April 10, 2016
LIN477
Arnold Rosenbloom
Luke Sawczak

Odeon

The Prose to Poetry Converter
v 0.2

Overview of the project

With Odeon,[1] you can turn any prose into poetry.

The basic philosophy of Odeon is that literary content can be separated from literary genre. Meaningful prose can also be meaningful poetry. All that needs to change is the external form.

The particulars of the form, however, are a very deep source of potential research at the interface of all three of Odeon's components: literature, linguistics, and computer science. (Because these three are close to my heart, the overlap was a compelling project.)

In terms of literature, we must investigate what poetry is, a problem not only recognized as rich for the first few millennia of its existence but particularly questioned in the last century, in which the rise of free verse and other atypical forms was rapid. What is the difference between poetry and prose with line breaks? We might begin with a short list of rhyme, meter, and word choice, and already we have several very complex topics to deal with.

Linguistics becomes heavily involved in each of these concepts, because each is ultimately rooted in several linguistic branches. Rhyme involves phonetics, phonotactics, and phonology. Meter involves phonology, prosody, and semantics. Word choice involves semantics, syntax, and pragmatics. As the field of linguistics develops—and we must remember that it has had a strictly scientific bent for less than a century—we are empowered to explore whether the previously tacit knowledge of poetic concepts can be described generatively.

In order to do this in the context of a computer program, we must make extensive use of natural language tools, including the NLTK, well-selected corpora, a pronunciation dictionary, and a thesaurus, as well as a robust environment in which to use them. By representing poetic concepts algorithmically and presenting a wide range of programmable options,[2] we enable the rapid and powerful exploration of many different operations on natural language. Because of its range, part of the delight of using Odeon is simply discovering the vast number of new poems that can be come up with that a human might not have considered.

---

[1] An ancient Greek building used for "singing exercises, musical shows, poetry competitions, and the like" (Wikipedia: Odeon). From Greek ἀείδω "I sing". This verb is also the root of "ode".
[2] The current set of options has some 250 billion combinations.

Results

Here are a few sample poems produced from prose in various configurations.

A: Prose

"Not a nasty, dirty, wet hole, filled with the ends of worms and an oozy smell, nor yet a dry, bare, sandy hole with nothing in it to sit down on or to eat."
— *The Hobbit*, J.R.R. Tolkien

A: Poem 1

Not a nasty, dirty, wet hole,
Filled with the ends of worms
And an oozy smell, nor yet a dry, bare,
Sandy hole with nothing in it to sit
Down on or to eat.

A: Poem 2

Not a nasty, dirty, wet
Hole, filled with the ends
Of worms and an oozy smell, nor yet
A dry, bare, sandy hole with nothing in it to sit
Down on or to eat.

A: Analysis

This prose has few perfect rhymes, resulting in a rhythm-heavy poem in the first configuration. Notice that the lines all have a similar number of stresses, with the exception of the orphaned last line, resulting in a fairly pleasant reading out loud. In a second configuration, a high degree of slant rhyme is allowed, which permits "wet, "yet", "sit", and "eat" to be recognized as rhymes, and hence we get a rhyme-heavy poem.

B: Prose

"For six years now he had been a man's hunter. For six years now he had heard the best of all talking. It was of the wilderness, the big woods, bigger and older than any recorded document."
— "The Bear", William Faulkner

B: Poem

six years man's hunter six years
heard best talking wilderness large woods
bigger and older recorded document

B: Analysis

In this configuration I have enabled punctuation and stopword removal,[3] lowercasing, and some synonymization. As a result, we get an abstract rendering yielding numerous interpretations. This style is also useful for users who are only looking for inspiration, rather than completed poems.

C: Prose

"Shall I compare thee to a summer's day? Thou art more lovely and more temperate. Rough winds do shake the darling buds of May, And summer's lease hath all too short a date. Sometimes too hot the eye of heaven shines, And often is his gold complexion dimmed; And every fair from fair sometime declines, By chance, or nature's changing course, untrimmed; But thy eternal summer shall not fade, Nor lose possession of that fair thou owes, Nor shall death brag thou wand'rest in his shade, When in eternal lines to Time thou grows. So long as men can breathe, or eyes can see, So long lives this, and this gives life to thee."
— "Sonnet 18", William Shakespeare, disguised as prose

C: Poem

Shall I
Compare thee
To a summer's day?
Thou art more lovely and more temperate. Rough winds do shake the darling buds of May,
And summer's lease hath all too short a date. Sometimes too hot the eye
Of heaven shines,
And often is his gold complexion dimmed; And every fair from fair sometime declines,
By chance, or nature's changing course, untrimmed; But thy
Eternal summer shall not fade,
Nor lose possession of that fair thou owes,
Nor shall death brag thou wand'rest in his shade,
When in eternal lines
To Time thou grows.
So long as men can breathe, or eyes can see,
So long lives this, and this gives life to thee.

