Automatic Classification of Poetry by Meter and Rhyme

MARGENTO (Chris Tanasescu), Bryan Paget, and Diana Inkpen

University of Ottawa
800 King Edward Ave.
Ottawa ON Canada, K1N 6N5

Abstract

In this paper, we focus on large scale poetry classification by meter. We repurposed an open source poetry scanning program (the Scandroid by Charles O. Hartman) as a feature extractor. Our machine learning experiments show a useful ability to classify poems by poetic meter. We also made our own rhyme detector using the Carnegie Melon University Pronouncing Dictionary as our primary source of pronunciation information. Future work will involve classifying rhyme and assembling a graph (or graphs) as part of the *Graph Poem Project* depicting the interconnected nature of poetry across history, geography, genre, etc.

Introduction

The huge amount of data available in the digital age has attracted the attention of major scholars and has developed into its own research paradigm. There is no consensus as to when data are large or complex enough to qualify as the object of data-intensive research, especially since huge or massive may mean completely different things in different fields and disciplines, but Levallois, Steinmetz, and Wouters advance a relevant and potentially useful definition: "data-intensive research [is] research that requires radical changes in the discipline" involving "new, possibly more standardized and technology-intensive ways to store, annotate, and share data," a concept that therefore "may point toward quite different research practices and computational tools" (Levalois, Steinmetz, and Wouters 2012). This paper introduces our endeavour to redefine the scholarly approach to poetry analysis by applying data-intensive research methods and eventually mathematical graph theory.

The earliest stages of the *Graph Poem Project* (MARGENTO 2015) resulted in the publication of a bulky collection of poems entitled *Nomadosophy* (MARGENTO 2012), subtitled "a graph poem". The author of this collection assembled the graphs "manually" by identifying and connecting the related poems himself, while also writing poems of his own inspired by this very process.

The goal of our work is to create computational tools necessary for identification and analysis of specific features in large collections of written poetry, such as meter, stanzaic patterns, verse forms, sonic devices (rhyme, alliteration, euphonious characteristics in general), and style (diction, tropes, syntax and anatomy of line breaks and enjambments).

The purpose of the *Graph Poem* is to augment the study of poetry. A poet's oeuvre, a poetry school's publications, or all of the poetry ever published by a journal can be thus represented as graphs. Then by studying the features of those graphs (connectivity, cliques, cut edges or cut vertices, etc.), a number of relevant conclusions can be drawn about that particular oeuvre/corpus/database or school/period. Such conclusions can then be measured against the established literary criticism.

We will now discuss the groundwork that has been done for the *Graph Poem*. The graph needs features to quantify and qualify the relationships between poems.

We decided that the poetic meter was a fundamental feature of a poem, followed by line structure and use of rhyme. The rest of this paper introduces our efforts to extract metrical related features from raw text and our initial classification experiments. We also present early experiments in rhyme detection in a later section.

Iambic pentameter is the most common meter used in English poetry. In its pure, traditional form, it consists of five pairs (feet) of alternating unstressed and stressed syllables, in that order. Variations are commonly used by poets intentionally for a myriad of emotional, sensual and interpretive effects. Common variations include reversal of the ordering of the first foot (stressed and unstressed) and a feminine ending (adding an extra unstressed syllable to the last foot as in "To be, or not to be: that is the question", where the last foot has three syllables, unstressed, stressed and unstressed). There are of course many other variations, therefore identifying iambic pentameter is not a straightforward task, as exceptions are the rule. We used a binary notation for scansion (the act of scanning a line of poetry to determine its rhythm); a "0" for an unstressed syllable and a "1" for a stressed syllable. Therefore an iambic foot was represented by a "01".

We experimented with different classifiers and discovered that J48 with bootstrap aggregating provided the most accurate results (Frank et al. 2009). Briefly, J48 is an open source and updated implementation (written in Java) of C4.5 (written in C, also open source but superseded by only the partially open source C5.0) by Ross Quinlan. These algorithms belong to a class of decision tree algorithms that utilize the concept of information entropy when creating new branches (Quinlan 2013).

