Identifying Assonant Clusters in Poetry

Steve Nathaniel and Samvat Rastogi

Professor Dickinson

Computational Linguistics

2 May 2017

Contents

Table of Figures

Introduction

This project investigates sound patterns in poetry, and specifically it attempts to develop a tool for identifying assonant clusters. While some research in computational linguistics has been directed toward metrical measurements, which consist of syllabic stress patterns, other discernable concentrations of consonance, assonance, alliteration, or rhyme are of interest to analysts of poetry. These have largely gone unstudied.

Stepping beyond simple binaries of stressed and unstressed syllables multiplies the challenges of this endeavor. In the case of stress, most software can draw on phonetic dictionaries that mark stress and consequently most of the computational challenges lie in attaching a time signature to a poem. In the case of assonance, vowels have more than one sound quality; for example, in the IPA vowel chart there are 39 characters placed along two axes according to "closeness" and "backness."

One might begin by counting recurrences of a specific sound, such as /æ/, or further by identifying patterns such as clusters in a given sound. However, this level of specificity discriminates between sounds that are qualitatively similar, for example a labio-dental fricative /f/ and labio-dental approximate /v/. To analyze consonant clusters, then, it becomes necessary to measure the sonic proximity of consonants as well as their qualitative properties (such as place and manner), and then to identify locales of recurrence.

To limit the scope of this challenging inquiry, we focus on assonance. To further simplify the project, we assume that the qualities of closeness and backness adequately distinguish individual vowel sounds. Further we assume the IPA's vowel diagram to parse these sonic qualities evenly and consequently to facilitate regular Cartesian quantification. Stated otherwise, a vowel sound may be represented by a coordinate {a, b} in which a and b are numbers pertaining to closeness and backness respectively. In effect, the similarity of non-equivalent vowel sounds may be measured. Below we can see how these vowel sounds might be quantified according to their coordinate position between the open-front vowels, {1, 1}, in the bottom left corner of the chart and the close-back vowels, {7, 7}, in the upper right corner of the chart.