C: Analysis

Here only rhyme is allowed, and it has done an interesting job of re-realizing Shakespeare's most famous sonnet. Since our program has a keen eye for rhyme, it actually finds more line breaks to make than Shakespeare chose to make. In fact, any long, poetic text is usually rich with internal rhyme and produces many possible poetic variations—if we continued to look through the vast number that are generated from Sonnet 18, we would likely find Sonnet 18 recreated.[4]

[3] As described below, since custom stopwords are defined, 'and' survives when it would not normally in the NLTK.
[4] As a caveat, since Shakespeare rhymes 'intemperate' with 'date', we would not find it without using slant rhyme.

Running

Odeon has been designed with users in mind, so running it is very straightforward.

The user will need to have Python 3 and the NLTK installed.[5]

You can run the script directly using the Python launcher, or from a command line by invoking "Odeon" with Python in the relevant directory. It is possible to supply command line arguments to provide a text in advance, or to use demo mode only.

The full list of commands is as follows:

Commands

Short name	**Long name**	**Purpose**	**Example**
-h	--help	Show command usage	python Odeon -h
-d	--demo	Run predefined demos	python Odeon -d
-t <text>	--text <text>	Supply literal text	python Odeon -t "Cats eat bats"
-f <filename>	--file <filename>	Supply file with text	python Odeon -f greatprose.txt

For example, the following command supplies the name of a file from which to read text:

python Odeon -f <filename>[6]

Note that you can only supply one of --demo, --text, or --file. If you do not supply any of the three at the command line, Odeon will prompt you for a text when it starts up.

After a text has been chosen, you will be asked to pick options. There are a number of presets included to select from. If you wish to customize the options further, you can go through them one by one. They are all clearly described for easy comprehension with no prior experience.

The program will load the resources it needs, process the text, and generate the poems. Then it will present poems to you one by one. Hit "Enter" to see each new poem or type "q" to quit.

[5] Starting points for these resources are supplied in the "References" section.

[6] When supplying text or a filename at the command line, surround it with quotation marks if it includes spaces. This restriction does not apply to the in-program prompt.

How it works

There are four main technical components of Odeon:

1. The **controller**, responsible for getting the user's text selection and options, sending them to the manipulator, and displaying poems to the user.
2. The **manipulator**, responsible for processing the base text and sending it out to tools that find synonyms and break lines via rhyme and meter.
3. The **resources**, responsible for providing access to a pronunciation dictionary, a synonym dictionary (thesaurus), and prose and poetry corpora.
4. The **language package**, responsible for modelling poems, lines (verses), and words. All of these models draw heavily on a comprehensive syllable model.

The controller

The controller is able to source text from the user via either a literal string input, a filename containing a string, or a predefined set of demo texts.

Once it has this, it proceeds to ask the user to set various options, either via a preset configuration or a custom choice of options. These options consist of how to format the text, how to use rhyme, how to use meter, how to use synonyms, whether to remove stopwords, and which resources to use in terms of corpora and dictionaries. The options must be freely available to all modules across the program, since they are drawn on extensively.

Finally, the controller submits the text to the manipulator, retrieves the generated poems, and prints them for the user.

The controller package also defines a logger that simply saves important events with their timestamps.

The manipulator

The manipulator does two things when it receives its text: it processes it and it discovers variants.

The processing of the text mostly consists of removing unwanted characters (but note that most punctuation is handled by the program without needing to be removed). It also consists of removing certain stopwords if the user has chosen the "ellipsis" option.

The discovery of variants essentially means finding synonyms. Each word reports its synonyms, limiting them to those that are either rarer or more poetic than itself. A word's rarity is defined as the inverse of its frequency across both the prose and poetry corpora, and its poeticness is defined as its frequency across the poetry corpus less its frequency across the prose corpus: a positive poeticness indicates a word used more often by poets than non-poets.

The search for synonyms can be simple or complicated. Certain words have no synonyms, as in the case of function words, or no very different ones, as in the case of "thing". But others have a wide range of homonymic meanings, and these vary widely not only within a part of speech but especially when a word's part of speech is ambiguous. Our thesaurus implementation, based on the synsets in the NLTK's WordNet interface, uses NLTK POS-tagging on each isolated word to

make a rudimentary guess at its part of speech, in the hopes of identifying whether it's likely to be a noun, adjective, verb, or adverb (the categories that WN recognizes). There are also user-defined upper bounds on the number of homonyms to comb through for synonyms and on the number of synonyms themselves.

