

AUTOMATIC POETRY CLASSIFICATION USING NATURAL LANGUAGE  
PROCESSING

BY

VAIBHAV KESARWANI

Thesis submitted in  
partial fulfillment of the requirements  
for the Master of Computer Science degree

School of Electrical Engineering and Computer Science  
Faculty of Engineering  
University of Ottawa

© Vaibhav Kesarwani, Ottawa, Canada, 2018

# ABSTRACT

Poetry, as a special form of literature, is crucial for computational linguistics. It has a high density of emotions, figures of speech, vividness, creativity, and ambiguity. Poetry poses a much greater challenge for the application of Natural Language Processing algorithms than any other literary genre.

Our system establishes a computational model that classifies poems based on similarity features like rhyme, diction, and metaphor.

For rhyme analysis, we investigate the methods used to classify poems based on rhyme patterns. First, the overview of different types of rhymes is given along with the detailed description of detecting rhyme type and sub-types by the application of a pronunciation dictionary on our poetry dataset. We achieve an accuracy of 96.51% in identifying rhymes in poetry by applying a phonetic similarity model. Then we achieve a rhyme quantification metric *RhymeScore* based on the matching phonetic transcription of each poem. We also develop an application for the visualization of this quantified *RhymeScore* as a scatter plot in 2 or 3 dimensions.

For diction analysis, we investigate the methods used to classify poems based on diction. First the linguistic quantitative and semantic features that constitute diction are enumerated. Then we investigate the methodology used to compute these features from our poetry dataset. We also build a word embeddings model on our poetry dataset with 1.5 million words in 100 dimensions and do a comparative analysis with GloVe embeddings.

Metaphor is a part of diction, but as it is a very complex topic in its own right, we address it as a stand-alone issue and develop several methods for it. Previous work on metaphor detection relies on either rule-based or statistical models, none of them applied to poetry. Our methods focus on metaphor detection in a poetry corpus, but we test on non-poetry data as well. We combine rule-based and statistical models (word embeddings) to develop a new classification system. Our first metaphor detection method achieves a

precision of 0.759 and a recall of 0.804 in identifying one type of metaphor in poetry, by using a Support Vector Machine classifier with various types of features. Furthermore, our deep learning model based on a Convolutional Neural Network achieves a precision of 0.831 and a recall of 0.836 for the same task. We also develop an application for generic metaphor detection in any type of natural text.

*To my wife and parents for their unconditional support and love.*

# ACKNOWLEDGMENTS

First and foremost, I would like to infinitely thank my supervisor Dr. Diana Inkpen for the fabulous research direction, innovative ideas, endless patience, invaluable feedback and support for me and my work. It is truly an honor working with such a great mentor.

I would like to thank my co-supervisor Dr. Chris Tanasescu for the invaluable insight, unfaltering dedication, creative genius and never-say-never attitude that propelled this dissertation and our team with such enthusiasm.

Special thanks to Dr. Stan Szpakowicz for the invaluable support, feedback and immense assistance in the metaphor detection chapter of this research.

Many thanks to my research team Bryan Paget and Blair Drummond for their help and creative ideas.

Special thanks to Ehsan Amjadian, Prasadith Buddhitha, Haifa Alharthy, Zunaira Jamil, Hanqing Zhou, Parinaz Sobhani, Jelber Shirabad, and Romualdo Alves for the help and feedback. Thanks to all the participants and presenters of TAMALE seminars and NLP-chat group meetings for sharing ideas, debating, and providing feedback.

Thanks to Dr. Ekaterina Shutova and Dr. Saif M. Mohammad for providing their metaphor datasets.

Thanks to the Social Sciences and Humanities Research Council of Canada and the Natural Sciences and Engineering Research Council of Canada for providing funding for this research.

Finally, I thank my beautiful wife Prachi for her support, understanding, patience, and encouragement throughout my research. And last but not the least, my parents for their faith in me and my abilities.

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	viii
LIST OF FIGURES . . . . .	x
LIST OF ABBREVIATIONS . . . . .	xi
CHAPTER 1 INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	1
1.2 Goals . . . . .	2
1.3 Intended Contributions . . . . .	3
1.4 Research Hypotheses . . . . .	3
1.5 Outline . . . . .	4
CHAPTER 2 BACKGROUND . . . . .	5
2.1 Natural Language Processing . . . . .	5
2.2 Feature Engineering for Textual Data . . . . .	5
2.3 NLP Tasks . . . . .	11
2.4 Machine Learning . . . . .	13
2.5 Deep Learning . . . . .	16
2.6 Word Embeddings . . . . .	20
CHAPTER 3 RELATED WORK ON POETRY COMPUTATIONAL ANALYSIS . . . . .	23
CHAPTER 4 RHYME CLASSIFICATION . . . . .	27
4.1 Introduction . . . . .	27
4.2 Related Work . . . . .	28
4.3 Background . . . . .	29
4.4 Methodology . . . . .	32
4.5 Results . . . . .	35
4.6 Web Application for Rhyme Visualization . . . . .	37
4.7 Conclusion and Future Work . . . . .	39

CHAPTER 5	DICTION ANALYSIS . . . . .	42
5.1	Introduction . . . . .	42
5.2	Related Work . . . . .	43
5.3	Method . . . . .	43
5.4	Example of Results . . . . .	51
5.5	Web Application for Diction Analysis . . . . .	54
5.6	Conclusion and Future Work . . . . .	55
CHAPTER 6	METAPHOR DETECTION . . . . .	57
6.1	Introduction . . . . .	57
6.2	Related Work . . . . .	58
6.3	Method . . . . .	59
6.4	Results . . . . .	64
6.5	Error Analysis . . . . .	68
6.6	Deep Learning Classification . . . . .	69
6.7	Web Application for Generic Metaphor Detection . . . . .	71
6.8	Conclusion and Future Work . . . . .	75
CHAPTER 7	CONCLUSIONS AND FUTURE WORK . . . . .	77
7.1	Conclusions . . . . .	77
7.2	Future Work . . . . .	78
REFERENCES	. . . . .	80

# LIST OF TABLES

2.1	Alphabetical list of part-of-speech tags used in the Penn Treebank Project. . . . .	8
4.1	ARPAbet phoneme set. Ph denotes phonemes, Ex denotes example words and Tr denotes translations. . . . .	30
4.2	Full Rhyme Example. . . . .	31
4.3	Slant Rhyme Example. . . . .	31
4.4	Rich Rhyme Example. . . . .	32
4.5	Mosaic Rhyme Example. . . . .	32
4.6	Multiple pronunciation example. . . . .	33
4.7	Rhyme Example. . . . .	33
4.8	Rhyme Weights. . . . .	34
4.9	Dataset statistics and classification results. . . . .	35
4.10	Poem RhymeScore. . . . .	36
4.11	Poem RhymeScore. . . . .	37
4.12	Poem RhymeScore. . . . .	37
5.1	LOVE in GloVe model . . . . .	50
5.2	LOVE in PoFo model . . . . .	50
5.3	SEE in GloVe model . . . . .	51
5.4	SEE in PoFo model . . . . .	51
6.1	Results for the class <i>metaphor</i> . . . . .	64
6.2	Results for classifiers trained on PoFo+TroFi+Shutova data, and tested on the 487 poetry sentences (sorted on <i>metaphor</i> F-score) . . . . .	65
6.3	Statistical significance tests for the classification results given in Table 6.2. We compare each row with the previous row via t-tests, based on the F-score for the <i>metaphor</i> class. . . . .	66
6.4	Results for the class <i>non-metaphor</i> . . . . .	66
6.5	Results of the direct comparison with related work (Rule+Stat = rule-based and statistical) . . . . .	67
6.6	A selection of incorrectly predicted PoFo sentences (L = literal, M = metaphorical) . . . . .	68
6.7	Ranges of parameters tested. . . . .	70



6.8	Top results for CNN classification tested on variable hyper-parameters where e denotes epochs, b denotes batch size, n denotes the number of neurons in 1st layer and i denotes the number of inputs. All other hyper-parameter remain constant, input activation : ReLU and output activation : SOFTMAX. The difference between the first and the last row is statistically significant ( $p < 0.05$ ). . . . .	71
-----	--	----

# LIST OF FIGURES

2.1	Stanford basic dependencies example . . . . .	6
2.2	A natural language classifier . . . . .	11
2.3	Layers in a convolutional neural network. . . . .	18
2.4	An unrolled recurrent neural network. . . . .	20
2.5	Schematic diagram of the CBOW model. . . . .	21
2.6	Architecture diagram of the Skip-gram model. . . . .	22
4.1	A poem (excerpt) converted to its phonetic transcription. Title: Summer begins to have the look. Poet: Emily Dickinson	31
4.2	A 3D scatter plot of PoFo poems . . . . .	38
4.3	A 2D scatter plot of PoFo poems . . . . .	39
4.4	Poem RhymeScore displayed on mouse over event in the plot .	40
4.5	On clicking a poem node, that specific poem is displayed with rhyme types in colored legend. . . . .	40
5.1	WordNet hypernym hierarchy for <i>happiness</i> (abstract) and <i>shark</i> (concrete). . . . .	44
5.2	Screenshot of the diction analysis web application that quantifies linguistic quantitative features. . . . .	54
5.3	Screenshot of the concordance web application that uses semantic query expansion to retrieve poems from PoFo corpus.	55
6.1	Schema diagram of the CNN text classifier. . . . .	70
6.2	Screenshot of the single-line metaphor detection web application.	73
6.3	Screenshot of the multi-line metaphor detection web appli- cation. The full text is not captured in this screenshot. . . . .	73

# LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
DL	Deep Learning
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbor
LSTM	Long Short Term Memory
ML	Machine Learning
NLP	Natural Language Processing
POS	Part of Speech
PoFo	Poetry Foundation
ReLU	Rectified Linear Unit
SVM	Support Vector Machines

# CHAPTER 1

## INTRODUCTION

”Genuine poetry can communicate before it is understood.” - T.S. Eliot.

Poetry has existed even before humans could read or write. It is said to be the most free literary genre. While all the other genres need characters, plot or narrative, poetry is free from all these restrictions. A poem can be as abstract or as specific as the poet needs it to be. The words in a poem may not be employed in their literal sense and may have a deeper contextual connotation. This is what makes poetry intriguing for computational linguistics.

### 1.1 Motivation

We cannot have enough data for prediction. This problem is the highlight of any Artificial Intelligence algorithm or software in the present day. The more data we have, the better is the prediction, but this increased precision has a cost in terms of computational cost. This results in the development of specialized corpora that work for a very focused number of tasks. In this context, the place of poetry is quite fascinating. Poetry is abundant with emotions, despite being succinct or terse. The level of ambiguity and subtlety is remarkable. The complexity of computational study of poetry is the prime motivation for this dissertation. Even the most basic rules for common tasks like sentence boundary detection fail in the case of poetry since capitalization, punctuation, end-of-line do not provide the complete picture, as is the case with other genres like news articles or novels. And we can develop algorithms for poetry that generalize well for non-poetry data as well. This is quite interesting as it motivates us to apply new language statistical models for poetry and get immediate results.

The motivation for this poetry computational analysis project and for applying graph theory in poetry arose when a collection of poems named *Nomadosophy* was published (MARGENTO, 2012). Initially, the author manually identified the connections between various poems and depicted them as a graph poem. But later it was observed that those tasks can be achieved computationally through the use of advanced state-of-the-art NLP techniques. A graph poem is a visual representation where poems are connected to each other by edges that denote similarity features intrinsic to poetry like rhyme, meter, topic, diction, tropes, etc. These features are what uniquely differentiates a poem from another poem and aid in their interpretation and understanding.

Moreover, it has been observed that the field of computational linguistics has not contributed much to the classification and analysis of poetry. Poetry being such a complex literary genre demands that we indeed apply these novel techniques. There have been approaches to applying them to poetry but most of them are limited to a specific genre or to a certain time period. This dissertation takes on a more holistic approach to poetic analysis.

To the best of our knowledge, the *GraphPoem* is the first and the only project with such a holistic computational approach to poetry analysis. A lot of tasks that have been carried out as an integral part of this dissertation or that we intend to carry out in the future, have never been done before. This in itself is a big motivation.

## 1.2 Goals

The goal of this thesis is to augment the study of poetry by applying Natural Language Processing, Machine Learning, and Deep Learning techniques. It aims to build the framework on which poets and others can quantify, visualize, and retrieve poetry on some similarity features. These features can be meter, stanzaic patterns, verse forms, sonic devices (rhyme, alliteration, euphonious characteristics in general), and style (diction, tropes, syntax and anatomy of line breaks) (Tanasescu et al., 2016). Of all these features that can be used for classification, this dissertation focuses on rhyme, diction, and metaphor.

## 1.3 Intended Contributions

The intended contributions of this thesis are as follows:

1. We introduce a rhyme quantification metric called RhymeScore.
2. We develop a novel alt-n rhyme detection system.
3. We develop a novel diction classification system with multiple features like verbal density, inflection ratio, etc. and build word embeddings on our poetry corpus.
4. We introduce a metaphor poetry dataset. This dataset is for a single type of metaphor and was annotated by us. We specifically developed it for poetry data, because all existing metaphor datasets are for other types of text.
5. We develop a novel metaphor detection using word embeddings as features for machine learning classifiers. We also experiment with deep learning classifiers.