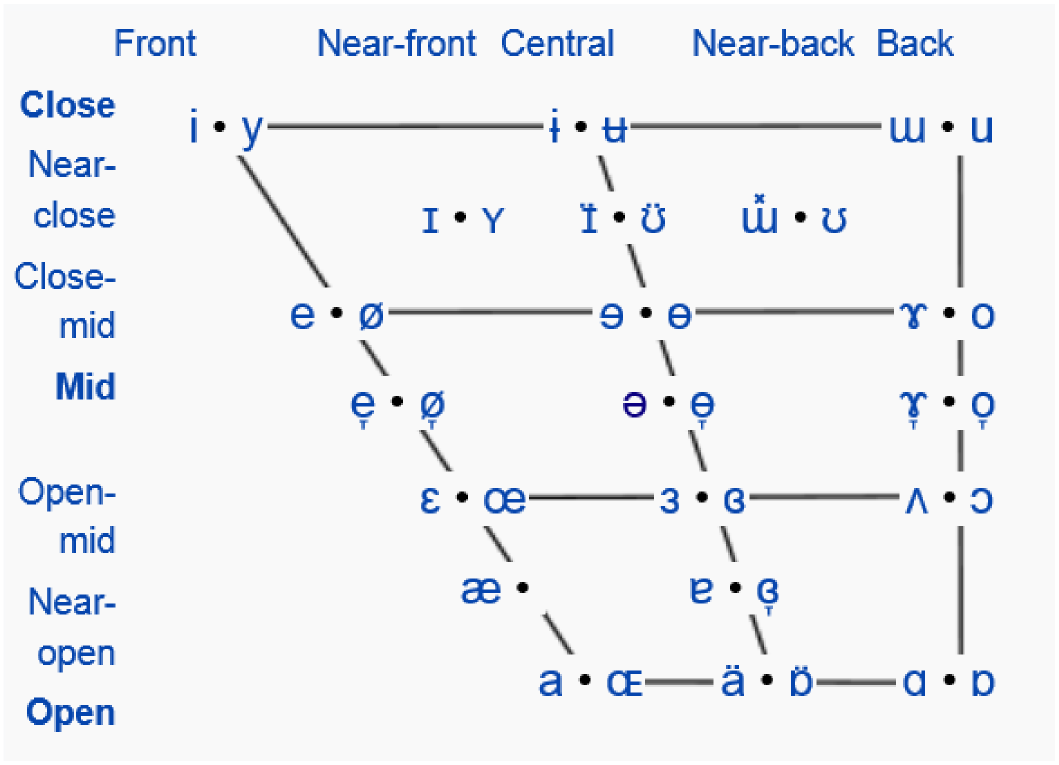

Figure 1: IPA Vowel Chart

Literature Review

Topic

The question of assonance in poetry involves a series of inquiries that each present unique challenges. Most scholarship that pertains to poetry involves rhythmic analyses of stress, whereas scholarship pertaining to assonance is generally interested in diction and dialect. Accordingly, our project inhabits a largely untapped region where poetic formalism and phonetics intersect. We attempted to examine a variety of scholarship in order to establish our project's academic geography.

Assonance

As we investigated the topics of assonance and poetry, we felt affirmed in that our project fills a research gap. Stated otherwise, very few people are doing research on either topic. To substantiate that claim, I note that the ACL database includes no papers that report an interest in assonance and only five that consider rhyme. One of these proved a valuable vision into what work has been done with topics related to assonance: rhyme and meter. Dmitriy Genzel, Jakob Uszkoreit, and Franz Och, in "'Poetic' Statistical Machine Translation: Rhyme and Meter" (2010), mainly concern themselves with the question of translating poetry, that describe the challenges implicit in line breaks, meters, and rhymes. This overview prepared us for the daunting set of tasks that our project includes

Poetry

The first article that truly seemed to address the problem that we had in mind was Nina McCurdy's, Vivek Srikumar's, and Miriah Meyer's (2015) "RhymeDesign: A Tool for Analyzing Sonic Devices in Poetry." In their article they describe the utility of their sound-analysis tool, which they designed specifically for poetry. Their stated purpose is to "supported by standard text analysis techniques. We introduce a formalism for analyzing sonic devices in poetry. In addition, we present RhymeDesign, an open-source implementation of our formalism, through which poets and poetry scholars can explore their individual notion of rhyme" (12). Initially, their project seems to encompass (or perhaps exclude the need for) ours, but as we studied their program, we found that our purposes were related, but not wholly competitive. However, their study does cast a broad vision for what sound patterns could involve:

> We classify these sonic patterns, which we define as instances of sonic devices, into four distinct types of rhyme: sonic rhyme involves the pronunciations of words; phonetic rhyme associates the articulatory properties of speech sound production, such as the location of the tongue in relation to the lips; visual rhyme relates words that look similar, such as cough and bough, whether or not they sound alike; and structural rhyme links words through their sequence of consonants and vowels. (12)

As we studied the methods applied through RhymeDesign, we began to see opportunities for our research to cover new territory. Notably, RhymeDesign was applied largely to analyze corpora of poems and answer generic questions. For example, the designers study the most common stress patterns on the poems identified on Wikipedia. However, such inquiries posited from the digital humanities do not often serve as a useful supplement to stricter adherents of the discipline of literary studies. While RhymeDesign tool is versatile and could be implemented for purposes of analyzing sound patterns in individual texts, its breadth actually stands as an impediment to its implementation as a tool to aid critical inquiry. We intend to produce a tool that offers a concise set of utilities catered to the research interests of contemporary critics. Accordingly, we will adopt a philosophy of specificity rather than possibility and thereby produce a tool that could find an eager audience.

I also note that RhymeDesign is hardly the only software of its type. It is preceded by PoemViewer (2013) and Prose Vis (2012), to name a pair, which offer visualizations of the patterns we are interested in. However, like RhymeDesign, these tools stem from an exuberant interest in what can be analyzed rather than what is useful to critics. By identifying a few specific assonant clusters, we hope to break outside the generic questions that have restricted the sense of sound that the inquiries of poetry critics have largely superseded. That being said, we intend to draw heavily on the work that has been done by these scholars.