Once the initial list of alternatives for each word has been identified, we generate variants, i.e. the product of each alternative. Each variant will be used equally to extend the possible poems. Since any large text will have an exponentially large number of variants, it is also possible to select only the two that respectively score highest in poeticness and rarity.

When the manipulator is then asked to generate poems, it employs the core strategy of Odeon: breaking the stream of text into lines at appropriate places. Essentially, we have two means of doing so: rhyme and meter. In rhyme mode, we break lines to create end rhymes. In meter mode, we break lines to create metrical patterns (or at least consistent numbers of beats per line). When the two are combined, we can choose to operate with a "both and" or an "either or" mindset. The former yields poems that are more formally correct, but in smaller number.

The manipulator finally applies any requested filters, such as requiring a strict rhyme scheme of each poem, and is done its work.

The resources

There are three resources to draw on: the pronunciation dictionary, the synonym dictionary, and the poetry and prose corpora.

The pronunciation dictionary is perhaps easiest to model, since Carnegie Mellon University provides an excellent dictionary of some 160,000 entries, available through the NLTK. This dictionary is also possible to download, which I have done in order to slightly customize it.

The synonym dictionary also comes in a local and NLTK copy, but in this case the two employ different strategies. One is a direct mapping of some 16,000 words to synonyms,[7] and the other is a complex reading of the NLTK's WordNet synset interface, as described above, which provides more control and flexibility.

The poetry and prose corpora are sourced from Project Gutenberg, the prose indirectly through the NLTK and the poetry hand-picked according to two genres: classical (pre-1850 or so) and modern.[8] The prose corpus contains some 1 million words and the poetry about 1.25 million. Both were carefully cleaned, in particular because they do not go through Odeon's native text-processing but are entrusted to the NLTK's tokenizers, and hence must be free of noise. There is also a mechanism in place for users or developers to add genres of poetry and prose, which the program will automatically detect, as documented in the project's data folder.

These corpora are modelled by NLTK PlaintextCorpusReaders and FreqDists.

---

[7] Provided by Karo Castro-Wunsch, who cleaned data from a well-known thesaurus.

[8] Since Project Gutenberg only provides public domain texts, the "modern" genre really does refer to the modern era as it existed about a century ago, not more recent developments in today's poetry.

The language package

Perhaps the true heart of Odeon is the language model; all the components depend on the poem, the poem on the line, the line on the word, and the word on the syllable.

Syllables are modelled using four pieces of information: the stress level, the onset, the nucleus, and the coda. The first is present in the CMU dictionary, and the rest can be deduced by a correct application of English phonotactics.

Every word's pronunciation is a list of its syllables. Several strategies are also employed to make non-alphabetic words pronounceable. For numbers, we use an external library that can yield text versions of cardinals and ordinals, which can then be treat as individual words. Compound words with hyphens can have a joint pronunciation. Punctuation and words not found in our dictionary are treated as unpronounced.

Words are responsible for three things: reporting their syllables for use in meter; reporting their rhyming segments for use in rhyme; and reporting their synonyms for use in synonymization, as described above. When rhyming, words can either require an exact rhyme, or they can allow for a user-defined threshold of "slant rhyme", or difference. A simplified phonetics model, described below, is used for comparing the similarity of a given pair of pronunciations.

Lines are mainly containers for lists, but can report their end rhyme and their number of stresses.

Poems are containers for lines, but also contain a scoring system. A poem can score itself on various vectors, such as density of end rhymes and consistent use of meter. This strategy is used because we overgenerate poems, owing to the large number of word choice variants and the fact that we can create various arrangements of rhyme and meter by breaking lines in various places. When presenting poems to the user, therefore, we want to present the best ones first.

Project Lessons

What poetry is

On the surface, poetry can be very simply defined as “rhyme and rhythm”. But though these are indeed a large part of Odeon’s concern, it’s clear that not only are they non-trivial to model in themselves, they are also far from the whole story of poetry.

To give an overview of what that whole story is, we need only consider that poets regularly make use not only of end rhyme, but of slant rhyme, eye rhyme, and internal rhyme; not only of rhyme schemes, but of mixed rhyme schemes; not only of rhythm, but of formal feet in well-defined patterns, as well as mixed feet and mixed patterns; or, in the other direction, away from formal beat and purely to stress- or syllable-timing; and they use the interface between the two, such as the sonnet form, which has both a metric requirement (iambic pentameter) and a rhyme scheme requirement (ABABCDCDEFEFGG or variations). Besides this, they also make use of with other aspects of sound: alliteration, assonance, consonance, dissonance, intonation (when read aloud, or signalled by punctuation, blank space, etc.), and more.

