Final Year Project

SimilarSpeak

Fíonán Byrne

Student ID: 16360121

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Julie Berndsen - julie.berndsen@ucd.ie

UCD School of Computer Science
University College Dublin

May 21, 2020

Table of Contents

Abstract

Many of today's current technologies, including those within the domain of speech recognition, rely on a means of finding similar sounding words. This project outputs a ranked list of similar sounding words and words sequences for any valid word defined by an American English dictionary. This input word is first translated into its phonetic representation by means of an online pronunciation dictionary and then it is subsequently broken up into its syllable components as specified by a provided syllabifier. The python program then searches for and store all possible defined and undefined words whose *phonetic distance* from the target word falls within a maximum distance threshold. The program will then provide the user with two lists containing all the resulting defined and nonsense words respectively in ascending order of phonetic distance. Additional functionality will allow the user to judge the degree to which each list corresponds with their own intuition about word similarity.

Chapter 1: Project Specification

1.1 Core Tasks

The core goal of this project is to create a system, called SimilarSpeak, which can accurately identify similar sounding words and word sequences for any given input word. By default, this output list will display the words and phrases in order of most to least similar according to some predetermined distance metric. The user will also be able to alter the maximum distance value of the displayed ranked list, so that fewer or more results can be shown.

1.2 Advanced Tasks

The project also requires three advanced ancillary tasks to be completed, the first of which is to extend the system to be able to find and return similarly-sounding nonsense words that still adhere to the *phonotactic constraints* of American English (i.e. generated words are pronounceable by a native speaker). An additional requirement is to provide a means of selecting whether the beginning of syllables, the ending of syllables, everything but the last syllable ending or all parts of the syllables in the input word are changed. As a note of clarification, the beginning of a syllable is considered to be any consonant sound or sounds before the singular vowel sound, while the ending of a syllable includes both the vowel sound and any consonant sounds that succeed it. The final specified task is to provide the user with some interface for evaluating the accuracy of the entire output list, so that each rating can be stored in a database along with the metric used for quantifying sound distances and the associated input word.

Chapter 2: **Introduction**

My vision for this project is a robust system that can find similarly sounding words and phrases, with a particular emphasis on *adaptability*. Calculating the phonetic distance between whole words and word phrases will require storing the distances between phonemes[1] in a two-dimensional matrix. My plan is to design a system with a back end that can easily *adapt* to new matrices derived from alternative phoneme distance metrics, so that differences between the ranked lists produced by difference metrics can be noted for future analysis. I also intend on developing a simple and intuitive user interface for taking in the input word, displaying the ranked list in an ascending order of phonetic distance and receiving each user's rating for the accuracy of each particular list.

But this all begs the question - why create a system that can find similar sounding words? How do I envision SimilarSpeak being used once all of the promised functionalities have been implemented? First of all, the adaptable nature of my program will certainly be of benefit to the domain of speech recognition. While other implementations of applications that can find similar sounding words are forced to operated within the confines of a single English dialect, SimilarSpeak will be usable as a means for reducing the word error rates of more dialects for the latest state-of-the-art **automatic speech recognition** (ASR) systems. Essentially, the system could be used to find the most accurate phoneme similarity metric for many possible native and non-native dialects of English.

In fact Emma O'Neil, a Computer Science PhD student in University College Dublin, has already informed me that she plans on using the finished SimilarSpeak program for one of her future studies to generate candidate words/word phrases for a list of target words for several experiments (i.e. the two words with the lowest calculated phonetic distance to each target word will be generated). These target words and candidate words will be presented to native speakers to see which of the two options they deem more similar to the target. The results of this experiment will show how native speakers' intuition about word similarity compares to the judgement of a phonological feature-based similarity metric[2]. A second experiment will involve investigating how the similarity judgements of native speakers with different accents compares with the similarity judgements of non-native speakers. O'Neill has told me that she hopes that the data gathered from this study will be used to improve the accuracy of ASR systems in deciphering speech by non-native speakers.

In addition to helping ASR systems learn a wider spectrum of different dialects, I envision my finished program assisting primarily non-native English speakers find the exact spelling of words that they can only recognize by sound alone. As I can attest from my own experiences learning German and Irish, listening to the sounds, rhythms and intonations of a words in a foreign language greatly benefits one's speaking, reading and writing comprehension skills. As explained by Renukadevi [2], *"listening plays a vital role, as it helps the language learner to acquire pronunciation, word stress, vocabulary, and syntax and the comprehension of messages conveyed"*. English in particular is

[1] *Phonemes* are the smallest, perceptually distinct units of sound that distinguish one word from another when spoken aloud [1]. For example the same phoneme /p/ can be heard in '**p**in' and 's**p**in'. However it's worth noting that in this example, the first /p/ is aspirated (denoted as $[p^h]$), but the second /p/ is not ($[p]$), meaning they are not identical sounds, but they are still considered to belong to the same phoneme, as substituting one for the other would not change the perceived meaning of the word. These two variations are said to be "allophones" of the phoneme /p/. Finally, a *phone* is the distinct unit of speech sound, regardless of whether the exact sound is critical to a listeners interpretation of the word.

[2] This will mean that in generating each pair of candidate words, a feature-based phoneme distance matrix will have to be used. I go into more detail about how the data for feature-based distance matrices are generated in Chapter 3 (Related Work and Ideas).

notorious for its disparities between the individual spellings and actual pronunciations of its words. However with the aid of the SimilarSpeak program, a non-native English speaker could easily find similar sounding words to "cash", if they had trouble remembering the spelling of "cache". Or if they were *languishing* in despair from trying to find the spelling of "language", a quick search of the very word "languish" would hopefully help them.

Finally along with language learning, I intend for my SimilarSpeak system to be able to assist lyric writers in finding the perfect words for their songs, raps or poems. Many other rhyme-finding applications (e.g. *Rhyme Time* on the Google Play Store) are functionally sub-optimal as they only attempt to find words that share the exact same vowel and final consonant sound in the last syllable. I plan for my program to be far more flexible with regard to finding rhyming words, as in addition to finding possible words that share the exact same final syllable ending, it will be capable of finding words whose ending is similarly sounding to the target word. Essentially it will be capable of discovering more subtle nuanced rhymes by, for instance, only finding words whose final syllable has the same vowel sound or has a similarly sounding vowel sound to that of the final syllable in the target word.[3]

[3] To put it more coarsely, my system will be capable of finding *assonant* words. Assonance takes place in a song or a poem when at least two words shared the same or a similarly sounding final vowel sound. For example, in *Do Not Go Gentle into the Good Night* by Dylan Thomas, the poet makes use of assonance with the "ah" sound of "rave" and "day" in the following line: *"Old age should burn and r**a**ve at close of d**a**y"*.

Chapter 3: Related Work and Ideas

Much research has been conducted within the field of psycholinguistics in identifying the optimal method for accurately quantifying the similarity between the words of a language. Constructing and then evaluating the performance of a system that can calculate the perceived similarity between phonetic words will naturally rely upon some distancing metric that can observe any two phonemes and then assign a number to their calculated similarity. In the past several decades linguists have identified two practical approaches to comparing phonemes: 1) theoretical based approaches based on the linguistic descriptions of the relevant phonemes and 2) empirically derived "bottom up" approaches that are based on data pertaining to the level of confusion (i.e. confusability) between phonemes in human trials. In this section, I will discuss the merits and drawbacks of each approach, as discussed by T. M. Bailey & Hahn [3], . Finally, I will describe the functionality of online programs, such as RhymeZone, which can produce a list of similar sounding English words and word phrases for a valid input.

3.1 Comparing Phoneme Similarity Metrics

In a paper for the Journal of Memory and language, Bailey & Hahn investigate the differences between feature and confusability based measures for phoneme similarity. The authors vouch for the straightforwardness of the former approach, as they argue that the level of phonetic similarity of two sounds can be easily derived from the degree of feature overlap between phonemes. The major class features of phonemes are traditionally place of articulation, manner of articulation and voicing, as they provide clear and convenient descriptions of phoneme contrasts, they often feature in feature geometry theories of phonological, and they are regularly used in similar psycholinguisic studies. However one notable concern is that these are all articulatory features, and so they have higher relevance in short-term memory and speech production tasks rather than word recognition tasks, such as the SimilarSpeak system I will be developing. In any case, a metric of simple feature similarity metric, S_{PMV}, is used in Bailey and Hahn's two experiments in which a variety of similarity metric are tested for their accuracy in predicting participant responses to word similarity.

Bailey and Hahn also refer to an alternative feature based approached proposed by Frisch in 1997, which is based on "mono-valent" features which are either present or absent in a phoneme. His similarity metric $S_{Frisch}$is based on a comparison of the *natural classes* to which two phonemes belong, where natural classes are the set of phonemes in a language that share certain distinctive features. Although Frisch concluded that his metric was better than S_{PMV} in predicting rates of phonological speech errors and predicting the strength of phonotactic constraints between adjacent and non-adjacent phonemes, S_{Frisch} could be more relevant to short-term memory and speech production tasks rather than to word recognition tasks.

With regard to phoneme confusability approaches, Bailey and Hahn consider metrics formed from confusion data from three different types of language processing tasks; short-term memory, speech production, and syllable perception. S_{wick} is derived from Wicklegren's table of consonant confusions, obtained from a controlled short-term memory task, in which the number of times one consonant in a consonant vowel syllable would be mistaken for another was counted. From these trials, the chance probability of one phoneme y, being substituted with (or rather confused with) another phoneme x, can be calculated as follows:

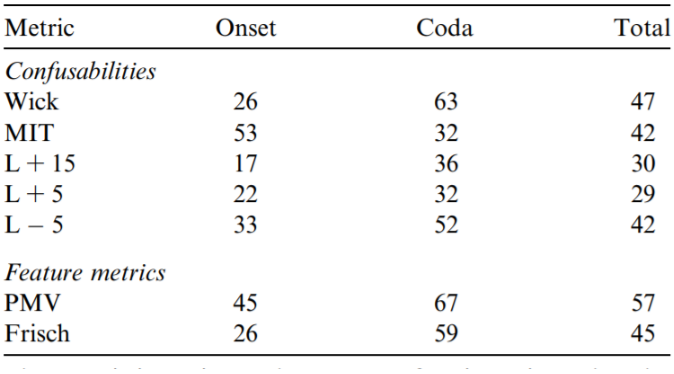
$$p(x \rightarrow y) = p(target = x) \cdot p(intrusion = y) for x \neq y \tag{3.1}$$

From this equation, the number of expected confusions for a particular phoneme pair can be calculated as follows, where N is the number of total confusions in the corpus:

$$Expected(x \rightarrow y) = \frac{p(x \rightarrow y)}{\sum_{i \neq j} p(x_i \rightarrow y_i)} \cdot N \tag{3.2}$$

The ratio between Observed(x→y) and Expected(x→y) results in a similarity score Confusion(x→y), which ranges between 0 (no confusion similarity) to positive infinity (high confusion similarity). The similarity score S_{wick} is then simply the average of Confusion(x→y) and Confusion(y→x). A second confusion metric S_{MIT} is obtained from speech error data from the MIT corpus of single consonant intrusion errors in spontaneous speech. As this dataset was constructed from substitution errors(*"mell"→ "well"*) and substitution errors (*"made possible"→ "pade mossible"*), S_{MIT} acts as an appropriate measure of phoneme confusability for speech production tasks. The similarity score is computed in a similar fashion to S_{Wick} by comparing Observed(x→y) to Expected(x→y) for any particular phoneme pair and then finding the average of Confusion(x→y) and Confusion(y→x). Finally as a measure of perceptual (i.e. aural) phoneme confusability, Bailey and Hahn use Luce's tables for phoneme errors under signal-to-noise ratios of +15 dB, +5 dB, and -5 dB. Again these metrics of similarity are computed by comparing the number of observed errors with the number of expected errors.