## 1.4 Research Hypotheses

Here are the research hypotheses that we want to prove:

1. Automatic rhyme analysis and rhyme quantification is possible.
2. Poetic diction features can be quantified and visualized.
3. Automatic metaphor detection in poetry is possible by using word embeddings as features for classification.

## 1.5 Outline

This dissertation is divided into 7 chapters. Chapter 1 is the current chapter that introduces our tasks and the motivation to work on them. Chapter 2 explains the concepts and techniques that form the basis of this dissertation. Chapter 3 enumerates the related works in computational poetry that have inspired or influenced our work. In chapter 4, we explain about rhyme classification in its sub-sections. In chapter 5, we talk about diction analysis with all of the features enumerated, except metaphor. Though metaphor is a part of diction analysis, due to the complexity of its detection, we explain our work on metaphor detection in Chapter 6. Lastly, we conclude and discuss future directions of research in Chapter 7.

Parts of Chapter 6 are published with the title *Metaphor Detection in a Poetry Corpus* in Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (pp. 1-9) at ACL (Annual Meeting of the Association for Computational Linguistics) 2017, Vancouver, Canada (Kesarwani et al., 2017).

Parts of Chapter 4, 5, 6 are published with the title *Access(ed) Poetry. The Graph Poem Project and the Place of Poetry in Digital Humanities* in DH2017 (Digital Humanities 2017) at McGill University in Montreal, Canada (Tanasescu et al., 2017).

# CHAPTER 2

## BACKGROUND

Note: This chapter can be skipped safely by a reader who has a background in NLP, Machine Learning, and Deep Learning. This chapter touches upon the concepts and techniques used in this thesis in a succinct manner.

### 2.1 Natural Language Processing

Natural Language Processing (NLP) is at the intersection of Computer Science, Artificial Intelligence, and Linguistics and it consists of processes for analyzing, understanding, and deriving meaningful information from textual data. By utilizing NLP and its components, one can process and analyze large amounts of textual data and solve a wide range of problems such as sentiment analysis, machine translation, named entity recognition, relation extraction, automatic summarization, speech recognition, and topic modeling.

Since text is a highly unstructured form of data, it has various types of noise and it is not useful until data pre-processing is performed on it. This process of cleaning noise from text, converting it into a standard format and ready for analysis is known as *text preprocessing*.

### 2.2 Feature Engineering for Textual Data

Preprocessed data needs to be converted into machine-readable features. Depending on the usage, following techniques can be utilized for this conversion:

- Syntactic parsing
- Entities / n-grams / word-based features



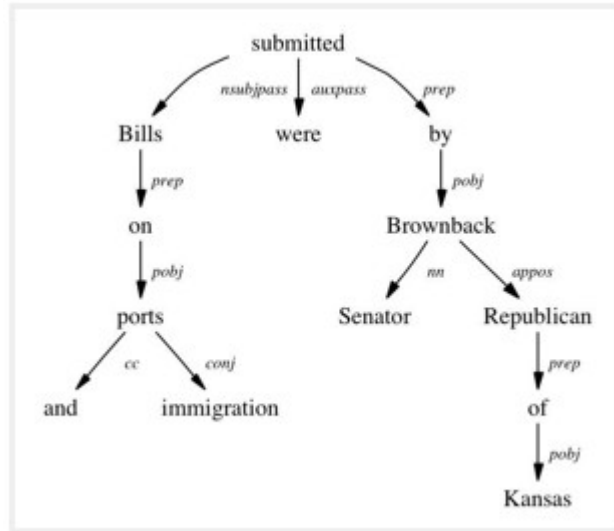


Figure 2.1: Stanford basic dependencies example

- Statistical features
- Word embeddings

### 2.2.1 Syntactic Parsing

Syntactic parsing involves the arrangement of sentences in a manner that all the dependency relations and grammar associations between the words are shown. These are very important to understand the structure of the sentences.

**Dependency Trees:** Sentences are words sequences. A basic dependency grammar determines the relationship between the words in a sentence. A dependency grammar is a category of syntactic analysis that involves labeling of asymmetrical binary relations between two lexical items (words). Every relation can be represented in the form of a triplet (relation, governor, dependent). For example<sup>1</sup>, consider the sentence *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas*. The relation between the words can be seen in the form of a tree representation as shown in Figure 2.1.

<sup>1</sup><https://nlp.stanford.edu/software/stanford-dependencies.shtml>

The tree shows that *submitted* is the root word of this sentence, and is linked by two sub-trees (subject and object subtrees). Each subtree in itself is a dependency tree with relations such as (*Bills* – *ports* by *proposition* relation), (*ports* – *immigration* by *conjugation* relation).

On parsing this type of tree recursively in a top-down manner, grammar relation triplets are obtained. These triplets can be used as features in many NLP problems like entity recognition, entity-wise sentiment analysis, and text classification. Two tools to generate dependency trees are StanfordCoreNLP (Manning et al., 2014) (from the Stanford NLP Group) and NLTK (Bird et al., 2009) dependency parser.

**Part-of-speech tagging:** In addition to the dependency relations, every word in a sentence is also associated with a part of speech (POS) tag like adjective, adverb, noun, verb, etc. These POS tags define the grammar function of a word in the sentence. Table 2.1 shows the list of POS tags defined by the Penn TreeBank Project (not including punctuation tags).

POS tagging is used for the following important tasks in NLP:

**Word sense disambiguation:** Some words in a language have multiple meanings depending upon their context and usage in a sentence. For example, in the two sentences below:

- Please book my flight.
- I am going to read this book in the flight.

The word *book* is used in a different context in both sentences, with different POS. In the first sentence, the word *book* is used as a verb, while in the second, it is used as a noun. Lesk’s algorithm (Lesk, 1986) is an algorithm for resolving these disambiguation cases.

**Improving word-based features:** Words are in itself very useful features, but when appended with POS tags, the resulting features are even more powerful, as they are preserving the context in a better way.

**Normalization and Lemmatization:** POS tags are very useful in the lemmatization process for converting a word to its lemma or base form.

<b>Tag</b>	<b>Description</b>
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VCN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

Table 2.1: Alphabetical list of part-of-speech tags used in the Penn Treebank Project.

**Efficient stopwords removal:** POS tags are also useful in removal of stopwords. Some POS tags can be easily removed without much loss in meaning like *IN* that denotes prepositions or subordinating conjunctions in the Penn Treebank tag list.

## 2.2.2 Entities / N-grams / Word-based Features

Entities are the constituent parts of a sentence, like noun phrase, verb phrase, etc. Entity Detection algorithms are an ensemble of rule-based parsing, POS tagging, lexicon lookup, and dependency parsing. Entity detection are used in automated chat bots, content analyzers, etc.

Some important entity detection methods in NLP are:

- Named Entity Recognition
- Topic Modeling
- N-Grams as Features

**Named Entity Recognition:** NER is the process of obtaining named entities such as person names, location names, date time, etc. from text. For example:

*Steve Jobs, the founder of Apple Inc. is walking on the streets of San Jose.*

Named Entities : *person : Steve Jobs, org : Apple Inc., location : San Jose*

**Topic Modeling:** Topic modeling is a process that identifies the topics present in a text document by detecting the latent pattern within the words. It is an unsupervised approach. Topics are defined as *a repeating pattern of co-occurring terms in a corpus*. A good topic model will classify words like *school, college, teaching* in the topic *Education* and so on. Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a popular topic modeling technique.

**N-Grams as Features:** N-Gram is a contiguous sequence of N words together. For N=1, it is termed as unigrams, N=2 are termed as bigrams, and so on.

### 2.2.3 Statistical Features

There are several techniques for quantification of words into numbers:

**Term Frequency–Inverse Document Frequency (TF-IDF):** TF-IDF is a term-weighted model that aims to convert the text documents into vectors (in order to represent documents as vector models) on the basis of occurrence of words in the documents by considering the corpus as a bag of words (BOW). A BOW model is such that it does not consider the exact ordering of words but just their frequency. For a dataset of  $N$  text documents, in any document  $D$ , TF and IDF will be defined as:

Term Frequency (TF) : TF for a term  $t$  is defined as the count of a term  $t$  in a document  $D$ .

Inverse Document Frequency (IDF) : IDF for a term is defined as the logarithm of the ratio of total documents available in the corpus and number of documents containing the term  $T$ .

TF–IDF formula gives the relative importance of a term in a corpus (list of documents), given by the following formula below:

$$W_{i,j} = tf_{i,j} \times \log \frac{N}{df_i}$$

$tf_{i,j}$  = number of occurrences of term $_i$  in document $_j$

$df_i$  = number of documents containing term $_i$

$N$  = total number of documents

### 2.2.4 Word Embeddings

Word embeddings are models where words are represented as vectors of real numbers in a low-dimensional vector space. Since the words are represented as vectors, various compositionality operations can be performed (Mikolov et al., 2013b) or similarity metrics or equivalence relations (word associations) can be computed.

Word embeddings will be explained in detail in Section 2.6.

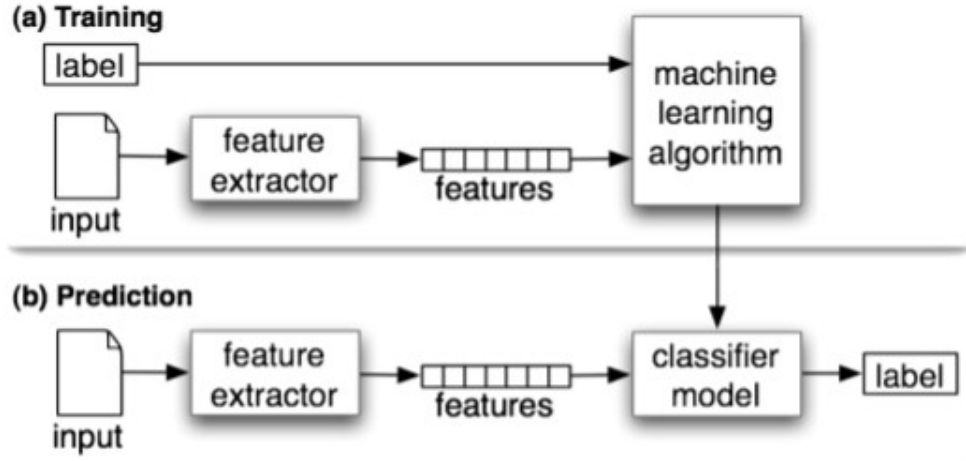


Figure 2.2: A natural language classifier

## 2.3 NLP Tasks

This section enumerates several applications in the field of natural language processing that have been used in this dissertation.

### 2.3.1 Text Classification

Text classification is a technique to classify a text object (document or sentence) in one of a fixed number of classes or categories. It is applied to even larger amounts of text and helps to organize text for information filtering and other tasks. Some applications include topic classification of news articles, sentiment classification, email spam detection, etc.

A natural language classifier consists of two parts: (a) Training a classifier (b) Predictions on new texts, as shown in figure 2.2. First, the text input is processed and features are generated. Then the machine learning model learns the association between features and the categories while training. During the prediction stage, the model makes predictions on new instances based on the training data.

The quality of predictions is dependent on the number of instances in training data and the quality of the generated features. Therefore, to improve the prediction, a lot of training data is needed.

### 2.3.2 Text Similarity

Text Similarity is a very important area of NLP that involves matching of text objects to find similarities. Some applications include semantic analysis, data de-duplication, etc. There are a number of text matching techniques. Some of them are as follows:

- Levenshtein Distance
- Phonetic Matching
- Cosine Similarity
- Block Distance
- Euclidean distance
- Latent Semantic Analysis

**Levenshtein Distance** : The Levenshtein distance (Yujian and Bo, 2007) between two strings is defined as the minimum number of edits required to convert one string into the other, with the allowable edit operations being an insertion, deletion, or substitution of a single character.

**Phonetic Matching** : A Phonetic matching algorithm (Zobel and Dart, 1996) takes a keyword as input (location, name, etc.) and produces a list of phonetically similar words. It is very useful for spell check, searching large text corpora, and matching relevant names. Soundex (Odell and Russell, 1918) and Metaphone (Philips, 2000) are examples of phonetic algorithms used for this purpose.

**Cosine Similarity** : If the text is represented as vectors, cosine similarity measures the similarity between the two word/phrase vectors. This is very useful in question answering, recommender systems, etc.

**Block Distance** : Block Distance (Krause, 1975) is also known as Manhattan distance. It computes the distance that would be traveled to get from one data point to the other if a grid-like path is followed. The Block distance between two items is the sum of the differences of their corresponding components (Gomaa and Fahmy, 2013).

**Euclidean distance** : Euclidean distance or L2 distance is the square root of the sum of squared differences between corresponding elements of the two vectors.

**Latent Semantic Analysis** : LSA (Landauer and Dumais, 1997) is the most popular technique of Corpus-Based similarity. LSA assumes that words that are close in meaning will occur in similar pieces of text. A matrix containing word counts per paragraph (rows represent unique words and columns represent each paragraph) is constructed from a large piece of text and a mathematical technique called singular value decomposition (De Lathauwer et al., 2000) (SVD) is used to reduce the number of columns while preserving the similarity structure among rows. Words are then compared by taking the cosine of the angle between the two vectors formed by any two rows (Gomaa and Fahmy, 2013).