Method

A second effort in our research involved a search for specific routines and algorithms that would streamline our efforts. In large part, our tasks were either too elementary or too obscure to feature in contemporary scholarship. For example, a simple English syllabification rule hierarchy seemed to us a fundamental kind of tool. In fact, it is too general to be considered in contemporary research. We would likely have more success if we consulted an introductory morphology textbook, but we chose instead to implement a home-grown rule hierarchy that served our purposes adequately.

Syllabification

Syllabification was a minor interest of our project, so we attempted to avoid becoming mired in obscure distinctions. Specific syllabic boundaries are not significant to long-term phonetic analyses, so we were looking for an efficient and elegant hierarchy of rules. Instead, we found abstruse inquiries into minor distinctions in marginal dialects and in abnormal contexts. One more straightforward inquiry was Karin Müller's (2006) "Improving Syllabification Models with Phonotactic Knowledge." She recommends preliminary morphological delineations that eliminate many ambiguities in typical parsing scenarios. While hers seemed a clear and cogent argument it would have added distracting trappings to a minor stage of our project to make morphological segments. Instead, we focused on patterns of consonants and vowels.

Susan Bartlett, Grzegorz Kondrak, and Colin Cherry's (2008) "Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion" actually includes a lot of basic terminology in its introduction that was helpful to recognize syllabification conventions. Our task, we realized, could be referred to as "hyphenation." This was not revelation per se, but we did gradually become more competent with the task by adopting the proper jargon. The article was especially useful as we imagine a method for analyzing sound in terms of rhythmic patterns rather than just clusters. The authors of the study recognize what we have: the hardest part of rhythm is imperfect conformity, which requires analysis of elements according to a full scale of conformity rather than a binary, yes or no.

Phonetic Translation

We drew on an IPA translator offered by Michael G. Phillips, a student at Syracuse University. While his method was one of many available on the vast and largely undifferentiated database of the internet, we selected Phillips' because we favored the Carnegie Melon phonetic dictionary that he draws on. We noted that this particular dictionary was featured often in contemporary phonetics conversations and thus we established its reputability. Phillips' website may be accessed here: https://github.com/mphilli/English_to_IPA. We adopted his python script and modified it for our purposes. Notably, his script includes some pre-processing that was useful, such as recognizing words without corresponding entries in the dictionary.

Project Outline

To research, develop, and implement this analytical tool, we propose the following project:

1. We select appropriate software.
2. We select a text of interest for analysis.
3. We replace a poetical text with its phonetic equivalent.
4. We identify consonants, vowels, and pauses—“symbol functions.”
5. We syllabify phonetic words.
6. We augment the existing table with a time signature pertaining to the sequence of syllables and pauses.
7. Vowels are mapped (quantitatively) to phonetic qualities.
8. A cluster analysis recognizes major correspondences of sound-positions. We will refer to these clusters as “resonant-clusters.”
9. Below we demonstrate the analytical process for a short line from Longfellow’s “Paul Revere’s Ride.”

    a. masses and moving shapes of shade — *Original Text*

    b. ˈmæsɪz ænd ˈmuːvɪŋ ʃeɪps ɒv ʃeɪd — *Phonetic Transcription*

    c. mæsɪz ænd muvɪŋ ʃeɪps ɒv ʃeɪd — *Cleaned (note: spaces remain)*

    d. mæsɪz ænd muvɪŋ ʃeɪps ɒv ʃeɪd
    cvcvcsvccscvcvcscvvccsvcscvvc — *Symbol Functions (c, v, s, p)*

    e. mæs ɪz ænd muv ɪŋ ʃeɪps ɒv ʃeɪd
    cvc vc vcc cvc vc cvvcc vc cvvc — *Syllabification*

    f. mæs ɪz ænd muv ɪŋ ʃeɪps ɒv ʃeɪd
    cvc vc vcc cvc vc cvvcc vc cvvc
    111 22 333 444 55 66666 77 8888 — *Time Signature*

    g. mæs ɪz ænd muv ɪŋ ʃeɪps ɒv ʃeɪd
    cvc vc vcc cvc vc cvvcc vc cvvc

```
111  22 333  444 55 66666 77 8888
x1x  2x 1xx x5x 2x x12xx 5x x12x            Backness
x2x  6x 2xx x7x 6x x47xx 1x x46x            Closeness
```