Related Work

Computational analyses of small, manually labeled corpora of poetry have been conducted before, notably by Malcolm Hayward (Hayward 1991). In 1991, Hayward published a paper on a connectionist constraint satisfaction model based on a parallel distributed processing model by McClelland and Rumelhat for analyzing the metrical features of poetry. He was motivated by traditional metrical analytic methods' inability to account for interactions between all aspects of a poem and their overall effect on prosody during reading performances. To improve analysis of iambic verse, he manually labeled each syllable in the opening line of Wordsworth's "Tintern Abbey" with the likelihood of stress (what he called activation) according to intonation, lexical stress, prosodic devices, syntactic patterns and interpretive emphasis.

Hayward divided the integral of activation in the soft position by the integral of activation in the hard position to measure metrical regularity in iambic verse. Ratios close to 0 indicated a more regular iambic verse while ratios that approached 1 indicated a less regular use of iambic pentameter (or none at all) (Hayward 1991).

Convinced that this model could be used to uncover individual poet's unique metrical style, in 1996 Hayward published a follow up paper comparing 10 poets' treatment of iambic verse and was able to fingerprint them and determine their period of writing (Hayward 1996).

While sophisticated, Hayward's approach can not scale as it relies on hand annotated data. Additionally, Hayward analyzed only 1000 lines of poetry and it is unknown how well his model will generalize for different corpora.

More recently, researchers in the digital humanities have begun working with much larger datasets. Algee-Hewitt, Heuser, et al. from the Stanford Literary Lab performed data intensive poetry analysis via the Trans-Historical Literary Project. As of this writing they have not published a paper, however an abstract of their work is available from dharchive.org (Algee-Hewitt et al. 2014) and their source code is available on Heuser's github page (Heuser 2015).

According to their presentation notes (available on github), their parser was able to correctly divide words into syllables with an accuracy of 88%. They were also able to correctly classify a sample of 205 poems as either binary or ternary footed and according to the location of the stressed syllable in the feet (either first or last position) with an accuracy of 99%.

As we will show in later sections, our scansion tool may be more useful for poetry analysis than the Stanford Literary Lab's parser.

Methodology

Text classification requires that text be represented as vectors of statistical features (see Figure 1).

Figure 1: Truncated feature vector (due to space)

Feature (%)	Data Type	Example (Sonnet 133)
hard rhyme	Float	0.285714285714
soft rhyme	Float	0.0
slant rhyme	Float	0.214285714286
eye rhyme	Float	0.0
unknown rhyme	Float	0.5
couplet (AABB)	Float	0.0714285714286
cross (ABAB)	Float	0.142857142857
envelope (ABBA)	Float	0.0
iamb	Float	0.571428571429
spondee	Float	0.128571428571
trochee	Float	0.142857142857
pyrrhus	Float	0.114285714286
anapest	Float	0.0142857142857
dactyl	Float	0.0142857142857
amphibrach	Float	0.142857142857

For the purpose of poetry classification we were interested in prosodic (the rhythm of the poem) and phonetic (the pronunciation of words) features. These two classes of features were used to identify poetic meter and rhyme. Rhyme detection is a work in progress; for more information see the future work section.

We originally used the Carnegie Melon University's Pronouncing Dictionary (cmudict) to determine phonetic stress. The cmudict contains entries for over 134,000 words and their pronunciations.

The cmudict's embedded stress information left a lot to be desired. Other researchers have expressed dissatisfaction with the cmudict as well (Greene, Bodrumlu, and Knight 2010). Specifically, the cmudict was not helpful if a word was not already in the dictionary and most importantly, as a dictionary it could not take into consideration the variable nature of English prosody. See Figure 2 for an example entry.