### 2.3.3 Coreference Resolution

Coreference resolution (Soon et al., 2001) is a process of resolving relational links among the objects (or words) within a sentence. Consider an example:

*Michael went to Peter's office to see the new board. He looked at it for an hour.*

Humans can easily figure out that *he* denotes Michael (and not Peter), and that *it* denotes the board (and not Peter's office). By using Coreference resolution, this task can be done automatically. It is used in question answering, document summarization, and information extraction.

## 2.4 Machine Learning

Machine learning (ML) is a subfield of Artificial Intelligence that allows software applications to do predictions by learning from data points. By using machine learning, one can build algorithms that can receive input data and learn patterns from the data in order to do statistical analysis.

There are broadly three types of ML algorithms:

- **Supervised Learning** : These algorithms consist of classes/categories that are to be predicted from a labeled set of predictors. By using the labeled set, a function is generated that maps input values to output classes. Usually, the training process continues until a minimum loss threshold is reached. Examples: Decision Tree (Quinlan, 2014),



Random Forest (Breiman, 2001), Logistic Regression, Support Vector Machine (Boser et al., 1992), etc.

- Unsupervised Learning : In these algorithms, we do not have any labeled data or categories for prediction. The data points need to be grouped or clustered together in  $k$  clusters. Example : K-means algorithm (Arthur and Vassilvitskii, 2007).
- Reinforcement Learning : Using these algorithms, a machine is trained to reward certain actions, while penalizing others. The machine is designed so that the rewarded actions are maximized and losses minimized. Therefore, it is able to make smart decisions and continually learn from past failures. Example : Markov Decision Process (Levin et al., 1998).

Some commonly used machine learning algorithms are:

### 2.4.1 Linear Regression

Linear Regression is used to estimate real values (price of an item, future sales, etc.) based on continuous variable(s). A linear function is used to map the input values to the output variables by fitting a best line. This best fit line is known as the regression line and is represented by the equation  $Y = a * X + b$  (Kutner et al., 2004).

### 2.4.2 Logistic Regression

Logistic Regression (Cox, 1958) is used to estimate discrete values (yes/no, true/false, binary values like 0/1) based on a given set of input variable(s). It is also known as logit regression as it predicts the probability of the occurrence of an event by fitting data to a logit function. Its output values lie between 0 and 1.

### 2.4.3 Decision Tree

Decision Tree (Quinlan, 2014) is a type of supervised learning algorithm. In this algorithm, we split the data into two or more heterogeneous groups based

on various techniques like Info Gain, Chi-square, Entropy, etc. The aim is to find the most significant attributes and make as distinct heterogeneous groups as possible. Overfitting is a major problem in most decision tree models and tree pruning is done to overcome it.

#### 2.4.4 Support Vector Machine

Support Vector Machine (SVM) (Boser et al., 1992) is a supervised machine learning algorithm that is based on the objective of finding a hyperplane that best divides a dataset into two classes. A hyperplane is a multidimensional plane that linearly separates and classifies a set of data points. A hyperplane is dependent on support vectors that are the data points nearest to it. If there are many hyperplanes for a dataset, the hyperplane with the maximum margin (distance between the hyperplane and the nearest data points) is chosen. SVM is very popular for classification tasks and works well for normal sized datasets. Training time may be high for very large datasets. It is less effective if the dataset is not linearly separable and on noisier datasets.

#### 2.4.5 Naive Bayes

Naive Bayes (John and Langley, 1995) is a classification technique that is based on Bayes theorem and assumes that the predictors (features) are independent of each other. In other words, it assumes that the presence of a feature in a class is unrelated to the presence of any other feature. Naive Bayes makes this assumption, in order to simplify the computation and assumes independence even if the features are related to one another.

#### 2.4.6 K-Nearest Neighbors

K-Nearest Neighbors (KNN) (Aha et al., 1991) is a simple algorithm that classifies a new data points based on the majority vote of its k neighbors. The data point is assigned to the most common class among its k neighbors that is measured by a distance function. These distance functions can be Manhattan, Euclidean, Minkowski, etc. KNN is very susceptible to noise

and outliers, therefore such points should be normalized in preprocessing stages. KNN can be used for classification and regression.

### 2.4.7 K-Means Clustering

K-Means (Arthur and Vassilvitskii, 2007) is a type of unsupervised algorithm that is useful in clustering of data. The aim of this algorithm is to find groups in the data, where  $k$  is the number of groups or clusters. The algorithm works iteratively to assign each data point to one of the  $k$  groups based on the features provided such that the cluster remains homogeneous inside and heterogeneous to other groups.

### 2.4.8 Random Forest

Random Forest (Breiman, 2001) is a term used for an ensemble of decision trees. In this algorithm, a collection of decision trees is termed as a *forest*. Each tree in a random forest gives a classification result or a *vote* for a class, to classify a new object based on attributes. The class that obtains the most votes from all the trees in the forest is the winner in classification for that object.

## 2.5 Deep Learning

Deep learning is a class of machine learning algorithms that (Deng et al., 2014):

- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manner.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.
- use some form of gradient descent for training via backpropagation.

Some commonly used terms in deep learning are explained below:

- **Activation Function** : Also called the transfer function, it is applied to some of the layers of a neural network to allow it to learn complex decision boundaries. Some commonly used activation functions are ReLU (Rectified Linear Unit), tanh, sigmoid, etc.
- **Attention Mechanism** : It is inspired by human visual attention, that is the ability to focus on specific parts of an image. Attention mechanism can be utilized in image recognition and language processing to help the network learn what to *focus* on when making predictions.
- **Backpropagation** : It is an algorithm to efficiently calculate the gradients in a feedforward neural network. It involves starting from the final layer of weights and propagating the gradients backwards to the previous layer. This backward flow of the error information allows for efficient computation of the gradient at each layer.
- **Batches** : While training a neural network, instead of sending the entire input at once, it is divided into several parts of equal size randomly. Training the data in batches makes the model more generalized as compared to the model built when the entire data is input to the network at once.
- **Dropout** : It is a regularization technique for neural networks that prevents overfitting (Srivastava et al., 2014). It prevents neurons from co-adapting (highlighting the same features repeatedly) by randomly setting a fraction of them to zero output at each training iteration. Dropout can be applied to CNNs and recurrent networks.
- **Epochs** : An epoch is defined as a single training iteration of all batches in both forward and backpropagation. One epoch is a single forward and backward pass of the entire input data. At the end of one epoch, the network will have been exposed to every record in the dataset once.
- **LSTM** : Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) networks were invented to prevent the vanishing gradient problem in RNNs by using a memory gating mechanism. By using LSTM, we help

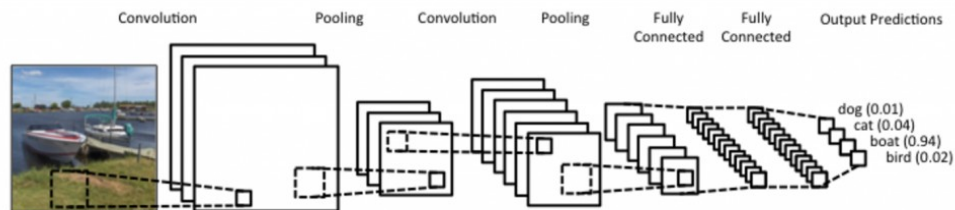


Figure 2.3: Layers in a convolutional neural network.

the network to efficiently propagate gradients and learn long range dependencies to calculate the hidden states in an RNN.

- Pooling : It is periodically introduced between the convolution layers to reduce the number of parameters and to prevent over-fitting. The most common type is a pooling layer of filter size (2, 2) using the MAX operation. Generally, max pooling works better than average pooling.
- ReLU : Rectified Linear Units are often used as activation functions in deep neural networks (Nair and Hinton, 2010). They are defined by  $f(x) = \max(0, x)$ . The advantages of ReLUs over other activations like tanh are that they tend to be sparse (their activation is easy to be set to 0), and that they suffer less from the vanishing gradient problem.
- Sigmoid : Sigmoid is a widely used activation function. It is of the form  $f(x) = \frac{1}{1+e^{-x}}$ . This is a smooth function and is continuously differentiable. The biggest advantage that it has over linear function is that it introduces non-linearity and hence the network can learn complex patterns.
- Softmax : The softmax function is used to convert a vector of raw scores into class probabilities at the output layer of a neural network used for classification. It is of the form  $f(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$ . It normalizes the scores by exponentiating and dividing by a normalization constant.

### 2.5.1 Convolutional Neural Networks

A convolutional neural network (CNN) (LeCun et al., 1998) contains one or more convolutional layers, pooling or fully connected, with nonlinear acti-

vation functions like ReLU (Nair and Hinton, 2010) or tanh applied to the results. In a traditional feedforward neural network (simplest form of network without any cycles or loops), each input neuron is connected to each output neuron in the next layer, which is called a fully connected layer. In CNNs, instead, convolutions over the input layer are used to compute the output and this results in local connections, where each input region is connected to a neuron in the output layer. Each convolution layer applies different filters, typically in hundreds or thousands, and the results are combined. During training, a CNN automatically learns the values of the filters based on the respective task. For example, in image classification, in the first layer a CNN may learn to detect edges from raw pixels. Then in second layer, use the edges to detect simple shapes. And then in higher layers, use these shapes to detect higher level features like facial shapes. The last layer is then a classifier that utilizes these high-level features to make predictions, as shown in figure 2.3.

In the context of NLP, instead of image pixels, the input to most tasks are sentences or documents represented as a matrix. Each row of the matrix is a vector that corresponds to one token, typically a word. Typically, these vectors are word embeddings (low-dimensional representations) like Word2Vec (Mikolov et al., 2013a) or GloVe (Pennington et al., 2014), but they could also be one-hot vectors that index the word into a vocabulary. For example, for a 20-word sentence using a 100-dimensional embedding, we would have a 20x100 matrix as our input.

## 2.5.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are used for sequential data where the order of input/output is important. These networks have loops within them and the previous output is used to predict the next one. The loops within the neurons in the hidden layers provide them with memory or the capability to store information for a transient duration in order to predict the output. The output of the hidden layer is sent iteratively to the next hidden layer for  $t$  timestamps. The unfolded neuron is shown in figure 2.4. After completing all the timestamps, the output of the recurrent neuron goes to the next layer. Since, the previous information is stored for a short duration, these networks

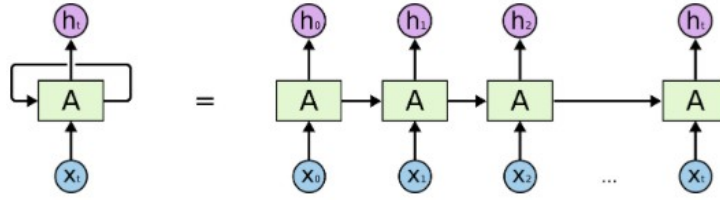


Figure 2.4: An unrolled recurrent neural network.

can learn sequences and encode contextual information for later decoding.

## 2.6 Word Embeddings

Word embeddings are a novel way of representing words as vectors in which a high dimensional word feature space is converted to a low dimensional one and the context is also preserved. Word embeddings are often used as input to weighted layers of a Convolutional Neural network or Recurrent Neural network.

The two popular models for word embeddings are Word2Vec (Mikolov et al., 2013a) and GloVe (Pennington et al., 2014). We can either use the pre-trained models or train our own word embeddings, if needed. These models take text as input and provide the word vectors as output for each word in the corpus.

Word2Vec model is composed of a preprocessing module, a shallow neural network model called Continuous Bag of Words (Mikolov et al., 2013a), and another shallow neural network model called Skip-gram (Mikolov et al., 2013a). The model first constructs a vocabulary from the training corpus and then learns word embedding representations from them. The vectors obtained from the model can be used as features/attributes for classification tasks, measuring text similarity using cosine similarity, etc.

### 2.6.1 Continuous Bag of Words Model

Continuous Bag of Words model (CBOW) is a shallow neural network that predicts the probability of a word given a context. A context may be defined

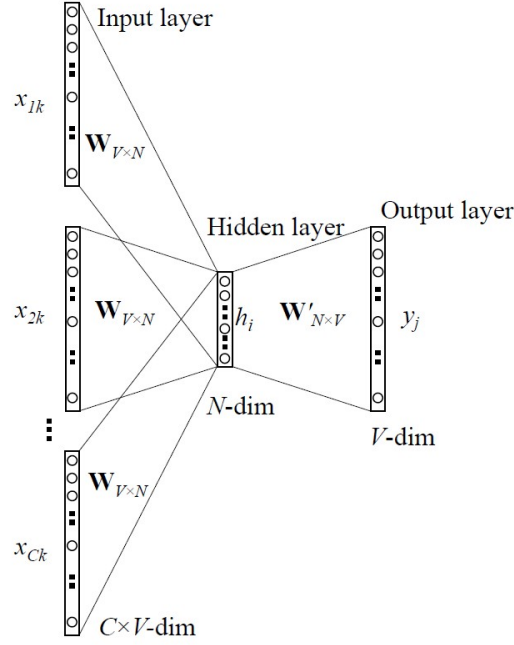


Figure 2.5: Schematic diagram of the CBOW model.

as a single word or a group of words that are in the vicinity of the word in question. First, a one-hot encoded input matrix is sent into this network with three layers: an input layer, a hidden layer, and an output layer. The output layer is a softmax function that is used to convert raw values to class probabilities. Figure 2.5 shows a schematic diagram of the CBOW model. The major advantage of CBOW is that it does not need much memory to store matrices and is computationally economical.