10. We calculate resonant clusters over this set of vowels.

11. We print out the poem with clusters marked.

```
a.  masses and moving shapes of shade
    mæsɪz ænd muvɪŋ ʃeɪps ɒv ʃeɪd
         ^            ^    ^         ^
```

Method

Please visit our website to investigate the unabridged method. Much of the data recorded here is too lengthy to reproduce here. https://samvat.github.io/assonant-clustering/

1. Import

We selected a popular text, “The Love Song of J. Alfred Prufrock,” to analyze, which, nevertheless, offered a number of challenges. The text had 3646 individual characters, not including spaces, which gave us enough raw data to implement a variety of techniques. We began by importing the data from the kind of text file in which we might expect to receive a poem. This preliminary step might require some minor modification depending on the intended mode of analysis. For example if this assonant analysis was linked to meter analysis software, some of the initial cleaning might already have taken place.

```
In [1]:
import nltk
import pandas as pd
In [2]:
f = open("Dataset.txt", encoding="utf8")
text = str(f.read())
print(text)

    Let us go then, you and I,
    When the evening is spread out against the sky
    Like a patient etherized upon a table;
```

2. Clean

Again, this process might depend on the state of the data. Many notable poetry texts are available through a variety of databases, such as poetryfoundation.org. These are transcribed rather than scanned and do not require considerations for copy fidelity, for instance. However, even texts accessed on such databases may include footnotes, unusual pagination, or antiquated spellings that could cause problems in cleaning as well as in subsequent stages of analysis.

```
In [3]:
#removed all punctuations
words = nltk.word_tokenize(text)
words=[word.lower() for word in words if word.isalpha()]
```

```
In [3]:
#removed all punctuations
words = nltk.word_tokenize(text)
```

```
words=[word.lower() for word in words if word.isalpha()]
```

```
['let', 'us', 'go', 'then', 'you', 'and', 'i', 'when', 'the', 'evening', 'is', 'spread',
'out', 'against', 'the', 'sky', 'like', 'a', 'patient', 'etherized', 'upon', 'a', 'table'
```

3. Phonetic Translation

As anticipated in our preliminary outline, many phonetic transcription algorithms exist and we were able to adopt one to our purposes. We found this particular transcriber effective because it recognizes a variety of potential aberrations. Notably, this algorithm selects from among various phonetic spellings with subtlety whereas others could not tolerate diverse spellings.

```
#Adopted from Michael G. Phillips (https://github.com/mphilli/English_to_IPA)
```

```
'lɛt əs goʊ ðɛn ju ənd aɪ hwɪn ðə ivnɪŋ ɪz sprɛd aʊt əgenst ðə skaɪ laɪk e
peʃənt iθəraɪzd əpɑn e tebəl
```

4. Symbol Function

To implement the various symbol functions, we used a simple transducer that drew on IPA consonant and vowel lexicons. Here, we retain spaces as unique characters to be replaced later as string delineations. Although some metrical parsers would apply a minor time signature to spaces, we ignore them according to more typical scansion practices.

```
In [7]:
sounds = list(ipa)
tags = ''
for sound in sounds:
   if sound in sound_tags:
      tags += sound_tags[sound]
```

```
'cvcsvcscvvscvcscvsvccsvvsccvcscvsvccvcsvcscccvcsvvcsvcvcccscvsccvv
scvvcsvscvcvccsvcvcvvccsvcvcsvscvcvcscvcsvcscvvsccvscvccvcsccc
```

5. Syllable Parsing

We do not make note of the intermediate step mentioned above. We parse words before parsing syllables, and the former step involves a basic transducer. Syllable parsing proved challenging because our initial algorithms parsed word by word and in a non-recursive manner. Consequently, words such as /iθəraɪzd/, which contains three syllables, would not be fully parsed. Surprisingly, we decided to implement our home-grown syllabification hierarchy because most contemporary linguistic scholarship concerned more esoteric parsing questions. Although many methods assumed a baseline English syllabification method, they focused on marginal dictions or dialects which were actually too finely-nuanced for our broader purposes. Fortunately, questions of syllabification are trivial to the question of assonance because at worst a syllable's signature is off by a beat, which is not likely to distort the cluster effect.

```
In [9]:
rules = {"vccccv":"vcc*ccv","vcccv":"vc*ccv","vccv":"vc*cv","vcvc":"vc*vc",
       "cvcv":"cv*cv","cvvcv":"cvv*cv","cvvvcv":"cvvv*cv"}
rules

    Out[9]:
    {'cvcv': 'cv*cv',
     'cvvcv': 'cvv*cv',
     'cvvvcv': 'cvvv*cv',
     'vccccv': 'vcc*ccv',
     'vcccv': 'vc*ccv',
```

