109/ICSLP.1996.607240.

[86] O. Kuchaiev, J. Li, H. Nguyen, *et al.*, *Nemo: A toolkit for building ai applications using neural modules*, 2019. DOI: 10.48550/ARXIV.1909.09577. arXiv: 1909.09577 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1909.09577.

[87] H. Kui, J. Pan, R. Zong, H. Yang, and W. Wang, "Heart sound classification based on log mel-frequency spectral coefficients features and convolutional neural networks," *Biomedical Signal Processing and Control*, vol. 69, p. 102 893, Aug. 2021, ISSN: 1746-8094. DOI: 10.1016/j.bspc.2021.102893. [Online]. Available: http://dx.doi.org/10.1016/j.bspc.2021.102893.

[88] E. C. Lallana and M. N. Uy, *The Information Age*. UNDP Asia Pacific Development Information Programme (UNDP-APDIP), May 2003, The original URL is no longer available, the archived version was used. [Online]. Available: http://www.apdip.net/publications/iespprimers/eprimer-infoage.pdf, archived at https://web.archive.org/web/20110520042925/http://www.apdip.net/publications/iespprimers/eprimer-infoage.pdf on May 20, 2011.

[89] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, ISSN: 1530-888X. DOI: 10.1162/neco.1989.1.4.541. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.4.541.

[90] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: http://dx.doi.org/10.1038/nature14539.

[91] G. von Leibniz, J. Child, and C. Gerhardt, *The Early Mathematical Manuscripts of Leibniz: Translated from the Latin Texts Published by Carl Immanuel Gerhardt with Critical and Historical Notes*. Open court publishing Company, 1920, ISBN: 9780486445960. [Online]. Available: https://openlibrary.org/works/OL1146833W/The_early_mathematical_manuscripts_of_Leibniz.

[92] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, Translation of [193]., Union of Soviet Socialist Republics, vol. 10, Feb. 1966, pp. 707–710. [Online]. Available: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf, archived at https://web.archive.org/web/20240728122644/https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf on Jul. 28, 2024.

[93] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, *Visualizing the loss landscape of neural nets*, 2017. DOI: 10.48550/ARXIV.1712.09913. arXiv: 1712.09913 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1712.09913.

[94] J. Li, V. Lavrukhin, B. Ginsburg, *et al.*, "Jasper: An end-to-end convolutional neural acoustic model," in *Interspeech 2019*, ser. interspeech$_2$019, International Speech Communication Association (ISCA), Sep. 2019. DOI: 10.21437/interspeech.2019-1819. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2019-1819.

[95] Y. Li, Y. Fu, H. Li, and S.-W. Zhang, "The improved training algorithm of back propagation neural network with self-adaptive learning rate," in *2009 International Conference on Computational Intelligence and Natural Computing*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009. DOI: 10.1109/cinc.2009.111. [Online]. Available: https://dx.doi.org/10.1109/CINC.2009.111.

[96] T. Likhomanenko, Q. Xu, V. Pratap, *et al.*, *Rethinking evaluation in asr: Are our models robust enough?* 2020. DOI: 10.48550/ARXIV.2010.11745. arXiv: 2010.11745 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2010.11745.

[97] T. Likhomanenko, Q. Xu, V. Pratap, *et al.* "Rasr release, Acoustic model." Models trained for the corresponding paper [96]. (Apr. 16, 2021), [Online]. Available: https://github.com/flashlight/wav2letter/tree/47949f479722000b49670295f9135efac1609a74/recipes/rasr#acoustic-model (visited on Aug. 1, 2024), archived at https://web.archive.org/web/20240801114051/https://github.com/flashlight/wav2letter/tree/47949f479722000b49670295f9135efac1609a74/recipes/rasr on Aug. 1, 2024.

[98] H. Liu, D. Tam, M. Muqeeth, *et al.*, *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*, 2022. DOI: 10.48550/ARXIV.2205.05638. arXiv: 2205.05638 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2205.05638.

[99] Y. Liu, Y. Zhang, Q. Li, *et al.*, *Hift: A hierarchical full parameter fine-tuning strategy*, 2024. DOI: 10.48550/ARXIV.2401.15207. arXiv: 2401.15207 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2401.15207.

[100] T. Lohrenz, Z. Li, and T. Fingscheidt, *Multi-encoder learning and stream fusion for transformer-based end-to-end automatic speech recognition*, 2021. DOI: 10.48550/ARXIV.2104.00120. arXiv: 2104.00120 [eess.AS]. [Online]. Available: https://arxiv.org/abs/2104.00120.

[101] G. Lours, M. Bowman, Docker, *et al.*, *Docker*, version 7.1.0, May 2024. [Online]. Available: https://pypi.org/project/docker/ (visited on Jul. 31, 2024), archived at https://web.archive.org/web/20240731080749/https://pypi.org/project/docker/ on Jul. 31, 2024.

[102] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu, *Full parameter fine-tuning for large language models with limited resources*, 2023. DOI: 10.48550/ARXIV.2306.09782. arXiv: 2306.09782 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2306.09782.

[103] J. Lyons, *Language and Linguistics* (Cambridge textbooks in linguistics). Cambridge University Press (CUP), 1981, ISBN: 9780521297752. [Online]. Available: https://www.cambridge.org/universitypress/subjects/languages-linguistics/english-language-and-linguistics-general-interest/language-and-linguistics.

[104] G. Macintosh and J. W. Gentry, "Decision making in personal selling: Testing the "k.i.s.s. principle"," *Psychology and Marketing*, vol. 16, no. 5, pp. 393–408, Aug. 1999, ISSN: 1520-6793. DOI: 10.1002/(sici)1520-6793(199908)16:5<393::aid-mar2>3.0.co;2-n. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1520-6793(199908)16:5%3C393::AID-MAR2%3E3.0.CO;2-N.

[105] F. Mak, A. Govender, and J. Badenhorst, "Exploring asr fine-tuning on limited domain-specific data for low-resource languages," *Journal of the Digital Humanities Association of Southern Africa (DHASA)*, vol. 5, no. 1, Feb. 2024, ISSN: 3006-6492. DOI: 10.55492/dhasa.v5i1.5024. [Online]. Available: http://dx.doi.org/10.55492/dhasa.v5i1.5024.

[106] R. Marcoux, L. Richard, and A. Wolff, "Estimation des populations francophones dans le monde en 2022: Sources et démarches méthodologiques," Observatoire démographique et statistique de l'espace francophone, Université Laval, Québec, Canada, Note de recherche de l'ODSEF, Mar. 2022, p. 177. [Online]. Available: https://observatoire.francophonie.org/wp-content/uploads/2022/03/odsef-lfdm-2022.pdf (visited on Jan. 17, 2025), archived at https://web.archive.org/web/20241218203955/https://observatoire.francophonie.org//wp-content/uploads/2022/03/odsef-lfdm-2022.pdf on Dec. 18, 2024.

[107] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, Mar. 2014, ISSN: 1075-3583. [Online]. Available: https://dl.acm.org/doi/10.5555/2600239.2600241; https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment, archived at https://web.archive.org/web/20240728121114/https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment on Jul. 28, 2024.

[108] J. Meyer, D. I. Adelani, E. Casanova, *et al.*, *Bibletts: A large, high-fidelity, multilingual, and uniquely african speech corpus*, 2022. DOI: 10.48550/ARXIV.2207.03546. arXiv: 2207.03546 [eess.AS]. [Online]. Available: https://arxiv.org/abs/2207.03546.

[110] G. Mokotoff. "Soundexing and genealogy." (1997), [Online]. Available: https://www.avotaynu.com/soundex.htm (visited on Jun. 23, 2024), archived at https://web.archive.org/web/20240624004543/https://www.avotaynu.com/soundex.htm on Jun. 24, 2024.

[111] C. Moseley, *Atlas of the World's Languages in Danger* (Memory of peoples Series). United Nations Educational, Scientific and Cultural Organization (UNESCO), 2010, ISBN: 9789231040962. [Online]. Available: https://openlibrary.org/books/OL27680143M/Atlas_Of_The_World%5C%27s_Languages_In_Danger.

[112] Mozilla Foundation. "Why common voice?" (Jun. 2017), [Online]. Available: https://commonvoice.mozilla.org/about (visited on Jun. 5, 2024), archived at https://archive.ph/S1CsN on Jul. 28, 2024.

[113] A. J. Naden *et al.*, *Dagbani Dictionary*, Web, A. J. Naden, Ed. Summer Institute of Linguistics International, Oct. 2014. [Online]. Available: https://www.webonary.org/dagbani/files/Dagbani_Dictionary_24_Oct_2014.pdf, archived at https://web.archive.org/web/20221014180337/https://www.webonary.org/dagbani/files/Dagbani_Dictionary_24_Oct_2014.pdf on Oct. 14, 2022.

[114] National Archives and Records Administration. "Soundex system." (May 2007), [Online]. Available: https://www.archives.gov/research/census/soundex (visited on Jun. 23, 2024), archived at https://web.archive.org/web/20240728122330/https://www.archives.gov/research/census/soundex on Jul. 28, 2024.

[115] H. Naveed, A. U. Khan, S. Qiu, *et al.*, *A comprehensive overview of large language models*, 2023. DOI: 10.48550/ARXIV.2307.06435. arXiv: 2307.06435 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.06435.

[116] O. Ndimele, *Language Endangerment: Globalisation and the Fate of Minority Languages in Nigeria.* M & J Grand Orbit Communications, 2015, ISBN: 9789785421514. [Online]. Available: https://openlibrary.org/books/OL48745766M/Language_Endangerment.

[117] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, Jan. 1965, ISSN: 1460-2067. DOI: 10.1093/comjnl/7.4.308. [Online]. Available: http://dx.doi.org/10.1093/comjnl/7.4.308.

[119] M. K. Odell, "The profit in records management," *Systems (New York)*, vol. 20, p. 20, 1956.

[120] OpenAI, "Gpt-4v(ision) system card," OpenAI, Tech. Rep., Sep. 2023. [Online]. Available: https://cdn.openai.com/papers/GPTV_System_Card.pdf (visited on Nov. 1, 2024), archived at https://web.archive.org/web/20241110212211/https://cdn.openai.com/papers/GPTV_System_Card.pdf on Nov. 10, 2024.

[121] OpenAI, J. Achiam, S. Adler, *et al.*, *Gpt-4 technical report*, 2023. DOI: 10.48550/ARXIV.2303.08774. arXiv: 2303.08774 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2303.08774.

[122] R. Osborne, "Cultures as languages and languages as cultures," in *Multilingualism in the Graeco-Roman Worlds.* Cambridge University Press (CUP), Sep. 2012, pp. 317–334. DOI: 10.1017/cbo9781139012775.013. [Online]. Available: http://dx.doi.org/10.1017/CBO9781139012775.013.

[123] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, ISSN: 1041-4347. DOI: 10.1109/tkde.2009.191. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2009.191.