In Bailey and Hahn first experiment, participants listened to 20 triplets of consonant-vowel-consonant (CVC) syllables that differed in the onset (consonant before the vowel) and 20 triplets of CVC syllables that differed in the coda (consonant after the vowel). Each target word (T) began and ended with the same consonant. The first syllable choice (A) was generated from the target word by swapping out one consonant with another that differed by *one* articulation feature (i.e. either different manner, place or voice). The second choice (B) was generated by bringing in a consonant that differed from the original by *two* class features. So for example, when /p∇sp/ was presented as a target word, the two choice syllables A and B were /b∇asp/ and /g∇asp/ respectively. Each of the seven similarity metrics returned two values Similarity(TA) and Similarity(TB) for each triplet in the experiment. The table below shows the the fraction of response variance that coincided with the predictions made by each of the similarity metrics, for both the onset and coda positions as well as a combination of the two.



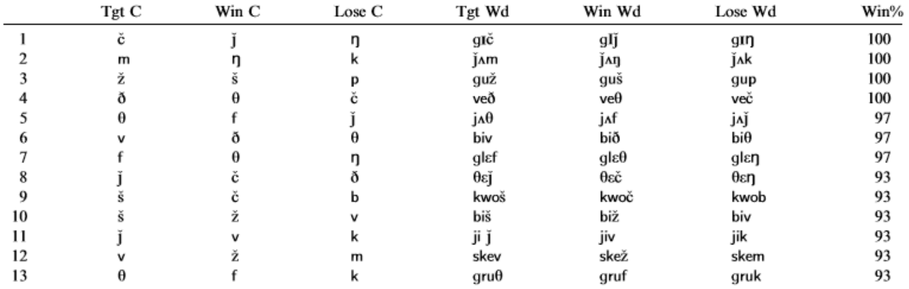
Metric	Onset	Coda	Total
Confusabilities			
Wick	26	63	47
MIT	53	32	42
L + 15	17	36	30
L + 5	22	32	29
L − 5	33	52	42
Feature metrics			
PMV	45	67	57
Frisch	26	59	45

Figure 3.1: *Results from Bailey and Hahn's first experiment which shows the performance of each similarity metric in predicting participant similarity judgements for a series of CVC syllables.* [3]

S_{PMV} proved to be the most accurate measure in this first experiment, which I found particularly surprising considering that Frisch's natural class metric was more successful in prediction phoneme errors in speech and it considers far more sub-features in its evaluation of phoneme similarity.

It is perhaps possible that this feature metric is a victim of *The Curse of Dimensionality* [4], a phenomenon that occurs in classification where too many dimensions can make it difficult to cluster (i.e. quantify the similarity between) observations (input phonemes). Additionally, Frisch's metric was made for identifying errors in speech production tasks, and so it relies on identifying the articulatory based features of phonemes. This suggests that S_{Frisch} is perhaps not well suited to the two word recognition experiments conducted by Bailey and Hahn.

In the paper's second experiment, the same procedures are repeated except this time the substitutions are made only in the final coda position. In contrast to the first experiment, Bailey and Hahn testes a significantly larger range of possible phoneme comparisons by selecting a representative sample of 180 combinations from the total possible 1680 consonant combinations. Additionally seven *dissimilarity* metrics are derived from each of the aforementioned similarity measures. For the feature metrics, D_{PMV} = 3 - S_{PMV} and D_{Frisch} = 1 - S_{Frisch}. For the confusability measures, each dissimilarity metric was calculated by taking the multiplicative inverse of the respective similarity metric. So for example, $D_{Wick} = \frac{1}{S_{Wick}}$.



	Tgt C	Win C	Lose C	Tgt Wd	Win Wd	Lose Wd	Win%
1	č	ǰ	ŋ	gɪč	gɪǰ	gɪŋ	100
2	m	ŋ	k	ǰʌm	ǰʌŋ	ǰʌk	100
3	ž	š	p	guž	guš	gup	100
4	ð	θ	č	veð	veθ	več	100
5	θ	f	ǰ	jʌθ	jʌf	jʌǰ	97
6	v	ð	θ	biv	bið	biθ	97
7	f	θ	ŋ	glɛf	glɛθ	glɛŋ	97
8	ǰ	č	ð	θɛǰ	θɛč	θɛŋ	93
9	š	č	b	kwoš	kwoč	kwob	93
10	š	ž	v	biš	biž	biv	93
11	ǰ	v	k	ji ǰ	jiv	jik	93
12	v	ž	m	skev	skež	skem	93
13	θ	f	k	gruθ	gruf	gruk	93

Figure 3.2: *This table shows a sample of the results from Experiment 2. The "Win" word is the choice where the final consonant differs from the final consonant of the target word by just one feature. The other choice word, referred to as the "Lose" word, differed by two or more phonetic features. The final column of this table shows the percentage of the 34 participants who selected the win word as the most similar sounding to the target word.* [3]

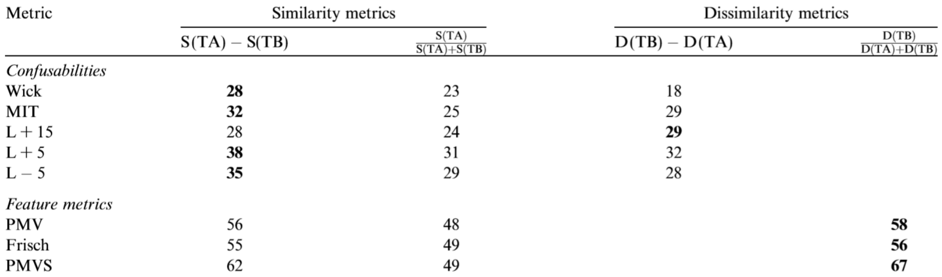
In contrast to the first experiment, rather than take the difference between S(TA) and S(TB) as a measure of each metrics response probability, an additional decision rule is implemented which predicts the probability of a particular response choice, by dividing the strength of that response by the sum of the response strength for all alternatives. What this means within the context of this experiment is that the predicted probability of a participant selecting word A is the ratio:

$$\frac{S(TA)}{S(TA) + S(TB)} \tag{3.3}$$

where S one of the seven measures for phoneme similarity. It's also worth noting that while similarity is intrinsically linked to dissimilarity, when used with either the difference or ratio decision rule to predict participant responses, the similarity and dissimilarity metrics produce different results for predicted responses. The table below shows the overall results from the experiment, where the best result for each metric is highlighted in bold.

The first conclusion that Bailey and Hahn draw from these results is that the measure of synesthetic[1] distance between two words is derived not from their level of similarity (i.e. number of features in common), but rather from the psychological estimates of difference between them. This is quite evident from the results shown above, as the ratios of PMV dissimilarities (fourth column) provided better predictions of similarity judgements than either the similarity ratio or similarity difference. The second conclusion that the authors arrive at is that as the best performing metric,

[1] Synesthesia is the perceptual phenomenon in which one sense (i.e. the hearing of a word) triggers another sense simultaneously (i.e. visual distance between two words). As Bradford [5] explains "a person with synesthesia may hear sounds while also seeing them as colorful swirls".



Metric	Similarity metrics		Dissimilarity metrics	
	S(TA) – S(TB)	$\frac{S(TA)}{S(TA)+S(TB)}$	D(TB) – D(TA)	$\frac{D(TB)}{D(TA)+D(TB)}$
Confusabilities				
Wick	**28**	23	18	
MIT	**32**	25	29	
L + 15	28	24	**29**	
L + 5	**38**	31	32	
L – 5	**35**	29	28	
Feature metrics				
PMV	56	48		**58**
Frisch	55	49		**56**
PMVS	62	49		**67**

Figure 3.3: *These numbers shows the amount of variance in participant choices relative to chance performance that can be explained by each similarity/dissimilarity metric. For the confusability metrics, the dissimilarity ratios were found to be functionally equivalent to their similarity counterpart, so their column was left blank. Likewise for the feature-based metrics, the dissimilarity differences are equivalent to the similarity differences, so they are intentionally left blank.* [3]

D_{PMV} is also the simplest measure of phoneme similarity in that it is easy to calculate not just for American English, but for any language where the manner, place and voice features of phonemes is known. Additionally, while D_{Frisch} is only 2% worse in its prediction accuracy than D_{PMV}, it involves significantly more complexity in computing how many of the 24 possible natural classes describe both phonemes.

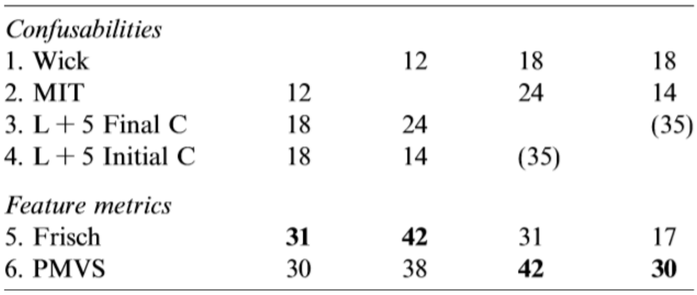
So while D_{PMV} *"recommends itself as a starting point in the search for better measures of phoneme similarity"* (p. 352), Bailey and Hahn note that the manner feature of a phoneme has been shown to carry significantly more weight than voicing or place of articulation with regard to word similarity. For this reason, they propose another feature based similarity and dissimilarity metric S_{PMVS} and D_{PMVS} by expanding the manner feature to include a fourth binary feature that distinguishes obstruent consonants from sonorant ones. As an example, /k/ is said to be obstruent as it formed by blocking airflow in the vocal tract, while /l/ does not so it is sonorant. As the Experiment 2 results show, this new metric is the best performer overall, as ratios of PMVS dissimilarities could about explain 67% of the variance.

One concern that Bailey and Hahn bring up is that while D_{PMVS} may be the best predictor in similarity judgements by participants, there is no guarantee that it is the best phoneme similarity metric for short-term memory, speech production and word recognition tasks. Rather than perform three more experiments to test the performance of D_{PMVS} in these domains, the authors decide to test the generality of each metric using the data they have already acquired for the three data-driven metrics (i.e. the phoneme confusability tables from the Wickelgren, MIT and Luce corpora which were generated from short-term memory, speech production and word recognition tasks respectively). They caution that although perceptual confusability data should not be considered as the standard method for assessing phoneme similarity (based on their poor performance in the two previous experiments), it is widely assumed by linguists that phoneme confusability accurately reflects the underlying similarities between phonemes to a significant degree.

The authors examine how well each metric predicts confusabilties across a range of domains by calculating the bivariate Spearman rank correlations[2] of each similarity metric across 171 phoneme pairs common to all the confusion datasets. The table below shows the squared correlation coefficients of each similarity metric against the Wickelgren, MIT, Luce final consonant and Luce initial consonant datasets (+5db signal-to-noise ratio for both).

The two theoretical metrics once again come out on top and are shown to highly correlate with

[2] The Spearman's Rank correlation coefficient ($-1 \leq r \leq 1$) is the statistical measure of the strength of the *monotonic* relationship between two variables. Within the context of comparing similarity metrics to confusion data, monotonic means that as the level of confusion between phonemes increases, their similarity score never decreases. The closer r is to ± 1, the stronger the monotonic relationship is.