6. Time Signature

The challenge we ran into was parsing both the letter function string and the phonetic symbol string at the same time. We ended up counting the positions of the syllable breaks in the letter function string ("tags"), inserting breaks in the IPA at the same location, then splitting the strings accordingly. To account for words with various syllable breaks we applied this operation recursively.

Once both strings, IPA and tags, were parsed syllabically, the time signature was simply their location in the array.

```
In [15]:
for i, row in dataf_4.iterrows():
      dataf_4.loc[i, 'tags'] = sound_tags[dataf_4.loc[i,'ipa']]

dataf_4.index = dataf_4.index+1
dataf_4
```

7. Quality

Quality proved significantly easier to implement. First we broke syllables up into individual characters with both IPA and tag elements. A simple transducer then allowed us to generate new arrays by replacing all IPA elements with place and manner according to their IPA chart locations.

```
In [16]:
#Closeness and Backness calculators FST
```

```
closeness  =
{"i":"1","y":"1","e":"1","ø":"1","ḛ":"1","ø̰":"1","ɛ":"1","œ":"1",
            "æ":"1","a":"1","ɶ":"1","ɪ":"2","ʏ":"2","ɨ":"3","ʉ":"3","ɪ̈":"3",
            "ʊ̈":"3","ɘ":"3","ɵ":"3","ə":"3","ɜ":"3","ɞ":"3","ɐ":"3","ä":"3",
            "ʊ":"4","ɯ":"5","u":"5","ɤ":"5","o":"5","ɤ̰":"5","o̰":"5","ʌ":"5",
            "ɔ":"5","ɑ":"5","ɒ":"5"}

backness =
{"i":"7","y":"7","e":"5","ø":"5","ḛ":"4","ø̰":"4","ɛ":"3","œ":"3",
            "æ":"2","a":"1","ɶ":"1","ɪ":"6","ʏ":"6","ɨ":"7","ʉ":"7","ɪ̈":"6",
            "ʊ̈":"6","ɘ":"5","ɵ":"5","ə":"4","ɜ":"3","ɞ":"3","ɐ":"2","ä":"1",
            "ʊ":"6","ɯ":"7","u":"7","ɤ":"5","o":"5","ɤ̰":"4","o̰":"4","ʌ":"3",
            "ɔ":"3","ɑ":"1","ɒ":"1"}
In [17]:
#Calculating backness and closeness for each vowel sound
for i, row in dataf_4.iterrows():
    for closescore in closeness:
        if dataf_4.loc[i, 'tags'] == 'v':
            if dataf_4.loc[i, 'ipa'] == closescore:
                dataf_4.loc[i, 'closeness'] = closeness[closescore]
    for backscore in backness:
        if dataf_4.loc[i, 'tags'] == 'v':
            if dataf_4.loc[i, 'ipa'] == backscore:
                dataf_4.loc[i, 'backness'] = backness[backscore]
```

	ipa	tags	times_signature	closeness	backness
1	l	c	1	NaN	NaN
2	ɛ	v	1	1	3
3	t	c	1	NaN	NaN
4	j	c	2	NaN	NaN
5	u	v	2	5	7
6	ɛ	v	2	1	3
7	s	c	2	NaN	NaN
8	g	c	3	NaN	NaN

We then cut all non-vowels to create a table that pertains exclusively to the assonant elements.

```
dataf_5 = pd.DataFrame()
dataf_5 = dataf_4.loc[dataf_4['tags']=='v']
dataf_5 = dataf_5.reset_index(drop=True)
dataf_5.index += 1
dataf_5
```

	ipa	tags	times_signature	closeness	backness
1	ɛ	v	1	1	3
2	u	v	2	5	7
3	ɛ	v	2	1	3
4	o	v	3	5	5
5	ʊ	v	3	4	6
6	ɛ	v	4	1	3
7	u	v	5	5	7
8	æ	v	6	1	2