[124] L. Philips, “Hanging on the metaphone,” *Computer Language*, vol. 7, no. 12, pp. 39–44, Dec. 1990. [Online]. Available: https://archive.org/details/sim_computer-language_1990-12_7_12/page/n41/mode/2up.

[125] L. Philips, “The double metaphone search algorithm,” *C/C++ Users Journal*, vol. 18, no. 6, pp. 38–43, Jun. 2000, The original URL is no longer available, the archived version was used., ISSN: 1075-2838. [Online]. Available: http://www.cuj.com/documents/s=8038/cuj0006philips/; https://dl.acm.org/doi/10.5555/349124.349132, archived at https://web.archive.org/web/20030701172517/http://www.cuj.com/documents/s=8038/cuj0006philips/ on Jul. 1, 2003.

[126] L. Philips. “Anthropomorphic; metaphone 3.” (Oct. 2009), [Online]. Available: https://www.amorphics.com/ (visited on Jul. 5, 2024), archived at https://web.archive.org/web/20240416232531/http://amorphics.com/ on Apr. 16, 2024.

[127] V. Pratap, A. Hannun, Q. Xu, *et al.*, “Wav2letter++: A fast open-source speech recognition system,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Institute of Electrical and Electronics Engineers (IEEE), May 2019. DOI: 10.1109/icassp.2019.8683535. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2019.8683535.

[128] V. Pratap, Q. Xu, A. Sriram, G. Synnaeve, and R. Collobert, “Mls: A large-scale multilingual dataset for speech research,” in *Proc. Interspeech 2020*, ser. interspeech$_2$020, International Speech Communication Association (ISCA), Oct. 2020, pp. 2757–2761. DOI: 10.21437/interspeech.2020-2826. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2020-2826.

[129] Public Relations Office of the Ministry of Finance, “2018 tiŋbihi bajɛti - 2018 bajɛti lahibali mini b□mma ni nyamma zaligu shɛli din be asama,” Accra, Ghana, Jul. 2018. [Online]. Available: https://mofep.gov.gh/sites/default/files/basic-page/2018-Citizens-Budget_Dagbaani.pdf (visited on Jul. 25, 2024), archived at https://web.archive.org/web/20240728124530/https://mofep.gov.gh/sites/default/files/basic-page/2018-Citizens-Budget_Dagbaani.pdf.

[130] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, *Robust speech recognition via large-scale weak supervision*, 2022. DOI: 10.48550/ARXIV.2212.04356. arXiv: 2212.04356 [eess.AS]. [Online]. Available: https://arxiv.org/abs/2212.04356.

[131] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” OpenAI, Tech. Rep., 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (visited on Nov. 1, 2024), archived at https://web.archive.org/web/20241102235715/https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf on Nov. 2, 2024.

[132] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," OpenAI, Tech. Rep., Feb. 2019, Originally released at https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf, but now unavailable, archived at https://web.archive.org/web/20190214181645/https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf on Feb. 14, 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (visited on Nov. 1, 2024), archived at https://web.archive.org/web/20241111161352/https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf on Nov. 11, 2024.

[133] P. Rajkovic and D. Jankovic, "Adaptation and application of daitch-mokotoff soundex algorithm on serbian names," in *XVII Conference on Applied Mathematics*, The original URL is no longer available, the archived version was used., Novi Sad, Serbia: Department of Mathematics and Informatics, Novi Sad, 2007, pp. 193–204. [Online]. Available: http://sites.dmi.pmf.uns.ac.rs/events/2006/prim2006/Papers/pdf/193-Rajkovic-Jankovic.pdf, archived at https://web.archive.org/web/20120307233007/http://sites.dmi.pmf.uns.ac.rs/events/2006/prim2006/Papers/pdf/193-Rajkovic-Jankovic.pdf on Mar. 7, 2012.

[134] K. Reitz *et al.*, *Requests*, version 2.32.2, May 2024. [Online]. Available: https://pypi.org/project/requests/ (visited on Jul. 31, 2024), archived at https://web.archive.org/web/20240731072653/https://pypi.org/project/requests/ on Jul. 31, 2024.

[135] A. Rodichevski. "Approximate string-matching algorithms, part 1." (), [Online]. Available: https://morfoedro.art/en/articolo/222 (visited on Jul. 19, 2024), archived at https://web.archive.org/web/20240728121643/https://morfoedro.art/en/articolo/222 on Jul. 28, 2024.

[136] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. DOI: 10.48550/ARXIV.1505.04597. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[137] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 0033-295X. DOI: 10.1037/h0042519. [Online]. Available: https://dx.doi.org/10.1037/h0042519.

[138] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://dx.doi.org/10.1038/323533a0.

[139] R. C. Russell, U.S. Patent 1 261 167, Apr. 2, 1918. [Online]. Available: https://patents.google.com/patent/US1261167.

[140] R. C. Russell, U.S. Patent 1 435 663, Nov. 14, 1922. [Online]. Available: https://patents.google.com/patent/US1435663.

[141] R. C. Russell, U.S. Patent RE15582E, Apr. 17, 1923. [Online]. Available: https://patents.google.com/patent/USRE15582E.

[142] F. Saa-Dittoh and Tingoli and Nyankpala Communities of the Northern Region of Ghana, *Recordings of spoken dagbani for the letters zero to ten and the words yes and no*, dag, Zenodo, 2024. DOI: 10.5281/ZENODO.13284005. [Online]. Available: http://dx.doi.org/10.5281/ZENODO.13284005.

[143] B. L. Samuelson and S. W. Freedman, "Language policy, multilingual education, and power in rwanda," *Language Policy*, vol. 9, no. 3, pp. 191–215, Jun. 2010, ISSN: 1573-1863. DOI: 10.1007/s10993-010-9170-7. [Online]. Available: http://dx.doi.org/10.1007/s10993-010-9170-7.

[145] S. Schmidgall, J. Achterberg, T. Miconi, *et al.*, *Brain-inspired learning in artificial neural networks: A review*, 2023. DOI: 10.48550/ARXIV.2305.11252. arXiv: 2305.11252 [cs.NE]. [Online]. Available: https://arxiv.org/abs/2305.11252.

[146] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[148] A. Schuringa, *Speechmatching: Phonetically match spoken words*, 2025. DOI: 10.5281/ZENODO.14765268. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.14765268.

[149] A. Shapson-Coe, M. Januszewski, D. R. Berger, *et al.*, "A petavoxel fragment of human cerebral cortex reconstructed at nanoscale resolution," *Science*, vol. 384, no. 6696, May 2024, ISSN: 1095-9203. DOI: 10.1126/science.adk4858. [Online]. Available: https://dx.doi.org/10.1126/science.adk4858.

[150] F. Sharifian, *Cultural Linguistics: Cultural conceptualisations and language* (Cognitive Linguistic Studies in Cultural Contexts). John Benjamins Publishing Company, Oct. 2017, ISBN: 9789027264992. DOI: 10.1075/clscc.8. [Online]. Available: http://dx.doi.org/10.1075/clscc.8; https://www.jbe-platform.com/content/books/9789027264992.

[151] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 04, no. 12, pp. 310–316, May 2020, ISSN: 2455-2143. DOI: 10.33564/ijeast.2020.v04i12.054. [Online]. Available: https://dx.doi.org/10.33564/IJEAST.2020.v04i12.054.

[152] Y. Shi, J. Wang, P. Ren, *et al.*, *Fine-tuning bert for automatic adme semantic labeling in fda drug labeling to enhance product-specific guidance assessment*, 2022. DOI: 10.48550/ARXIV.2207.12376. arXiv: 2207.12376 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2207.12376.

[153] R. Shwartz-Ziv and N. Tishby, *Opening the black box of deep neural networks via information*, 2017. DOI: 10.48550/ARXIV.1703.00810. arXiv: 1703.00810 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1703.00810.

[154] D. Spinellis, "The decay and failures of web references," *Communications of the ACM*, vol. 46, no. 1, pp. 71–77, Jan. 2003, ISSN: 1557-7317. DOI: 10.1145/602421.602422. [Online]. Available: http://dx.doi.org/10.1145/602421.602422.

[155] A. Srivastava, A. Rastogi, A. Rao, *et al.*, *Beyond the imitation game: Quantifying and extrapolating the capabilities of language models*, 2022. DOI: 10.48550/ARXIV.2206.04615. arXiv: 2206.04615 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2206.04615.

[156] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html, archived at https://web.archive.org/web/20240803101358/https://jmlr.org/papers/v15/srivastava14a.html on Aug. 3, 2024.

[157] Stack Exchange, Inc. "Stack overflow developer survey 2023." (May 2023), [Online]. Available: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies (visited on Jul. 28, 2024), archived at https://web.archive.org/web/20240726191529/https://survey.stackoverflow.co/2023/ on Jul. 28, 2024.

[158] G. V. Stan, "”small” language limited-vocabulary automatic speech recognition using machine learning," Master's thesis, Vrije Universiteit Amsterdam and University of Amsterdam, Amsterdam, The Netherlands, Aug. 2021. [Online]. Available: https://w4ra.org/wp-content/uploads/2021/08/ICT4D_voice_recognition_project-1.pdf, archived at https://web.archive.org/web/20240728121841/https://w4ra.org/wp-content/uploads/2021/08/ICT4D_voice_recognition_project-1.pdf on Jul. 28, 2024.

[159] G. V. Stan, A. Baart, F. Dittoh, H. Akkermans, and A. Bon, "A lightweight downscaled approach to automatic speech recognition for small indigenous languages," in *14th ACM Web Science Conference 2022*, ser. WebSci '22, Adapted from Master's Thesis [158]., Association for Computing Machinery (ACM), Jun. 2022. DOI: 10.1145/3501247.3539017. [Online]. Available: http://dx.doi.org/10.1145/3501247.3539017.

[160] Stanford Phonology Archive, "Dagbani sound inventory (spa)," in *PHOIBLE 2.0*, S. Moran and D. McCloy, Eds. Jena: Max Planck Institute for the Science of Human History, 2019. [Online]. Available: https://phoible.org/inventories/view/139.

[161] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of North American Fuzzy Information Processing*, ser. NAFIPS-96, Berkeley, California, United States of America: Institute of Electrical and Electronics Engineers (IEEE), Jun. 1996. DOI:

10.1109/nafips.1996.534789. [Online]. Available: http://dx.doi.org/10.1109/NAFIPS.1996.534789.

[162] R. Storn and K. Price, “Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997, ISSN: 0925-5001. DOI: 10.1023/a:1008202821328. [Online]. Available: http://dx.doi.org/10.1023/A:1008202821328.

[163] V. N. Sukhadia and S. Umesh, “Domain adaptation of low-resource target-domain models using well-trained asr conformer models,” in *2022 IEEE Spoken Language Technology Workshop (SLT)*, Institute of Electrical and Electronics Engineers (IEEE), Jan. 2023, pp. 295–301. DOI: 10.1109/slt54892.2023.10023233. [Online]. Available: http://dx.doi.org/10.1109/SLT54892.2023.10023233.