Figure 2: Example from CMU Pronouncing Dictionary

Word: ``meter''	
Entry: ``M IY1 T ER0''	
M	``m'' sound.
IY1	as in ``bee''; the ``1'' denotes lexical stress activation.
T	``t'' sound.
ER0	as in ``her''; the ``0'' denotes a lack of lexical stress.

Note: only vowel sounds include lexical stress information.

To improve accuracy of lexical stress detection, we reused code from the Scandroid by Charles O. Hartman (Hartman 2004). While the cmudict relies entirely on its dictionary, the Scandroid uses a rule based algorithm to determine stress and consults its own dictionary for words known to be exceptions. We modified the original Python source code, enabling the Scandroid to scan arbitrarily large collections of poems and migrated most of its functions into our own command line Python scripts.

The Scandroid parses words into syllables and then follows a set of rules to guess which syllables were stressed and which were unstressed. The Scandroid contains a dictionary of a few hundred words which introduce ambiguity into rule based pronunciation — such as ”reach” and ”react” — and most monosyllabic pronouns, conjunctions and prepositions. It also has a method for grouping stresses together into feet. Grammatical words, such as monosyllabic prepositions and pronouns are considered soft by default. The Scandroid assigns a strong stress to all monosyllabic words not found in the dictionary.

Hartman built the Scandroid's syllable division system on earlier work done by Paul Holzer (Holzer 1986). Stressed syllables are determined using a method developed by Bernstein and Nessly known as Nessly's Default (Bernstein and Nessly 1981). Nessly's Default assumes that most disyllabic words have the stress in the first of the two syllable positions. Briefly, when the Scandroid scans a poem, two algorithms are deployed and the one producing the most regular scansion with the lowest complexity is chosen over the other (Hartman 2005). The first algorithm — Corral the Weird — checks for abnormal line endings and determines metricality by excluding them. The second algorithm — Maximize the Normal — searches for the longest run of iambic feet.

Figure 3: Features used to classify poems by meter

1.	The number of lines in the poem.
2.	The average number of metrical feet per line and the standard deviation.
3.	The count of each type of foot (iamb, spondee, trochee, pyrrhus, anapest, dactyl, amphibrach, tribrach, bacchius, antibacchius, cretic, molossus).
4.	The maximum number of feet of the same type in a row.
5.	The number of times the foot changed from one type to another throughout the poem.

We worked mainly with poetry from the poetryfoundation.org with some additional sonnets from sonnets.org. We chose these two sources because their corpora were already labeled and the former provided us with a well rounded — in terms of verse form — selection of poetry from the 16th century onward, including sonnets and free verse. We employed a machine learning technique called bootstrap aggregating (also known as bagging) (Breiman 1996), which is a meta-algorithm that can greatly improve decision tree accuracy. The decision tree algorithm used for this paper was J48, as part of Weka. Bagging creates multiple training subsets of the original training set by uniformly sampling with replacement. Multiple decision trees are made on each subset. The trees are then combined to make a more accurate tree (Witten, Frank, and Hall 2011).

Experiments and Results for Meter Classification

Scansion results for “Sonnet 133” by William Shakespeare will be displayed first as an example, followed by classification results from our corpus. “Sonnet 133” was chosen randomly from a selection of sonnets by various poets. A sonnet is a traditional verse form usually written in iambic pentameter and usually following a rhyming scheme (see Figure 4).

Figure 4: Sonnet 133 by William Shakespeare (1564-1616)

Beshrew that heart that makes my heart to groan
For that deep wound it gives my friend and me:
Is't not enough to torture me alone,
But slave to slavery my sweet'st friend must be?
Me from myself thy cruel eye hath taken,
And my next self thou harder hast engrossed;
Of him, myself, and thee I am forsaken,
A torment thrice threefold thus to be crossed.
Prison my heart in thy steel bosom's ward,
But then my friend's heart let my poor heart bail;
Whoe'er keeps me, let my heart be his guard:
Thou canst not then use rigour in my jail.
And yet thou wilt; for I, being pent in thee,
Perforce am thine, and all that is in me.