Poetry also involves a strong semantic component, from word choice to ellipsis to anaphora to disguised or nonsensical passages. We might also posit (and indeed I think it fairly likely) that the whole pragmatics can change as well: what precisely the poet chooses to say and by what means (metaphor, symbolism, etc.) might be unique to the form. In this sense poetry is its own discursive genre, whence the inherent oddity of, for example, Alexander Pope’s “Essay on Man”: poetry is generally not used as an essay format.

For the purposes of Odeon, however, it is necessary to consider this aspect of poetry moot from a computational standpoint: it is still difficult to model meaning algorithmically, at least out of the scope of this project. Instead, I make the assumption that the poeticness of a poem’s content is a function of the poeticness of the input text, and avoid changing it in any fundamental way. The mantra is garbage in, garbage out; but by the same token, meaning in, meaning out.

Finally, poetry also involves peculiarities of formatting when printed. The use of whitespace includes the use of different indentation levels, stanza breaks, and the arrangement of text to form concrete shapes (on occasion) or purposefully to deter the eye. Punctuation can also be highly singular in poetry, such as Emily Dickinson’s use of the emdash —, as can be the strategic use of lowercasing and capitalization, such as in the work of e e cummings.

The definition I prefer is hence: “The formal aspect of poetry is linguistic play.” Whatever linguistic tool the mind can identify and configure, it will. This is why seemingly inanimate features of text, such as its arrangement on the page or its use of punctuation marks, can be put to the service of poetry, just as each concept in essentially every branch of linguistics can. Even the discursive form can be played with, resulting in locutions that appear to affect content as well.

Given this, how can we hope to model poetry until we can model language?

The answer is that we must use a reduced and simplified model (particularly for the scope of an undergraduate course project). Even with this simplification, though, it has been possible to make a few interesting observations in the development of Odeon’s model.

Pronunciation

The pronunciation of a word is not an easy thing to model. Its true depth is difficult to plumb, because every phonetic segment is simultaneously represented as an entity at the phonetic, phonological, and prosodic levels, and its representations at those levels are subject to higher abstractions: the stress of any segment, for example, is not innate but interacts with emphasis as given by semantic and pragmatic considerations.

There is also a great deal of variation. Even a very good pronunciation dictionary like the CMU one fails to account totally for dialectical differences. For example, in Canadian English we do not distinguish between 'AA' and 'AO' (/ɑ/ and / ɒ/ as marked in the CMU dictionary),[9] resulting in unintuitive failures to rhyme, such as between 'walk' and 'block'. The CMU dictionary offers a secondary pronunciation or 'block' to use 'AA', but not vice versa. I therefore flattened this distinction in the local "Canadianized" version I offer to users. But more importantly, no word is independent of others and the phonetic realization of the phonemes listed will vary by context. The phonological rules that determine this are complex and differ for every language. Many phenomena are impossible to represent on a word-by-word basis. For example, a French pronunciation dictionary would have to account for inter-word liaison.

In any case, the list of phonemes and syllables is an acceptable starting point and the only one available to me at the moment.

For certain uses of pronunciation, however, I had to model syllables more effectively. Syllables are themselves complex topics in linguistics and their structure a very active field of research and debate, but the basic model of onset, nucleus, and coda, the latter two of which make up what is called the rhyme, suited my purposes. In order to identify these groupings, which are not marked in the CMU dictionary, I had to draw on my linguistic knowledge and create a mapping of sounds to (a) the sounds that can follow them in an onset cluster and (b) the sounds that can precede them in a coda cluster.[10] I traverse the phonemes backwards in order to privilege onsets,[11] find nuclei, and then search forwards for a coda and backwards for an onset.

Interestingly, this representation only arose fairly late in the process, as part of what was to be a fairly minor aspect of the program: slant rhyme. Slant rhyme (or half-rhyme) is when two words do not strictly rhyme but are close enough to get away with in some contexts, such as 'get' and 'deaf' or 'always' and 'play'. Some genres, such as rap lyrics, make heavy use of slant rhyme. The question for the natural language analyst is: what makes two rhymes similar?