Confusabilities				
1. Wick		12	18	18
2. MIT	12		24	14
3. L + 5 Final C	18	24		(35)
4. L + 5 Initial C	18	14	(35)	
Feature metrics				
5. Frisch	**31**	**42**	31	17
6. PMVS	30	38	**42**	**30**

Figure 3.4: *Squared correlation coefficients between similarity metrics and each of the four confusability data. [3]*

the confusability data mined from short-term memory, speech production and word recognition tasks. However before Bailey and Hahn can come to the conclusion that feature based phoneme similarity metrics are the optimal measures of perceptual confusion, they first seek to understand the level of commonality between the data from these three task domains. In other words, they try to calculate a single latent variable which they refer to as the *common core of confusability* between the three types of language processing tasks. First they assumed that the various sets of confusion data (along with a fourth set named Similarity Judgements which was derived from their second experiment) could be explained as follows:

$$MIT = CommonCore + SpeechErrorFactors + Noise \tag{3.4}$$

$$Wick = CommonCore + MemoryConfusionFactors + Noise \tag{3.5}$$

$$Luce = CommonCore + PerceptualConfusionFactors + Noise \tag{3.6}$$

$$SimJudgements = CommonCore + SimJudgementFactors + Noise \tag{3.7}$$

As they explain "the relative contribution of the common core determines the extent to which the various confusabilities measure the same thing". Using factor analysis, they identified the linear combination of confusion metrics that minimized the total square difference between this combination and each confusability metric. What they found from this analysis is that the common core could explain 52% of the total variance across the four domains, and 66, 45, 68 & 75% of the variance within the MIT, Luce and Wickelgren and Experiment 2 similarity judgement data respectively. What these results tell us is that while the Wickelgren metric is relatively uncorrelated, a significant common core exists between these four language domains which explains the underlying levels of confusion between words. To put it in simpler terms, if we used a certain metric for evaluating the level of perceptual confusion (and this level of similarity) between two words within the similarity judgement context (i.e. the SimilarSpeak application), then this same metric should be adequate at detecting confusion levels between words in a word recognition task. Additionally in a test to measure the robustness of the S_{PMVS} metric, the authors that a considerable 71% of the similarity rank variance could be explained by the dissimilarity ratios of PMVS. Essentially, what all these results show is that much of the variance within the confusability metrics can be explained by a hidden common factor "within the meta-linguistic judgements of similarity". More coarsely speaking, confusability data adequately reflects perceptual phoneme similarity.

In their concluding remarks, Bailey and Hahn decide that the perceptual data-driven measures offer no substantial benefit over the feature-based ones. Additionally since theoretical metrics were 1) better at predicting phoneme confusabilities across a range of transformations (similarity/dissimilarity, difference/ratio decision rules), and 2) can offer interpretable explanations of the similarity relationships observed, they clearly emerge as the more practical approach of the two. Finally, although Frisch's metric was better at predicting speech error data from the MIT dataset, the PMVS dissimilarity metric should be sufficient as the standard metric for computing phoneme similarities across short-term memory, speech production, word recognition and word similarity judgement tasks.

However in recent years, many have further explored some of the potential advantages of similarity measures driven by perceptual phoneme similarity. As Gallagher and Graff state in their paper *The Role of Similarity in Phonology*[6], perceptual similarity does not necessarily coincide with the natural classes or feature values used in a language's phonology. In their article, they show how the contrast between an *apical* and *retroflex* sibilant has been shown to be have less confusability than the contrast between an *apical* and *retroflex* stop, despite the fact that both pairs differ in the same exact feature. Additionally Gallagher and Graff found that the the perceptual similarity of a given phonotactically incorrect onset cluster (e.g. /**fn**ak/) can result in different levels of confusion four the four types of modifications to the onset phoneme (e.g. deletion(/nak/), substitution (/snak/), insertion (/F$_e$nak/) and prothesis[3] (/$_e$fnak/)). They found that while theoretical similarity metrics would judge the deletion modification to be more similar to the target word than the insertion modification, in speech production tasks fricative-nasal clusters and stop-nasal clusters (both phonotactically incorrect for native English speakers) would be more often misproduced with insertion modifications instead.

In an article by O'Neill and Carson-Berndsen [7], they agree with this sentiment which argues that feature-based methods are far from perfect. They explain that the construction of theoretical based distance metrics depends greatly on "exactly which features are used to distinguish between phonemes and the method in which the feature differences are calculated". This high dependency can consequently lead to a disconnect between a native speakers' perception of phoneme similarity and the calculated similarity defined in terms of phonological features and natural classes. In their paper, the authors try to uncover the source of this discrepancy by observing how the distributional patterns of phonemes influences our perception of phoneme similarity. They explain how research has shown that there exists a clear link between distribution and perception. For instance in a study by Scharenborg et al. [8], it was proved that the psychological representations of sound can change with different contexts. A phoneme halfway between /r/ and /l/ was presented to participants in either an /r/ or /l/ context, resulting in the listeners adapting to the particular phoneme representation accordingly. This phenomenon is in many way similar to the audio illusion [9] that spread across social media in 2018, in which some people heard a man saying the word "yanny", while others heard "laurel", depending on the psychological context provided by the listener's brain.

In an attempt to examine phoneme similarity O'Neill and Carson-Berndsen generated phoneme embeddings for all 24 consonant phonemes using a syllable corpus of 1.5 million syllables. This was done by using a simple skip-gram model which sought to maximise the probability of the phonemic environment given a target phoneme (similar in theory to how whole-word embedding operates). This model was then implemented with a Word2Vec tool to create 20-dimensional vector representations, with a window size of 1 phoneme either side of the target. Finally to visual the similarities captured by the model, the Euclidean distance between phoneme embeddings was calculated and the Ward clustering method was used to create the similarity hierarchy shown in figure 3.5 below.

Regarding feature-based approaches to similarity, O'Neill and Carson-Berndsen apply the Ward clustering method to Bailey and Hahn's D_{PMVS} metric to produce the model shown in 3.6 below. Although the model uses the place, manner, sonority and voicing features with equal weight, the model shows that the first distinguishing feature is sonority, while the most fine-grained differences are related to the voicing feature. The perceptual similarity model 3.7 was also generated using data from an aural word recognition experiment carried out by Cutler et al. [10]. Both of these

[3] Prothesis is the addition of a phoneme to the start word that doesn't change its intended meaning. For example, the word "far" has diachronically evolved into the word "afar", with the prothesis of the /a/ sound.

similarity models were also created using the Ward clustering method and and the same Euclidean distance formula.

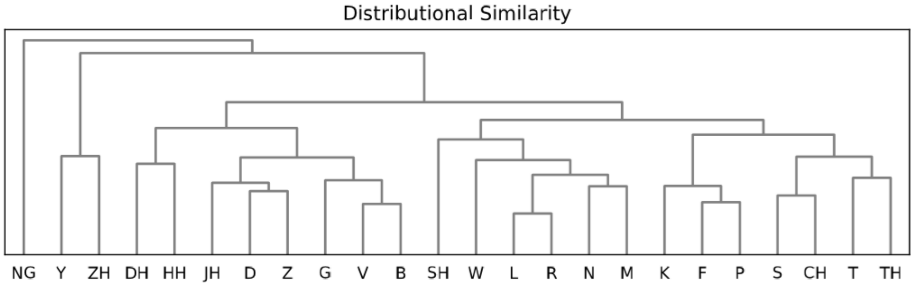


Figure 3.5: Similarity hierarchy of the distribution-based model. [7]

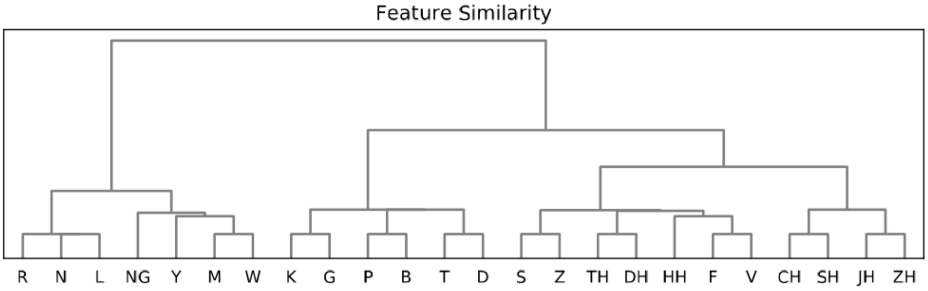


Figure 3.6: Similarity hierarchy of the feature-based model. [7]

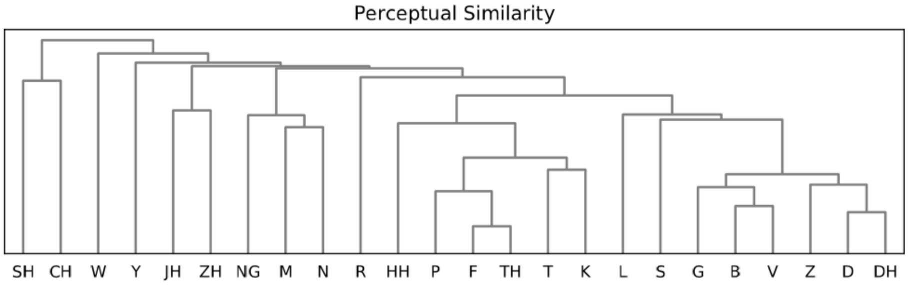
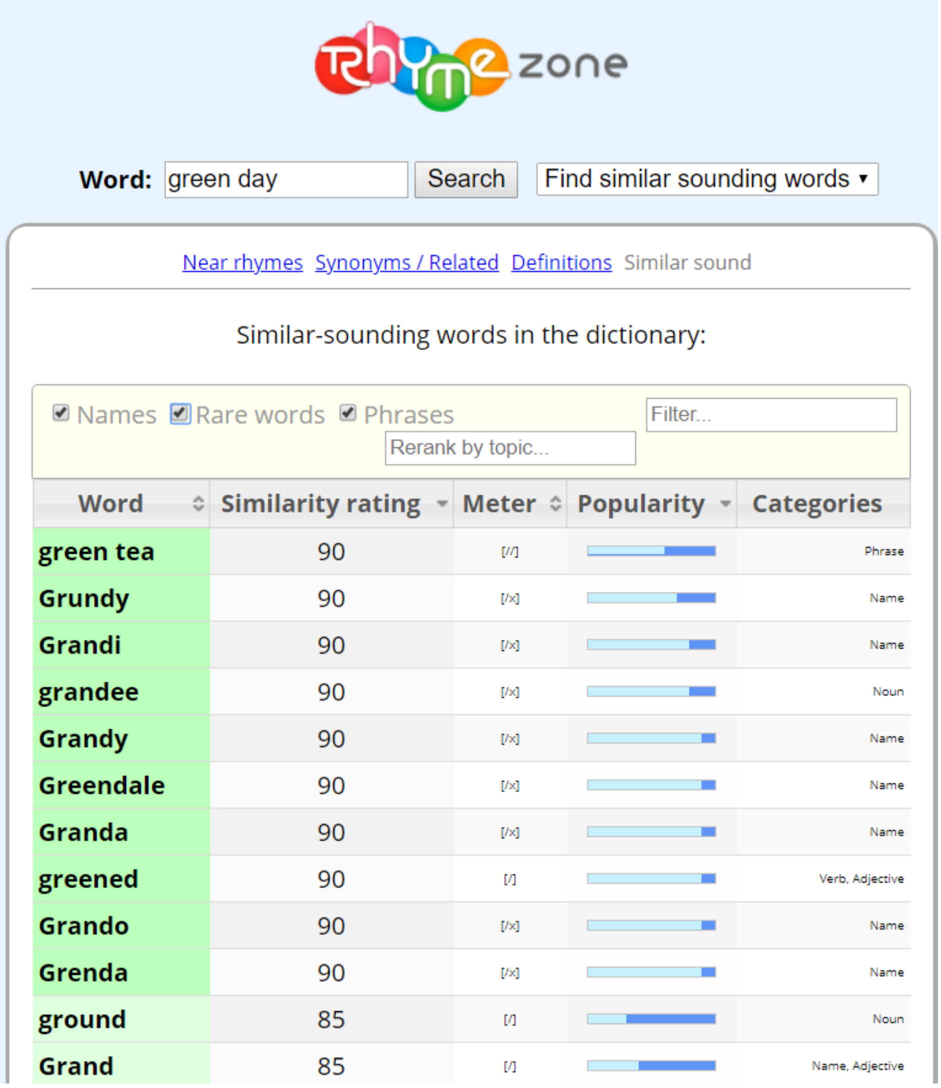


