LING 388

Language and Computers: Final Project

Modeling Poetic Dactylic Hexameter

Nicholas Wolf

May 3, 2016

Meta-information

The following files are contained in my project submission:

code/classify.py
: Interactive classification script. Prompts the user to enter a line of poetry and, if it is valid, returns the line with metrical scansion markings added as diacritics.

code/data_processing.py
: Script to process and build a labeled corpus from text files.

code/aeneid_classifier
: The trained NLTK classifier is written to file with the Python `pickle` module for later recall. This eliminates the need to re-train the model every time the script is run.

data/BookOneNoScansion.txt
: Book One of Virgil's Aeneid, Latin [1].

data/BookOneScansion1_33.txt
: Lines 1-33 of Book One of Virgil's Aeneid with metrical scansion, Latin [2].

Introduction

My interest in language was sparked by Latin classes I took in high-school. Latin is an interesting language for a number of reasons — the aspect of it that always fascinated me is the structure of Roman epic poetry. Virgil's *Aeneid*, the 12-book long poem that tells the story of the founding of Rome, uses a rhythmic scheme called dactylic hexameter. *Dactylic*

hexameter[1] is a meter that consists of lines made from six bars or "feet". Each foot can be either a *dactyl*, which contains a long syllable followed by two short syllables ("dum-ditty"), and is marked with the notation:

$$- \smile\smile$$

or a *spondee*, which contains a long syllable followed by another long syllable ("dum-dum"),and is marked as:

$$--$$

The only exception to this rule is in the sixth foot, where the last syllable in the line is an anceps syllable. *Anceps* can function as either long or short and are either marked with a $\times$ or left blank. The metrical pattern of a line of dactylic hexameter looks something like:

$$- \smile\smile \,|\, - \smile\smile \,|\, - - \,|\, - - \,|\, - \smile\smile \,|\, - \times$$

Here's an example from the *Aeneid* in which Virgil uses a series of dactyls to express the speed of a running horse:

qūadrŭpĕ | dāntĕ pŭ | trēm sŏnĭ | tū qŭatĭt | ūngŭlă | cāmpum

"a hoof shakes the crumbling field with a galloping sound"

And another in which he does the opposite — using a series of spondees to mimic the pounding sound of blacksmiths at work [3]:

īll(i) īn | tēr sē | sē mūl | tā vī | brācchĭă | tōllunt

"take up their arms with great strength one to another"

The second line illustrates something that makes all of this even more complicated — a phenomenon called elision. If a word-terminal syllable ends in a vowel and the next word in the line begins with a vowel, the syllable can be completely omitted from the meter and the two words blend together when pronounced. In the line above, '*īll(i) īntēr*' would be pronounced as '*īllīntēr*'. This is marked by wrapping the omitted vowel in ( ).

Latin poetry does not normally come with the syllable lengths and meters marked. Presumably, Latin speakers naturally knew how to determine syllable length. It is harder for a non-native speaker to learn how to do. Mainly because the rules for what makes a syllable long or short are really more of suggestions that the author can choose to break for poetic effect.[2] This allows the author more freedom for creative expression but also makes metrical scansion more difficult than just learning a set of rules. In my project, I write a computer program that uses a statistical model to scan lines of Latin poetry. I first create a naive Bayesian model to estimate the emission probability distribution of a Hidden Markov Model (HMM). I then find the Viterbi path on the HMM to predict the most likely sequence of syllable lengths for the line of poetry.

[1]The "dactylic" in dactylic hexameter comes from the Greek: $\delta\acute{\alpha}\kappa\tau\upsilon\lambda o\varsigma$, dáktylos, "finger". A poetic dactyl is one long syllable followed by two short syllables just like the human finger is made up of one long bone followed by two short bones!

[2]I will touch more on these rules when I discuss computationally extracting features from a line of poetry.

Building a Corpus

I obtained two corpora — a normal copy of Virgil's *Aeneid* (in Latin) [1] and a copy of the first 33 lines of the first book of the *Aeneid* with metrical scansion (also in Latin) [2]. I will refer to them as the non-scanned corpus and the scanned corpus, respectively. As an example, here is the first line from the non-scanned corpus:

Arma virumque cano, Trojae qui primus ab oris.