[164] R. Taft, *Name Search Techniques* (Special report (New York State Identification and Intelligence System) 1). Albany, New York, United States of America: Bureau of Systems Development, New York State Identification and Intelligence System, 1970. [Online]. Available: https://books.google.nl/books?id=CdU8AAAAMAAJ.

[165] M. Tanaka, *Weighted sigmoid gate unit for an activation function of deep neural network*, 2018. DOI: 10.48550/ARXIV.1810.01829. arXiv: 1810.01829 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1810.01829.

[166] Telegram FZ LLC and Telegram Messenger Inc. “Telegram bot api.” version 7.7. (Jul. 2024), [Online]. Available: https://core.telegram.org/bots/api (visited on Jul. 20, 2024), archived at https://web.archive.org/web/20240728124150/https://core.telegram.org/bots/api on Jul. 28, 2024.

[169] TIOBE Software BV. “Tiobe index for july 2024.” (Jul. 2024), [Online]. Available: https://www.tiobe.com/tiobe-index/ (visited on Jul. 25, 2024), archived at https://web.archive.org/web/20240728124749/https://www.tiobe.com/tiobe-index/ on Jul. 28, 2024.

[170] H. Touvron, T. Lavril, G. Izacard, *et al.*, *Llama: Open and efficient foundation language models*, 2023. DOI: 10.48550/ARXIV.2302.13971. arXiv: 2302.13971 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2302.13971.

[171] D. Tryon, “7. language endangerment and globalisation in the pacific,” in *Language Diversity in the Pacific*. Multilingual Matters, Dec. 2006, pp. 97–111. DOI: 10.21832/9781853598685-010. [Online]. Available: http://dx.doi.org/10.21832/9781853598685-010.

[172] J. Turk *et al.*, *Jellyfish*, version 1.0.4, May 2024. [Online]. Available: https://pypi.org/project/jellyfish/ (visited on Jul. 26, 2024), archived at https://web.archive.org/web/20240720175436/https://pypi.org/project/jellyfish/ on Jul. 28, 2024.

[173] UCLA Phonological Segment Inventory Database, “Dagbani sound inventory (upsid),” in *PHOIBLE 2.0*, S. Moran and D. McCloy, Eds. Jena: Max Planck Institute for the Science of Human History, 2019. [Online]. Available: https://phoible.org/inventories/view/299.

[174] United Nations: Conference on Trade and Development, *Information economy report 2009*. New York, NY: United Nations, Nov. 2009. [Online]. Available: https://unctad.org/system/files/official-document/ier2009_en.pdf, archived at https://web.archive.org/web/20220119031137/https://unctad.org/system/files/official-document/ier2009_en.pdf on Jan. 19, 2022.

[175] G. Van Rossum and F. Drake, *Python 3 Reference Manual: (Python Documentation Manual Part 2)* (Documentation for Python). CreateSpace Independent Publishing Platform, 2009, ISBN: 9781441412690. [Online]. Available: https://dl.acm.org/doi/book/10.5555/1593511.

[176] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017, pp. 6000–6010. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html, archived at https://web.archive.org/web/20240803093920/https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html on Aug. 3, 2024.

[177] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Feb. 2020, ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: https://dx.doi.org/10.1038/s41592-019-0686-2.

[179] L. Wang and S. Roberts, *The instabilities of large learning rate training: A loss landscape view*, 2023. DOI: 10.48550/ARXIV.2307.11948. arXiv: 2307.11948 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2307.11948.

[180] Y.-X. Wang, D. Ramanan, and M. Hebert, "Growing a brain: Fine-tuning by increasing model capacity," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Jul. 2017, pp. 3029–3038. DOI: 10.1109/cvpr.2017.323. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.323.

[181] J. Wei, Y. Tay, R. Bommasani, *et al.*, *Emergent abilities of large language models*, 2022. DOI: 10.48550/ARXIV.2206.07682. arXiv: 2206.07682 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2206.07682.

[182] P. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting* (Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control). Wiley, 1994, ISBN: 9780471598978. [Online]. Available: https://openlibrary.org/works/OL3929269W/The_roots_of_backpropagation.

[183] S. Wichmann, "How to distinguish languages and dialects," *Computational Linguistics*, vol. 45, no. 4, pp. 823–831, Jan. 2020, ISSN: 1530-9312. DOI: 10.1162/coli_a_00366. [Online]. Available: http://dx.doi.org/10.1162/coli_a_00366.

[184] Wikivoyage contributors. “Dagbani phrasebook – travel guide at wikivoyage.” (Jul. 2023), [Online]. Available: https://en.wikivoyage.org/w/index.php?title=Dagbani_phrasebook&oldid=4696558 (visited on Jul. 26, 2024), archived at https://web.archive.org/web/20240728124437/https://en.wikivoyage.org/w/index.php?title=Dagbani_phrasebook&oldid=4696558 on Jul. 28, 2024.

[185] W. E. Winkler, “String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.,” United States Bureau of the Census, Washington, DC, United States of America, Report, version ED325505, 1990, pp. 1–8. eprint: ERIC. [Online]. Available: https://eric.ed.gov/?id=ED325505, archived at https://web.archive.org/web/20240728123533/https://eric.ed.gov/?id=ED325505 on Jul. 28, 2024.

[186] C. Wong, “Cubic millimetre of brain mapped in spectacular detail,” *Nature*, May 2024, ISSN: 1476-4687. DOI: 10.1038/d41586-024-01387-9. [Online]. Available: https://dx.doi.org/10.1038/d41586-024-01387-9.

[187] J. Wu and C. Chan, “Isolated word recognition by neural network models with cross-correlation coefficients for speech dynamics,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1174–1185, 1993, ISSN: 0162-8828. DOI: 10.1109/34.244678. [Online]. Available: http://dx.doi.org/10.1109/34.244678.

[188] Z. Wu, A. Arora, Z. Wang, *et al.*, *Reft: Representation finetuning for language models*, 2024. DOI: 10.48550/ARXIV.2404.03592. arXiv: 2404.03592 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2404.03592.

[189] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, *Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment*, 2023. DOI: 10.48550/ARXIV.2312.12148. arXiv: 2312.12148 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2312.12148.

[190] M. Yamaguchi, D. Tay, and B. Blount, *Approaches to Language, Culture, and Cognition: The Intersection of Cognitive Linguistics and Linguistic Anthropology.* Palgrave Macmillan UK, 2014, ISBN: 9781137274823. DOI: 10.1057/9781137274823. [Online]. Available: http://dx.doi.org/10.1057/9781137274823; https://link.springer.com/book/10.1057/9781137274823.

[191] K. Zhou, C. Grover, M. Klein, and R. Tobin, “No more 404s: Predicting referenced link rot in scholarly articles for pro-active archiving,” in *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, ser. JCDL ’15, Association for Computing Machinery (ACM), Jun. 2015. DOI: 10.1145/2756406.2756940. [Online]. Available: http://dx.doi.org/10.1145/2756406.2756940.

[192] J. Zittrain, K. Albert, and L. Lessig, “Perma: Scoping and addressing the problem of link and reference rot in legal citations,” *Legal Information Management*, vol. 14, no. 2, pp. 88–99, Jun. 2014, ISSN: 1741-2021. DOI: 10.1017/s1472669614000255. [Online]. Available: http://dx.doi.org/10.1017/S1472669614000255.

[193] В. И. Левенштейн, “Двоичные коды с исправлением выпадений, вставок и замещений символов,” *Доклады Академии Наук СССР*, vol. 163, no. 4, pp. 845–848, Jan. 1965. [Online]. Available: `https://www.mathnet.ru/rus/dan31411`, archived at `https://web.archive.org/web/20240728122513/https://www.mathnet.ru/rus/dan31411` on Jul. 28, 2024.

Attachment A

Speech Matching documentation

Following on the next pages is the documentation of the ‘speechmatching’ software. This is compiled using the documentation also available in the repository, see appendix B.

Speech Matching

Release 1.0.0

Auke Schuringa <auke.schuringa@proton.me>

Feb 27, 2025

CONTENTS

CHAPTER

ONE

README

The speechmatching package allows for calculating similarity scores between short audio fragments containing speech. It then uses these similarity scores to find matches over multiple groups of recordings.

This is especially useful for rare languages from which not enough data exists to train a fully capable ASR model, or when the resources for training such a model are not available.

This package, along with the acoustic binary was created in combination with the thesis on Speech Matching [thesis] by Auke Schuringa for the University of Amsterdam and the Vrije Universiteit in the Netherlands.

Warning

This project does not currently run on machines with the ARM64 architecture due to the intel-mkl library not supporting this.

1.1 Setup

The repository is made up of two parts. The first is the speechmatching package itself, and the second is the acoustic binary located in the acoustic/ directory.

For details on setting up the acoustic package, please see its documentation. Because the acoustic package has specific dependencies and can be somewhat complex to build, it is recommended to run it using Docker.

The speechmatching Python package will by default look for the aukesch/speechmatching Docker image locally, and attempt to pull it if not available. For this, Docker needs to be installed.

Get Docker by installing it:

```
curl https://get.docker.com/ | sudo sh
```

Then, grant permissions to the current user so Docker can be run without sudo:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

The speechmatching package requires ffmpeg for converting files with an audio stream to a usable format. Using apt, this can be installed with:

```
sudo apt-get update
sudo apt-get install ffmpeg
```

Finally, install the `speechmatching` package from PyPI [package]:

```
pip install speechmatching
```

Or from the cloned repository:

```
pip install .
```

Optionally, create a virtual environment first:

```
sudo apt-get install python3-venv
python3 -m venv env
source env/bin/activate
pip install .
```

1.1.1 Development

During development, it may be useful to have a separate Docker image that is not named `aukesch/speechmatching`, which can then contain changes made during development. To support this, the `speechmatching` package will look for a Docker image named `speechmatching` before it looks for the Docker image `aukesch/speechmatching`. If the first docker image is available, the image under `aukesch/` will not be used.

1.2 Usage

There are two ways to use the `speechmatching` package - either locally or from within Docker.

1.2.1 Local

Running locally requires the (automatically pulled) Docker image containing the `acoustic` binary. This image sets the environment variable `ACOUSTIC_RUNNING_IN_DOCKER=1`, which is not available when running outside the container.

When running `speechmatching` and an audio file needs to be processed, the package detects that it is not running in Docker, then attempts to start a container from the `speechmatching` or `aukesch/speechmatching` Docker image (and attempt to pull it if it is not available locally). It will communicate with this container to process the audio file. When the program stops, the package tries to stop and remove the created container.

In other words, all that is needed is for Docker to be installed and set up correctly, and then the package can be run normally. These steps were explained in the previous section.

Warning

Upon stopping or closing the program, one may see a note about "cleaning up." If this process is aborted, the Docker container may continue running even after the Python process closes.