Figure 3.7: Similarity hierarchy of the perception-based model. [7]

In their analysis of the three models, the authors conclude how native speaker perceptions of phoneme similarity were not accurately captured by the feature-based model and that certain observations of the models seem to indicate that these perceptions are influenced to some degree by the distributional features of phonemes. For instance the cluster of nasal sounds (/m/, /n/ and /ng/) was absent from the feature-based model (as the place of articulation feature differs among them), while the perceptual model was able to identify the phoneme triad. Of the three sounds, /ng/ appeared to be the most dissimilar of the three, as shown by its leaf node being further away from /n/ and /m/. This small observation can be explained by the distribution model, which groups /m/ and /n/ on the same branch, but /ng/ very far away. This distance is due to the unique property of /ng/ only appearing in the final syllable position and never in the onset position.

3.2 Similar Implementations

RhymeZone [11] (originally called the *Semantic Rhyming Dictionary* until April 2000) is an online rhyming dictionary and thesaurus that was originally developed by Doug Beeferman in 1996. The website allows users to enter any valid English word or word phrase and produces a list of similar sounding words, which can be ordered by spelling, meter, popularity (according to a text corpus of 1 million words), category or, by default, a "similarity score". Three tick boxes below the input box allow the user to include/exclude proper nouns, rare words and phrases in the search. Additionally the metric column shows the stress pattern of each results, where '/' denotes a stressed syllable and 'x' denotes an unstressed syllable. The "filter" text field allows users to only see results that include a specific substring or belong to a certain word category, while the "Rerank by topic" field filters the list to only include words that are associated with a topic word (e.g. animal, place, music etc.).



Similar-sounding words in the dictionary:

Word	Similarity rating	Meter	Popularity	Categories
green tea	90	[//]		Phrase
Grundy	90	[/x]		Name
Grandi	90	[/x]		Name
grandee	90	[/x]		Noun
Grandy	90	[/x]		Name
Greendale	90	[/x]		Name
Granda	90	[/x]		Name
greened	90	[/]		Verb, Adjective
Grando	90	[/x]		Name
Grenda	90	[/x]		Name
ground	85	[/]		Noun
Grand	85	[/]		Name, Adjective

Figure 3.8: *Ranked list of similar sounding words, generated from RhymeZone.com* [11]

Chapter 4: **Data Considerations**

4.1 CMU Pronouncing Dictionary

For obtaining the *phonemic transcriptions* of English words, I will be using the word-to-pronunciation mappings provided by the The Carnegie Mellon University (CMU) Pronouncing Dictionary[12]. The listing below show four entries in the dictionary along with their respective pronunciations in the ARPAbet[1] phoneme set. Each vowel sound carries a lexical stress marker, where 0 indicates no stress, 1 indicates primary stress and 2 indicates secondary stress. Since lexical stress alone does not have a substantial enough impact on whole word similarity, these numerical markers will be filtered out after importing the CMU dictionary into the python program. Additionally, some identically spelled words will have several pronunciations associated with them. In these cases, the dictionary has an entry for each possible pronunciation, with a parenthesized number that increments for each subsequent entry. In cases like "ORANGE" where each pronunciation refers to the same word, only the first entry will be considered and all others will be disregarded. However in the case of *heteronyms* like "READ" (where alternative pronunciations of identically spelled words result in a change in meaning), each pronunciation will be exempt from being discarded. This will require constructing a small static dataset of English heteronyms (e.g. "record", "bass", "excuse" etc.).

```
APPLE         AE1  P  AH0  L
BANANA        B  AH0  N  AE1  N  AH0
ORANGES       AO1  R  AH0  N  JH
ORANGES(1)    AO1  R  IH0  N  JH
READ          R  EH1  D
READ(1)       R  IY1  D
```

Listing 4.1: Formatting of translation data provided by the CMU dictionary [12]

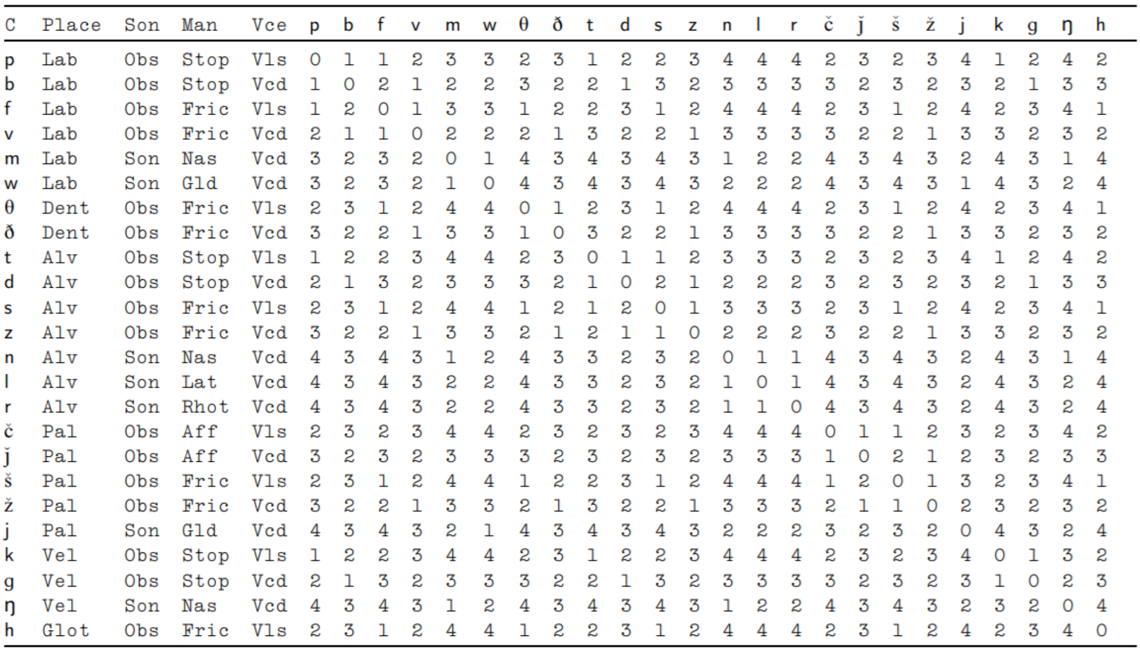
For importing the data from the CMU dictionary, I plan using **cmudict**, a python wrapper package that includes a data file (cmu.dict) with all entries in the CMU dictionary formatted like in the examples in Listing 4.1 above. This translation data will be stored locally as a hash map (where each key-value pair is a unique syllabified pronunciation mapped to the corresponding word it represents).

[1] While there exist various other alphabets for representing the English phonemes, such as IPA (International Phonetic Alphabet) or SAMPA, ARPAbet is unique in that it uses interpretable, machine-readable ACII characters to represent each sound defined by the IPA.

4.2 Phoneme Distance Matrices

As mentioned in the introductory section (chapter 2), finding the phonetic distances between words will require prior knowledge of the distances between individual phonemes. This knowledge will be represented locally as a series of separate consonant and vowel phoneme-distance matrices. Each immutable 24x24 consonant matrix and 15x15 vowel matrix will be stored locally as a .csv file and referenced by a Matrices Container class, so that during execution the currently selected consonant and vowel matrix will be used to calculate the phonetic distances between words. My intention is to ensure through experimentation that the system can function properly when using each possible combination of matrices. If, for whatever reason, a distance between two phonemes is unknown, the maximum possible distance (lowest possible similarity) will be assumed. As this project is not concerned with discovering the actual distances between consonant and vowel phoneme, each matrix will either provided by my supervisor or will be taken from an publicly available report. Each distance metric uses different value ranges (e.g. Bailey and Hahn's PMVS metric contains integer values from 0 - 4 inclusive, while some confusability metrics contains float values from 0 to positive infinity), so all values will have to be normalized to the same format (possibly as float between 0 and 1 inclusive).

Below (Fig 4.1) is an example of consonant similarity matrix proposed by Bailey and Hahn [3], which is based on their phonological feature-based metric, PMVS:



C	Place	Son	Man	Vce	p	b	f	v	m	w	θ	ð	t	d	s	z	n	l	r	č	ǰ	š	ž	j	k	g	ŋ	h
p	Lab	Obs	Stop	Vls	0	1	1	2	3	3	2	3	1	2	2	3	4	4	4	2	3	2	3	4	1	2	4	2
b	Lab	Obs	Stop	Vcd	1	0	2	1	2	2	3	2	2	1	3	2	3	3	3	3	2	3	2	3	2	1	3	3
f	Lab	Obs	Fric	Vls	1	2	0	1	3	3	1	2	2	3	1	2	4	4	4	2	3	1	2	4	2	3	4	1
v	Lab	Obs	Fric	Vcd	2	1	1	0	2	2	2	1	3	2	2	1	3	3	3	3	2	2	1	3	3	2	3	2
m	Lab	Son	Nas	Vcd	3	2	3	2	0	1	4	3	4	3	4	3	1	2	2	4	3	4	3	2	4	3	1	4
w	Lab	Son	Gld	Vcd	3	2	3	2	1	0	4	3	4	3	4	3	2	2	2	4	3	4	3	1	4	3	2	4
θ	Dent	Obs	Fric	Vls	2	3	1	2	4	4	0	1	2	3	1	2	4	4	4	2	3	1	2	4	2	3	4	1
ð	Dent	Obs	Fric	Vcd	3	2	2	1	3	3	1	0	3	2	2	1	3	3	3	3	2	2	1	3	3	2	3	2
t	Alv	Obs	Stop	Vls	1	2	2	3	4	4	2	3	0	1	1	2	3	3	3	2	3	2	3	4	1	2	4	2
d	Alv	Obs	Stop	Vcd	2	1	3	2	3	3	3	2	1	0	2	1	2	2	2	3	2	3	2	3	2	1	3	3
s	Alv	Obs	Fric	Vls	2	3	1	2	4	4	1	2	1	2	0	1	3	3	3	2	3	1	2	4	2	3	4	1
z	Alv	Obs	Fric	Vcd	3	2	2	1	3	3	2	1	2	1	1	0	2	2	2	3	2	2	1	3	3	2	3	2
n	Alv	Son	Nas	Vcd	4	3	4	3	1	2	4	3	3	2	3	2	0	1	1	4	3	4	3	2	4	3	1	4
l	Alv	Son	Lat	Vcd	4	3	4	3	2	2	4	3	3	2	3	2	1	0	1	4	3	4	3	2	4	3	2	4
r	Alv	Son	Rhot	Vcd	4	3	4	3	2	2	4	3	3	2	3	2	1	1	0	4	3	4	3	2	4	3	2	4
č	Pal	Obs	Aff	Vls	2	3	2	3	4	4	2	3	2	3	2	3	4	4	4	0	1	1	2	3	2	3	4	2
ǰ	Pal	Obs	Aff	Vcd	3	2	3	2	3	3	3	2	3	2	3	2	3	3	3	1	0	2	1	2	3	2	3	3
š	Pal	Obs	Fric	Vls	2	3	1	2	4	4	1	2	2	3	1	2	4	4	4	1	2	0	1	3	2	3	4	1
ž	Pal	Obs	Fric	Vcd	3	2	2	1	3	3	2	1	3	2	2	1	3	3	3	2	1	1	0	2	3	2	3	2
j	Pal	Son	Gld	Vcd	4	3	4	3	2	1	4	3	4	3	4	3	2	2	2	3	2	3	2	0	4	3	2	4
k	Vel	Obs	Stop	Vls	1	2	2	3	4	4	2	3	1	2	2	3	4	4	4	2	3	2	3	4	0	1	3	2
g	Vel	Obs	Stop	Vcd	2	1	3	2	3	3	3	2	2	1	3	2	3	3	3	3	2	3	2	3	1	0	2	3
ŋ	Vel	Son	Nas	Vcd	4	3	4	3	1	2	4	3	4	3	4	3	1	2	2	4	3	4	3	2	3	2	0	4
h	Glot	Obs	Fric	Vls	2	3	1	2	4	4	1	2	2	3	1	2	4	4	4	2	3	1	2	4	2	3	4	0