Figure 5: Metrical breakdown of *Sonnet 133*

Foot Type	Binary	Composition
Iamb	01	59.2%
Spondee	11	12.7%
Trochee	10	8.5%
Pyrrhus	00	12.7%
Anapest	001	1.4%
Dactyl	100	1.4%
Amphibrach	010	1.4%
Tribrach	000	0.0%
Bacchius	011	0.0%
Antibacchius	110	0.0%
Cretic	101	0.0%
Molossus	111	2.8%

Average number of feet per line: 5.07
Longest run of a single foot: 4
Percentage of foot changes 49.30%

We used a Python script to compare the scansions of *Sonnet 133* produced by the Scandroid and the Stanford Literary lab with one manually generated scansion produced by one of the authors (see Figures 6, 7, 8, 9, 10 and 11). Dropped and inserted syllables were taken into consideration by a fail-safe which avoided underestimating similarity due to frame-shifting (where the whole line is considered different due to a missing or added character). The following results relate to *Sonnet 133*.

Figure 6: Comparison of scansions of *Sonnet 133*

Similarity with Manual Scansion	
Scandroid	90.55%
Stanford	67.78%

Similarity between Scandroid and Stanford
68.07 %

Figure 7: Manual scansion of *Sonnet 133*

Original	Digitized
01 01 01 01 01	01 01 01 01 01
00 11 01 01 0+	00 11 01 01 01
0+ 01 01 0+ 0/+	01 01 01 01 011
01 01 00 0/1 -1	01 01 00 011 11
+0 01 0-1 01 0	10 01 011 01 0
00 /+1 01 0+ 01	00 111 01 01 01
0+ 01 0+ 0+ 01 0	01 01 01 01 01 0
01 01 -+ /0 01	01 01 11 10 01
10 01 00 11 01	10 01 00 11 01
0+ 01 11 01 11	01 01 11 01 11
001 01 01 001	001 01 01 001
01 /+ 11 0+ 01	01 11 11 01 01
0+ 01 0+ 001 0+	01 01 01 001 01
01 0+ 0+ 01 0+	00 01 01 01 01

1 represents lexical stress and 0 represents a lack of it. + denotes a promoted half-stress, / denotes an ordinary half-stress and - denotes a demoted half-stress. The digitized scansion was achieved by changing the half stresses to 1.

We also randomly selected 41 poems and compared the average similarity between the Scandroid and Stanford scansions was 63.70%. We found a 50.87% similarity between the Scandroid and cmudict scansions, and a 37.91% similarity between the Stanford and cmudict scansions.

Figure 8: Comparison within sample of 41 Poems

Similarity	
Scandroid & Stanford	63.70%
Scandroid & cmudict	50.87%
Stanford & cmudict	37.91%

We also made a comparison with the double zeros in the Stanford scansion replaced with single zeros, in which case there was an 81.48% similarity for "Sonnet 133" and a similarity of 83.55% for the 41 scansions overall.

Figure 9: Scansion comparison: The Scandroid and cmudict

The Scandroid	cmudict
10 01 01 01 01	?11111111
01 11 01 01 00	1111111101
11 01 01 00 01	11101110101
01 01 00 01 100	11110011?111
00 01 01 10 10	11211101110
00 11 01 01 01	0111110101
00 01 01 01 010	11210111010
01 01 10 10 01	0121121111
10 01 01 11 01	101101110111
01 01 11 01 11	1111111111
10 10 10 11 01	?011111111
01 11 11 00 01	1?111?011
01 01 001 01 01	01111110101
10 01 01 01 00	0111011101

Note: our code does not yet implement the promoted stress algorithm as in the original Scandroid by Hartman.