### 2.6.2 Skip-Gram Model

The skip-gram model is also a shallow neural network, but it has the reverse architecture of CBOW. The objective here is to predict the context given a word. The input vector for skip-gram is similar to a 1-context CBOW model. The loss function is the same as in CBOW model. It also contains three layers: an input layer, a hidden layer, and an output layer that contains neurons equal to the number of context words. Figure 2.6 shows the architecture diagram of the skip-gram model. The major advantage of skip-gram model is that it can learn multiple representations for the same word depending on



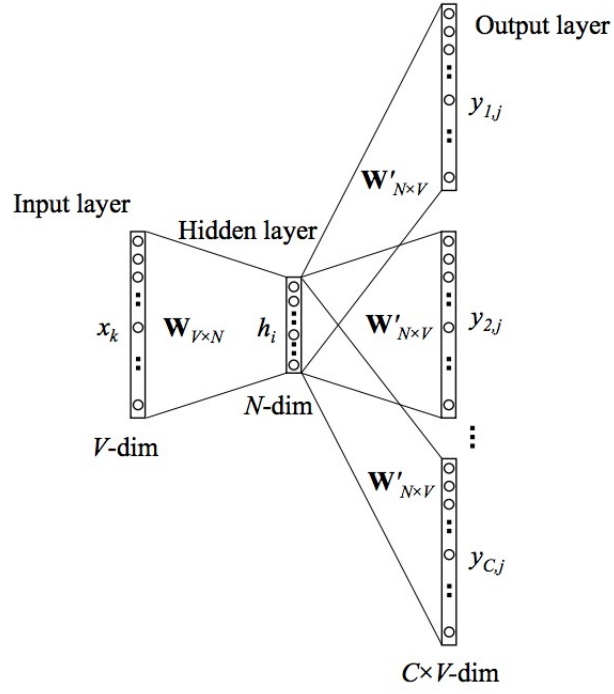


Figure 2.6: Architecture diagram of the Skip-gram model.

the context or usage of the word.

# CHAPTER 3

## RELATED WORK ON POETRY COMPUTATIONAL ANALYSIS

There have been numerous works on computational poetry in the past that have aimed to analyze poetry. SPARSAR (Delmonte and Prati, 2014) is one such system that aims to study poetry by the use of NLP tools like tokenizers, sentence splitters, NER (Named Entity Recognition) tools, and taggers. In addition, the system adds syntactic and semantic structural analysis and prosodic modeling. The authors analyze poems by computing metrical structure and rhyming scheme. Their corpus of 500 poems is smaller than ours and their evaluation is done by randomly sampling 50 poems out of 500. Our system includes 12,830 poems from Poetry Foundation(PoFo) and aims to not just perform semantic and syntactic analysis of poetry, but also visualize the dataset as a network graph. Delmonte (2015) generates three relational views of a poem: phonetic, poetic and semantic. Each of these views shows a color-coded line-by-line representation of a single poem, whereas our system aims to compare multiple poems.

POEMAGE (McCurdy et al., 2016), another system for poetry analysis is closer to ours in the sense that it aims to visualize the sonic elements of a poem primarily rhyme types as a path view diagram. While we consider sound an important factor for poem analysis, we consider it a pillar and not the foundation in multi-faceted poem analysis. For that reason, we delve into metaphor detection (explained in Chapter 6) considering it a semantic activity rather than a sonic one. Moreover, this generalization allows us to compare easily with related works in the fields of machine learning, deep learning, and NLP. Consequently, our algorithms work not just for poetry but also for other genres and other types of natural language texts.

VerseVis (Milton and Lu) is another interesting work that aims to visualize rhyme and meter in a poem in a color-coded visual representation where height represents stress on a word and color represents a phoneme. The major difference from our rhyme analysis (explained in Chapter 4) is that we do

automatic detection of rhyme by matching phonemes in a word, whereas VerseVis focuses on manual detection and just shows the phoneme distribution of a poem visually.

Emily (Madnani, 2005) is another tool for visual poetry analysis in which each of the poems is represented by a group of colored lines proportional to the poem’s length. A user can also perform an unweighted and weighted search on the poem content. We have also applied keyword search in our diction application, but we not only show the results for keyword occurrence but also semantically expand our query to include similar terms (explained in detail in Section 5.3.7). Abdul-Rahman et al. (2013) visualize a poem by phonetic and semantic connections with a latitudinal layout representation. They utilize sentiment ontology to attach semantic information to words and classify words grammatically. Musaoğlu et al. (2017) generate a *poetry barcode* that visualizes the structure of a poem with verb conjugations, passive/active verb usage, and emotional tone. They also visualize alliterations in a poem along with adjectives, word length, and verb tense. Meneses and Furuta (2015) visualize poetry by developing Graphwave that creates a wave-like pattern on all the unique terms of a poem; SentimentGraph and SentimentWheel that does sentiment analysis on a poem and represents it as a graph and wheel respectively.

All of the above systems visualize poems on an individual basis, but our system is more comprehensive as it takes all the poems at once and graphically visualizes them with connected edges. In other words, we do not aim to analyze poems individually, but as a corpus and connect them graphically based on quantified similarity features.

Kaplan and Blei (2007) developed a quantitative method to assess the style of American poems and visualization in relation to one another. They use metrics for orthographic, syntactic and phonemic features. For visualization, they use PCA (Principal Component Analysis) that reduces the dimensionality to two dimensions. Our diction analysis (explained in Chapter 5) is similar to their orthographic and syntactic features, but we did not apply PCA and SVD (Singular Value Decomposition) since they are computationally expensive. Instead, we built word embeddings by using Word2Vec(Mikolov et al., 2013a) on our PoFo corpus that reduces dimensionality without the resource overheads. Rhyme analysis (explained in Chapter 4) is similar to their phonemic feature, but we drill down to rhyme sub-types as well.

Lately, there has been growing interest in poetry generation by applying deep generative models like RNNs. One such system is Hafez (Ghazvininejad et al., 2017), that uses topical words (words related to a seed) to generate a poem of fixed length using an RNN decoder. A seed may be any word that decides the theme of that generated poem. There is style control as well that modulates encourage/discourage words, curse words, repetition, alliteration, word length, topical words, sentiment and concrete words. iPoet (Yan, 2016), another poetry generation system uses RNN encoder and decoder to compose a poem character-wise and additionally uses an iterative polishing mechanism to refine the generated poem to mimic a human poet. They also use evaluation metrics like PPL (Perplexity), BLEU (Bilingual Evaluation Understudy) and human evaluation to judge the quality of generated poems. Zhang and Lapata (2014) is another such system that generates Chinese poetry by using Recurrent Context model and Recurrent Generation model. Although our main purpose is not poetry generation, we nevertheless keep a scanner on the field to see which metrics are the most critical for relevance (and high scores according to our tools) of poems, and we make sure to include them in our complementary task of poetry analysis.

Manurung (2004) approached poetry generation as a state space search problem and hence applied evolutionary algorithms (mainly evaluation and evolution) for NLG (Natural Language Generation). He defines a poem as a natural language artifact which simultaneously fulfills the properties of *meaningfulness*, *grammaticality* and *poeticness*. While such rules may well be needed for NLG and poetry generation, they fail to address subtler issues such as the creativity and literary freedom of a poet. Our analysis in a way stems from this challenge, that for such a creative field like poetry, generation will always have to play catch up with human inventiveness and critical thinking. Moreover, our notion of poeticness is more comprehensive and holistic as we consider more features than just *rhythmic patterns and rhyme*.

Kantosalo and Riihiaho (2014) developed a tool called *Poem Machine* that aims to aid the poem writing process by generating a template poem with a specific theme (people, pets, nature, etc.) and suggesting rhyming words (based on assonance, consonance, full rhyme, and alliteration) for refining the generated poem word-by-word. This tool is a great example of how poem generation can help the poets to streamline the writing process instead of

replacing human creativity altogether.

Each of the next chapters of this thesis contains a dedicated related work section to describe the work for that specific task.

# CHAPTER 4

## RHYME CLASSIFICATION

### 4.1 Introduction

Rhyme is a repetition of similar sounds (or the same sound) in two or more words, most often in the final syllables of lines in poems and songs. Rhyme is one of the most important aspects of a poem and provides a poem with phonemic agreement. There are many ways that rhyme can be assessed in a poem. The two major ones that we will be working on are positional and phonemic.

#### **Positional rhyme types:**

1. End rhyme: Rhyme that occurs at line ends, which is the most used type of rhyme.
2. Internal rhyme: Rhyme that occurs within a single line or passage. Like the words *dreary* and *weary* in the example below:

Once upon a midnight *dreary*, while I pondered, weak and *weary*,  
Over many a quaint and curious volume of forgotten lore.

#### **Phonemic rhyme types:**

1. Full rhyme: Refers to the immediately recognizable form: *true/blue*, *mountain/ fountain*.
2. Slant rhyme: Refers to rhymes that are close but not exact: *lap/shape*, *glorious/nefarious*.

3. Identical rhyme: A word rhymes with itself like *ground* in below example:

We paused before a house that seemed,  
A Swelling of the *Ground*.  
The Roof was scarcely visible,  
The Cornice in the *Ground*.

4. Eye rhyme: Refers to rhymes based on similarity of spelling rather than sound. Example: *love/move/prove, why/envy*.
5. Rich rhyme: A word rhymes with its homonym: *blue/blew, guessed/guest*. (Ros, 2000)

## 4.2 Related Work

Most of the earlier work on rhyme detection in poetry was either based on rhyme patterns like AABBBCC, AABBCDD, etc. (Plamondon, 2006) (Delmonte, 2013) or based on just a few rhyme types (no rhyme sub-types) (Mittmann et al., 2016).

RhymeDesign (McCurdy et al., 2015) is a tool for analyzing sonic devices in a poem by expressing rhymes as combinations of object components onset O, nucleus N and coda C. They represent all rhyme sub-types as combinations of these components. Our work differs in the sense that we focus not only on transcribing rhymes but their quantification as well, so that they can be visualized on a graph. Another work by Reddy and Knight (2011) that is in certain respects similar to ours is the only one except our own that uses a rhyme quantification methodology, but the context is different. They use quantification with a generative model in performing machine translation of poems to another language while maintaining the rhyme and the meter in the process. Our focus is on the detection and visualization aspects, to augment the recommendation of poems based on their rhyme similarity.

Aoidos (Mittmann et al., 2016) is a system that analyzes rhymes in Portuguese poetry. The notable difference with our system is that rhyme sub-types are not analyzed. Portuguese being more phonetic than English,

presents less flexibility while transcribing words. Testing for Aoidos is based on human judges giving acceptable/unacceptable ratings for rhyme detected by the system.

SPARSAR, another system for automatic poetry analysis by Delmonte (2013) detects meter, rhyme and semantic structures of poetry. This system uses CMUDict (Weide, 2005) for phonetic transcription just like we did but uses rule-based patterns (AABBCC, AABCDD, etc.). Similarly, AnalysePoems by Plamondon (2006) also used rule-based patterns for classifying rhymes in poetry and this system handles a large corpus of poems (3162 poems) from Representative Poetry Online (<http://rpo.library.utoronto.ca>).

The poetic style analysis carried out by Kao and Jurafsky (2015) involves rhyme sub-types such as Identity/Perfect/Slant end rhyme, alliteration, consonance, and assonance that are very close to our sub-type categorization. But their work is focused on a selected set of 159 poems, written by 19th century non-Imagist and Imagist poets, to differentiate their style. Our work involves a significantly larger number of poems from various periods. Moreover, the patterns extracted, if any, will not be manual, but automatically-detected.

### 4.3 Background

To detect rhyme, we need the phonetic transcription of the poem. We use CMUDict (Weide, 2005), a pronunciation dictionary that has a set of 134,000 words and their pronunciations. Its phoneme set is based on ARPAbet that has a 39 class categorization as shown in Table 4.1. To denote stress, it has 3 levels: 0 - No stress, 1 - Primary stress and 2 - Secondary stress.

Sometimes, CMUDict fails to get pronunciation for a word for multiple reasons. One reason is that the word is in its inflectional form and a very rare one, thus it is absent from the dictionary. For these cases, we have to consider generating phonetic sequence based on some rules. Logios Lexical tool (CMU, 2007) is one solution. We have not used this tool as of this writing, but it can definitely be tried in future work. The second reason for failure is that some words in a poem have been coined by the poet or are the result of the poet’s own idiosyncratic use of language, hence they are not present in the dictionary. In other words, the connotation is captured by the



Ph	Ex	Tr
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY

Ph	Ex	Tr
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY
SH	she	SH IY
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY L D
Z	zee	Z IY
ZH	seizure	S IY ZH ER

Table 4.1: ARPAbet phoneme set. Ph denotes phonemes, Ex denotes example words and Tr denotes translations.

overall contextual setup and not by the individual word itself. Lastly, archaic words are very poorly represented in the phonetic dictionary. For all the above cases, rule-based phoneme generation is able to give an approximate pronunciation and may be enough to detect stress fall and rhyme. Figure 4.1 shows a poem excerpt with its phonetic transcription.