If that happens, the container can be manually identified and stopped:

```
docker ps
```

Copy the `CONTAINER ID` of the container in question, then stop and remove it with:

```
docker stop CONTAINER_ID
docker rm CONTAINER_ID
```

1.2.2 Docker

Alternatively, the speechmatching package can be run entirely inside Docker. In this scenario, the package detects that the ACOUSTIC_RUNNING_IN_DOCKER=1 environment variable is present and attempts to interact with the acoustic binary locally within the same container.

To do this, include a Dockerfile in the directory of the code with a structure like:

```
FROM aukesch/speechmatching
COPY . .
# more code...
CMD ["python3", "main.py"]
```

After which the Docker image can be built and run.

1.2.3 Example

Most of the functions that are needed to run this on a basic level are in the speechmatching.recording submodule.

In the following example, we assume the user has several audio files:

```
./audio/speech1.mp3
./audio/speech2.3gp
./audio/speech3.wav
./audio/house1.mp3
./audio/house2.mp4
./audio/tree1.3gp
./audio/tree2.mp4
./audio/tree3.mp3
./audio/tree4.wav
./audio/unknown.mp3
```

The last file in this list is named unknown.mp3, and it is believed this audio file belongs to one of the three spoken words speech, house, or tree, for which we have several samples available.

A single recording can be loaded and analyzed:

```
>>> from speechmatching.recording import Recording

>>> # load the recording
>>> speech1 = Recording('./audio/speech1.mp3')

>>> # get the transcript
>>> transcript = speech1.transcript

>>> # print the most likely text
>>> print(transcript.text)

>>> # print the top likely texts
>>> print(transcript.probable_texts())

>>> # and calculate similarity scores to other recordings
>>> speech2 = Recording('./audio/speech2.3gp')
>>> print(speech2.similarity(speech1))
```

(continues on next page)

(continued from previous page)

```
>>> # same as
>>> print(speech2.transcript.similarity(speech1.transcript))
```

These samples are sorted into groups:

```
>>> from speechmatching.recording import Group, Recording

>>> groups = [
...     Group(
...         identifier='speech',
...         recordings=[
...             Recording('./audio/speech1.mp3'),
...             Recording('./audio/speech2.3gp'),
...             Recording('./audio/speech3.wav')
...         ]
...     ),
...     Group(
...         identifier='house',
...         recordings=[
...             Recording('./audio/house1.mp3'),
...             Recording('./audio/house2.mp4')
...         ]
...     ),
...     Group(
...         identifier='tree',
...         recordings=[
...             Recording('./audio/tree1.3gp'),
...             Recording('./audio/tree2.mp4'),
...             Recording('./audio/tree3.mp3'),
...             Recording('./audio/tree4.wav')
...         ]
...     )
... ]

>>> # load the unknown recording and match it
>>> unknown = Recording('./audio/unknown.mp3')
>>> match = unknown.match(groups)
>>> print(match.identifier)
>>> # best matching group is printed here
```

There are more possibilities, with much greater control over the process by using various arguments available to the functions around normalization of strings, calculating similarity scores, and strategies for finding the best matching group.

Using the function `.match(...)` of the `Recording` instance uses the combination of normalization and matching algorithms that was found to work best in the thesis for which this software was written.

More examples can be found in the `examples/` directory.

1.3 Sample implementations

1.3.1 Basic

This basic example downloads a zip file [zipfile] with directories of recordings, unpacks this in the current directory, and peforms various operations with the sound recordings.

In "basic example 1", basic loading, transcript calculation, and similarity score calculation is done. In "basic example 2", a small number of three groups of recordings are created, with each having between 2 and 4 recordings. A final recording is then separately loaded and matched with the three groups to see which group matches the unknown recording best.

Finally, in "basic example 3", all downloaded recordings are loaded from their directories. One group is created per directory, and all recordings from that directory are added to that group.

A group is then taken to be tested with, and 20 random chosen recordings from this group are matched with all groups, after which the best matching group is chosen as matching group, or no decision can be made.

Please see the code in `example.py` for comments on how this works.

Note

When processing the audio files in the downloaded `recordings.zip` file, they are also processed by the `acoustic` binary to have information extracted from them. This may take some time.

Setup

The basic example of how to use the speechmatching package can be run through either Docker, or locally (manually).

Manual

It is advised to create a virtual environment using in a directory `env/`:

```
python3 -m venv env
```

and to load this with:

```
source env/bin/activate
```

In the virtual environments (which you have activated two commands ago), install the `speechmatching` package. From the two directories up with:

```
(cd ../..; pip install .)
```

and make sure that the `acoustic` binary is already built and available using instructions in the main `README` of this repository.

All requirements from `requirements.txt` need to be installed:

```
pip install -r requirements.txt
```

And one should then be all set to run the script with:

```
python3 main.py
```

Docker

Alternatively, Docker can be used, for which a Dockerfile is available.

Build the image using:

```
docker build . -t speechmatching-basic
```

After this completes successfully, run the image using:

```
docker run --rm -it speechmatching-basic
```

1.3.2 Telegram bot

This is a simple example of a Telegram bot. The bot can receive text and voice message to create groups of recordings, and can receive voice messages to match against these groups to find the best matching group.

The message expected by the bot depends on the state it is in. There are three states:

- Text state: the bot expects a text as representation of a word that is about to be registered. After receiving this word, the bot will move to the next state in this list.
- Recording registration state: the bot expects a voice message to register with the previously given textual representation of it. One or more recording may be given after each other. If instead of a recording, a text message is received, the bot will abandon the previous word if no recordings were registered to it, or keep it, and move to the "text state".
- Interpretation state: the bot expects a voice message to match against the previously registered recordings.

The commands to move between the states are as follows:

- `\new`: remove all recordings and switch to the text state.
- `\interpret`: move from the text or recording registration state to the interpretation state.

Setup

For the Telegram bot, one needs to get a token from Telegram for which the guide [telegramguide] can be followed. When this token is available, change the name of file `env.list.CHANGEME` to `env.list`, and replace dummy token `123456789:abcdefghijklmnopqrstuvwxyzABCDEFGH` by the token from Telegram.

When the token is in place, the bot can be run manually or using Docker.

Manual

Please see the manual setup in the README for the Basic example. The setup for the Telegram bot requires one extra step, setting the environment variable from `env.list`. Right after loading the Python virtual environment using `source env/bin/activate`, the Telegram bot token can be loaded into the environment with:

```
set -a
source env.list
set +a
```

After which the instructions from the setup for the Basic example are the same and can be followed further.

Docker

As for the Basic example, the Telegram bot can also be run using Docker.

Build the image for the Telegram bot under name `speechmatching-bot`:

```
docker build . -t speechmatching-bot
```

After this completes successfully, run the image using:

```
docker run --rm --env-file env.list -it speechmatching-bot
```

where the previously filled in `env.list` file is used.

1.3.3 Experiments

This example contains the IPython Notebook `final.ipynb`, which has the code for the experiments and plots for the thesis for which `speechmatching` was originally written. The Notebook may be not well organized and does not contain very clear notes and descriptions.

Setup

After setting up the `speechmatching` package and the `acoustic` binary, install the dependencies from `requirements.txt` and run:

```
jupyter notebook
```

after which the `final.ipynb` file can be opened, and experimented with.

CHAPTER

TWO

SPEECHMATCHING PACKAGE

2.1 Submodules

The `speechmatching` package contains various submodules. Of these submodules, usually only the submodule `speechmatching.recording` should be used. This submodule contains the functions for loading directories, and creating and handling `Group`, `Recording`, and `Transcript` instances.

2.1.1 speechmatching.audio module

Functions to prepare an audio or video file or raw transcript.

For processing, an audio file is expected of a WAV format with a single channel and a rate of 16kHz, which is what the function in this module creates.

`speechmatching.audio.`**`format_audio`**(*input_filepath*, *output_filepath=None*, *overwrite=True*)

Convert an audio or video file to a WAV file with 1 channel and 16 kHz.

After conversion, the new audio file is stored on disk for reuse next time. The filename of the created WAV file is the input filepath with `_processed.wav` appended to it.

Examples

```
>>> format_audio('example.mp4')
'example.mp4_processed.wav'
>>> format_audio('example2.mp3')
'example2.mp3_processed.wav'
```

Parameters

- **`input_filepath`** (*str*) – The audio or video file to convert.
- **`output_filepath`** (*str | None*) – A filepath ending with `.wav` to store the result in. If this is not given, the `input_filepath` has `_processed.wav` appended to it and is used instead.
- **`overwrite`** (*bool | None*) – Whether to overwrite the output file or not. Has default value `True`.

Returns

The filepath to which the created WAV file was written.

Raises

- **`ValueError`** – If the output filepath does not end in `.wav`.
- **`ffmpeg.Error`** – in case of an error by ffmpeg.
- **`FileNotFoundError`** – If the input file was not found, or the output file was not created.

Return type
str

2.1.2 speechmatching.config module

Module containing global configuration for Docker and local based models.

`class speechmatching.config.`**`Config`**

Bases: `object`

The configuration around interaction with the `acoustic` binary.

`MODEL_DOCKER_IMAGE`

The name(s) of the docker images with the `acoustic` binary to use. By default these are either:

- `speechmatching`: A locally build docker image to be used.
- **`aukesch/speechmatching`: The docker image on docker.com,**
  which can also be pulled.

`MODEL_DOCKER_BIN_LOCATION`

The file path within the Docker container where the `acoustic` binary can be found.

`CACHE_DIR_LOCAL`

The directory path on the local machine used to store the models for the `acoustic` binary if not running with Docker.

`CACHE_DIR_DOCKER`

The directory in the container used to store the models when using Docker for transcribing.

`CACHE_DIR_DOCKER = '/root/.cache/acoustic'`

`CACHE_DIR_LOCAL = '/home/uni/.cache/acoustic'`

`MODEL_DOCKER_BIN_LOCATION = '/opt/bin/acoustic'`

`MODEL_DOCKER_IMAGE = ['speechmatching', 'aukesch/speechmatching']`

2.1.3 speechmatching.match module

Module containing the functions for calculating matching scores.

The functions help with creating the algorithms for normalization or matching of strings, and combinations of the two.

`speechmatching.match.`**`ensure_algs_dict`**`(`*norm_algs*, *match_algs*`)`

Create a dictionary of normalization and matching algorithms.

Please see the documentation of the *`ensure_norm_algs_dict()`* function for how a representation of normalization algorithms is processed.

The matching of audio files is performed using the results from the normalization algorithms. To construct the dictionary that describes how these matches are done, the normalization algorithm needs to be available.