And from the scanned corpus

Ārmă vĭr | ūmqŭe că | nō, Trōj | āe qūi | prīmŭs ăb | ōris

In order to train the naive Bayes classifier, I needed to build a labeled corpus that contained a list of (*syllable*, *length*) tuples and preserved the word and line structures from the original text so that I could easily answer questions like "is this syllable the last one in the line?" when generating feature-sets for the syllables.

The author of the scanned corpus, Daniel Rodde, marked the metrical scansion with accented characters. I was able to extract the syllable lengths from the scanned corpus by iterating over the unicode decomposition of every character in the text — any decomposition that contained U+0304 (macron, e.g. ā) denoted a long syllable and any decomposition that contained U+0306 (breve, e.g. ă) denoted a short one. Elisions were marked at every "("[3] and in-determinate syllables were marked at the end of every line.

One of the most difficult parts of this project was writing a regular expression to correctly syllabify Latin words. I eventually settled on the regular expression:

```
syllable_pattern = re.compile(
        r"""(?ix)
        (.*?(ty|y|ae|au(?!m)|ei|oe|qui|oi|gui|\biu|iur|iac|[aeioy
           ]|(?<!q)u)
        (?:[^aeioutdpbkgq](?=[^aeioyu]|\b)|
        [tdpbkg](?![lraeiou])|
        q(?!u)|\b
        )*)
        """
)
```

When passed into `re.findall(syllable_pattern, word)`, it returns a list of all the syllables in `word`.[4] I used this to break the non-scanned corpus into syllables, and then zipped the array of syllables with the array of syllable lengths from the scanned corpus to create my labeled corpus.

[3] The corpus was pre-processed to remove all of the parenthesis that occurred in the original text.

[4] The only issue I've noticed with this regular expression is that it has a hard time recognizing the syllable "tyr".

Extracting Features

My next task was building a feature set for all of the syllables in my labeled corpus. In Latin dactylic hexameter, there are four categories of traits that influence syllable length. They are ordered here roughly by the strength with which they contribute to the determination of the length of a syllable. [4, 5] were immensely helpful in choosing these features.

1. 'Ultima'
   (a) The first syllable on a line is always long.
   (b) The second to last syllable on a line is always long vowel.
   (c) The last syllable on a line always has an in-determinant length.
2. Length by Nature
   (a) Word-terminal syllables (i.e. the last syllable in a word) that end in *-o, -i, -u* are usually long.
   (b) Word-terminal syllables that end in *-as, -es, -us* are usually long.
   (c) Word-terminal syllables that end in *-a, -is* are often short.
   (d) Word-terminal syllables that end in *-e* are usually short.
   (e) Word-terminal syllables that end in *-us* are usually short.
   (f) Word-terminal syllables that end in *-am, -em, -um* are usually short.
3. Length by Position
   (a) Vowels that are followed by two consonants are usually long.
   (b) Vowels that are followed by stop-liquids (e.g. *tr*) are sometimes long.
   (c) Diphthongs (e.g. *ae*) are usually long.
4. Ability to Elide
   (a) Word-terminal syllables that end in a vowel or *-m* and are followed by a word that begins with a vowel or an *h-* can, but don't have to, elide.

The ways that these features influence the length of a syllable are qualified with words that are frustratingly ambiguous — words like 'often', 'usually' and 'sometimes'. One of the nice side-effects of building a statistical model to classify the lengths of syllables is that it can estimate the distributions that underlie those fuzzy qualifiers. Another nice side-effect is that training the classifier on a single author's work will produce a sort of fingerprint that captures that author's tendencies to break, bend and ignore rules.

Training

My labeled corpus comprises 33 lines (506 syllables) of poetry. I trained the built-in NLTK naive Bayesian classifier on 75% of the data and withheld the remaining 25% for testing.