Figure 10: Scansion comparison: The Scandroid and Stanford Literary Lab

The Scandroid	Stanford Literary Lab
10 01 01 01 01	1 00 1 00 1 0 1 0 1
01 11 01 01 00	1 0 1 00 1 0 1 0 1
11 01 01 00 01	1 00 1 0 1 0 1 0 1
01 01 00 01 100	0 1 0 1 0 1 0 1 0 1 0
00 01 01 10 10	1 0 1 00 1 0 1 0 1 0
00 11 01 01 01	00 1 00 1 0 1 0 1
00 01 01 01 010	00 1 0 00 1 00 1 0
01 01 10 10 01	1 0 1 0 1 0 1 00 1
10 01 01 11 01	1 00 1 0 1 0 1 0 1
01 01 11 01 11	1 00 1 0 1 0 1 0 1
10 10 10 11 01	1 0 1 0 1 0 1 00 1
01 11 11 00 01	1 0 1 00 1 00 1
01 01 001 01 01	0 1 0 1 1 0 1 0 1 0 1
10 01 01 01 00	0 1 00 0 1 00 1 0

These results were obtained from Python code found on Ryan Heuser's github page. We followed the instructions found in README.md (Heuser 2015).

We will now discuss classification results. With J48 and bootstrap aggregating, we were able to achieve a 94.39% success rate of correctly classifying poetry as metrical or not (see Figure 11). We compared these results with the results from two simple baseline algorithms, ZeroR (which uses class frequency to predict the majority class) and OneR (a one level decision tree) (Witten, Frank, and Hall 2011). ZeroR classified these data with an 82.53% success rate, and 92.10% with OneR. These results were attained with a training dataset consisting of 871 metrical poems and 4115 non metrical poems. The class imbalance was representative of the relative scarcity of metered poetry when compared to the abundance of non metered poetry. We used 10-fold cross-validation.

Figure 11: Classification results

Classification	Metrical	Not Metrical
Metrical	682 (87.66 %)	189 (4.44 %)
Not Metrical	96 (12.34 %)	4019 (95.56 %)

Correctly Classified Instances : 4701 (94.39%)

We also trained additional classifiers with two balanced datasets derived from the original by undersampling the non metrical class (to create a balanced dataset with 871 of each class) and oversampling the metrical class by 400% using SMOTE (Synthetic Minority Over-sampling TEchnique) which resulted in a dataset with 4355 metrical poems and 4115 non metrical poems. SMOTE is an algorithm that creates synthetic data by examining similar data — determined by proximity in Cartesian space — from the minority class (Chawla et al. 2002). A separate test set was used to compare these two classifiers with the original classifier (which was also retrained on data that was free from the test instances). The test dataset consisted of 42 metrical poems and 207 non metrical poems. The three training sets produced classifiers that were accurate to within 1% of each other.

Early Experiment on Rhyme Detection

Work done for meter detection led naturally to methods for rhyme detection. For rhyme classification we considered only rhymes located at the end of each line in the poem, otherwise known as terminal rhymes. From this we were also able to identify couplet (AABB), cross (ABAB) and envelope (ABBA) terminal rhyme schemes. Issues arose when trying to identify possible rhyming pairs. Rhymes come in many varieties and poets often use the fact that words can be made to rhyme by playing with their pronunciation, so we had to design a flexible and inclusive way of classifying them.

A rhyme has at least two components, the last stressed (sometimes unstressed) vowel and all letters following that vowel. Rhyming pairs can therefore be classified based on which of these elements they share in common. To identify rhymes, we compared phonemes (the simplest linguistic units of sound) of pairs of words and then classified them based on what they shared in common. Often more than one name is used for the same class of rhyme, so we chose to classify rhymes as either perfect, slant (not perfect) and eye (words that don't rhyme but used by poets because they look like they would). We felt that these three classes were sufficient to describe all rhyme types.