My answer is that the similarity of two rhyming portions is a function of the similarity of their segmental components. This raises a further question: what makes two segments[12] similar? Here we have a theoretically rich opportunity involving the entire theoretical model of the segment. The average English speaker might know that there are "vowels" and "consonants", and the language enthusiast that there are "plosives" and "sibilants" and "glides" and so forth. The

[9] Except when R-coloured, i.e. /ɑɹ/, as in 'car'.
[10] There is some room for theoretical improvement here, because these rules are not arbitrary but are defined (mostly) by the sonority of the consonants: in general, sonority must be higher closer to the nucleus.
[11] Some sequences, such as 'R K', are parsed as a coda if traversing forwards, since 'K' can follow 'R' in a coda, but as a separate coda and onset if traversing backwards, since 'R' cannot precede 'K' in an onset. The latter is desired.
[12] Used here in the phonetic sense, corresponding more or less to "phone", the realization of a phoneme.

linguist, however, breaks down every segment into its constituent features. Vowels are described by close/open (height of the tongue in the mouth), front/back (how far back the peak of the tongue is in the mouth), roundedness (of the lips), and, in some models, tense/laxness (of the tongue muscle). Consonants are described by voiced/unvoiced (the vibration of the glottis), place of articulation, and manner of articulation (both numerous). All segments are also described by their sonority, which leads some linguists even to place vowels and consonants on a continuum entailing no formal distinction between the two classes.

The question of what makes two segments similar, then, is a question first of whether they are commensurable and then of whether their comparison can be quantified. I chose to define vowels and consonants as incommensurable, but all vowels as commensurable with all other vowels (even diphthongs, or vowel clusters, represented as a single vowel in the CMU dictionary), and all consonants as commensurable with all other consonants. I defined a system of quantification for each.[13] Vowels' closeness, backness, and roundedness are each given a rating based on their relative position in the mouth. Consonants' voicedness, backness, and sonority are each given a similar rating. Each factor is then weighted, and the difference between two segments is thus 1.0 less the summed and weighted difference between their measures.

From this, we can extrapolate the similarity of clusters of consonants. We align clusters rather than individual consonants because when comparing, for example, 'bake' to 'brake', we ought to compare 'b' and 'br'. The similarity of clusters of equal length is trivial: it is the product of the similarity of their components. For the similarity of clusters of unequal length, such as 'str' and 'pr', a recursive algorithm matches various subclusters with different weights to arrive at an overall similarity. There is room for theoretical improvement here, because it might be possible to intelligently identify subclusters as more important to be matched to each other, such as the two 'r' sounds in the preceding example.

Finally, we need to identify which clusters to compare. It was while doing this and puzzling over the problem of aligning 'ache' and 'bake', which are the same after the onset and ought to be recognized as so, and of 'arcane' and 'cocaine', whose first two sounds must be flipped in order to compare the 'a' to the 'o', that it became clear that a syllable model was needed. Then I could compare onsets to onsets, nuclei to nuclei, and codas to codas, each at a predefined weight.

The pronunciation similarity engine, while used in Odeon only for slant rhyme, is therefore fairly powerful and theoretically informed. Minor adaptations could further improve it and produce applications in a broad range of contexts, including, for example, accent categorization.

Rhyme

In one sense, rhyme is easy to define: the rhyming segment is the series of phonemes from the last stressed syllable to the end, and two words rhyme if their rhyming segments are identical. Hence 'cake' and 'lake' rhyme, because the rhyming segment begins after the onset. However, onsets do become of interest when the last stressed syllable is not the last syllable, as in 'matron' and 'apron', which do not rhyme owing to the difference between 'tr' and 'pr'. As we have just seen above, slant rhyme also complicates matters.

[13] Benefiting from consultation with another linguist, my mother, who also suggested the syllable model.

Certain rhymes are also considered more impressive, partly as a function of the perceived rarity or difficulty of the words involved or, paradoxically, the "unlikeliness" or near-unfitness of the rhyme. For example, 'dogmatic' and 'pragmatic' is an interesting rhyme because of the rarity of the words, and 'jeweller' and 'furor' is an interesting slant rhyme because of how tenuous the similarity is. Rhymes can also be formed across multiple words, such as 'visit' and 'is it'. In some genres of poetry, eye rhymes such as that between 'stone' and 'done' are also acceptable. Odeon makes no attempt to account for any of these facts in its current iteration.

Two elements of rhyme raised interesting theoretical questions during development.

One is the rhyme scheme. Odeon's strategy is overgeneration and then filtering, rather than intentionally seeking out schemes in advance. As a result, identifying scheme-matching is paramount. They are also not exhaustively defined in Odeon. I defined, for example, 'AA' and 'ABAB' and English sonnet 'ABABCDCDEFEFGG'. Eventually, I realized that schemes should also have a wildcard option, such as 'A*': this would allow a rhyme scheme to be detected if the same rhyming sound came up every other line, but the intermediate lines did not rhyme.

In order to check whether it matches a rhyme scheme, a poem is symbolized: for example, a poem with end rhymes 'dog', 'cat, 'lion', 'bat' would become 'ABCB'. This matches the rhyme scheme 'AB*B'. The poem is then shifted over to see whether a reassignment of symbols might yield other matches. For example, we consider just the end rhymes 'cat', 'lion', 'bat', get the symbolic poem 'ABA', and can now match the scheme 'A*A'. Similarly, a poem represented as 'ABBCB' cannot match scheme 'AA*A', but can after being shifted and reassigned as 'AABA'.