Rhyme is the similarity of phonemes and stress fall and is dependent on the pronunciation of the words in a poem. First, the position of the words is chosen based on the type of rhyme that is to be checked. If End rhyme is to be checked, then the last word of each line is considered. If Internal rhyme is to be checked, then the word just before the caesura and the last word of the line are considered. Caesuras are identified by detecting punctuation like comma, dot, and semicolon in the enclosed words in a line. If there are multiple punctuation marks, then only the first one is considered. End rhyme has two further sub-categories: Consecutive End and Alternate End rhyme. For Consecutive, the last words on consecutive lines are compared

Summer begins to have the look,	S_AH1_M_ER0B_IH0_G_IH1_N_ZT_UW1#T_IH0#T_AH0
Peruser of enchanting Book,	HH_AE1_VDH_AH0#DH_AH1#DH_IY0L_UH1_K
Reluctantly but sure perceives,	AH1_V#AH0_V
Again upon the backward leaves ,	EH0_N_CH_AE1_N_T_IH0_NG#EH0_N_CH_AE1_N_IH0_NG_B_UH1_K
Autumn begins to be inferred,	R_IH0_L_AH1_K_T_AH0_N_T_L_IY0B_AH1_TSH_UH1_R
By millinery of the cloud,	P_ER0_S_IY1_V_Z
Or deeper color in the shawl,	AH0#EY1G_EY1_NAH0_P_AA1_NDH_AH0#DH_AH1#DH_IY0
That wraps the everlasting hill.,	B_AE1_K_W_ER0_DL_IY1_V_Z
The eye begins its avarice,	AO1_T_AH0_MB_IH0_G_IH1_N_ZT_UW1#T_IH0#T_AH0
A meditation chastens speech,	B_IY1#B_IY0IH2_N_F_ER1_D
Some Dyer of a distant tree,	B_AY1AH1_V#AH0_VDH_AH0#DH_AH1#DH_IY0K_L_AW1_D
Resumes his gaudy industry.,	AO1_R#ER0D_IY1_P_ER0K_AH1_L_ER0#K_AO1_L_ER0
Conclusion is the course of All,	IH0_N#IH1_NDH_AH0#DH_AH1#DH_IY0SH_AO1_L
At most to be perennial,	DH_AE1_T#DH_AH0_TR_AE1_P_SDH_AH0#DH_AH1#DH_IY0
And then elude stability,	EH2_V_ER0_L_AE1_S_T_IH0_NG_HH_IH1_L
Recalls to immortality.	

Figure 4.1: A poem (excerpt) converted to its phonetic transcription. Title: Summer begins to have the look. Poet: Emily Dickinson

and for Alternate, the last words on alternate lines. Internal rhyme has no such sub-categorization.

End and Internal rhymes have another sub-categorization based on the nature of similarity of phonemes (Ros, 2000). Full rhyme refers to a perfect match as shown in Table 4.2.

True	T	R	UW1
Blue	B	L	UW1

Table 4.2: Full Rhyme Example.

In the above example, stress fall is exactly in the similar region of phonemes, hence we have a Full rhyme. Slant rhyme refers to a non-perfect match as shown in Table 4.3.

Nefarious	N	AH0	F	EH1	R	IY0	AH0	S
Glorious		G	L	AO1	R	IY0	AH0	S

Table 4.3: Slant Rhyme Example.

In the above example, although many end phonemes match, because primary stress does not fall on that part, it is a Slant rhyme. Rich rhyme refers to cases when a word rhymes with its homonym as shown in Table 4.4.

Blue	B	L	UW1
Blew	B	L	UW1

Table 4.4: Rich Rhyme Example.

In the above case, the two words have exactly the same phonemes. For all the above rhyme sub-types, the similarity is detected by phonetic matching. In Eye rhyme, instead of using phoneme matching, spelling matching is considered (Ros, 2000). For example, *why* and *envy* is an Eye rhyme pair. Identical rhyme refers to when a word rhymes with itself in different positions in a poem. There are other rhyme sub-types based on similarity like Assonant, Consonant, Scarce, and Macaronic rhymes. We have not implemented these as of this writing, but they can be included in our future work depending on their prominence in our corpus.

There is another category of rhyme based on similarity across word boundaries: Mosaic rhyme that refers to rhyme using more than one word as shown in Table 4.5. This type is also reserved for future work.

Dismay	D	IH0	S	M	EY1
This may	DH	IH1	S	M	EY1

Table 4.5: Mosaic Rhyme Example.

## 4.4 Methodology

First, the whole poem is converted to its phonetic transcription. A list is generated for all the end words and another for all the words around the caesura. To detect the end rhyme, the list of end words is checked twice. It is checked for consecutive rhyming and then for alternate rhyming. There is a two-step processing for each sub-category as well. First, the type of rhyme based on similarity is established (Rich/Full/Slant). Then a RhymeScore is calculated to quantify the magnitude of rhyming. RhymeScore is a number between 0 and 1 (in most cases). When there is an overlap in rhyme sub-type, this number can attain a value greater than 1.

RhymeScore is computed first at word level to denote the similarity score of

two words positioned as per their rhyme type. Then the scores for words are summed up and normalized to denote rhyming at poem level. Each rhyme type has its own RhymeScore.

$$RhymeScore_{wordpair} = \frac{Matched\ phoneme\ count}{Total\ phonemes\ in\ bigger\ word}$$

As stated earlier, a word may have different pronunciations. For example, the word *process* has two pronunciations as shown in Table 4.6.

PROCESS	P	R	AA1	S	EH2	S
PROCESS(1)	P	R	AO1	S	EH2	S

Table 4.6: Multiple pronunciation example.

Thus all possible phoneme combinations are tried and all possible values of  $RhymeScore_{wordpair}$  are calculated for that word pair and the maximum of all scores is taken as the final  $RhymeScore_{wordpair}$ . Thus, if A & B are two words on different lines being tested for rhyming, and A has 3 pronunciations A1, A2 & A3, and B has 2 pronunciations B1 & B2, then:

$$RhymeScore_{wordpair}(A, B) = Max(RhymeScore_{wordpair}(A_i, B_j))$$

Since, rhyme is defined both by phonemes and stress position, we need to consider stress similarity in addition to phoneme comparison. While quantifying, there has to be some intermediate scoring so as to give not-perfect cases some weight. Table 4.7 shows an example.

Nefarious	N	AH0	F	EH1	R	IY0	AH0	S
Glorious		G	L	AO1	R	IY0	AH0	S
		n	nc	nv	yc	yv	yv	yc

Table 4.7: Rhyme Example.

In the absence of a phoneme at a position, no score is given. n is vowel-consonant mismatch, nc is consonant mismatch, nv is vowel mismatch, yc is consonant match, yv is vowel match without stress and \*yv is vowel match with stress. We assign weights to each parameter to stipulate its impact on the overall rhyme of the word pair. We choose 0 for a total mismatch and incrementally add 0.2 to each category until we reach 1 as a perfect match for \*yv.

$$*yv > yc > yv > nc > nv$$

Thus,  $*yv$  has the most impact on rhyme and  $nv$  the least. The impact of  $nc$  and  $nv$  is very small as compared to the others. Now to quantify the scores, we assign values to these parameters:

Parameter	$*yv$	$yc$	$yv$	$nc$	$nv$
Weight	1.0	0.8	0.6	0.4	0.2

Table 4.8: Rhyme Weights.

Now  $RhymeScore_{wordpair}$  is calculated based on these weights. After computing all  $RhymeScore_{wordpair}$  for each pair of words, they are all summed up to calculate  $RhymeScore$  of a poem:

$$RhymeScore_{total} = \Sigma(RhymeScore_{wordpair})$$

$RhymeScore_{total}$  is a measure of rhyme density of a poem. To make this score independent of the length of the poem, it is divided by a Normalization factor  $norm_f$ .

$$RhymeScore_{norm} = \frac{RhymeScore_{total}}{norm_f}$$

The normalization factor is the maximum number of possible chances of matching in a poem. For consecutive end rhyming, it is equal to:

$$norm_f = \text{Number of lines in poem} - 1$$

For Internal rhyme, it is equal to:

$$norm_f = \text{Number of lines in poem}$$

In the Alternate rhyme detection, we are considering alternation with a line difference up to the full length of the poem (and not just 1). This is done to detect rhyme when it is present further apart that goes undetected manually. We name this alt-n rhyme pattern, where  $n$  is an integer greater than 1 and smaller than the length of the poem minus 2.

For alt-n rhyme, normalization is equal to:

$$norm_f = \frac{(n-1)*(n-2)}{2}$$

After calculating RhymeScores for all rhyme types and sub-types for all poems, we can perform the graph visualization.

## 4.5 Results

Our input set is a corpus of 12,830 poems from the Poetry Foundation. This set contains poetry from geographically and culturally diverse regions like USA, Russia, China, UK, India, Greece, etc. The poetic themes also belong to broadly diverse categories like Religion, Nature, Relationships, Mythology & Folklore, Philosophy, Love, Humour & Satire, Death, Social Commentaries, History & Politics, Music, Parenthood, etc.

A subset of 100 annotated poems is used to check the accuracy of our classification. These poems have been annotated by two human judges, with an inter-annotator agreement of at least 90%. This set has been subdivided into two sets of 50 poems each. The first set is our development set on which all thresholds and weights are based. The second set is our test set on which classification testing is done. Accuracies for both are given in Table 4.9.

<b>Title</b>	<b>Dev Set</b>	<b>Test Set</b>
Poems	50	50
Total annotations	449	344
Correct annotations	436	332
Incorrect annotations	13	12
Accuracy of classification	97.10%	96.51%

Table 4.9: Dataset statistics and classification results.

Here is a rhyme annotation result by our rhyme detection algorithm for a poem by Kay Ryan (Ryan et al., 2005):

Title : Thin

Author : Kay Ryan

How anything  
is known #cons=slant#  
is so thin #cons=slant#  
a skin of ice  
over a pond #alt=slant#  
only birds might  
confidently walk  
upon. A bird's

worth of weight #alt=slant# #alt2=slant#  
or one bird-weight #cons=full# #alt3=slant#  
of Wordsworth. #alt=slant#

We encounter *cons=slant* in line 3 that denotes that there is a consecutive slant rhyme, hence the word *thin* and *known* are the slant rhyming words. Similarly, *cons=full* denotes a consecutive full rhyme on the words *weight* and *bird-weight*.

For alt-n rhyme, *alt2=slant* on line 9 denotes that there is slant rhyme on the current line with 2 lines alternation. Hence, slant rhyming words are *weight* and *might*. Similarly, for *alt3=slant* on line 10, slant rhyming words are *bird-weight* and *might*.

Some examples of results of poetry analysis based on their RhymeScore are given below.

Title	Statement with Rhymes
Author	Weldon Kees
End Rhyme	0.044
Internal Rhyme	0.027
Eye Rhyme	0
Full Rhyme	1.873
Rich Rhyme	0
Identical Rhyme	0
Slant Rhyme	0.600

Table 4.10: Poem RhymeScore.

Table 4.10 shows RhymeScore values for a poem by Weldon Kees. It has very few End rhymes. Internal rhyme is present; this is rare, as it appears in very few poems. It has a modest score for all the other rhyme types.

Table 4.11 shows RhymeScore values for a poem by Charles Dickens that shows heavy usage of End rhyme. This high score is due to very high Full rhyme score. Slant rhyme is not prominent. Eye and Internal rhyme are absent. There is no Rich rhyme as well.

Table 4.12 shows RhymeScore values for a poem by William Shakespeare that is rich in End rhyme. This high score is due to an almost equal score for Full and Slant rhyme.

Title	The Song of the Wreck
Author	Charles Dickens
End Rhyme	0.202
Internal Rhyme	0
Eye Rhyme	70
Full Rhyme	12.770
Rich Rhyme	0
Identical Rhyme	0
Slant Rhyme	2.666

Table 4.11: Poem RhymeScore.

Title	Sonnet 104
Author	William Shakespeare
End Rhyme	0.259
Internal Rhyme	0.020
Eye Rhyme	0
Full Rhyme	3.552
Rich Rhyme	0
Identical Rhyme	0
Slant Rhyme	3.101

Table 4.12: Poem RhymeScore.

## 4.6 Web Application for Rhyme Visualization

After quantifying and classifying poems, we developed a web application for visualizing poem on a 2D and 3D scatter plot. The objective for visualization is to find similar poems based on their different rhyme sub-types. All the poems with similar RhymeScores will be projected closer on a plot and the dissimilar ones will be further apart.

We used the Plotly framework with Javascript for developing these plots and deployment was done on the Apache HTTP server. We can select the rhyme type to be displayed on the axes. If only 2 rhyme types were selected, a 2D scatter plot is generated on the PoFo poems set, and on selecting 3 rhyme types, a 3D scatter is generated.