Figure 4.1: *Consonant distance matrix provided by Bailey and Hahn, where each value is determined by the differences in the four major class features between the two phonemes (place, sonority-obstruent, manner, and voicing). These values will be normalized and then stored locally in .csv data file.*

Chapter 5: **Outline of Approach**

My current plan for the SimilarSpeak system is to develop a base python program initially using Jupyter Notebook to ensure proper functionality within each component and then moving on to a more comprehensive development environment that is suitable for larger scale projects (e.g. Visual Studio Code or Pycharm). While I will only begin to develop the system's front end once the core python program can output ranked lists including and excluding nonsense words, I will nevertheless need a simple user interface which will take in the user's input word and query preferences and then display the list of the most similarly sounding words/word phrases to the input word. I have decided that the best approach to achieving these goals is to use Django, a powerful open-source framework for displaying the user interface as a local web application. Although there are many alternative python web frameworks available to use (such as Flask and Grok), Django seems to be the best option for me, as I am quite unfamiliar with web development and there seems to be a lot of great resources online for learning how to use this framework. Of course my decision to use Django isn't final and I may discover a better means of implementing the user interface while developing the primary python program, but for now it is my first preference choice.

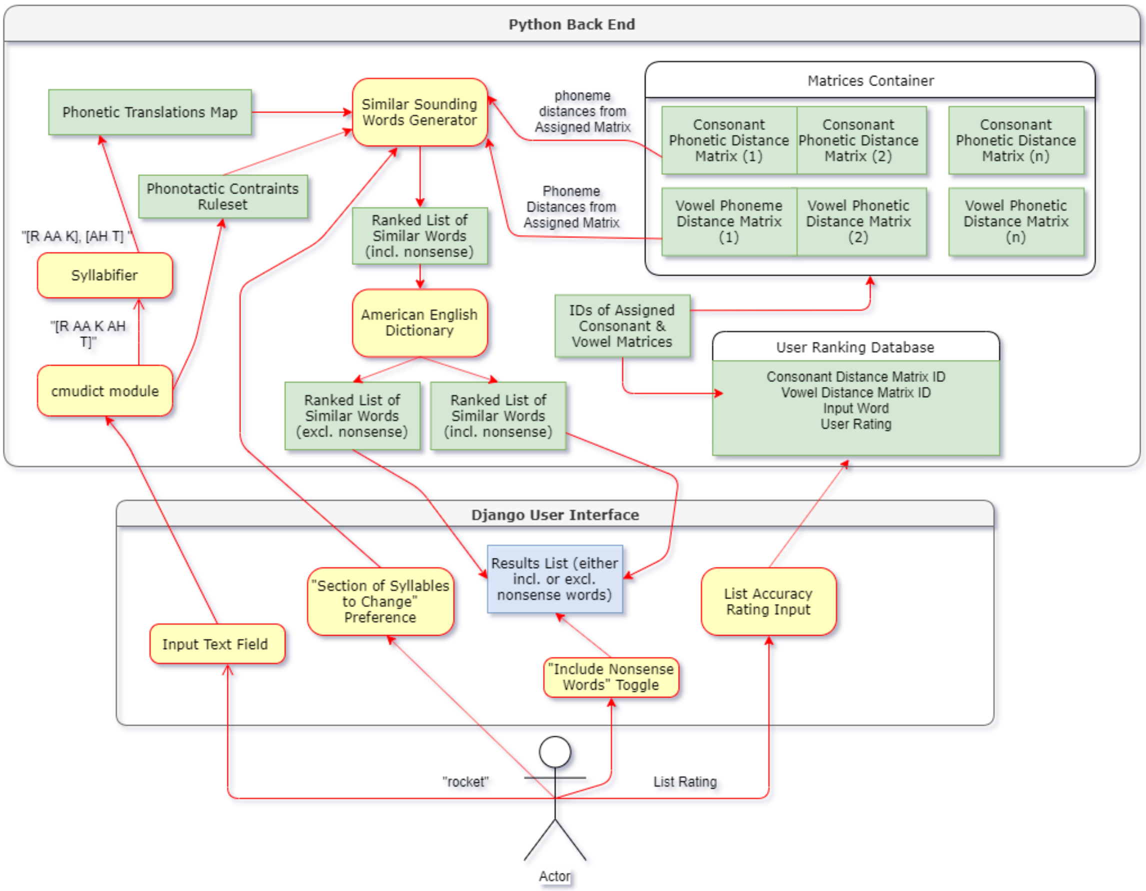

Python Back End
Phonetic Translations Map
Similar Sounding Words Generator
phoneme distances from Assigned Matrix
Matrices Container
Consonant Phonetic Distance Matrix (1)
Consonant Phonetic Distance Matrix (2)
Consonant Phonetic Distance Matrix (n)
Phonotactic Contraints Ruleset
Phoneme Distances from Assigned Matrix
Vowel Phoneme Distance Matrix (1)
Vowel Phonetic Distance Matrix (2)
Vowel Phonetic Distance Matrix (n)
"[R AA K], [AH T] "
Syllabifier
Ranked List of Similar Words (incl. nonsense)
American English Dictionary
IDs of Assigned Consonant & Vowel Matrices
"[R AA K AH T]"
User Ranking Database
Consonant Distance Matrix ID
Vowel Distance Matrix ID
Input Word
User Rating
cmudict module
Ranked List of Similar Words (excl. nonsense)
Ranked List of Similar Words (incl. nonsense)
Django User Interface
Results List (either incl. or excl. nonsense words)
"Section of Syllables to Change" Preference
List Accuracy Rating Input
Input Text Field
"Include Nonsense Words" Toggle
"rocket"
List Rating
Actor

Figure 5.1: *High-level overview of my vision for the SimilarSpeak system. The word "rocket" is shown as an example of how the input word will be broken up into its syllabified phonetic representation. Yellow boxes denote a functioning system component, while green boxes indicated data of some kind. The final displayed results list is shown in blue for emphasis.*

Any effective approach to a solving a problem requires a clear yet flexible idea of how go about doing so. With this in mind, I decided to outline my plan for constructing the SimilarSpeak system with a UML diagram (Fig 5.1 on the previous page). Although the diagram may appear rustic in its design due to the exclusion of specific methods calls, I believe it accurately depicts how each component in the system will communicate and work harmoniously with one another.

To begin, a simple text field in the user input will take in the target word from the user and pass it into the python back end program. From there the cmudict python package will be used to construct a map of all valid English words to their syllabified phonemic translation using a provided syllabifier. The diagram shows how an input word will be first translated into into phonemic representation ("rocket" -> "R AA K AH T") and then into its determined syllable components ("[R AA K], ["AH T]").

The component for generating similar sounding word structures is the most important part of the entire system, as a poor implementation of this will subsequently result in long wait times and inefficient use of memory. My very first idea for finding similar words was to design two entirely separate systems: one for finding well-defined words and another for generating nonsense words (possibly using BK-tree structure [13] for the defined words). However upon further contemplation, I realized that it would be far more efficient to first design a component that would find all possible unique phonemic strings that are 1 edit away from the original target phonetic word, according to the Levenshtein distance formula. This is a method of finding how dissimilar two words are from one another by calculating the minimum number of insertions, deletions and substitutions needed to transform one string to the other. So for example, the distance between "**c**at" and "**b**at" is 1, while the distance between "w**h**at" and "wat**er**" is 3. The Levenshtein distance between strings a and b is given by the formula below:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 5.2: *Levenshtein distance between strings a and b, where $1(_{a_i \neq b_j})$ denotes 0 when a = b and 1 otherwise. The operations in the minimum are ordered as follows: deletion, insertion, substitution.* [14]

The diagram below shows how I envision the work flow of the 1-edit distance hierarchy component. Essentially each syllable of the phonemic translation of the target word would be dealt with separately, with an end-of-word symbol (EOW), '-', appended to the end of the word. Within each syllable branch, the component generates a large dataset of all possible unique phonemic strings found by applying all three Levenshtein edits on each phoneme, apart from the EOW symbol which would only have the insertion edit applied to it. Taking the target word "glowing" as an example, the word generated from the "G deletion" operation would be "L OW IH NG", while the words generated from the "G substitution" operation would range from "**P** L OW IH NG" to "**HH** L OW IH NG". Similarly the words generated from "G insertion" would range from "G **PP** L OW IH NG" to "G **HH** L OW IH NG"[1]. One all 1-edit possible phonemic strings have been added to the set of generated words, the following operations will need to be completed:

1. Remove unpronounceable generated words by first creating a large dataset of every possible pronounceable English syllable based of all those found in the syllabified phonetic dictionary. If a generated word contains a onset-vowel-coda combination that is not found in this dataset (e.g. "T L AH"), then it will be deemed unpronounceable and will be subsequently removed from the list.

[1] It's worth noting that the insertion edit will add the phoneme *before* the particular target phoneme

2. Generate a phoneme distance score for each generated word using the values provided by the currently selected consonant and vowel matrices. As all generated words will contain the exact same number of syllables as the target word, the distance score will be found by comparing each coda, vowel and onset cluster in n$_{th}$ syllable of the generated word with each equivalent coda, vowel and onset cluster in the n$_{th}$ syllable of the target word. As there could be an unequal number of phonemes between codas or onsets clusters, a string alignment tool will be used to calculate the cluster distance score. The total distance score for each generated word will be found by averaging the scores of each onset, coda and vowel comparison (with more weight given to the vowel sound).

3. Remove all generated words from the set whose score falls below an arbitrary threshold. This threshold value is different from the user-controlled displayed maximum distance.

4. Look up each generated word in the stored phonetic dictionary and set a Boolean value to be true if the entry appears. This value will be later used to split the entire set of generated words into two lists: one excluding nonsense words and another including them.