We defined perfect rhymes as pairs of words sharing a final stressed vowel sound and all remaining phonemes, e.g. "snake" and "bake". It is also possible for a perfect rhyming pair to share a final unstressed vowel sound, for example "handing" and "standing".

We included multiple imperfect rhyme definitions in our slant rhyme class, including pairs of words sharing a final vowel sound but with different remaining letters, e.g. "band" and "sang", or words that do not share a final vowel sound but share the remaining letters, e.g. "hand" and "bind". We included what are sometimes called embedded rhymes, which are like perfect rhymes, but one of the two words is a compound word, e.g. "band" and "sandcastle" — this type of rhyme is sometimes used with an enjambment (poetic line break) splitting the compound word in two.

We defined eye rhymes as pairs of words that do not share much at all in pronunciation but have similar spelling, such that they may look like a perfect rhyming pair but sounding completely different, e.g. "slaughter" and "laughter". See Figures 12, 13 and 14 for examples of how we compared words.

For the purpose of this paper, we consider word pairs as simply rhyming or not, i.e. the type of rhyme was not considered. This choice was made in light of the amount of work we still need to do to improve the generalization ability of the rhyme detector for large scale poetry analysis. Additionally, rhyme detection is just the beginning in our sonic analysis. Our software for detecting rhyme has the resolution and versatility for other tasks such as identifying alliteration, assonance, consonance and sonic devices in general.

We have completed preliminary work on rhyme detection. To identify rhymes, we collected the last words from each line of the poem. We then used the cmudict to compute the Levenshtein edit distances for each phoneme, for each word, for each line. We considered phonemes to be the same when the Levenshtein edit distance was 0. We were able to detect different kinds of rhymes because we were able to compare assonance and consonance separately.

Figure 12: A perfect rhyming pair

battle : cattle				
B K	AE1 AE1	T T	AH0 AH0	L L
nc	*yv	yc	yv	yc

Similarities are denoted by a `y` for a match or an `n` for no match. A $\star$ indicates a match on a stressed vowel. `v` and `c` represent vowels and consonants, respectively. A solitary `n` denotes a complete mismatch, i.e. not only do they not match, but one is a vowel and the other is a consonant. Some examples: `*yv` represents a match on a stressed vowel and `nc` represents a non matching pair of consonants.

Figure 13: An example of a mosaic rhyme

But-oh! ye lords of ladies intellectual,
Inform us truly, have they not hen-peck'd you all?

intellectual : hen-peck'd you all										
IH2	N HH	T EH1	AH0 N	L P	EH1 EH1	K K	CH Y	UW0 UW1	AH0 AO1	L L
	n	n	n	nc	*yv	yc	n	yv	nv	yc

Figure 14: Examples of slant rhymes

hand : hang			
HH HH	AE1 AE1	N NG	D
yc	*yv	nc	

band : bind			
B B	AE1 AY1	N N	D D
yc	nv	yc	yc

Conclusion and Future Work

As part of the Graph Poem project's holistic approach to poetry analysis, we plan on incorporating additional features like metaphor, diction (choice of words), stanzaic patterns, verse forms, alliteration (and other euphonic devices), syntax, the anatomy of enjambments and tropes in general, etc. With relation to meter, a modern neural network could be used to reclaim some of the rhythmical richness lost in binary encoding by allowing the simultaneous analysis of all features influencing stress (Hayward 1991).

Our binary classification experiments (metered or not metered) were better than the baseline. The unbalanced test experiment resulted in an accuracy of 94.39% (Figure 10), which was significantly better than ZeroR (82.53%) but only marginally better than OneR (92.10%). The two balanced test results (from over and under sampling) were not significantly different from the original test (less than 1% difference between the three). Classification may improve with more advanced classification algorithms and may improve further still as we continue to add more training data to our corpus.