Examples

```
>>> norm_algs = {'metaphone': 0.25, 'nysiis': 0.7499999999999999}
>>> ensure_algs_dict(norm_algs, 'hamming')[1]
{'metaphone': {'hamming': 1.0}, 'nysiis': {'hamming': 1.0}}
>>> ensure_algs_dict(norm_algs, ['hamming', 'jaro'])[1]
{'metaphone': {'hamming': 0.5, 'jaro': 0.5}, 'nysiis': {'hamming': 0.5, 'jaro': 0.5}
↪}
>>> ensure_algs_dict(norm_algs, {'hamming': 0.1, 'jaro': 0.5})[1]
{'metaphone': {'hamming': 0.16666666666666669, 'jaro': 0.8333333333333334}, 'nysiis
↪': {'hamming': 0.16666666666666669, 'jaro': 0.8333333333333334}}
>>> ensure_algs_dict(norm_algs, {'metaphone': {'hamming': 0.1, 'jaro': 0.3}, 'nysiis
↪': {'levenshtein': 0.1, 'jaro': 0.2}})[1]
{'metaphone': {'hamming': 0.25, 'jaro': 0.7499999999999999}, 'nysiis': {'levenshtein
↪': 0.3333333333333333, 'jaro': 0.6666666666666666}}
```

Parameters

- **norm_algs** (*str | List[str] | Tuple[str, ...] | Dict[str, float]*) – The representation of normalization algorithms.
- **match_algs** (*str | List[str] | Tuple[str, ...] | Dict[str, float] | Dict[str, str | List[str] | Tuple[str, ...] | Dict[str, float]]*) – The representation of matching algorithms.

Returns

A tuple of the dictionary of normalization algorithms, and the dictionary of matching algorithms, where the dictionary of the matching algorithms has used information from the dictionary of normalization algorithms.

Return type

Tuple[*Dict*[str, float], *Dict*[str, float] | *Dict*[str, *Dict*[str, float]]]

`speechmatching.match.`**`ensure_norm_algs_dict`**(*norm_algs*)

Format the representation of a normalization algorithm dictionary.

The given input is a representation of a dictionary describing the normalization algorithms and their factors in a final result.

Examples

```
>>> ensure_norm_algs_dict('metaphone')
{'metaphone': 1.0}
>>> ensure_norm_algs_dict(['soundex', 'metaphone'])
{'soundex': 0.5, 'metaphone': 0.5}
>>> ensure_norm_algs_dict({'soundex': 0, 'metaphone': 0.1, 'nysiis': 0.3})
{'soundex': 0.0, 'metaphone': 0.25, 'nysiis': 0.75}
```

Parameters

norm_algs (*str | List[str] | Tuple[str, ...] | Dict[str, float]*) – The representation to make into a dictionary of normalization algorithms.

Returns

The dictionary of normalization algorithms and their factors with sum up to 1.

Raises
: `TypeError` – If `norm_algs` is not a string, list, tuple, or dict.

Return type
: *Dict*[str, float]

`speechmatching.match.`**`find_name`**(*name*, *return_f=True*)

Find the function in the the `jellyfish` [jellyfish] module based on the name.

This function is cached.

The name for the functions and their function name in `jellyfish` are mapped as follows:

```
{
    "soundex": "soundex",
    "nysiis": "nysiis",
    "metaphone": "metaphone",
    "hamming": "hamming_distance",
    "levenshtein": "levenshtein_distance",
    "damerau": "damerau_levenshtein_distance",
    "jaro": "jaro_similarity",
    "winkler": "jaro_winkler_similarity"
}
```

Parameters

- **`name`** (*`str`*) – The name to map against and find the `jellyfish` function for.
- **`return_f`** (*`bool`*) – Whether to return the found callable, or the string of the name of the function in `jellyfish`.

Returns
: The `jellyfish` function if `return_f` is `True`, or the name of the function.

Raises
: `ValueError` – If the given name of an algorithm is not found, if multiple algorithms are found for a certain name, or if the name of an algorithm could not be matched for a given name.

Return type
: str | *Callable*[[str, str], float] | *Callable*[[str], str]

`speechmatching.match.`**`match`**(*normed1*, *normed2*, *match_algs*)

Calculate a matching score between two strings.

The strings are expected to be normalized, however this is not strictly necessary. This function will not further normalize the given strings.

Parameters

- **`normed1`** (*`str`*) – The first string to compare.
- **`normed2`** (*`str`*) – The second string to compare to the first string `normed1`.
- **`match_algs`** (*`Tuple[Tuple[str, float]]`*) – The matching algorithms to use and the factor with which they should count in the final result. An example of a correct value is:

```
(
    ('hamming', 0.3),
    ('winkler', 0.5)
)
```

which notes that both the Hamming distance and the Jaro-Winkler similarities should be used, and the returned list of values will have the respective scores be multiplied by factors of respectively 0.3 and 0.5.

If a distance function is given, we convert the result into a similarity score using similarity_score=1-(distance_score/max_length)

Returns
A list of values of the result of match between the two given strings and each method with their factor.

Return type
List[float]

`speechmatching.match.`**`normalize`**`(`*s*, *name=None*`)`

Normalize a string with a certain algorithm.

This function is cached.

The names for the algorithms that are supported are:

- `soundex` for the Soundex algorithm
- `nysiis` for the NYSIIS algorithm
- `metaphone` for the Metaphone algorithm

Parameters

- **s** (*str*) – The string to normalize.
- **name** (*str | None*) – The name of the function to use. If not given, the string `s` is simply returned.

Returns
The normalized string `s` or the string itself when no name of the algorithm to use was given.

Return type
str

2.1.4 speechmatching.model module

Functions and classes for interacting with the `acoustic` binary.

The `acoustic` binary is responsible for transcribing audio data into text by using a neural network. Objects here allow for downloading these models, caching them locally or for Docker usage, and running the binary either locally or inside a Docker container.

`class speechmatching.model.`**`DockerModel`**`(`**args*, *pull_image=True*, ***kwargs*`)`

Bases: `Model`

A model that runs the `acoustic` binary in a Docker container.

This model communicates with a container of the Docker image under (normally) name `aukesch/speechmatching` or `speechmatching`. The model starts the container, interacts with it, and stops it when asked to.

This is the model that should be used when the `acoustic` binary and its libraries have not been compiled locally, but are only available through Docker, and the software using the `speechmatching` package does not run in Docker.

Parameters

- **pull_image** (*bool*) – Whether to pull the `aukesch/speechmatching` Docker image when a local image or this one is not found.
- ***args** – See the initialization arguments of *Model*.
- ****kwargs** – See the initialization arguments of *Model*.

_abc_impl = <_abc._abc_data object>

_alive_checker(*container*, *interval=2*)

Continuously check if the container is still alive.

If the container is not alive anymore, the logs of the container are printed to the `LOGS_FILEPATH` global variable filepath, and the program is exited with exit code 1.

Parameters

- **container** (*dict*) – The container to be checked.
- **interval** (*int*) – The interval in seconds with which should be checked, default is 2.

property _client: APIClient

Create the Docker API client for managing containers.

Returns
A `docker.APIClient` instance.

Raises
Exception – If the client could not be started, which is an indication that Docker is not installed. Or, if the Docker could not be used due to permission problems.

property _container: dict

Starts the container containing the `acoustic` binary.

The container is started with the command to start the `acoustic` binary and let it wait for input over standard input for audio files to transcribe.

Returns
Information about the container created by the client.

property _container_stdout: SocketIO

The socket for standard output from the running container.

Returns
The socket to the container to read information from.

property _container_stdout_lines: Iterator[str]

Iterator over lines from the standard output socket.

Yields
A line read from standard output from the container.

_copy_file(*src_filepath*, *dst_dirpath*)

Copy a file from the host to the container.

After copying, the file becomes available in the container and to the `acoustic` binary for further processing. However, the binary still needs to be sent the message that it should process the copied file.

If a file already exists at the location the local file will be copied to, this file will be overwritten.

Parameters

- **src_filepath** (*str*) – Path to the file on the host machine.

- **dst_dirpath** (*str*) – Path to copy the file from `src_filepath` to on the container.

Returns
The location in the container the file has been copied to.

Return type
str

property _image: str

Find or pull the image to use for the `acoustic` binary.

The names for the Docker images listed in *speechmatching.config.Config* are checked in order, and if one is found to exist, it is returned.

If no images are found to exist, the first image name containing `/` is pulled if `pull_image` was not set to `False` on initialization of this class. After pulling, the name of the pulled Docker image is returned.

Returns
An existing Docker image.

Raises
Exception – If no Docker image could be found and no Docker image should be pulled because `pull_image` was set to `False`, or if no Docker image name could be found that can be pulled.

property _socket: SocketIO

The socket for standard input to the running container.

Returns
The socket to the container to write information to.

_stop_client()

Stop the Docker communication client.

_stop_container()

Stop and remove the running container gracefully.

_stop_socket()

Close the sockets to the container.

read()

Read a line from the container over standard output.

Returns
A single line of output.

Return type
str

read_result(*filepath=None*)

Read output from the transcribing container.

The file written by the `acoustic` binary is copied into the host machine and then read, and is not being read directly from the container.

Parameters
filepath (*str | None*) – The filepath to read from. This is by default the default file the `acoustic` binary writes results to in the container, which is defined in the *speechmatching.config.Config* class.

Returns
The result from the model after processing an audio file.

Raises
`ValueError` – If the file copied from the container is somehow not found.

Return type
str

`start()`

Start the container and create a socket for writing to it.

`stop()`

Stop the sockets, container and Docker client.

`write(`*message*`)`

Write a message to the container over standard input.

Parameters
`message` (*`str`*) – The message to write.

`class speechmatching.model.`**`LocalModel`**`(`**args*, *acoustic_location=None*, ***kwargs*`)`

Bases: *`Model`*

A model that runs the `acoustic` binary locally on the host machine.

This model can be used when the `acoustic` binary and the required libraries have been compiled and are available locally. This is the case when this code is run in Docker with the compiled binary, or when effort has been put in to compile the binary locally outside of Docker.

When not running in Docker from the `aukesch/speechmatching` image, this model should likely not be used, and the *`DockerModel`* should be used instead.

Upon initialization of an instance, the required files for transcribing are downloaded if not downloaded yet.

Parameters

- **`acoustic_location`** (*`Optional[str]`*) – Optional. This is the location of the `acoustic` binary if this is different from one of the expected locations.
- **`*args`** – See the initialization arguments of *`Model`*.
- **`**kwargs`** – See the initialization arguments of *`Model`*.

`_abc_impl = <_abc._abc_data object>`

`property _process`

Create the process for transcribing.

If the location of the `acoustic` model was not given upon initialization of the instance, an attempt is made to locate the binary in one of the following locations:

- `/opt/bin/acoustic`
- `/usr/local/bin/acoustic`
- `acoustic` (in the local directory)

If the binary is found, the process is started and waits for input over standard input for files to transcribe.

Returns
A `subprocess.Popen` instance that communicates with the `acoustic` binary over standard input and output.

Raises
`Exception` – In case the binary is not found.

read()

Read a line from the process over standard output.

Returns
A single stripped line of output.