Evaluation

After the classifier was trained, it classified syllables in the test fold with an accuracy of 81.8% — baseline performance is 54.0%[5]. The tendencies of the classifier can be seen in the confusion matrix in Table 1.

	Long	**Short**	**In-determinate**	**Elision**
Long	43.7%	9.5%	–	0.8%
Short	7.1%	28.6%	–	0.8%
In-determinate	–	–	6.3%	–
Elision	–	–	–	3.2%

Table 1: Confusion matrix for naive Bayesian classifier (rows = reference).

Below is the output from the classifier on the entire testing set. Green lines are correctly labeled. Red lines are incorrect and illegal dactylic hexameter.

ĀRMĂ vĭrūmquĕ cānō Trōjāe qŭi prĭmŭs ăb ōris

Ītălĭām fātō prōfŭgŭs Lăvĭnăquĕ vēnit

Lītōrā mūl(tum) īl(le) ēt tērrīs jāctătŭs ēt ālto

Vī sŭpĕrūm sāevāe mĕmōrēm Jūnōnīs ŏb īram

Mūltă quŏ(que) ēt bēllō pāssŭs dūm cōndĕrēt ūrbem

īnfĕr | rētquĕ dĕ | ōs Lătĭ | ō gĕnŭs | ūndĕ Lă | tīnum

Ālbănĭquĕ (pa)trēs āt(que) āltāe (moe)nĭă Rōmae

Mūsă mĭhĭ cāusās mĕmōrā quŏ nŭmĭnĕ lāeso

Extending the Model

The naive Bayesian classifier assumes independence between syllables — the model doesn't consider the lengths of the other syllables in a line when predicting the length of a particular syllable. Even though the classifier is more than 80% accurate at a syllable level, all but one of the output lines are illegal dactylic hexameter! The problem can be better modeled by

[5]Every syllable tagged as 'Long'.

a Hidden Markov Model (HMM). Figure 1 represents the state transition diagram for this HMM.

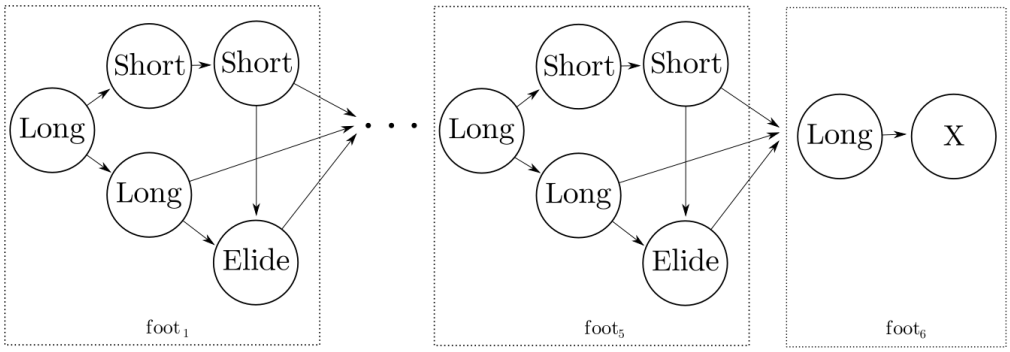

Figure 1: State transition diagram for vowel length in a line of dactylic hexameter.

Any path from the long syllable on the left side of the diagram (the first syllable in the line) to the indeterminate (×) syllable on the right side (the last syllable in the line) constitutes a sequence that is a legal line of dactylic hexameter. I assume that the transition probability distribution from any state is uniform over the set of possible next states and the emission probability for each state is given by the naive Bayesian model trained earlier.