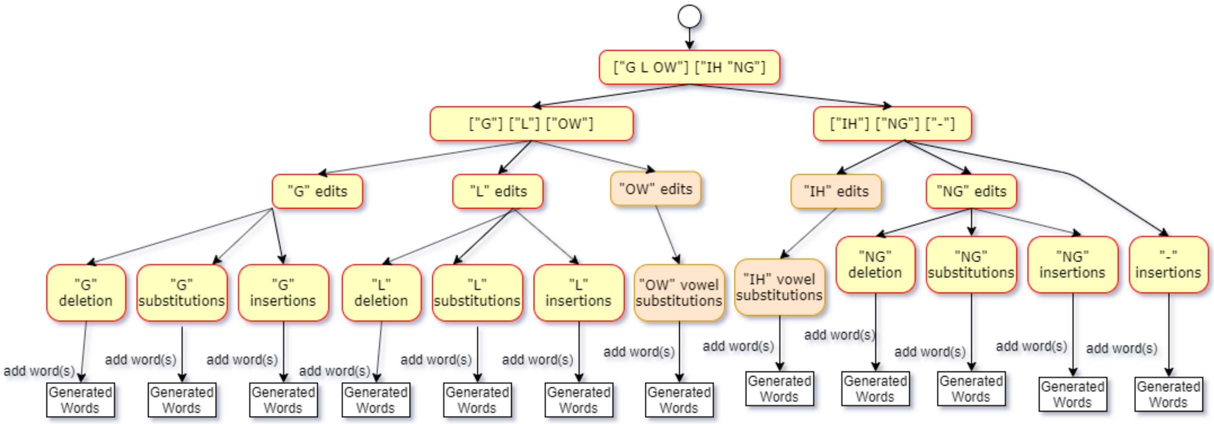


Figure 5.3: *Flow of operations for deriving all words that are 1-edit distance away from the target word. The example word here is "G L OW IH NG" ("glowing").*

These set of steps will result in a (potentially very large) dataset of nonsense and non-nonsense phonemic strings that are one deletion, insertion or substitution away from the phonemic representation of the target word. A simple Quicksort function will then rank these entries in ascending order of their distance score. In order to find words that are more than one edit distance away from the target word, a subset of the most similar generated words will be selected (either based on a predefined proportion or another distance threshold), and for each word in this subset, the same steps as described above will be applied with the same maximum distance threshold value from the original input word. This will result in the discovery of additional words that are 2 Levenshtein edits away from the target word. Further experimentation will determine if there is enough value in finding words that are 3 or even 4 Levenshtein edits away - at the cost of substantially higher computation times. Figure 5.4 below visualizes the order of operations within this component of the python back end.

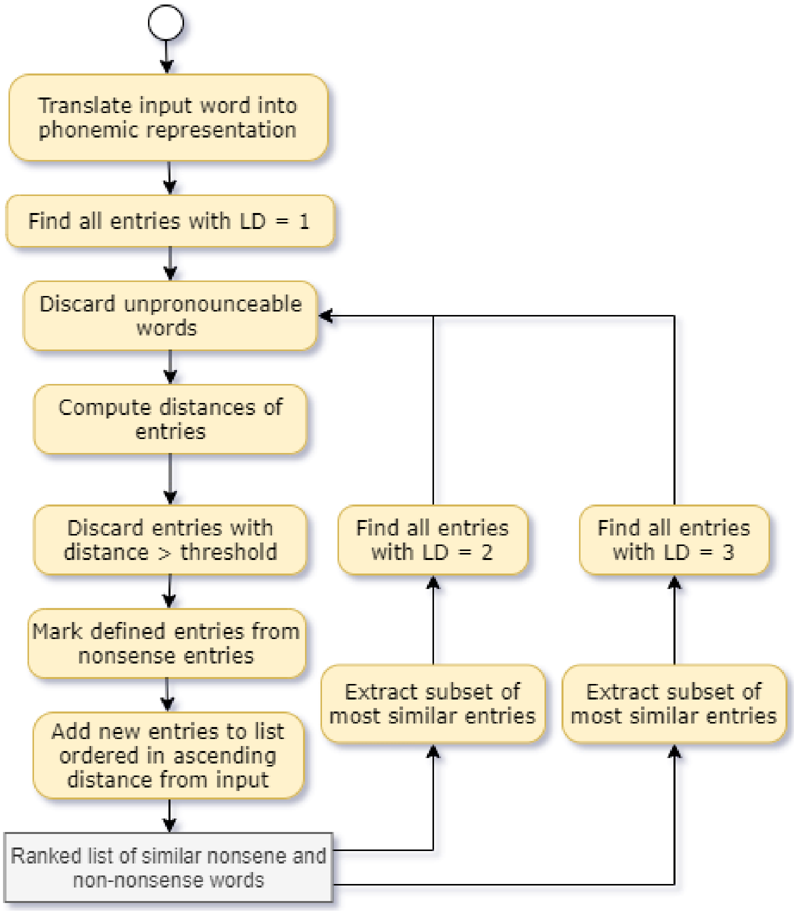

Translate input word into phonemic representation
Find all entries with LD = 1
Discard unpronounceable words
Compute distances of entries
Discard entries with distance > threshold
Mark defined entries from nonsense entries
Add new entries to list ordered in ascending distance from input
Ranked list of similar nonsene and non-nonsense words
Find all entries with LD = 2
Find all entries with LD = 3
Extract subset of most similar entries
Extract subset of most similar entries

Figure 5.4: *Order of operations within component for finding similar sounding words. LD refers to the Levenshtein distance between the generated phonemic string and the input phonemic string.*

Once all similar nonsense and non-nonsense words have been computed, the python program will produce two lists for the target word and then send them to the Django front end to be displayed to the user. A simple toggle will change which list is being displayed. An input field will let the user submit their rating for the ranked list. A database will contain each user rating alongside the target word, references to the consonant and vowel distance matrices used and the original input word.

Chapter 6: **Project Workplan**

6.1 Gannt Chart

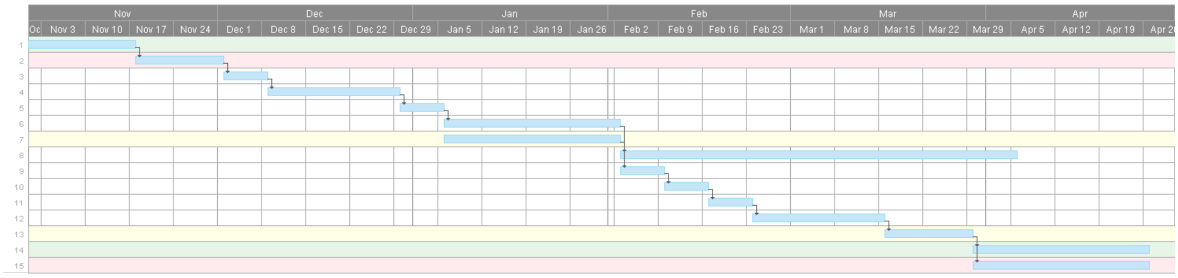
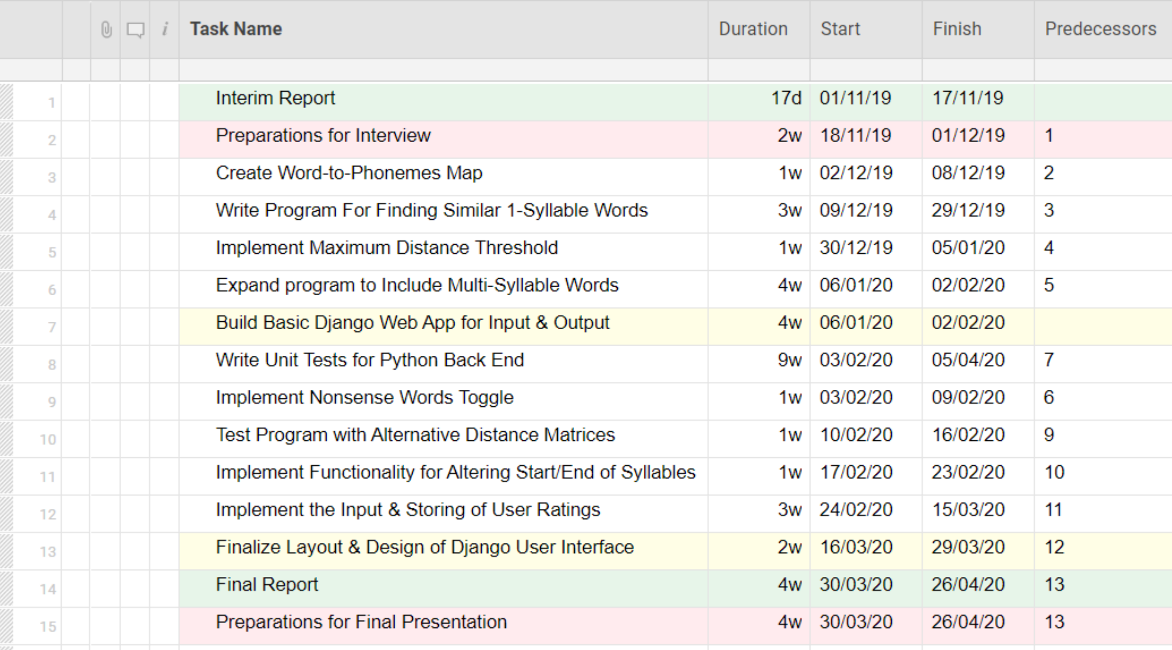


Figure 6.1: *Gannt Chart showing when each task should begin and be completed. Yellow rows indicate a task relating to the User Interface, while green and red rows denote reports and presentations respectively. Task details are shown in figure 6.2 below.*



	Task Name	Duration	Start	Finish	Predecessors
1	Interim Report	17d	01/11/19	17/11/19	
2	Preparations for Interview	2w	18/11/19	01/12/19	1
3	Create Word-to-Phonemes Map	1w	02/12/19	08/12/19	2
4	Write Program For Finding Similar 1-Syllable Words	3w	09/12/19	29/12/19	3
5	Implement Maximum Distance Threshold	1w	30/12/19	05/01/20	4
6	Expand program to Include Multi-Syllable Words	4w	06/01/20	02/02/20	5
7	Build Basic Django Web App for Input & Output	4w	06/01/20	02/02/20	
8	Write Unit Tests for Python Back End	9w	03/02/20	05/04/20	7
9	Implement Nonsense Words Toggle	1w	03/02/20	09/02/20	6
10	Test Program with Alternative Distance Matrices	1w	10/02/20	16/02/20	9
11	Implement Functionality for Altering Start/End of Syllables	1w	17/02/20	23/02/20	10
12	Implement the Input & Storing of User Ratings	3w	24/02/20	15/03/20	11
13	Finalize Layout & Design of Django User Interface	2w	16/03/20	29/03/20	12
14	Final Report	4w	30/03/20	26/04/20	13
15	Preparations for Final Presentation	4w	30/03/20	26/04/20	13

Figure 6.2: *Summary of key tasks and their dependencies as outlined by the project Gantt chart*

6.2 Breakdown of Project Workplan

Figures 6.1 & 6.2 above shows my current work plan for constructing the SimilarSpeak program. Although some of the dates of the later tasks will be updated when exact deadlines are given, the order of main tasks and their Finish-to-Start dependencies are presented.