8. Cluster

The final step was to identify clusters in this data. Initially we planned to use one of a number of clustering algorithms common to data analysis. However, the algorithms that we identified established particular assonant loci and then identified proximate sounds. This means that we would have to presuppose a specific sound that predominates, thus defeating the purpose of the tool, to identify such clusters for its users. Instead, we used a tool that measured each vowel's sound vector (time, closeness, backness) against its neighbors' sound vector, a difference that we refer to as resonance. The data set we produced is an nxn table in which n is the number of vowels in our data set. We then measured the relative resonance of each of the n elements by summing its various resonances. High summations indicated highly resonant elements.

```
#Datatype conversion from str to int for time signature, backness and
closeness
dataf_5['times_signature'] = dataf_5['times_signature'].astype('int')
dataf_5['closeness'] = dataf_5['closeness'].astype('int')
dataf_5['backness'] = dataf_5['backness'].astype('int')
In [20]:
score_vector=[]
for r in zip(dataf_5['ipa'], dataf_5['times_signature'], dataf_5['closeness'],
dataf_5['backness']):
    for s in zip(dataf_5['ipa'], dataf_5['times_signature'],
dataf_5['closeness'], dataf_5['backness']):
        score_vector.append((r[0],s[0],r[1]-s[1],r[2]-s[2],r[3]-s[3]))
score_vector
```

The selection below identifies a number of vector differences (resonances) between the vowel "e" in the poem's first word, "let."

```
[('ɛ', 'ɛ', 0, 0, 0),
 ('ɛ', 'u', -1, -4, -4),
 ('ɛ', 'ɛ', -1, 0, 0),
('ɛ', 'o', -2, -4, -2),
('ɛ', 'ʊ', -2, -3, -3),
('ɛ', 'ɛ', -3, 0, 0),
('ɛ', 'u', -4, -4, -4),
('ɛ', 'æ', -5, 0, 1),
('ɛ', 'a', -6, 0, 2),
('ɛ', 'ɪ', -6, -1, -3),
('ɛ', 'ɪ', -7, -1, -3),
('ɛ', 'i', -8, 0, -4),
('ɛ', 'i', -9, 0, -4),
```

Below, the differences of these vectors is calculated as a single magnitude. As we can see, these values generally increase as they fall later in the poem. However, qualitatively similar vowels do accomplish resonances in spite of this temporal displacement. I note that the values of time, backness, and closeness are equally weighted, and in future iterations of this tool, we will certainly normalize these and identify more appropriate weights--or else leave this to the user’s discretion.

```
('ɛ', 'ɛ', 1.0),
 ('ɛ', 'o', 4.898979485566356),
 ('ɛ', 'ʊ', 4.69041575982343),
 ('ɛ', 'ɛ', 3.0),
 ('ɛ', 'u', 6.928203230275509),
 ('ɛ', 'æ', 5.0990195135927845),
 ('ɛ', 'a', 6.324555320336759),
 ('ɛ', 'ɪ', 6.782329983125268),
 ('ɛ', 'ɪ', 7.681145747868608),
 ('ɛ', 'i', 8.94427190999916),
 ('ɛ', 'i', 9.848857801796104),
```

Results and Future Research

After the difficulties of largely building our code from the ground up, we are pleased to report that we have formulated a tool that already demonstrates useful utilities for the poetry critic. We are surprised to have arrived at our proposed destination in spite of numerous conceptual and methodological challenges along the way. However, this is not to say that we are content with what we have produced. While we are excited by the results that we are already seeing, we are equally eager to attend to the tool’s open ends.