In order to account for schemes used in longer poems, rhyme schemes are treated as repeatable blocks. Each rhyme scheme is matched to the whole poem and yields "runsets", which represent indices in the poem where the rhyme scheme can be found in whole. Various features are then extracted from the runset, including how many runs could be made and which lines are not covered. This allows us to say, for example, that a poem that matches both 'AA' and 'AABB' is better represented by 'AABB' because it covers all the lines.

Since all these decisions were made intuitively, there is much room for improvement of the method. But this last point brings us to the second question in rhyme, namely how to score it. As mentioned above, we want to present poems from best to worst since they are overgenerated. But how do you quantify a poem's use of rhyme? Is a poem that uses six of the same rhyme sound more skillful, because it requires finding more words that rhyme with each other, or is it better to vary the distinct rhyming sounds? Is a poem that matches one rhyme scheme four times more or less skillful than one that matches another, longer one twice? How do we account for the fact that 'June' and 'moon' are cliché while 'forced' and 'endorsed' are novel? The calculations remain rudimentary in this version of Odeon.

Meter

Meter is the least developed component of Odeon in this version. Still, it has been fruitful.

In order to break lines metrically, we have two options: we can break it using strictly defined meter, or mere rhythm.

Strictly defined meter consists of classical feet, such as the trochee (STRESSED + unstressed), the iamb (unstressed + STRESSED), and the anapest (unstressed + STRESSED + unstressed). These feet repeat a fixed number of times per line, and there are no leftover feet.

Sometimes lines can be enjambed; for example, if the final syllable needed to complete a line is stressed, but the next word to use is ‘honour’, a poet will use the word and consider the next line to invisibly begin with the leftover unstressed syllable. This becomes even more important in prose to poetry to conversion: a poet can choose to avoid enjambment using paraphrase and other hedges, but Odeon is bound to the words given it, barring the existence of lucky synonyms (here, for example, the one-syllable alternative ‘fame’).

There is also the practical problem of stress assignment. In our pronunciation dictionary, stress is inherent to a word. But in reality, stress can sometimes differ by context. For example, if the phrase ‘He ventured to our house’ occurs in an iambic context, it is read iambically, and ‘to’ naturally receives stress even though it is generally an unstressed word. One preliminary solution for this is to mark certain words as being either stressed or unstressed, but this will be ultimately impractical, since the number of words that behave this way is vast and includes large swathes of certain categories, such as single-syllable adjectives. There is also room here for the integration of theory, since an informed program could, for example, use POS-tagging to identify a word like ‘through’ as a preposition or an adverb and rate it unstressed or stressed, respectively.

Because of these complications, it’s very rare that arbitrary text fits a meter. There are various ways around this. For example, one could allow a certain degree of “fuzziness”, allowing an occasional unstressed syllable between feet. This is not implemented in Odeon at present. The rules for when it is ideal to use fuzzy meter are likely quite involved.

Another strategy is to discard the notion of feet and instead simply count beats, or “rhythm”. A beat is either a stress or, less frequently, a syllable. By ensuring that lines have more or less the same number of beats, you force the reader to parse text rhythmically (cf. the first Tolkien configuration in the “Results”). Odeon offers both, falling back to rhythm at the user’s option.

Once again the questions of schemes and scoring arise, and both are underdeveloped here. The use of a scheme might be, for example, checking that lines are iambic pentameter (five iambic feet per line) or anapestic tetrameter (four anapestic feet per line). As for scoring, the only measure currently used is consistency in the number of stresses and words per line. The more consistent, the better. Even this, though, is subject to consideration, since there are genres such as free verse in which consistently rhythmic lines are not desired.

It bears mentioning that in the case of both rhyme and meter, an entirely alternative approach in which lines are broke to create schemes, just as they are broken to create rhyme and meter in the first place, could be of value, but is incompatible with the shape of Odeon as it stands.

Formatting

Odeon takes a very basic approach to formatting: text can be lowercased (as in certain experimental poetry); the starts of lines can be capitalized (as in some style guides); and punctuation can be stripped (as in certain subgenres of poetry).

The two features I would have particularly liked to add are indentation and stanzas.

Indentation could be regular or algorithmic. That is, we could indent every other line, for example, or three out of every four, or else we could come up with a rule for when to indent. A fruitful one might be based on the POS of the word that ends up at the beginning of the line: conjunctions, for example, could be indented since they are continuations of unfinished thoughts.