Return type
str

read_result(*filepath=None*)

Read output from the transcribing process.

Parameters
filepath (*str | None*) – The filepath to read from. This is by default the default file the `acoustic` binary writes results to, which is defined in the *speechmatching.config.Config* class.

Returns
A line of output from standard output.

Return type
str

start()

Start the model by creating the process to transcribe with.

stop()

Stop the model by terminating the running process.

write(*message*)

Write a message to the process over standard input.

Parameters
message (*str*) – The message to write.

class speechmatching.model.**Model**(*model_size='70M'*)

Bases: `ABC`

Abstract base class representing a model for transcribing audio.

Parameters
model_size (*str*) – The model size to use. Either `70M` or `300M`. See documentation for function *download_files()* for more information.

PROCESSES: List[*Model*] = []

_abc_impl = <_abc._abc_data object>

abstract read()

abstract read_result(*filepath*)

Parameters
filepath (*str*)

Return type
str

classmethod record_process(*instance*)

Add a running model to keep track of.

It is important to keep track of the instances of the model that are in use in order to stop these instances when the program closes.

Parameters
`instance` (`Model`) – The model to register.

abstract start()

abstract stop()

abstract write(*message*)

Parameters
`message` (*str*)

class speechmatching.model.**Transcriptor**(*model_location=None*)

Bases: `object`

The transcriptor for guiding the transcribing process.

Parameters
`model_location` (*Optional[str]*) – The model to use, can have either value `local`, or value `docker`. If not set, the value will be attempted to be determined by looking for the `ACOUSTIC_RUNNING_IN_DOCKER` environment variable set by the `aukesch/speechmatching` or `speechmatching` Docker images, if one of them is being used.

_transcribe(*input_filepath*)

Run the process described in function *Transcriptor.transcribe()* in this class.

Parameters
`input_filepath` (*str*)

Return type
str

stop()

Stop the model.

transcribe(*input_filepath*)

Transcribe an audio or video file using the running model.

This process can only run once at a time for each running instance of the model.

The given audio or video file is first processed into a WAV file of a single channel and 16000 hertz. If the Docker container is running, the file is copied into the running container. The file is then processed using the running `acoustic` binary, and the result is retrieved when the signal is given that processing has finished. This signal is given by writing on standard output from the process in the Docker container.

Parameters
`input_filepath` (*str*) – The audio or video file to process.

Returns
The raw output from the transcribing process in the form described in the documentation of class *speechmatching.recording.Transcript*.

Return type
str

`speechmatching.model.`**`download_files`**(*model_size='70M'*, *overwrite=False*)

Download the tokens URL [tokens] and a certain model URL to the cache dir.

The model is either one of 70 million [70model] or 300 million [300model] parameters.

These files are required as input to the `acoustic` binary to transcribe an audio file to characters and their probabilities.

Parameters

- **`model_size`** (*`str`*) – The model to download. This can be either `70M` or `300M`. The default value is `70M`.
- **`overwrite`** (*`bool`*) – Whether to overwrite an existing file or not.

Raises

`AssertionError` – If the HTTP response code is not 200 or if the hash digest of the downloaded file does not match the expected digest.

`speechmatching.model.`**`get_cache_filepath`**(*location*, *make_dir=True*, *docker=False*)

Get the filepath for a URL or file in the cache dir.

The default cache dir can be found in the *`speechmatching.config.Config`* class.

Parameters

- **`location`** (*`str`*) – The URL or file for which to construct the cache filepath.
- **`make_dir`** (*`bool`*) – Make the target directory if it does not exist. By default valued `True`.
- **`docker`** (*`bool`*) – Whether to use the cache dir in the Docker environment, or in the local environment. By default set to `False`, which means the local environment is used.

Returns

A string representing the full path to the cached file. For URLs, only the filename portion (after the last "/") is used in the cache path.

Return type

str

`speechmatching.model.`**`stop`**()

Stop all running models, both local and in Docker containers.

This method is registered to be automatically called when the signal is given to stop. It will use the processes stored in the global class variable in the *`Model`* class to determine which processes are running and need to be stopped.

This function may take several seconds to complete.

Warning

If this process is stopped before it finishes, and a Docker container was used for transcribing, the container may not be stopped correctly and it needs to be stopped manually.

2.1.5 speechmatching.recording module

The functions and classes around storing, processing and handling audio.

These are the main objects that are likely used when using package `speechmatching` in code. Please see the README for instruction on how to use this, and have a look at the examples.

`class speechmatching.recording.`**`Group`**(*identifier=None*, *labels=None*, *recordings=None*)

Bases: `object`

A group to hold *`Recording`*s.

Parameters

- **`identifier`** (*`Optional[str]`*) – The identifier of the group, which should be unique. It is not required.
- **`labels`** (*`Optional[Dict[str, str]]`*) – The labels of the group. This could for example be names or translations:

  ```
  {
      "English": "Three",
      "Dagbani": "Ata"
  }
  ```

  An empty dictionary is used if nothing is given.
- **`recordings`** (*`Optional[List[Recording]]`*) – The list *`Recording`*s to initially load. If nothing is given, the group is initialized empty.

`add`(*recording*)

Add a *`Recording`*.

Parameters

`recording` (`Recording`) – The *`Recording`* to add under the identifier set in the *`Recording`* itself.

Returns

`True` if the recording was successfully added, or `False` if the identifier from the given recording is already in this group.

Return type

bool

`group`(*size=None*, *identifier=None*)

Create a new *`Group`* from this *`Group`*.

Parameters

- **`size`** (*`int | None`*) – The number of random *`Recording`*s from this *`Group`* to use. This is by default set to the number of *`Recording`*s currently in this *`Group`*.
- **`identifier`** (*`str | None`*) – The identifier for the new *`Group`*. This is by default set to the identifier of the current *`Group`* with _sub appended to it.

Returns

The new *`Group`* of given `size` and with given `identifier`.

Return type

Group

property identifier: str | None

The identifier of the *Group*.

label(*k*)

Get a label from the group.

Parameters
k (*str*) – The key of the label.

Returns
The label under key k.

Return type
str | None

random()

Return a single random *Recording*.

Return type
Recording

recording(*identifier*)

Get the *Recording* under a certain identifier.

Parameters
identifier (*str*) – The identifier of the *Recording*.

Returns
The *Recording* under label identifier.

Return type
Recording

recordings()

Return a list of all *Recording*s in this *Group*.

Return type
List[Recording]

remove(*recording*)

Remove a recording from the *Group*.

Parameters
recording (Recording | *str*) – The identifier to the *Recording*, or the *Recording* itself that should be removed from this *Group*.

Raises
ValueError – If the recording does not exist in this group.

sample(*k=1*)

A number of randomly picked unique recordings from the *Group*.

Parameters
k (*int*) – The number of *Recording*s to get. The default value is 1.

Returns
The list of k randomly picked unique *Recording*s.

Return type
List[Recording]

set_label(*k*, *v*)

Set or replace a label.

Parameters

- **k** (*str*) – The key of the label.
- **v** (*str*) – The value of the label.

class speechmatching.recording.**Recording**(*filepath*, *transcriptor=None*, *preload=True*, *identifier=None*, *raw_output_filepath=None*)

Bases: `object`

Representation of an audio recording and various processing tasks.

Parameters

- **filepath** (*str*) – The path to the audio file.
- **transcriptor** (*Optional[Transcriptor]*) – The *speechmatching.model.Transcriptor* to use for the transcribing the recording. This is optional. If not given, a global one is created and used.
- **preload** (*bool*) – Whether the audio file should be processed upon initialization of this class. This may significantly increase the time required for initializing the class, but prevents this processing time from happening later. `True` by default.
- **identifier** (*Optional[str]*) – The identifier of the *Recording* instance. This identifier can be used in the *Group* the recording is in to allow for looking up the *Recording*. If not given, this will be set to `filepath`.
- **raw_output_filepath** (*Optional[str]*) – The filepath to which the raw output of the audio file given by the *speechmatching.model.Transcriptor* will be written. If not given, this is set to the audio `filepath` with `_raw_output.txt` appended to it.

GLOBAL_TRANSCRIPTOR = None

property filepath: str

The filepath of the audio file.

classmethod get_global_transcriptor()

Create and/or return the created global *speechmatching.model.Transcriptor*.

Returns
The global *speechmatching.model.Transcriptor*.

Return type
Transcriptor

property identifier: str

The identifier of the current *Recording* instance.

match(*groups*, *size=9223372036854775807*, *use_min_group_size=False*, *algs_norm=['soundex', 'nysiis', 'metaphone']*, *algs_match=['damerau', 'jaro', 'winkler']*, *return_indecision=True*, **args*, ***kwargs*)

Match the current *Recording* with multiple *Group*s.

Matching of one recording with multiple groups will allow for a single group, multiple groups, or no groups to be found to match the recording depending on the given arguments, see the descriptions of the parameters for more information on this.

The normalization and matching algorithms used here are by default set to the values that were found to work best according to the thesis for which this software was written.

Examples

```
>>> group1 = Group(identifier="Group1", recordings=[rec2])
>>> group2 = Group(identifier="Group2", recordings=[rec3, rec4])
>>> rec1.match([group1, group2], size=1)
Group(identifier='Group2', ...)
```

Parameters

- **groups** (*List[Group]*) – The groups to match this recording against.
- **size** (*int*) – The number of recordings from each group to use for matching. If a group has fewer recordings than this `size`, then argument `use_min_group_size` decides how many recordings of each group will be used.
- **use_min_group_size** (*bool*) – If set to `True`, then if one of the groups has fewer than `size` recordings, `size` is readjusted to the size of this group so an equal number of recordings from each group is used.
- **algs_norm** (*str | List[str] | Tuple[str, ...] | Dict[str, float]*) – See the documentation for the *`Transcript.similarity()`* function. Here, the default value is set to:

  ```
  ['soundex', 'nysiis', 'metaphone']
  ```

- **algs_match** (*str | List[str] | Tuple[str, ...] | Dict[str, float] | Dict[str, str | List[str] | Tuple[str, ...] | Dict[str, float]]*) – See the documentation for the *`Transcript.similarity()`* function. Here, the default value is set to:

  ```
  ['damerau', 'jaro', 'winkler']
  ```

- **return_indecision** (*bool*) – Whether to return `None` if there are multiple groups that match the recording with the same similarity. When this happens, a decision cannot be made for a single group. If this argument is set to `False`, all matched groups are returned instead.
- ***args** – Arguments for the *`similarity()`* function.
- ****kwargs** – Arguments for the *`similarity()`* function.

Returns

One of the given groups is returned if this group was found to be a best match to the recording. Multiple groups are returned if argument `return_indecision` is `False`, and multiple groups match the recording equally well. `None` is returned if multiple groups match equally well and `return_indecision` is set to *True*.

Return type

None | Group | *List*[Group]

`similarity`(*others*, **args*, ***kwargs*)

Calculate similarity scores between this and other *`Recording`*s.