The Scandroid was more accurate at determining lexical stress and parsing text into metrical feet than the work done by the Stanford Literary Lab. It may be possible to train a neural network in a way similar to the work done by Hayward (but using more modern methods) to further improve accuracy and introduce sensitivity to different levels of stress (not just binary encoding).

There are known issues with the Scandroid, including lexical ambiguity and phrasal verbs (Hartman 2005). Lexical ambiguity affects the pronunciation of certain words depending on whether they are verbs or nouns. Hartman uses "con VICT" (verb) and "CON vict" (noun) as an example. This ambiguity could be lessened by including contextual information from a POS (Part of Speech) tagger. Phrasal verbs are lexical units that operate as a single word but do not look that way, e.g. "put out", "put down" or "put away". Stress tends to be evenly distributed over the verb and grammatical structure following it. A POS tagger could also be used to identify phrasal verbs and adjust stress appropriately. Finally, our implementation of the Scandroid is missing the ability to identify promoted stress. Source code cleanup should make that easier to implement.

Rhyme detection can be improved and extended by adding the ability to detect internal/nonterminal rhymes and rhymes spread out over multiple words (known as mosaic rhymes), e.g. "Poet" and "know it". If a word is not listed in the cmudict, we used the double metaphone algorithm by Lawrence Philips (Philips 1990) and spelling as crude backups. A more sophisticated approach may involve the use of a computer text to speech system.

References

Algee-Hewitt, M.; Heuser, R.; Kraxenberger, M.; Porter, J.; Sensenbaugh, J.; and Tackett, J. 2014. The Stanford Literary Lab Transhistorical Poetry Project Phase II: Metrical Form. Proceedings, Stanford University, Lausanne.

Bernstein, J., and Nessly, L. 1981. Performance Comparison Of Component Algorithms For The Phonemicization Of Orthography. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, 19–22. Stanford, California, USA: Association for Computational Linguistics.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; and Kegelmeyer, W. P. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16:321–357.

Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H.; and Hall, M. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11(1).

Greene, E.; Bodrumlu, T.; and Knight, K. 2010. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, 524–533. Association for Computational Linguistics.

Hartman, C. 2004. Charles Hartman Programs. Online; accessed 01-August-2015. Retrieved from http://oak.conncoll.edu/cohar/Programs.htm.

Hartman, C. 2005. The Scandroid Manual. Online; accessed 01-August-2015. Retrieved from http://oak.conncoll.edu/cohar/Programs.htm.

Hayward, M. 1991. A connectionist model of poetic meter. *Poetics* 20(4):303–317.

Hayward, M. 1996. Analysis of a corpus of poetry by a connectionist model of poetic meter. *Poetics* 24(1):1–11.

Heuser, R. 2015. Stanford Literary Lab Github Account. Online; accessed 01-August-2015. Retrieved from https://github.com/quadrismegistus/litlab-poetry.

Holzer, P. 1986. Machine Reading of Metric Verse. *Byte Magazine* 11(02):224–225.

Levalois, C.; Steinmetz, S.; and Wouters, P. 2012. *Sloppy Data Floods or Precise Social Science Methodologies? Dilemmas in the Transition to Data-Intensive Research in Sociology and Economics*. The MIT Press.

MARGENTO. 2012. *Nomadosofia/Nomadosophy*. Max Blecher Press. Bilingual, Romanian and English.

MARGENTO. 2015. The Graph Poem Project. Online; accessed 17-February-2016. Retrieved from http://artsites.uottawa.ca/margento/en/the-graph-poem/.

Moretti, F. 2013. *Distant Reading*. Versa.

Philips, L. 1990. Hanging on the Metaphone. *Computer Language* 7(12).

Quinlan, R. 2013. Data Mining Tools See5 and C5.0. Online; accessed 01-August-2015. Retrieved from http://rulequest.com/see5-info.html.

Witten, I. H.; Frank, E.; and Hall, M. A. 2011. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Burlington, MA: Morgan Kaufmann, 3rd ed edition.