Stanzas could be defined algorithmically. For example, after every run of a rhyme scheme, one could start a new stanza. This would produce couplets in a poem whose scheme was '*A*A*A', for example. Or one could start a new stanza every n lines, where n is some factor of the number of lines in the poem. More interesting would be a determination of when human break stanzas. Since humans write poetry with stanzas in mind in the first place, they tend to break conceptual ideas at the borders of stanzas (or, in experimental poetry, to intentionally continue an idea across a stanza break for the effect of suspension). While we can't analyze meaning at this stage, we could emulate this to some degree by seeking punctuation that delimits independent clauses.

Stopwords

Stopwords in Odeon are grouped under various headings where the common definition is "words that are irrelevant for some purpose". Specifically, some words have no synonyms (like 'or') or no useful ones (like 'man', which has synonyms in WordNet, but they are poor substitutions, including 'adult male' and 'piece'); other words are not desirable for rhyming, like subject pronouns (although this is configurable by the user); and others contribute little to the semantic interpretation, and can be removed to create an elliptical effect.

The NLTK's stopwords module is not very robust. It is a flat list of words without categorization or subsets. It is also inconsistent on homonyms: 'mine' is not a stopword, for example, though other possessive pronouns are, presumably because there is also a lexical word 'mine'; but 'just' is included in the adverbial sense, entailing the accidental removal of the adjective synonymous with 'fair'. Hence, custom subgroups must be defined for most purposes to get the desired list. One functional improvement would be an intelligent wrapper for requesting particular kinds of stopwords, such as determiners or wh-questions.

Synonyms

As discussed above, synonymization is easy with a thesaurus but good synonymization is extraordinarily difficult. This is true even with a sophisticated model of a word's meaning, such as WordNet synsets provide: all homonyms are labelled by part of speech and sense, so that it's easy to request *cat* 'feline' or *cat* 'cool person'. The difficulty arises rather in knowing which sense is meant in the original text. The POS can be identified, to some degree, even when considering words in isolation (for example, 'small' has both an adjective and a noun sense, but the adjective is more common). But—to echo our recurring theme—we have no means of examining semantics. More advanced natural language analysis tools could hypothetically interface with Odeon and attempt to determine the sense meant from context.

One more issue is that by default, the NLTK lemmatizes a word in order to produce its synsets. However, it cannot re-add inflection it has stripped, so a synonym will not fit grammatically back into the sentence. To avoid this, I only synonymize uninflected forms. A functional improvement would be a tool for using morphological rules to detect and re-add inflection.

Interfaces

As a minor note on command-line interfaces, Odeon has explored, in its several iterations, several different means of configuring and presenting the results of prose to poetry conversion.

Here are a few of my findings:

1. The user must be asked for text before they choose options. The options they decide on are dependent on the text they have in mind.
2. The user must be offered customization only as a last resort. Presets should be well-named and cover the most common use cases. Even when customizing option by option, there should be means of escaping individual customization branches, to avoid overwhelming and/or boring the user.
3. The user must be presented the best poems first. They will not click through all poems if they are disappointed by the first few.
4. The user must be presented poems at their own pace: one at a time, at their option to proceed or not. Too much text on the screen and they are overwhelmed. Similarly, they must have an exit opportunity at every possible juncture.
5. The user must see only a summary score by default, and only a detailed breakdown if they request it. To see the score components demystifies it a little, and is only useful for the curious.
6. The user must have poetry concepts explained to them in layman's terms.
7. The user must be given as many means as possible of doing every task, *without* each means being individually explained. This allows them to develop their own habits and do what feels familiar without the risk of unexpected behaviour.

New features

Finally, some possible directions for entirely new features bear mention:

- Indentation and stanza breaks, as described above.
- Breaking lines with neither rhyme nor meter, which is currently impossible. This could be accomplished, for example, by using a POS-tagged tree of a poem's grammar and identifying major phrases before or after which to break (or in the middle of which, in certain poetry styles).
- Scoring subtler phonoaesthetic features, such as assonance and alliteration.
- Introducing a statistical classifier-based model for poeticness, in order to classify unseen words, and to better understand which features are typical of words used by poets.
- A web app interface with a GUI that allows the simultaneous and rapid selection of text and options, the saving of individual poems and configurations, the sharing of poems, and the upvoting and downvoting of poems to produce feedback that can work in tandem with existing score factors to identify which ones are important for producing good poems.

References

Discussion

Online discussions on the subject of how to manually turn prose into poetry.