Figure 4.2 shows a 3D scatter plot of PoFo poems. The toolbox on the top right corner has multiple operations and we are allowed to:



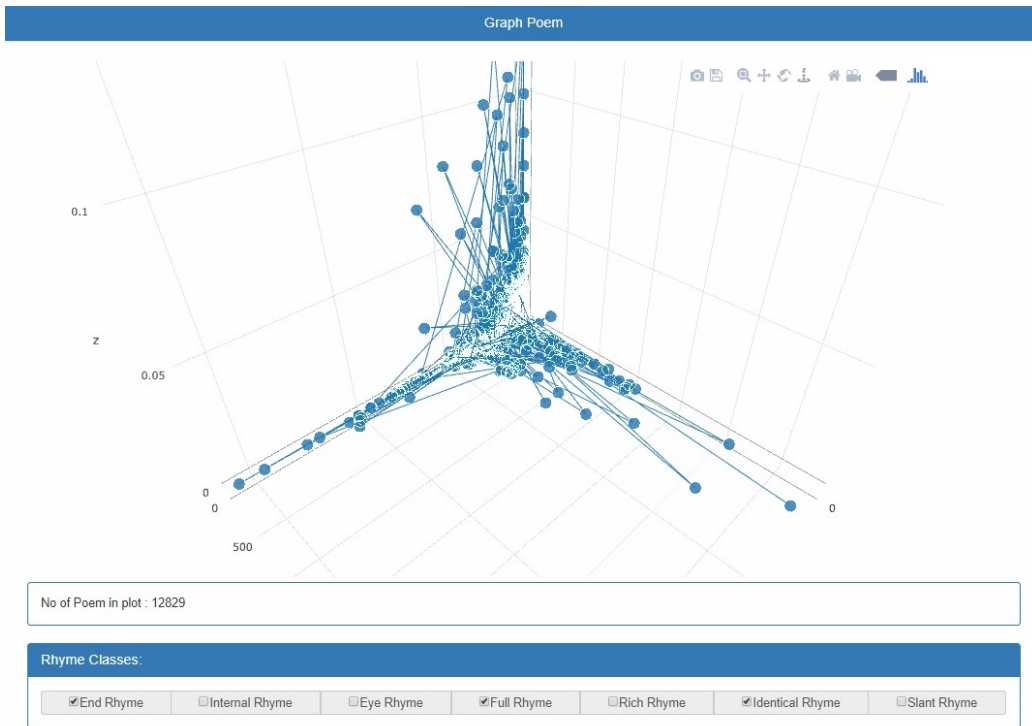


Figure 4.2: A 3D scatter plot of PoFo poems

- pan in all four directions
- zoom in and out
- perform orbital rotation in a 360-degree plane to rotate all the axes
- perform turntable rotation
- save a snapshot of the plot in the current state

Figure 4.3 shows a 2D scatter plot of PoFo poems. The toolbox in this plot is slightly different than the one with a 3D plot and we are allowed to:

- pan in all four directions;
- zoom in and out;
- box and lasso select to zoom in on a small region to view poems in that concentrated region only;
- save a snapshot of the plot in the current state.

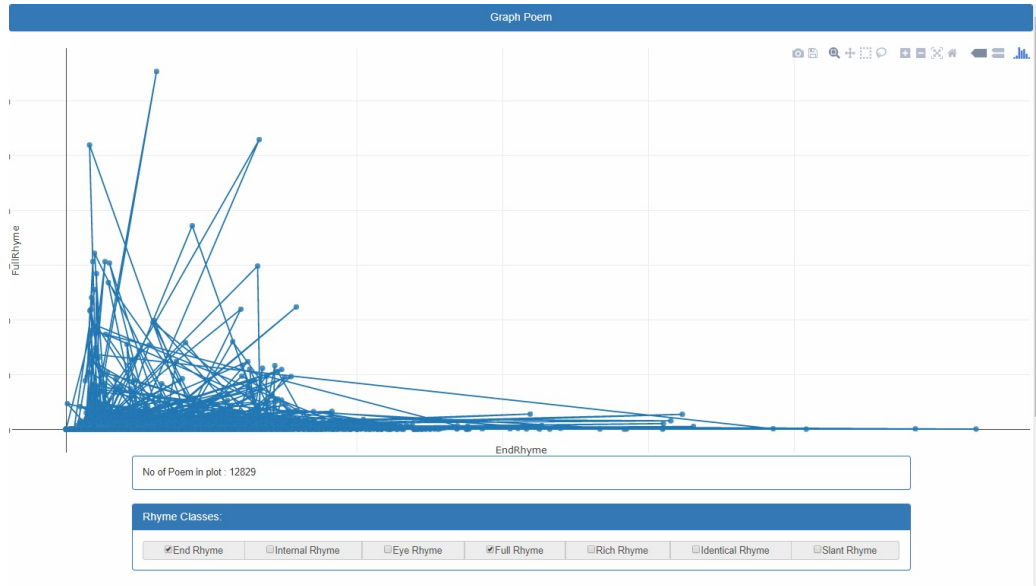


Figure 4.3: A 2D scatter plot of PoFo poems

Figure 4.4 shows a small region that was selected in the 2D scatter plot. In both the 2D and 3D plots, if the mouse hovers on a poem node, a small marker is seen on top of that node that shows the RhymeScore values for that specific poem.

Further, on clicking on a poem node, that specific poem is opened in a dialog with all rhyme types highlighted in color. This can be seen in figure 4.5.

The legend for rhyme types is:

- Identical : yellow
- Rich: red
- Full: lightgreen
- Slant: lightblue
- Eye: cyan

## 4.7 Conclusion and Future Work

We have built a rhyme classifier based on CMUDict phoneme dictionary that classifies poem lines into rhyme types and subtypes. Then we quantified

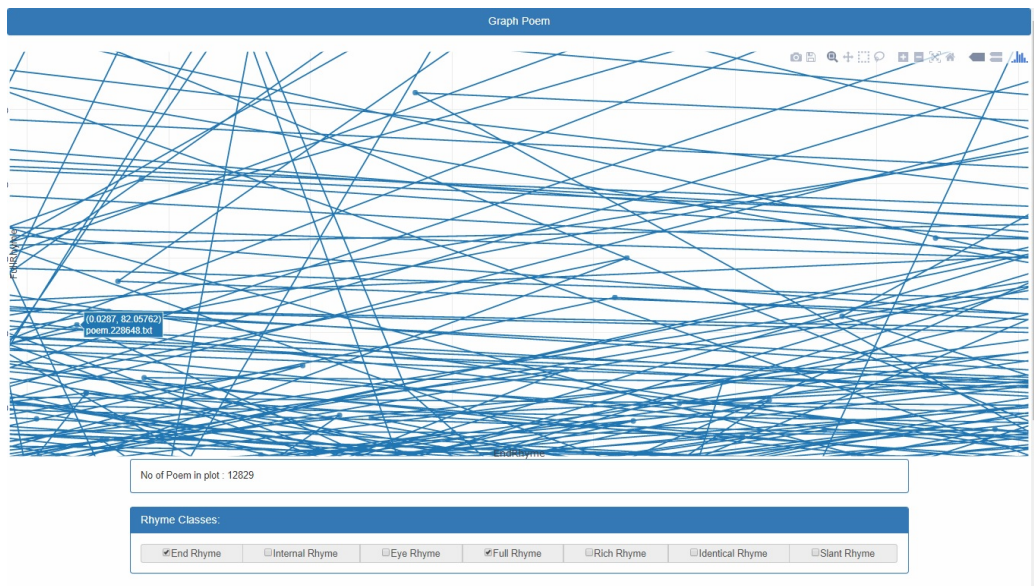


Figure 4.4: Poem RhymeScore displayed on mouse over event in the plot

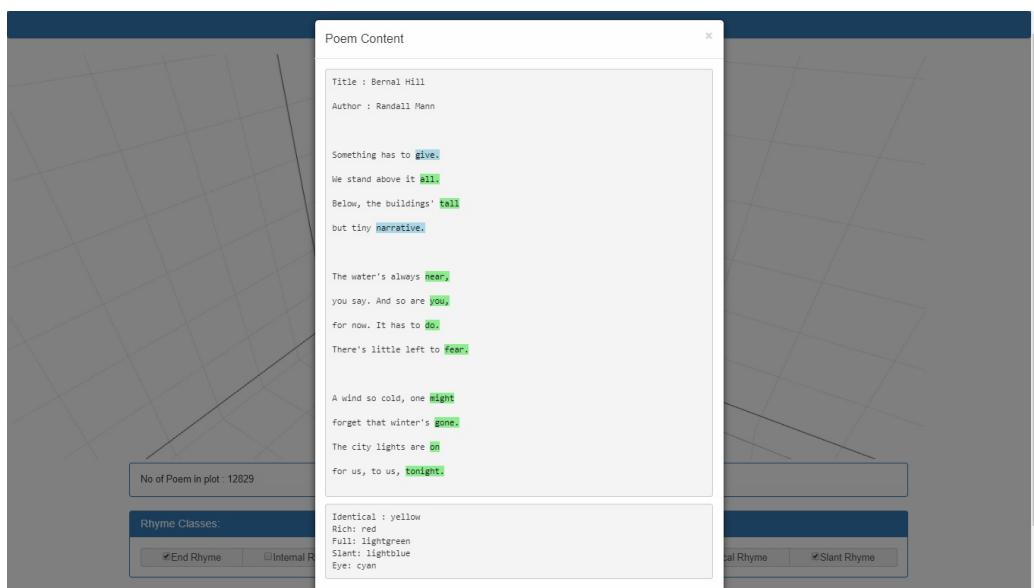


Figure 4.5: On clicking a poem node, that specific poem is displayed with rhyme types in colored legend.

rhymes into a normalized RhymeScore for each poem so that we may visualize a poem graphically on the basis of its rhyme types. For visualization, we built a web application for generating a scatter plot based on the metrics obtained.

In future work, an option would be to use the RhymeScores obtained for all poems as input data for machine learning classifiers like SVM, Naive Bayes, KNN, etc. to find out the similarities between various poems on the basis of rhyme types. A challenge that we may face is that these classifiers are binary, but our input is numerical and multi-class. Thus we may have to take some thresholds for each of rhyme type and sub-types to reduce this to a two-class problem. If we take rhyme types of a poem as binary, we ignore all the richness of all the intermediate cases. Moreover, the overlap between various rhyme types poses a major challenge on this reduction strategy.

Due to the overlapping nature of rhyme sub-types, Full/Slant/Rich rhyme types show values greater than 1. We need to find a method to normalize these overlap cases in order to make the analysis of these types independent of their parent type values. Currently, we always have to compare the parent rhyme types in order to make a non-biased analysis. We cannot compare sub-type scores of poems directly due to this length bias.

Further, we also aim to increase our corpus size in the future. We are considering including more poems from Representative Poetry Online repository by Plamondon (2006). This would increase the richness of our analysis. The inclusion of more rhyme types like Mosaic, Assonant, Consonant, Broken, etc., is also possible for future work.

# CHAPTER 5

## DICTION ANALYSIS

### 5.1 Introduction

Diction refers to the linguistic style of a poem. It is a very important classification criterion as it sets the tone or atmosphere of a poem and depends on the vocabulary choice of the poet. Diction encompasses linguistic quantitative features and semantic features.

The list of linguistic quantitative features that are computed in this thesis for each poem are as follows:-

- Abstract/Concrete ratio
- Inflection ratio
- Verbal density
- Noun density
- Adjective density
- Pronoun vs Noun ratio

The list of semantic features that are computed for each poem are as follows:-

- Metaphor detection
- Concordance
- Word Embeddings

Though metaphor detection is a subsection of diction analysis, it is illustrated extensively in the next chapter as it is a complex task. Leaving the metaphor apart, all the above features are explained in the Section 5.3.

## 5.2 Related Work

Kao and Jurafsky (2015) have employed various features such as object, abstract, imageability, concreteness, emotion, valence and arousal. Their aim was to compare Imagist poets with others and used the above-mentioned features in addition to several others for comparative analysis. Our work also uses some of the features like abstract, concrete and objects (nouns), but does not focus on the imageability and emotional aspects. Instead, we focus more on the semantic part, as our dataset is quite generic and not focused on a specific style of poetry.

Delmonte (2013) also employs many semantic measures to compute poetic style like concrete vs. abstract and eventive classes, specific vs. collective and ambiguous concepts, and so on. We also use similar features to theirs like abstract/concrete, inflection ratio, and densities of various part of speech (POS) words. But our aim is to develop a metric for poetic diction analysis that can be visualized on a graphical model. Furthermore, we also indexed all Poetry Foundation (PoFo) poems for concordance or semantic retrieval.

Stamatatos et al. (2000) also talk about quantifiable measures like ours, but their task is genre and author detection that is quite different than ours and their dataset is not poetry. They also use several features like densities of POS words and common word frequencies, among others, for their classification task.

This section on related works for diction analysis is not very extensive as there are very few works on poetic diction analysis.

## 5.3 Method

### 5.3.1 Abstract/Concrete ratio

The Abstract/Concrete ratio is the ratio of the total of abstract words to the total of concrete words in a poem. Abstract words are those that depict an abstract concept that is non-tangible like an emotion, a time period, an event, a shape, etc. Concrete words are those that depict a concrete object that is tangible like an artifact, a living being, a physical object, etc.