Examples

```
>>> rec1 = Recording('audio1.wav')
>>> rec2 = Recording('audio2.wav')
>>> rec1.similarity(rec2, algs_norm='metaphone', algs_match='jaro')
0.85
>>> rec3 = Recording('audio3.wav')
>>> rec4 = Recording('audio4.wav')
>>> rec1.similarity([rec2, rec3, rec4], algs_norm='metaphone', algs_match='jaro
↪')
[0.85, 0.3, 0.91]
```

Parameters

- `others` (`Recording | List[Recording]`) – The other recording to calculate a similarity to, or a list of multiple other recordings.
- `*args` – Arguments for the *`Transcript.similarity()`* function of the *`Transcript`* instance for this current *`Recording`*.
- `**kwargs` – Arguments for the *`Transcript.similarity()`* function of the *`Transcript`* instance for this current *`Recording`*.

Returns

A single similarity score if one *`Recording`* was given, or a list of scores if multiple *`Recording`*s were given.

Return type

float | *List*[float]

property transcript: *`Transcript`*

Get the *`Transcript`* of this recording.

The recording is transcribed with the *`speechmatching.model.Transcriptor`*, which may be a global one, as detailed earlier in this class. If the file in which the raw output would be stored exists, it is loaded and no new transcript is created. If it does not exist, it is created after transcribing.

Returns

The transcript of the *`Recording`*.

class `speechmatching.recording.`**`Transcript`**(*raw*)

Bases: `object`

The handling of raw output for a transcript.

The raw transcript is in a form as shown the description of the `raw` variable below, and usually comes from the *`speechmatching.model.Transcriptor`* used in the *`Recording`* class.

This class performs various operations on the raw data, like extracting probable texts, and calculating matching scores with different transcripts.

Parameters

`raw` (*`str`*) – The string with multiple lines for the probabilities of the characters in the transcript. The first line in the string should note the characters, and each subsequent line represents the probabilities of these characters for the specific time step. This is formatted as:

```
char1 char2 char3 [...] charx
prob1 prob2 prob3 [...] probx
```

(continues on next page)

(continued from previous page)

```
[...] (time steps)
prob1 prob2 prob3 [...] probx
```

for which an example is:

```
a b c [...] z
0.1 0.3 0.05 [...] 0.01
[...]
0.01 0.8 0.1 [...] 0.001
```

property probabilities: ndarray[Any, dtype[_ScalarType_co]]

Create the list of lists of probabilities per time step.

The first line of the raw input is left out of the returned data, and only the probabilities are included from the raw data.

Returns
The matrix of probabilities with the time step on the y-axis and the character probability on the x-axis.

probable_texts(*min_probability=0.2*, *cache=True*, *normalize=True*, *or_best=False*, *or_best_char=False*)

Calculate probable texts from the raw data and their probabilities.

The raw data given to the instance has probabilities at each time step for given characters. With these probabilities, probable texts or strings of characters can be calculated. This can be done by taking all characters in each time step with a probability above the minimum probability and combining these into multiple strings.

It can happen that multiple characters in a time step have a sufficient probability, in which case multiple string are calculated. As an example, if there are three time steps, where time step one has 1 character of sufficient probability, time step two has 3, and time step three has 2 such characters, then a total of $1 \cdot 3 \cdot 2 = 6$ texts will be calculated.

It can happen that none of the characters in a time step have a sufficient probability, in which case what happens can be controlled using the arguments `or_best` and `or_best_char`.

If `or_best` and `or_best_char` are both set to `False`, it is possible for a single empty string to be returned in the dictionary.

Parameters

- **min_probability** (*float*) – The minimum probability for a token to be considered. Has a default value of 0.2.
- **cache** (*bool*) – Use the cache to store the result for the given set of arguments, or to retrieve the result from there and return it.
- **normalize** (*bool*) – Normalize the probabilities for the returned texts to 1.
- **or_best** (*bool*) – If no text could be extracted due to all character not exceeding the minimum probability, simply take the most likely text, which is the text return by the *text* property of the instance.
- **or_best_char** (*bool*) – if no character for a time step could be selected due to none of them exceeding the minimum probability, use the character from the time step with the highest probability.

Returns

The most likely texts according to the given arguments in the form of a dictionary:

```
{
    "TEXTA": 0.3,
    "TEXTB": 0.2,
    "TEXTC": 0.5
}
```

where the probabilities add up to 1 if `normalize` is set to `True`.

Return type

Dict[str, float]

similarity(*others*, *algs_norm=['soundex', 'nysiis', 'metaphone']*, *algs_match=['damerau', 'jaro', 'winkler']*, *choose_best=False*, *min_probability=None*, *or_best=True*, *or_best_char=False*)

Calculate similarity scores over one or more other *Transcript*s.

These similarity scores are calculated by first normalizing texts taken from these transcripts, and then calculating a similarity score for each of them using the matching algorithms, and factors for how much they should count in the final result.

Please also see the description in the function *probable_texts()* of this class to read how the used probable texts are calculated.

Parameters

- **others** (Transcript | *List*[Transcript]) – The other transcripts to calculate a similarity score with.
- **algs_norm** (*str* | *List[str]* | *Tuple[str, ...]* | *Dict[str, float]*) – The algorithms to normalize the texts with before calculating similarity scores. The normalization algorithms are a representation of what should happen, which may include factors when combining algorithms. Possible algorithms are
  - `soundex` for the Soundex algorithm [soundex]
  - `nysiis` for the NYSIIS algorithm [nysiis]
  - `metaphone` for the Metaphone algorithm [metaphone]

  and these can be combined in the following ways
  - one normalization algorithm can be given as:

    ```
    algs_norm = 'soundex'
    ```
  - multiple normalization algorithms can be given as:

    ```
    algs_norm = ['soundex', 'nysiis']
    ```
  - and factors can be given which do not need to add up to 1:

    ```
    algs_norm = {'soundex': 0.1, 'nysiis': 0.5}
    ```

  The best performing combination of normalization algorithms and matching algorithms, according to the report, is used by default. For the normalization algorithms this is:

  ```
  ['soundex', 'nysiis', 'metaphone']
  ```
- **algs_match** (*str* | *List[str]* | *Tuple[str, ...]* | *Dict[str, float]* | *Dict[str, str* | *List[str]* | *Tuple[str, ...]* | *Dict[str, float]]*)

– The algorithms to use for calculating the actual similarities between strings after normalization using the algorithms in `algs_norm`. The possible algorithms are

- `hamming` for the Hamming distance (as a similarity) [hamming]
- `levenshtein` for the Levenshtein distance [levenshtein]
- `damerau` for the Damerau-Levenshtein distance [damerau]
- `jaro` for the Jaro similarity [jaro]
- `winkler` for the Jaro-Winkler similarity [winkler]

and these can again be combined, in the way in which they can be similarly combined for the `algs_norm` argument as:

```
algs_match = 'hamming'
algs_match = ['hamming', 'jaro']
algs_match = {'hamming': 0.1, 'jaro': 0.5}
```

and additionally as a dictionary taking the normalization algorithms into account as:

```
algs_match = {
    'metaphone': {'hamming': 0.1, 'jaro': 0.3},
    'nysiis': {'levenshtein': 0.1, 'jaro': 0.2}
}
```

Here, again the best performing combinations are used by default, which for the matching algorithms is:

```
['damerau', 'jaro', 'winkler']
```

- **choose_best** (*bool*) – If multiple probable texts for a single recording are used in the calculation of the similarity score, this argument determines if the total similarity score should be averaged, or if simply the higest similarity scores should be chosen. The default is `False`, meaning that the final similarity score is averaged over all used probable texts.
- **min_probability** (*float | None*) – Used in the *`probable_texts()`* function, see the documentation there.
- **or_best** (*bool*) – Used in the *`probable_texts()`* function, see the documentation there.
- **or_best_char** (*bool*) – Used in the *`probable_texts()`* function, see the documentation there.

Returns
A single similarity score if only a single transcript was given, or a list of similarity scores with one score for each of the given transcripts in order.

Return type
float | *List*[float]

property text: str

Calculate the text with most probable character in each time step.

After calculation of the raw string, it is still processed with the *`speechmatching.utils.process_string_alnum()`* function before being returned. The characters are uppercased before returning.

Returns
The most probable text by selecting the most likely token at each time step.

property tokens: List[str]

The list of characters in the raw data in order.

Returns

The characters in the first line of the raw data in the same order they have in this first line.

`speechmatching.recording.`**`load_directory`**(*directory*, *transcriptor=None*, *identifier=None*, *labels=None*, *return_empty=False*, *verbose=True*)

Create a *Group* with all audio files in a directory.

Any audio files ending with `_processed.wav` in the directory will be ignored, as this is the filename ending used for audio files that have been converted into the format used by the `acoustic` binary for transcribing.

Parameters

- **directory** (*str*) – The directory from which load the audio files.
- **transcriptor** (Transcriptor *| None*) – The *speechmatching.model. Transcriptor* to use for transcribing. If not given, a global one will be used.
- **identifier** (*str | None*) – The identifier of the new *Group*. If not set, the name of the directory will be used instead.
- **labels** (*Dict[str, str] | None*) – A dictionary of the labels to set.
- **return_empty** (*bool*) – If set to `True`, an empty *Group* is created if the directory contains no audio files. Default is `False`.
- **verbose** (*bool*) – Whether to output the progress of loading files. Default is `True`.

Returns

The *Group* with loaded audio files, or `None` if no audio files were found, and `return_empty` is `False`.

Return type

Group | None

`speechmatching.recording.`**`load_directory_groups`**(*directory*, *transcriptor=None*, *return_empty=False*)

Load multiple *Group*s of *Recording*s from a directory.

The given `directory` should contain multiple directories which each containing the audio files for a *Group*. If a directory does not contain audio files, it may be ignored depending on the `return_empty` argument.

If a `<directory>_metadata.json` file is present in the directory next to the directory containing *Recording*s for the *Group*, the identifier and labels from this JSON file will be adopted for the new *Group*, else default values documented in the arguments of this function will be used.

The `<directory>_metadata.json` is of format JSON and can hold one of the keys `identifier` and `label`, with `label` being a dictionary with strings as keys and values. An example is:

```
{
    "identifier": "mygroup",
    "labels": {
        "en": "my group",
        "nl": "mijn groep"
    }
}
```

Parameters

- **directory** (*str*) – The directory from which directories of audio files are loaded into *Groups*.
- **transcriptor** (Transcriptor | *None*) – The *speechmatching.model.Transcriptor* to use for the processing the audio files. If not given, a global one will be used.
- **return_empty** (*bool*) – Whether to create an empty *Group* if a subdirectory does not contain any audio files.

Returns
A dictionary of the created *Groups* with their identifier as key, and the created *Group* as value.