6.2.1 Tasks 3-5

As mentioned in the previous chapter, I intend to begin working on the functionality of the core python program first. As I have already successfully written a basic Jupyter Notebook program that can translate input words into their phonemic representations using the cmudict module, I expect that I should have a comprehensive word to phoneme map written by **8th December**. From there I have allocated myself with three weeks to construct the python program that can find similar sounding one-syllable words. This will involve first finding similar pronounceable words that are a single Levenshtein edit away from the target word and then finding similar words that are two and three edits away. Once this has been completed, the next task will be to implement the *Maximum Word Distance* variable, so that after the initial list of similar words has been generated (from generating similar words that are 1-, 2- & 3-Levenshtein edits away from the input), a changeable distance threshold can expand or shorten the output list. This task will also involve testing the way in which whole word distances are calculated, to ensure that the implementation accurately reflects the phoneme distances specified by the initial consonant and vowel distance matrices.

6.2.2 Tasks 6-8

I plan to spend much of the Christmas break and the first two weeks on the Spring trimester planning and constructing the foundations of the user interface. As I am quite unfamiliar with designing web deployed applications in python, I am confident that I have given myself enough time to learn how to properly use the Django framework. However as none of the core python tasks rely on the completion of the user interface foundations, I will be working on it while simultaneously implementing the syllabifier, so that similar nonsense and non-nonsense words can be found for input words with two or more syllables. Once these two tasks have been complete (by **2nd February**) , I will begin writing some of basic unit tests for the python program.

6.2.3 Tasks 9-11

As I intend to find the similar nonsense words concurrently with the non-nonsense words, I expect that it should only take one week to implement the toggle for switching between the two lists in the web app. Once this toggle has been implemented, I will also be able experiment with the maximum distance threshold values, so that an optimal default value can be found for each lists (i.e. a value that results in an average of possibly 10 similar words for both lists). By the **16th February** (end of Week 4 of the Spring Trimester) I want to have ensured that the program functions properly with normalised matrices derived from alternative distance metrics. From here I have given myself two weeks to implement the option of only changing the beginnings and endings of syllables within the input word. In addition to these two options, I will also present the option of changing everything except for the ending of the final syllable, so that words rhyming with the input word can be found.

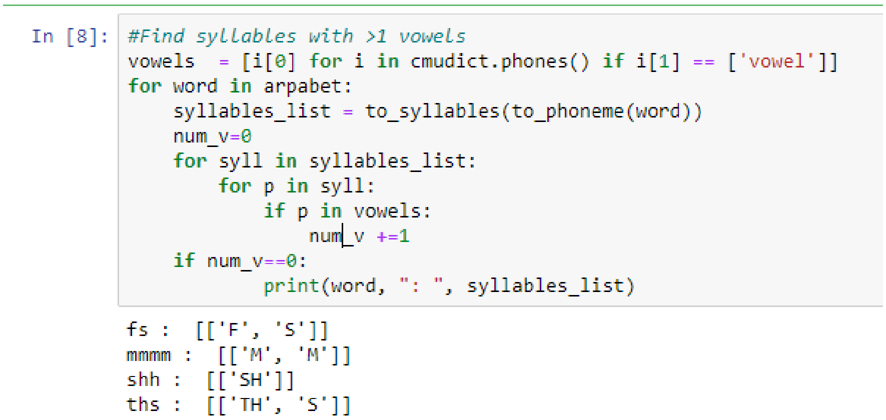
6.2.4 Tasks 12-15

By the **24th February**, I plan to begin implementing the user rating subsystem. This will involve creating both the input method within the Django framework and the database for storing information relating to each user rating. The final practical task is to finalize the user interface of the Django application. As shown in snippet from the Gannt chart, I expect this task to be completed by the **29th March**, however this date may change once the deadline for project submissions has been finalized. Finally, I anticipate that the majority of the work for the final report and presentation will take place in the month of April, so I have dedicated four weeks to both tasks in the Gannt chart as well.

Chapter 7: Data and Context

As mentioned in the Introductory section, one of the primary objectives of this project is to create an *adaptable* system for finding similar sounding words. In my finished application, this adaptability was delivered by allowing for alternative distance matrices. When a phoneme distance is to be calculated, the consonant-consonant or vowel-vowel distance for each phoneme pair is extracted from the matrix and normalised between 0 and 1. As I was unable to experiment with an alternative vowel distance matrix, the default matrix assumed all non-identical vowel pairs to be exactly 0.5 away (more specifically this equal to the *gap penalty* variable, which is the distance between a phoneme and a gap),

This adaptability also extends to a customisable dictionary for adding or removing custom text-to-phoneme translations. I found this to be a necessity as the default CMU dictionary contains several entries that would produce incorrect results, as the similar word generator assumes that all syllables contain exactly one vowel sound. In testing, I found two entries with more than one vowel sound when passed through the syllabifier (*couatre* & *hatheway*) and four instances of words with no vowel sounds at all (Fig 7).



```
In [8]: #Find syllables with >1 vowels
        vowels  = [i[0] for i in cmudict.phones() if i[1] == ['vowel']]
        for word in arpabet:
            syllables_list = to_syllables(to_phoneme(word))
            num_v=0
            for syll in syllables_list:
                for p in syll:
                    if p in vowels:
                        num_v +=1
            if num_v==0:
                    print(word, ": ", syllables_list)
```

```
fs :  [['F', 'S']]
mmmm :  [['M', 'M']]
shh :  [['SH']]
ths :  [['TH', 'S']]
```

Chapter 8: Core Contribution

8.1 Flask Web Framework

Based on the feedback I received from my moderators, I decided to use Flask as the python web framework. Compared to my original choice of Django, Flask is a more light-weight solution with a fast debugging tool and *RESTful* request dispatching. REST is an extension to flask that allows for the creation of an API sever by taking advantage of HTTP protocols to provide easy access to local resources. For the Similar Speak application this is the local SQLite database that is updated with similar sounding words as they are discovered. I created the flask application with API endpoints to make it possible to extend the program into an API application so that words could be search by accessing a URL.

Within the flask program the functionality for displaying the word table in the user interface is created in JavaScript, and was written to follow the MVC (Model-View-Controller) design pattern. The Model provides a connection to the words API. When a particular user interaction (e.g. search) is called, the Controller calls the corresponding Model API, in order to interact with the local connexion server. The view updates the HTML DOM (Document Object Model) to allow the word data to be displayed. When an update to the view is required, the Controller calls the View API. The controller creates the event handlers for user interaction, so that it call the Model to make requests to the people API, and call the View to update the DOM with new word data.

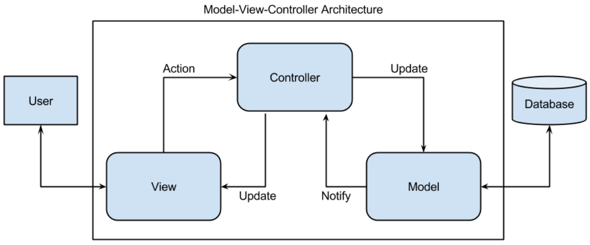


Figure 8.1: *MVC software pattern used for flask front-end [15]*

8.2 Word Generation

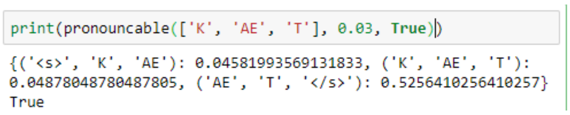
My final implementation for the word generation program is very similar to how I envisioned it in Figure 5.4, but with some key differences. The word generator splits the *syllabified* input word into its separate syllable components and then splits it once more into into pre-vowel phonemes (onset), vowel, and post-vowel phonemes (coda). Derivations of the coda and onset are made by inserting one phoneme to the beginning and end, removing one phoneme and swapping each phoneme for another one. As each syllable must have a vowel component, the vowel is only swapped out for another. Each modification is stitched back to the original word to form a new entry, which is then sent to the pronunciation module to determine its phonetic validity. As in my original plan, I initially followed figure 5.4 by calculating the distances of these first list of words, and fed all entries below a distance threshold of 0.5 back into the word generation program. However this increased the list of generated words exponentially, resulting in long execution times.

However this resulted in a conundrum, as removing the the functionality to recursively generate more words from the original list of generated words, would make the discovery of 2-edit Levenshtein entries impossible (e.g. "cat" (K AA T) is two edits away from "bad" (B AA D)). My solution to this problem was to generate syllables that had one edit in the onset position and one in the coda position and with the same vowel sound. This decision stemmed from intuition about the importance of vowels as the *nucleus* of the word.

8.3 Pronounceable Module

When constructing the module that would determine the pronouncability of a word, I initially considered creating a lookup database of all syllables that appear in the cmu dictionary. However due to the presence of obscure phoneme pairings. For instance, a "T", "L" phoneme pair is not valid, however the instance of *dietl* -> ["D" "AY"] ["AH" "T" "L"], would have contradicted this.

My solution to this problem was to create a trigram model containing the conditional probabilities of all phoneme tuples. This works by first converting all entries in the cmu dict into the phonetic representation and then passing them through the syllabifier. I then created a bigram model, which estimates the probability of a phoneme by using only the conditional probability of the previous phoneme. This is then generalized to the trigram model which calculated the conditional probability for each phoneme tuple.



```
print(pronouncable(['K', 'AE', 'T'], 0.03, True))
```

```
{('<s>', 'K', 'AE'): 0.04581993569131833, ('K', 'AE', 'T'):
0.04878048780487805, ('AE', 'T', '</s>'): 0.5256410256410257}
True
```

Figure 8.2: *Example conditional probabilities to determine pronouncability*

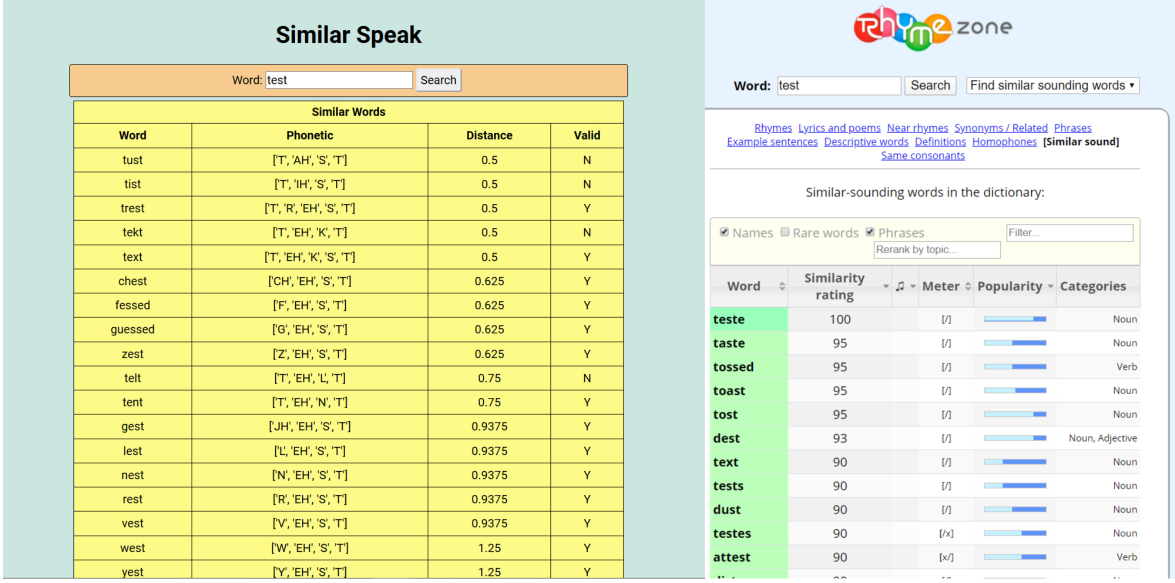
8.4 Word Alignment and Distance Metric

The phonetic distance from the input for all pronouncable words is then calculated by aligning the phonemes, inserting gaps if they are of different lengths. In an early attempt at creating an alignment algorithm, I aligned the phonemes of each syllable in a such way so that the vowels would always be aligned together. However this meant that final syllable of the longer word would always be compared to a gap, and words with similar consonant phonemes would not be properly aligned if they appear at different ends of the vowel sound (e.g. Each 'T' would not be aligned in the following alignment: ["T", "UW", "-"] ["AE", "T"]).