WordNet (Miller, 1995) is employed to get this categorization. In WordNet,

**happiness** -- (emotions experienced when in a state of well-being)  
 => feeling -- (the experiencing of affective and emotional states; "she had a feeling of euphoria"; "I disliked him and the feeling was mutual")  
 => state -- (the way something is with respect to its main attributes; "the current state of knowledge"; "in a weak financial state")  
 => attribute -- (an abstraction belonging to or characteristic of an entity)  
 => abstraction -- (a general concept formed by extracting common features from specific examples)  
 => **abstract entity** -- (an entity that exists only abstractly)  
 => entity -- (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

**shark** -- (any of numerous elongate mostly marine carnivorous fishes with heterocercal caudal fins and tough skin covered with small toothlike scales)  
 => cartilaginous fish, chondrichthian -- (fishes in which the skeleton may be calcified but not ossified)  
 => fish -- (any of various mostly cold-blooded aquatic vertebrates usually having scales and breathing through gills;)  
 => aquatic vertebrate -- (animal living wholly or chiefly in or on water)  
 => vertebrate, craniate -- (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain )  
 => animal, animate being, beast, brute, creature, fauna -- (a living organism characterized by voluntary movement)  
 => organism, being -- (a living thing that has (or can develop) the ability to act or function independently)  
 => living thing, animate thing -- (a living (or once living) entity)  
 => object, physical object -- (a tangible and visible entity; an entity that can cast a shadow; "it was full of rackets, balls")  
 => **physical entity** -- (an entity that has physical existence)  
 => entity -- (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Figure 5.1: WordNet hypernym hierarchy for *happiness* (abstract) and *shark* (concrete).

all nouns possess a hypernym hierarchy and the root is always the class *entity*. The immediate children of the root are *physical entity* or *abstract entity*. The level where physical/abstract is present depends on the individual word and its specificity in the English language. Figure 5.1 shows the hypernym hierarchy for an abstract word *happiness* and a concrete word *shark*.

Our algorithm iterates to get to this abstract/concrete level and classifies the word based on this. In each poem, the total count of abstract/concrete words is computed and consequently used to get the ratio. If a word is absent in WordNet, then it is skipped.

### 5.3.2 Inflection ratio

Inflection is a change in the root form of a word to denote a change in tense, mood, person, number, case, and gender. The inflection ratio is the ratio of inflectional words to non-inflectional words in a poem.

To compute inflectional words, the WordNet stemmer is employed. All words that remain the same after stemming are non-inflectional words and vice versa.

### 5.3.3 Verbal density

The verbal density is the ratio of total count of verbs to the total words in a poem. The total word count does not include any punctuation or numbers and contains only words. We count as verbs the following part-of-speech tags:

- VB : base form
- VBD : past tense
- VBG : gerund/present participle
- VBN : past participle
- VBP : sing. present, non-3d
- VBZ : 3rd person sing. present

All the POS nomenclature in sections 5.3.3 to 5.3.6 is based on the Penn TreeBank Project tags listed in Table 2.1 in the Background chapter. For finding the POS tags for all the above metrics, the Spacy library (Honnibal and Johnson, 2015) is employed.

### 5.3.4 Noun density

Noun density is the ratio of total count of nouns to the total words in a poem. Total word count does not include any punctuations or numbers and contain only words. Noun denotes all part-of-speech tags:

- NN : singular
- NNS : plural
- NNP : proper singular
- NNPS : proper plural



### 5.3.5 Adjective density

The adjective density is the ratio of total count of adjectives to the total number of words in a poem. The total word count does not include any punctuation or numbers and contains only words. Adjective denotes all part-of-speech tags:

- JJ : root
- JJR : comparative
- JJS : superlative

### 5.3.6 Pronoun vs. Noun ratio

The Pronoun vs Noun ratio is the ratio of the total number of pronouns to the total number of nouns in a poem. Pronoun denotes all part-of-speech tags:

- PRP : personal
- PRP\$ : possessive
- WP : wh-pronoun
- WP\$ : possessive wh-pronoun

Noun denotes all part-of-speech tags:

- NN : singular
- NNS : plural
- NNP : proper singular
- NNPS : proper plural

### 5.3.7 Concordance

Concordance refers to the retrieval of a queried term and all the related terms in the context of a text. A concordance, in itself, is not a semantic feature, but in our case, we find words that are semantically related to the words

in the query. To get the related terms of a queried term, query expansion is employed. First, the WordNet stemmer is used to find the stem of the queried word. For example, *seas* is stemmed to *sea* and both of the words are included in our query set. Secondly, WordNet is used to get the synonym sets for all senses of the stem word. For example, *ocean* is a synonym of *sea*. Now, our query set contains *seas*, *sea* and *ocean*.

The query *humans* is similarly expanded (in decreasing order of importance) to:-

- world
- human race
- humanity
- humankind
- human beings
- humans
- mankind
- man

The query *feeling* is expanded (in decreasing order of importance) to:-

- feeling
- impression
- belief
- notion
- opinion
- spirit
- tone
- feel

- flavor
- flavour
- look
- smell
- touch
- touch sensation
- tactual sensation
- tactile sensation
- intuitive feeling

After getting an expanded query set, Apache Lucene<sup>1</sup> is used to get all occurrences of the words in our set. We index all poems in the PoFo corpus and retrieval is done on that populated index. This makes the search operation quite fast and efficient.

The results from Lucene are sorted on the basis of the TF-IDF (explained in detail in section 2.2.3) parameter to get the most important poems with most occurrences on the top and less important results later. Results show the Lucene Score as well on top of every poem. The Lucene Score ranges from 0 to 1 and is an absolute doc score that depends on the following four metrics:

- Inverse Document Frequency
- Term Frequency
- Coordination Factor
- Field Length

Further, to ease the locating of search terms, the Lucene Highlighter is used to highlight the search terms in the poem. The highlighted terms are shown in cyan color.

---

<sup>1</sup><https://lucene.apache.org/core/>

### 5.3.8 Word Embeddings

Word embeddings are models where words are represented as vectors of real numbers in low dimensional vector space (explained in detail in section 2.6). Words are represented as vectors to perform various compositionality operations (Mikolov et al., 2013b) or to find similarity metrics or equivalence relations (word associations). Word embeddings are used for language modeling as they serve as input for all neural networks and deep networks.

We use word embeddings for the metaphor detection task (explained in chapter 6). We experimented with Word2Vec (Mikolov et al., 2013a) trained on Google News corpus and GloVe (Pennington et al., 2014) trained on the Gigaword corpus. We observed that although good for common speech analysis, we could not use the already available models for building word associations for poetry. The explanation is that none of the pre-trained models were trained on poetry and therefore contained less contextual information pertaining to poetic style and diction.

To verify the validity of the observation above, we did a comparison of related words in the GloVe Gigaword model and in our custom trained PoFo model. The results in Table 5.1 and Table 5.2 show the words related to the word *love* in the GloVe and PoFo models in decreasing order of similarity score. It can be seen that the words in the GloVe model are more conversational (pronouns like *me*, *my*, *you*, *I* and *she* reinforce this) and less thematic; whereas words from PoFo model are more focused and are very representative of the word *love*.

Table 5.1: LOVE in GloVe model

Word	Score
me	0.738
passion	0.735
my	0.733
life	0.729
dream	0.727
you	0.718
always	0.711
wonder	0.709
i	0.708
dreams	0.707
mind	0.706
friends	0.704
true	0.703
loves	0.700
feel	0.698
happy	0.698
fun	0.697
kind	0.696
soul	0.695
she	0.695

Table 5.2: LOVE in PoFo model

Word	Score
joy	0.791
sorrow	0.783
hope	0.781
desire	0.764
grief	0.759
despair	0.742
delight	0.737
pleasure	0.730
beauty	0.730
pain	0.729
bliss	0.729
hate	0.716
pity	0.714
true	0.709
comfort	0.706
shame	0.702
passion	0.701
faith	0.697
fear	0.697
hunger	0.695

Similarly, in Tables 5.3 and 5.4, words related to word *see* in GloVe and PoFo model are enumerated (again in decreasing order of similarity score). It can be observed again that the PoFo model captures the thematic representation of *see* with better objects (like *imagine*, *recall*, *understand*, etc.). The GloVe model captures associations more related to conversational words like *you*, *how*, *why*, etc..

Table 5.3: SEE in GloVe model

Word	Score
look	0.848
you	0.838
come	0.837
how	0.835
why	0.831
want	0.829
know	0.826
do	0.823
what	0.822
think	0.821
we	0.818
n't	0.817
if	0.808
go	0.806
sure	0.806
going	0.803
so	0.803
make	0.802
'll	0.800
say	0.799

Table 5.4: SEE in PoFo model

Word	Score
imagine	0.793
recall	0.758
remember	0.755
understand	0.733
enter	0.733
know	0.731
meet	0.724
think	0.724
tell	0.717
hear	0.709
find	0.704
forget	0.704
follow	0.702
guess	0.700
believe	0.696
count	0.693
show	0.691
feel	0.691
hide	0.688
stand	0.686

## 5.4 Example of Results

To illustrate Concordance, here is an example using Lucene Highlighter:

Query : seas OR sea OR ocean

Total poems : 2026

File : poem.248362.txt

Lucene Score : 0.8021394

Poem Title : The Ocean

Poet : Nathaniel Hawthorne

The Ocean has its silent caves,  
Deep, quiet, and alone;  
Though there be fury on the waves,  
Beneath them there is none.

The awful spirits of the deep  
Hold their communion there;  
And there are those for whom we weep,  
The young, the bright, the fair.

Calmly the wearied seamen rest  
Beneath their own blue sea.  
The ocean solitudes are blest,  
For there is purity.

The earth has guilt, the earth has care,  
Unquiet are its graves;  
But peaceful sleep is ever there,  
Beneath the dark blue waves.

All the words in the search set are seen in cyan color. The Lucene Score for this poem is 0.802, which is the top score for this search set.

For the same poem given above, the results of the linguistic quantification are given below:-

PROPN : 2  
TOKENS : 107  
DET : 13  
NOUN : 19  
ADP : 8  
CONJ : 3  
PUNCT : 19  
ADJ : 19  
VERB : 14  
PRON : 2

ADV : 8

Verbal density=0.130

Noun density=0.177

Adjective density=0.177

Pronoun vs. Noun ratio=0.105

For the same poem given above, the result of the Inflection analysis are below :-

**16 Inflectional words:** [has, caves, waves, is, spirits, are, seamen, solitudes, are, is, has, has, are, graves, is, waves]

**30 Non Inflectional words :** [silent, deep, quiet, be, fury, none, awful, deep, hold, communion, weep, young, bright, fair, wearied, rest, own, blue, sea, ocean, blest, purity, earth, guilt, earth, care, peaceful, sleep, dark, blue]

**Inflection ratio:** 0.533

For the same poem, the result of Abstract/Concrete analysis are :-

**Nouns :** [cave, fury, wave, none, spirit, communion, whom, seaman, sea, ocean, solitude, purity, earth, guilt, care, grave, sleep]

**9 Abstract words :** [fury, wave, none, communion, solitude, purity, guilt, care, sleep]

**7 Concrete words:** [cave, spirit, seaman, sea, ocean, earth, grave]

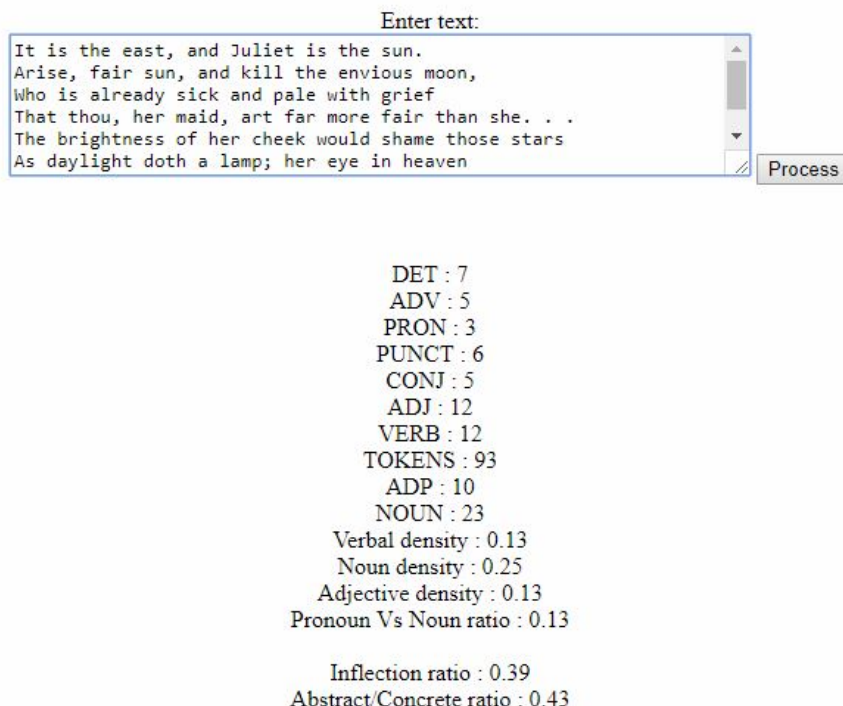
**Abstract/Concrete ratio:** 1.285

Though the results are shown for only one poem, diction analysis is done for all poems in the PoFo corpus. Qualitatively, these ratios speak to the linguistic style of a poem (Delmonte, 2013). High verb density is good for a poem, as it denotes more action elements. High adjective density is, stylistically speaking, bad, as it signals the fact that the poet has to explain everything and is, therefore, an indication of a very likely lack of subtlety. High pronoun-noun ratio is generally faulty, as it would signal lack of succinctness or, more simply, a potential sign of verbosity. High inflection ratio is preferable, as it shows that the poet employed variety in expressions. High abstract/concrete ratio is in principle an infelicitous characteristic, as it is a likely a sign of poor craftsmanship or lack of suggestiveness.