Computing the most likely sequence of syllable lengths for a line of poetry is fairly easy — each foot, except for the last, can have one of four scansions

$$--\tag{1}$$

$$--\ (elide)\tag{2}$$

$$-\smile\smile\tag{3}$$

$$-\smile\smile\ (elide)\tag{4}$$

and the last foot is a given

$$-\times$$

so the space of possible paths is actually relatively small ($4^5 = 1024$). For any given line, the space can be further restricted by only considering paths that have a length equal to the number of syllables in the line. A line with 15 syllables, for example, only has 120 different possible metrical sequences. This is a small enough search space that it is not very expensive to compute the log-likelihood of *every* possible path. The path with the smallest negative log-likelihood corresponds to the most likely sequence of syllable lengths. Classifying a line of poetry by calculating the most likely sequence of hidden states in the HMM improves syllable classification accuracy to 85.3% and (more importantly) ensure that every line of labeled poetry that comes out of the classifier is legal. Below is the output from the new classifier on the entire test set. Green lines are correctly labeled. Non-highlighted lines are incorrect but legal dactylic hexameter.

ĀRMĂ vĭr | ūmquĕ că | nō Trō | jāe q̆ui prĭ | mūs āb | ōris

Ītălĭ | ām fā | tō prō | fūgŭs Lă | vīnăquĕ | vēnit

Lītōrā mūl | (tum) īllĕ ĕt | tērrĭs jăc | tātŭs ĕt | ālto

Vī sŭpĕr | ūm sāe | vāe mĕmŏ | rēm Jū | nōnĭs ŏb | īram

Mūltă quŏ | (que) ēt bēl | lō pās | sūs dūm | cōndĕrĕt | ūrbem

īnfĕr | rētquĕ dĕ | ōs Lătĭ | ō gĕnŭs | ūndĕ Lăt | īnum

Ālbā | nīquĕ păt | rēs āt | (que) āltāe | mōenĭă | Rōmae

Mūsă mĭ | hī cāu | sās mĕmŏ | rā quŏ nŭ | mīnē | lāeso

Line-level accuracy has increased by a factor of four and every line is legal dactylic hexameter.

Future Work

There are a few things that could improve the classifier I've built, and a few things that I think would be fun to do with it.

1. I would like to construct a larger corpus to train the model on. The current corpus has 506 labeled syllables in it but only 33 lines of poetry. Given the influence that the structure of a line has on the lengths of the syllables in it, 33 is too small of a number to train an accurate model. An additional benefit of having more lines is that it would allow the construction of a prior distribution that could capture a belief in how likely the scansion pattern of a line is (Virgil hardly ever writes lines with only long vowels, for example, so that sequence should have a lower prior probability than others).

2. The Viterbi algorithm is a dynamic programming approach that reduces the computational complexity of calculating the most likely sequence of states in the HMM from exponential to linear time [6]. While it isn't necessary for the classifier currently, I would like to implement it because it would make it computationally feasible to extend the model to capture dependency between lines, which could be an interesting area of exploration.

3. I would like to develop a metric for measuring the character of a line of dactylic hexameter (e.g. lines with a lot of long syllables feel slower than lines with a lot of short syllables) so that the way the character of a poem changes over the course of a chapter, section, or book could be visualized.

4. A foot of dactylic hexameter can be thought of as a four-bar musical phrase where long syllables correspond to half notes and short syllables correspond to quarter notes. I think it would be interesting to run an entire book of the *Aeneid* through the classifier, write the metrical scansion to a MIDI file, and then use it in a musical composition as a rhythm track.

5. I'm curious to investigate how difficult it would be to extend this model to be able to scan other poetic meters.

References

[1] Virgil. *The Aeneid*. Project Gutenberg. Accessed at http://www.gutenberg.org/files/227/227-h/227-h.htm.

[2] Daniel Rodde. Aeneid Book 1 ll. 1-33 Scanned & Translated. Accessed at http://www.fenwickfriars.com/Page/246.

[3] Dactylic Hexameter. Accessed at https://en.wikipedia.org/wiki/Dactylic_hexameter.

[4] Joseph Farrell. Hexametrica, 1999. Accessed at http://www.skidmore.edu/academics/classics/courses/metrica/.

[5] Barbara Boyd. *Vergil's Aeneid: Selections from Books 1, 2, 4, 6, 10 & 12*. Bolchazy-Carducci Publishers, Inc, 2009.

[6] Christopher M CM Christopher M. Bishop. *Pattern Recognition and Machine Learning*, volume 4 of *Information science and statistics*. Springer, 2006.