Return type
Dict[str, Group]

speechmatching.recording.**sanitize_raw_transcript**(*transcript*, *no_dup=True*, *no_space=True*)

Sanitize a raw transcript by removing certain characters.

The character `|` is treated as a special character and will be transformed into a space if `no_space` is set to `False`. Optionally, duplicate characters can be deleted, which is set to `True` by default.

Examples

```
>>> sanitize_raw_transcript("he||llo||world!")
'heloworld'
>>> sanitize_raw_transcript("he||llo||world!", no_space=False)
'he lo world'
```

Parameters

- **transcript** (*str*) – The raw transcript.
- **no_dup** (*bool*) – Whether to remove duplicate characters or not. Default value is `True`.
- **no_space** (*bool*) – Whether to remove spaces or not. Default value is `True`.

Returns
The final sanitized string.

Return type
str

2.1.6 speechmatching.utils module

Various miscellaneous helper functions for the main code.

speechmatching.utils.**dicts_to_tuples**(*data*)

Convert a dictionary to `tuples`.

Conversion works iteratively, each key-value pair in the dictionary becomes a `(key, value) tuple`.

Examples

```
>>> d = {
...     'this': 'dictionary',
...     'is': {'nested': {'very': 'deep', 'extra': 'example'}},
...     'can': 'be',
...     'shallow': 'too'
... }
>>> dicts_to_tuples(d)
(('this', 'dictionary'),
 ('is', (('nested', (('very', 'deep'), ('extra', 'example'))),)),
 ('can', 'be'),
 ('shallow', 'too'))
```

Parameters
: `data` (*dict*) – The dictionary to convert.

Returns
: A tuple with nested tuples from the dictionary.

Return type
: tuple

`speechmatching.utils.`**`ensure_list`**`(`*data*`)`

Ensure the given data is of type `list`, or make a new list.

If the data is not yet a list, it is made into a list, and else it is returned as-is. Data of type `tuple` will be converted into a `list`, and else if the given `data` is not already a list, it will be returned as a list with `data` as single item in it.

Examples

```
>>> ensure_list([1, 2, 3])
[1, 2, 3]
>>> ensure_list((1, 2, 3))
[1, 2, 3]
>>> ensure_list("abc")
['abc']
```

Parameters
: `data` (*Any*) – The data to ensure is a list.

Returns
: The created or already given list.

Return type
: *List*[*Any*]

`speechmatching.utils.`**`print_docker_pull`**`(`*messages*`)`

Print the messages from a pull operation by `docker-py`.

The pull should be performed with `stream=True` and `decode=True`, for the messages to be yielded in the right order.

Parameters
: **messages** (*Iterator[dict]*)

`speechmatching.utils.`**`process_string_alnum`**(*s*)

Normalize a string to only alphanumerical characters and no duplicates.

Examples

```
>>> process_string_alnum("aaabbc!c???")
'abc'
```

Parameters
: **s** (*str*) – The string to process.

Returns
: The normalized string.

Return type
: str

`speechmatching.utils.`**`use_directory`**(*directory=None*)

Create a temporary directory or use an existing one.

Parameters
: **directory** (*str | None*) – The directory to use or create. If this directory already exists, it is yielded, else a temporary directory is created.

Yields
: The path to the directory being used (either the one provided or the newly created temporary directory).

Return type
: *Iterator*[str]

2.2 Changelog

Each release of the `speechmatching` package will have changes compared to the previous release documented here. The initial release was version 1.0.0.

2.2.1 1.0.0 – 2024-11-10

Initial release of the `speechmatching` package and the `acoustic` binary, see the changelog of the `acoustic` for more information on the latter.

CHAPTER

THREE

ACOUSTIC BINARY

The `acoustic` binary processes audio files as input using an acoustic model for a neural network. It maps outcomes to probabilities for tokens.

3.1 Installation

To compile the `acoustic` binary, several dependencies with specific versions are required. Due to the somewhat complex setup, it is recommended to use Docker.

3.1.1 Docker

The Docker image can be pulled under name `aukesch/speechmatching` with the `acoustic` binary already built in the image, with:

```
docker pull aukesch/speechmatching
```

Alternatively, to compile the `acoustic` binary and the `speechmatching` package into a Docker image, use the `Dockerfile` provided in the repositories root directory. Build the Docker image under the name `speechmatching` so the image can be easily found by the `speechmatching` package:

```
docker build . -t speechmatching
```

This process may take several minutes to complete.

The Docker image name `speechmatching` is used for manually built Docker images, which can be useful during for example testing, while the image `aukesch/speechmatching` is the one that is maintained by the maintainers of the `speechmatching` package. The `speechmatching` package will first look for the `speechmatching` Docker image, and else look for or pull the `aukesch/speechmatching` image.

3.1.2 Manual

The following versions of the dependencies were found to work well. It is possible other versions are also supported, but this was not thoroughly tested.

- ArrayFire [arrayfire] version `3.8.3` with options
  - `AF_BUILD_CPU=ON`
  - `AF_BUILD_CUDA=OFF`
  - `AF_BUILD_OPENCL=OFF`

  - AF_BUILD_EXAMPLES=OFF
  - AF_WITH_IMAGEIO=OFF
  - BUILD_TESTING=OFF
  - AF_BUILD_DOCS=OFF
- oneDNN [onednn] version 2.5.2 with options
  - DNNL_BUILD_EXAMPLES=OFF
- Gloo [gloo] branch 56b221c0a811491d2dc2a3254b468ad687bbdaab with options
  - USE_MPI=ON
- kenlm [kenlm] branch 9af679c38477b564c26917a5dcf52d2c86177fb9 with options
  - CMAKE_POSITION_INDEPENDENT_CODE=ON
- Flashlight [flashlight] version 0.3.1 with options
  - FL_BACKEND=CPU
  - FL_BUILD_ALL_APPS=OFF
  - FL_BUILD_PKG_TEXT=ON
  - FL_BUILD_PKG_RUNTIME=ON
  - FL_BUILD_PKG_SPEECH=ON
  - FL_BUILD_TESTS=OFF
  - FL_BUILD_EXAMPLES=OFF
  - FL_BUILD_APP_ASR=ON
  - FL_BUILD_APP_ASR_TOOLS=ON
- Library intel-mkl-64bit-2020.4-912 from Intel repos [intelrepos] as:

```
wget https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
↪PRODUCTS-2019.PUB
apt-key add GPG-PUB-KEY-INTEL-SW-PRODUCTS-2019.PUB
sh -c 'echo deb [trusted=yes] https://apt.repos.intel.com/mkl all main > /
↪etc/apt/sources.list.d/intel-mkl.list'
apt-get install -y --no-install-recommends intel-mkl-64bit-2020.4-912
```

All of the above should also be combined with option CMAKE_BUILD_TYPE=Release.

The acoustic binary itself can then be made and installed by:

```
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release \
         -DCMAKE_MODULE_PATH=/usr/local/share/flashlight/cmake
make
mkdir /opt/bin
cp acoustic /opt/bin
```

Please see the Dockerfile in the main directory of the repository for more detailed information.

3.2 Usage

The acoustic binary can be run with the following arguments:

--help Show the help message.

-i PATH, --input-filepath PATH Path to the WAV file to process, which should have a single channel of 16000 hertz.

--am PATH, --acoustic-model-filepath PATH Path to the acoustic model to use, this is usually either the 70 million parameter model [70model], or the 300 million parameter model [300model].

-o PATH, --output-filepath PATH Path to the file to write the probabilities to. If the options `-stdin` is used, this file is overwritten whenever a new WAV file is given for processing.

-t PATH, --tokens-filepath PATH Path to the file with tokens to use. Usually this is a default tokens file [tokens].

-stdin, --standard-input Listen to standard input for filenames of WAV files to process. If this option is used together with the `-i path`, `--input-filepath PATH` option, then the WAV file given over the command line is processed first, after which the program continues with and start listening on standard input.

When compiled into a Docker image under name `speechmatching`, the `speechmatching` package can be installed locally with `pip`, and it will look for and start a container for the `speechmatching` or `aukesch/speechmatching` Docker image. When the program is stopped, the package will attempt to stop and remove any created container.

See the documentation for the `speechmatching` package for further information.

3.3 Changelog

Each release of the `acoustic` binary will have changes compared to the previous release documented here. The initial release was version 1.0.0.

3.3.1 1.0.0 – 2024-11-10

Initial release of the `acoustic` binary.

BIBLIOGRAPHY

[thesis] Auke Schuringa, “Energy-efficient phonetic matching for spoken words of rare indigenous languages in low-resource environments,” Master’s thesis, Vrije Universiteit Amsterdam and University of Amsterdam, Amsterdam, The Netherlands, Jan. 2025.

[package] `speechmatching` on PyPI https://pypi.org/project/speechmatching/

[zipfile] https://zenodo.org/api/records/13284005/files/recordings.zip

[telegramguide] https://core.telegram.org/bots/features#botfather

[jellyfish] Repository of jellyfish on GitHub https://github.com/jamesturk/jellyfish

[soundex] Wikipedia page on the Soundex algorithm https://en.wikipedia.org/wiki/Soundex

[nysiis] Wikipedia page on the NYSISS algorithm https://en.wikipedia.org/wiki/New_York_State_Identification_and_Intelligence_System

[metaphone] Wikipedia page on the Metaphone algorithm https://en.wikipedia.org/wiki/Metaphone

[hamming] Wikipedia page on the Hamming distance https://en.wikipedia.org/wiki/Hamming_distance

[levenshtein] Wikipedia page on the Levenshtein distance https://en.wikipedia.org/wiki/Levenshtein_distance

[damerau] Wikipedia page on the Damerau-Levenshtein distance https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

[jaro] Wikipedia page on the Jaro similarity https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance#Jaro_similarity

[winkler] Wikipedia page on the Jaro-Winkler similarity https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance#Jaro%E2%80%93Winkler_similarity

[arrayfire] Repository of ArrayFire on Github https://github.com/arrayfire/arrayfire

[onednn] Repository of oneDNN on GitHub https://github.com/oneapi-src/onednn

[gloo] Repository of Gloo on GitHub https://github.com/facebookincubator/gloo

[kenlm] Repository of kenlm on GitHub https://github.com/kpu/kenlm

[flashlight] Repository of Flashlight on GitHub https://github.com/flashlight/flashlight

[intelrepos] Intel repository https://apt.repos.intel.com/mkl

[70model] Acoustic model with 70 million parameters https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_70Mparams.bin (archived at https://web.archive.org/web/20230603172217id_/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_

[300model] Acoustic model with 300 million parameters https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_stride3_letters_300Mparams.bin (archived at https://web.archive.org/web/20220729100332id_/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/am_transformer_ctc_

[tokens] Default tokens for both acoustic models https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/tokens.txt (archived at https://web.archive.org/web/20220729100540id_/https://dl.fbaipublicfiles.com/wav2letter/rasr/tutorial/tokens.tx

PYTHON MODULE INDEX

INDEX