1. There are a number of questions and ambiguities that our code leaves unattended. We list these below in no particular order.
    a. How does stress play into assonance? Should stressed syllables be weighted more heavily in calculations of resonance?
    b. What challenges would arise in translating this tool to accommodate consonant analyses as well? Are the qualities of place and manner, like backness and closeness, comprehensive and quantifiable?
    c. How well could this tool be integrated with existing metrical analysis tools? Could we generate a report that described meter, consonance, and assonance?
    d. What weights are most appropriate to apply to each element in the sounds’ vectors (closeness, backness, and position)? Or, if they were left open to a user at the beginning of an analysis, what kinds of guidance might a user receive in this decision so as to produce meaningful data?
    e. Further, what kind of interface is appropriate to a potential user of this tool? If a variety of results are eventually possible, including analyses of consonance and rhythm, how might a user select from among these?
    f. How successful is the quantification of backnesss and closeness at measuring the sonic similitude of vowels? Are there other, more precise measures of what we have called resonance?
    g. Diphthongs present a challenge to this method. In the case of repeated diphthongs, the tool might recognize two distinct assonant clusters, which perhaps should be recognized as one.

Finally, we can offer a vision of the tool that this could become. Currently, we have not packaged the code such that it would be conducive to a user unfamiliar with the python interface. Ideally, we would enact a series of command prompts that would offer the user options. These might include the following:

1. “Input text:”
2. “We could not identify the following words. Please identify alternatives that best maintain the sound quality of the text:
    teacups:
    braceleted:
    malingers:”
3. “Would you like to analyze consonance(1), assonance(2), or both (3)?:”
4. “Would you like to analyze clusters(1) or rhythms(2), or both(3)?”
5. “How many features per word should we analyze? We recommend between .05 and .005:”

6. “How would you like the weight the following qualities? A weight of 1 will cause a quality to be equally important as the others, while a weight of 2 will cause a quality to be twice as important as the others. The time signature is based on syllables:

closeness:
backness:
time:
place:
manner:

Once these inputs are established the user should receive a clear report that marks features pertaining to their selections. These would likely be marked diacritically, with different marks pertaining to unique features. A poem with, say, a dozen assonant clusters, might be marked alphabetically, with diacritical marks of “a,” “b,” “c,” and so on pertaining to cluster a, cluster b, cluster c, and so on. This would help distinguish intermingled clusters, which are not unlikely.

While we have tackled the challenge of assonance, the structures we have put in place could easily be adjusted to accept consonance analyses. In expanding the tool this would likely be the first direction we would take. While closeness and backness help to quantify the sound qualities of vowels, the IPA’s categories of place and manner would provide ideal substitutes for consonants. Further, most of the initial data-processing that such an analysis would require has already been taken care of by our existing code.

To cast an even more ambitious vision, we would like to supplement the clustering tool with a rhythm tool. In other words, we would like users to be able to choose between temporally limited and temporally regular patterns. In effect, we would be able to identify rhyme schemes or other metrically contingent patterns. The vector relationships that we have established in the assonance algorithm would be able to identify half- and slant-rhymes, which have proved an enduring difficulty to existing software.

Due to the success of this endeavor, we have determined to continue working on this tool during the summer to attempt to delve the potential we have seen. We would also like to inquire into potential opportunities to expand this project by drawing on the expertise of faculty from computer science, linguistics, and literature departments. With expert advisors, we believe that we could further develop the academic and pedagogical utility of the tool. This unique project and the unique collaboration that produced it may yet prove a valuable contribution to research in poetics, linguistics, and phonetics.

Works Cited

Bartlett, Susan, Grzegorz Kondrak, and Colin Cherry. 2008. “Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion.” *Proceedings of ACL-08: HLT. Association for Computational Linguistics*. pgs. 568-576.

Genzel, Dmitriy, Jakob Uszkoreit, and Franz Och. 2010. “‘Poetic’ Statistical Machine Translation: Rhyme and Meter" Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics. Pgs. 158-166.

Phillips, Michael G. “English_to_IPA.” Personal Website. https://github.com/mphilli/English_to_IPA. Accessed 28 April 2017.

McCurdy, Nina, Vivek Srikumar, and Miriah Meyer. 2015. "RhymeDesign: A Tool for Analyzing Sonic Devices in Poetry." Proceedings of the Fourth Workshop on Computational Linguistics for Literature. Association for Computational Linguistics. Pgs. 12--22.

Müller, Karin. 2006. "Improving Syllabification Models with Phonotactic Knowledge.” *Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology and Morphology at HLT-NAACL 2006*. Association for Computational Linguistics, pages 11-20.