- Oxfraud: http://oxfraud.com/100-converter
  - Discussion of the essential difference between prose and poetry, in reference to the doggerel-writing 17th Earl of Oxford (sometimes credited with Shakespeare authorship by doubters of Shakespeare manuscript authenticity)
- Zumpoems: https://zumpoems.com/tag/converting-prose-into-poetry/
  - Further discussion of the differences between prose and poetry
- Writerly Play: http://writerlyplay.com/poemize-it/
  - Discussion for poets who wish to manually turn prose into poetry

Related Work

There are very many examples of pure generators of poetry, often with a selector for theme. Odeon is not a poetry generator, because its philosophy is that meaning is only present in the output if it was present in the input. However, I include one poetry generator for reference.

- Reddit thread re: poetry generator: http://bit.ly/1XqeNQM
  - By no means the oldest, newest, or most professional, but fairly representative of the quality of most poetry generators out there
  - Managed to get published in Duke's literary magazine
  - Uses a context-free grammar in Backus-Naur notation
  - Author claims that it "can create poetry indistinguishable from real poets", but the text is nonsensical; perhaps its publication is more a comment on poetry critics
- Rhymer: http://www.lukeallen.com/rhymer.html
  - Eerily similar project and name of the author (Luke), yet just discovered now
  - Rhymes using CMU dictionary; no syllable model
  - Uses WordNet synonyms, hypernyms, and hyponyms to identify synonyms; good strategy, but used solely for the purpose of permitting rhyme
  - Identifies "deletable words", a potentially more robust ellipsis system
  - Uses deletable words to equalize line lengths in terms of syllables
- Poetweet (Twitter to poetry): http://poetweet.com.br/?lang=en
  - Offers predefined forms (using both meter and rhyme)
  - Appears to search for tweets that can be truncated to make form-matching lines
  - Poor rhyme detection
  - Mostly just for fun, clearly not very theoretical
- Poemize: http://webermartin.net/poem.php
  - Given a URL, returns a poem in one of three predefined rhyming styles
  - Quite poor; very little restriction on words gathered to make up rhyming lines
- Poemify: https://www.npmjs.com/package/poemify
  - Version 1.1.1; recent work, last updated late 2014
  - Line of various lengths, stanzas, indentation, punctuation
  - But everything is random, with bounds configured by the user

General References

Some resources to look into to better understand any concepts mentioned in this report.

Poetry

- Introduction to poetry: https://en.wikipedia.org/wiki/Poetry
- Introduction to prose: https://en.wikipedia.org/wiki/Prose
- Introduction to meter: https://en.wikipedia.org/wiki/Metre_(poetry)
- Prosodic foot (cf. iamb, trochee, anapest): https://en.wikipedia.org/wiki/Foot_(prosody)
- Introduction to rhyme: https://en.wikipedia.org/wiki/Rhyme
- Slant (half) rhyme: https://en.wikipedia.org/wiki/Half_rhyme
- Synonyms: https://en.wikipedia.org/wiki/Synonym
- Stopwords: https://en.wikipedia.org/wiki/Stop_words
- Project Gutenberg: https://www.gutenberg.org/
- Project Gutenberg poetry shelf: https://www.gutenberg.org/ebooks/bookshelf/60

Linguistics

- Introduction to linguistics: https://en.wikipedia.org/wiki/Linguistics
- Syllable: https://en.wikipedia.org/wiki/Syllable
- Phone: https://en.wikipedia.org/wiki/Phone_(phonetics)
- Phoneme: https://en.wikipedia.org/wiki/Phoneme
- Stress: https://en.wikipedia.org/wiki/Stress_(linguistics)
- Phonetics: https://en.wikipedia.org/wiki/Phonetics
- Phonology: https://en.wikipedia.org/wiki/Phonology
- Phonotactics: https://en.wikipedia.org/wiki/Phonotactics
- Morphology: https://en.wikipedia.org/wiki/Morphology_(linguistics)
- Syntax: https://en.wikipedia.org/wiki/Syntax
- Semantics: https://en.wikipedia.org/wiki/Semantics
- Pragmatics (discourse): https://en.wikipedia.org/wiki/Pragmatics

Computer Science

- Python: https://www.python.org/
- NLTK: http://www.nltk.org/
- Command-line interfaces: https://en.wikipedia.org/wiki/Command-line_interface

External Resources

- Local thesaurus adapted from https://github.com/elimirks/fuzzywuzzy
- num2words adapted from https://github.com/savoirfairelinux/num2words

Poets & Authors by Order of Mention

- J.R.R. Tolkien: https://en.wikipedia.org/wiki/J._R._R._Tolkien
- William Faulkner: https://en.wikipedia.org/wiki/William_Faulkner
- William Shakespeare: https://en.wikipedia.org/wiki/William_Shakespeare
- Alexander Pope: https://en.wikipedia.org/wiki/Alexander_Pope
- Emily Dickinson: https://en.wikipedia.org/wiki/Emily_Dickinson
- e e cummings: https://en.wikipedia.org/wiki/E._E._Cummings