In researching alternative ways to align phonemes, I came across a paper by Hixon et al. [16], in which they modify they use the Needleman-Wunsch protein alignment algorithm to find the maximum similarity score for two strings. This dynamically aligns two strings by first creating a scoring matrix from the strings and then tracing back to find the path that results in the smallest global distance. They explain how in nucleotides, a gap created by an insertion or deletion may have "significant biological consequences", therefore a severe penalty is usually applied, however for phonetic words, this penalty should not be as severe. In the final program, this value is set to 0.5.

Chapter 9: Evaluation

My original plan was to evaluate the quality of the generated words by conducting in-person experiments. This would have involved asking participants to enter one word of their choosing, which would be used to generate two lists: one from the SimilarSpeak application and another from the public RhymeZone API. Having the participants then select the list that they judge most accurate, would have perhaps highlighted the effectiveness of the SimilarSpeak application for different selections of words. Unfortunately this was not able to be conducted due to social distancing guidelines.



Similar Speak

Word: test Search

Similar Words			
Word	**Phonetic**	**Distance**	**Valid**
tust	['T', 'AH', 'S', 'T']	0.5	N
tist	['T', 'IH', 'S', 'T']	0.5	N
trest	['T', 'R', 'EH', 'S', 'T']	0.5	Y
tekt	['T', 'EH', 'K', 'T']	0.5	N
text	['T', 'EH', 'K', 'S', 'T']	0.5	Y
chest	['CH', 'EH', 'S', 'T']	0.625	Y
fessed	['F', 'EH', 'S', 'T']	0.625	Y
guessed	['G', 'EH', 'S', 'T']	0.625	Y
zest	['Z', 'EH', 'S', 'T']	0.625	Y
telt	['T', 'EH', 'L', 'T']	0.75	N
tent	['T', 'EH', 'N', 'T']	0.75	Y
gest	['JH', 'EH', 'S', 'T']	0.9375	Y
lest	['L', 'EH', 'S', 'T']	0.9375	Y
nest	['N', 'EH', 'S', 'T']	0.9375	Y
rest	['R', 'EH', 'S', 'T']	0.9375	Y
vest	['V', 'EH', 'S', 'T']	0.9375	Y
west	['W', 'EH', 'S', 'T']	1.25	Y
yest	['Y', 'EH', 'S', 'T']	1.25	Y

RhymeZone

Word: test Search Find similar sounding words ▾

Rhymes Lyrics and poems Near rhymes Synonyms / Related Phrases Example sentences Descriptive words Definitions Homophones [Similar sound] Same consonants

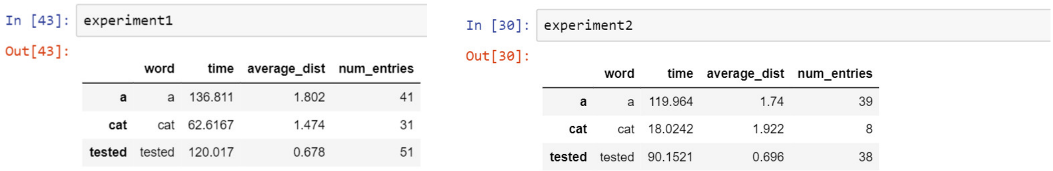
Similar-sounding words in the dictionary:

☑ Names □ Rare words ☑ Phrases Filter... Rerank by topic...

Word	Similarity rating	♫	Meter	Popularity	Categories
teste	100		[/]		Noun
taste	95		[/]		Noun
tossed	95		[/]		Verb
toast	95		[/]		Noun
tost	95		[/]		Noun
dest	93		[/]		Noun, Adjective
text	90		[/]		Noun
tests	90		[/]		Noun
dust	90		[/]		Noun
testes	90		[/x]		Noun
attest	90		[x/]		Verb

Figure 9.1: *Comparison of results from SimilarSpeak and RhymeZone[11]*

However as the program's performance dependant on several parameters, I was able to run 4 different experiments to determine the optimal balance of execution time and accuracy. In the first two experiments below, I ran the program with a pronunciation threshold (minimum conditional value required for a tuples in the syllable) of 0.2 and 0.4 respectively.



```
In [43]: experiment1
Out[43]:
```

	word	time	average_dist	num_entries
a	a	136.811	1.802	41
cat	cat	62.6167	1.474	31
tested	tested	120.017	0.678	51

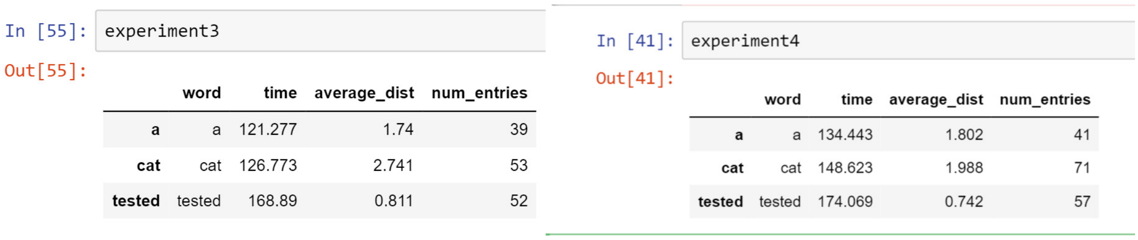
```
In [30]: experiment2
Out[30]:
```

	word	time	average_dist	num_entries
a	a	119.964	1.74	39
cat	cat	18.0242	1.922	8
tested	tested	90.1521	0.696	38

Figure 9.2: *Experiments 1 & 2 (Threshold values of 0.2 and 0.4 respectively)*

In the third and fourth experiments again run with a threshold value of 0.2 and 0.4 respectively but with one noticeable difference. These experiments allow for a swap/delete edits for both coda *and* onset, rather than just one or the other. This allows for the discovery of additional words that are two Levenshtein edits away (e.g. "K AE T" -> "L AE D").

As "cat" in experiment 2 shows, increasing the pronouncability threshold to 0.4 can significantly decrease the number of results produces. Additionally, experiment 3 & 4 showed inconsistent



```
In [55]: experiment3
Out[55]:
```

	word	time	average_dist	num_entries
a	a	121.277	1.74	39
cat	cat	126.773	2.741	53
tested	tested	168.89	0.811	52

```
In [41]: experiment4
Out[41]:
```

	word	time	average_dist	num_entries
a	a	134.443	1.802	41
cat	cat	148.623	1.988	71
tested	tested	174.069	0.742	57

Figure 9.3: *Experiments 3 & 4 (threshold values of 0.2 and 0.4 respectively)*

difference in execution times and average distances. While running these experiments on a much larger amount of words may have painted a cleared picture of the optimal set-up, I decided it would be sufficient to apply the alterations from experiment 3 to the final application.

Chapter 10: Conclusions and Future Work

Although I am satisfied with the final application and the means by which it finds phonetically similar words, there are a number of features I did not have the opportunity to implement. Although I wanted to provide an option to rate the ranked list and store the rating in a local database, I had to accept that the program would not be properly evaluated by real users, and so this feature was never realized. Other user interface features that were not implemented include an option to filter out nonsense words when generating the list, the mutation of only the beginning and endings of syllables, and an option save the list as a json file.

The final version of the application produces similar words by taking each syllable sequentially to create a new word with one modified syllable. While this works well for single syllable words, it means that words with at least one edit in two or more syllables are not found. Additionally, although similar sounding words tend to have the same number of syllables, alternative implementations like RhymeZone have shown that there are cases where a similar word has fewer or more syllables than the target (e.g. "rocked" and "rock-et"). Overall while some of these planned specifications were not realized, I feel like I have learned a lot about developing python web applications, having begun with no UI experience.

Finally, the component for calculating the distance for each possible word proved to be a bottleneck during the program execution. While I did experiment with alternative ways to calculate phonetic distance (specifically converting each word to a 4-character SOUNDEX representation and adding the distances for each character), the project specification required that these distances be derived from a predetermined distance metric (i.e. distance matrix). While finding a solution to the bottleneck exceeded the scope of this project, it is a problem I hope to return to in the future.

Code Repository: `github.com/FionanByrne/SimilarSpeak`

Bibliography

1. *An Introduction to Phonology* https://www.linguisticsnetwork.com/an-introduction-to-phonology/. (accessed: 09.11.2019).
2. Renukadevi, D. The Role of Listening in Language Acquisition; the Challenges & Strategies in Teaching Listening. *International Journal of Education and Information Studies* **4,** 59–63 (2014).
3. Bailey, T. M. & Hahn, U. Phoneme similarity and confusability. *Journal of Memory and Language,* 339–362 (2005).
4. Yiu, T. *The Curse of Dimensionality* https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e.
5. Bradford, A. *What Is Synesthesia?* https://www.livescience.com/60707-what-is-synesthesia.html.
6. Gallagher, G. & Graff, P. The role of similarity in phonology. *Lingua* **122.** Phonological Similarity, 107–111. ISSN: 0024-3841. http://www.sciencedirect.com/science/article/pii/S0024384111002221 (2012).
7. O'Neill, E. & Carson-Berndsen, J. The effect of phoneme distribution on perceptual similarity in English. *Interspeech 2019,* 1941–1945 (2019).
8. Scharenborg, O., Tiesmeyer, S., Hasegawa-Johnson, M. A. & Dehak, N. Visualizing phoneme category adaptation in deep neural networks. *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech,* 1482–1486 (2018).
9. Becker, R. & Lopatto, E. *Yanny or Laurel? The science behind the audio version of The Dress* https://www.theverge.com/2018/5/15/17358136/yanny-laurel-the-dress-audio-illusion-frequency-sound-perception.
10. Cutler A, e. a. Patterns of English phoneme confusions by native and non-native listeners. *The journal of Acoustical Society of America* **116,** 3668–3678 (2004).
11. *RhymeZone: rhyming dictionary and thesaurus* https://www.rhymezone.com. (accessed: 09.11.2019).
12. *CMU Pronouncing Dictionary* http://www.speech.cs.cmu.edu/cgi-bin/cmudict. (accessed: 09.11.2019).
13. *The BK-Tree – A Data Structure for Spell Checking* https://nullwords.wordpress.com/2013/03/13/the-bk-tree-a-data-structure-for-spell-checking/. (accessesd: 09.13.2019).
14. Nam, E. *Understanding the Levenshtein Distance Equation for Beginners* https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0. (accessed: 09.11.2019).
15. Kennedy, P. *Overview of Model-View-Controller (MVC)* 2015. https://www.patricksoftwareblog.com/overview-of-model-view-controller-mvc/.
16. Hixon, B., Scheider, E. & Epstein, S. L. Phonemic Similarity Metrics to Compare Pronunciation Methods. *Interspeech 2011* (2011).