In the next section, we will introduce our diction web application whereby we can do a comparative analysis of the diction features in a poem.



## Diction Analysis in Poetry



Enter text:

It is the east, and Juliet is the sun.  
Arise, fair sun, and kill the envious moon,  
Who is already sick and pale with grief  
That thou, her maid, art far more fair than she. . .  
The brightness of her cheek would shame those stars  
As daylight doth a lamp; her eye in heaven

Process

DET : 7  
ADV : 5  
PRON : 3  
PUNCT : 6  
CONJ : 5  
ADJ : 12  
VERB : 12  
TOKENS : 93  
ADP : 10  
NOUN : 23  
Verbal density : 0.13  
Noun density : 0.25  
Adjective density : 0.13  
Pronoun Vs Noun ratio : 0.13

Inflection ratio : 0.39  
Abstract/Concrete ratio : 0.43

Figure 5.2: Screenshot of the diction analysis web application that quantifies linguistic quantitative features.

### 5.5 Web Application for Diction Analysis

We have developed a web application for analyzing diction in poetry (and non-poetry) in general and not just for the PoFo corpus. This application incorporates all the linguistic quantitative features explained in the previous sections. For semantic features, we have build applications for metaphor detection that will be explained in detail in the next chapter.

Figure 5.2 shows a screenshot of the diction analysis application. The user enters the poem (or any text) in the textbox and clicks the process button to execute the script. All the quantified features are computed and displayed on the screen.

For concordance, we have developed another application that retrieves poems based on a search query and employs semantic query expansion as explained earlier. It is indexed on PoFo corpus with a total of 12830 poems. Figure 5.3 shows a screenshot of the application with a sample search result.

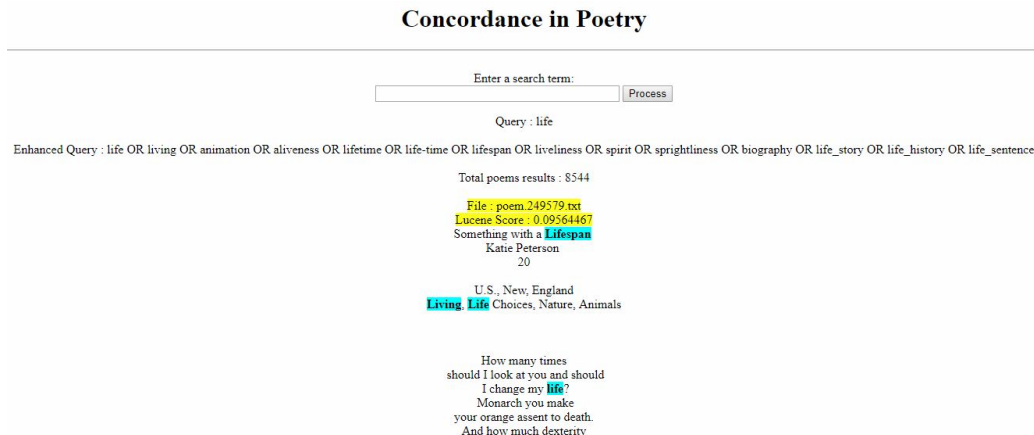


Figure 5.3: Screenshot of the concordance web application that uses semantic query expansion to retrieve poems from PoFo corpus.

The poems are retrieved on decreasing order of Lucene score and the search term is highlighted in color for ease of visibility. Currently, this application does not have an option to include more poems that are not part of the PoFo corpus, but in the future, we will expand it to index more poems as well.

## 5.6 Conclusion and Future Work

We have attempted to quantify the linguistic and semantic diction features (implicit or not) that are important in analyzing a poem holistically. Though we have employed all these techniques for the PoFo corpus, they can easily be applied to non-poetry data as well, since we do not use any poetry-specific algorithms.

The primary cause of error in diction analysis is due to the absence of Word Sense Disambiguation. We have used WordNet for almost all the analysis tasks, but in WordNet, words can have different senses. Though the first sense of a word always refers to the most common one, sometimes this extrapolation is incorrect and our classification gets flawed. In order to rectify this, we intend to apply Word Sense disambiguation methods such as Context Overlap in our future work.

The ultimate aim of diction analysis is not just the quantification of these features, but graph visualization as well. We aim to use the quantified parameters to plot the poems in a 3-dimensional space and connect them by edges dependent on the magnitude of the parameters. This will be imple-

mented in our future work. We aim to deduce patterns and similarities in our poetry plot based on our already quantified linguistic features.

# CHAPTER 6

## METAPHOR DETECTION

### 6.1 Introduction

Metaphor is crucial to the understanding of any literary text. A metaphor deviates from the normal linguistic usage. It intends to create a strong statement that no literal text can accomplish. Metaphor differs from idioms, because one can understand a metaphor even with no prior knowledge or context of an idiom. Here are examples of metaphors in poetry:

- The hackles on my neck are fear (Wright, 1958)
- My eyes are caves, chunks of etched rock (Lorde, 2000)

Literary metaphor operates not only in the local context where it appears. It also functions in the broader context of the whole work or even an author's oeuvre, and in the context of the cultural paradigms associated with a specific metaphor field (Ritchie, 2013). Contrary to the standard view, literary metaphor sometimes also maps not only in one direction (from *vehicle* to *tenor*) but in two. It thus helps reshape both concepts involved (Ritchie, 2013, p. 189). In other cases, a metaphor interconnects two concepts and so only develops each of them into independent sources of introspective and emotional stimulation (Ritchie, 2013, p. 193).

Literary metaphor is generally thought to be more stylistically colourful. It is placed somewhere at one extremity of a spectrum that has common-speech metaphor at the other end (Ritchie, 2013). In poetry sometimes the opposite is also true. The most unadorned and literal language can be strongly metaphorical by means of the symbolic import of whole passages or even entire poems: a poem or a longer passage figuratively alludes to an implicit concept. Such is the case, for instance, of Robert Frost's *The Road Not Taken* (Frost, 1962). The poem speaks in its entirety of a consequential choice made

in life, without apparently deploying any actual metaphor. Needless to say, it is a type of metaphor possibly even more difficult to process automatically.

## 6.2 Related Work

We used a few rule-based methods for metaphor detection as a baseline for our experiments. Turney et al. (2011) proposed the Concrete-Abstract rule: a concrete concept when used to describe an abstract one, represents a metaphor. A phrase like *Sweet Dreams* is one such example. We use the Abstract-Concrete rule as one of the many features in our model. In experiments, it has in fact proved to be quite useful in the case of poetry as well.

Neuman et al. (2013) propose to categorize metaphor by part of speech (POS) tag sequences such as Noun-Verb-Noun, Adjective-Noun, and so on. We follow the same methodology to extract the set of sentences that can be metaphorical in nature. Our method differs because we use word embeddings pre-trained on the Gigaword corpus (Pennington et al., 2014) to get word vector representations (vector difference and cosine similarity) of possible metaphorical word pairs. Another difference is the addition of two more types of POS sequences, which we have found to be metaphorical in our Poetry Foundation poetry corpus. We explain the types in section 6.3.1.

Neuman et al. (2013) describe a statistical model based on Mutual Information and selectional preferences. They suggest using a large-scale corpus to find the concrete nouns which most frequently occur with a specific word. Any word outside this small set denotes a metaphor. Our experiments do not involve finding selectional preference sets directly. Instead, we use word embeddings. We have found the selectional preference sets too limiting. The word span is to be set before the experiments. Some sentences exceed that limit, so the contextual meaning is lost.

Shutova et al. (2016) introduce a statistical model which detects metaphor. So does our method, but their work is more verb-centered, in that verbs are a seed set for training data. Our work looks more into the possible applications for poetry, not generically. We also concentrate on nouns, because our initial experiments concerned Type I metaphor: a copular verb plays only an auxiliary role, so the focus is on the two nouns.

A genre-based comparison of metaphor in literature would involve a wide-ranging theoretical and historical comparative analysis of literary genres and tropes. Such analysis is beyond the scope of this thesis, and outside the focus of our current research, which concerns itself only with poetry and selects its data accordingly.

## 6.3 Method

### 6.3.1 Building the Corpus

We have built our own corpus, because there is no publicly available poetry corpus annotated for metaphor. Annotating poetry line by line can be laborious. We have observed empirically that negative samples are too numerous. To ease this task, we applied Neuman’s approach: consider POS tag sequences to extract potential metaphor. We extracted all sentences from the 12,830 PoFo poems that match these tag sequences.

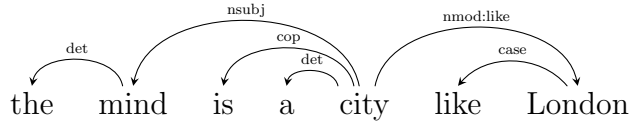
Type I metaphor has a POS tag sequence of Noun-Verb-Noun where the verb is a copula (Neuman et al., 2013). We have extended this to include the tag sequence Noun-Verb-Det-Noun, since we have found that many instances were skipped due to the presence of a determiner. Type II has a tag sequence of Noun-Verb-Noun with a regular, not copula, verb (Neuman et al., 2013). Type III has a tag sequence of Adjective-Noun (Neuman et al., 2013). We also propose two more metaphor types that we noticed in our poetry data: Type IV with a tag sequence of Noun-Verb, and Type V with a tag sequence of Verb-Verb. Here are examples:

- As if the *world were* a *taxi*, you enter it [Type 1] (Koch, 1962)
- I counted the *echoes assembling, thumbing* the *midnight* on the piers. [Type 2] (Crane, 2006)
- The moving waters at their *priestlike task* [Type 3] (Keats, 2009)
- The yellow *smoke slipped* by the terrace, made a sudden leap [Type 4] (Eliot, 1915)
- *To die – to sleep* [Type 5] (Shakespeare, 1904)

Though we prepare the corpus on Type I metaphor sequence, we also work on a method independent of POS tag sequences later on. We employ a dependency parser (de Marneffe et al., 2006) to identify all associations in a sentence. We use associations such as *nsubj*, *dobj* and so on to filter down to get word pairs that need to be checked for metaphor occurrence. Other irrelevant associations are discarded. We take this generic approach because we feel that POS sequences may be a little restrictive and some instances that do not follow the specific POS sequence could be missed. This word pair identification method is used to generate candidates for metaphor detection.

Identifying head words in a sentence is in itself a challenging task. It is like compressing a phrase to a word pair that may or may not be a metaphor. The POS tag sequence does not always provide an understandable word pair. Sometimes we lose critical words that may be of value. When the nouns highlighted by the POS tagger are not enough to identify the head of a sentence (or a phrase), we use the Stanford Parser (de Marneffe et al., 2006) for identification. As an additional step, we extract all *nsubj* associations from these sentences. If the head word is different from the earlier identified head (suggested by the POS tagger), then the head word is updated.

Here is an example (Schwartz, 1989):



In this example, *mind* and *city* connected by *nsubj* association would be the possible word pair that needs to be checked for metaphoricity.

### 6.3.2 Annotating the Corpus

We extracted around 1,500 sentences with the type I metaphor tag sequence, and annotated the first 720. We employed majority voting. First, two independent annotators annotated the 720 sentences without any communication. Then the value of kappa was calculated. Its value came to 0.39, and agreement to 66.79%. Next, we involved a third annotator who cast a majority vote in case of disagreement. If one of the two annotators agreed to the other's justification, then the disagreement was resolved without the

intervention of the third annotator. After this, kappa increased to 0.46 and agreement to 72.94%.

While annotating, we found several highly ambiguous sentences which required a wider context for assessment. In those rare cases, the annotators were allowed to go back to the poem and judge the metaphor candidate by looking at the context in which it appeared. This was done to avoid discarding a legitimate example for lack of sufficient information. In most cases, however, the sentence alone provided enough information.

All sentences given to the annotators were marked to indicate the head of the sentence. The point was to avoid confusion whenever there was more than one noun phrase. For example:

*my eyes are caves , chunks of etched rock @2@* (Lorde, 2000)

The number 2 denotes that the word at location 2, *eyes*, is a head word. Therefore the second head would be *caves*, because this is a sentence with a Type I metaphor tag sequence. Since this is obviously a metaphorical word pair, the annotator would write *y* at the end of the sentence.

The annotators were also allowed to skip a sentence if they could not make up their mind. All in all, a sentence can be labeled as *y* for metaphor, *n* for non-metaphor and *s* for a skipped sentence.

Annotating metaphor is not a trivial task. Borderline cases occur, and there is ambiguity. We have encountered many such situations while annotating. For example:

*to me the children of my youth are lords , @7@s* (Crabbe, 1950)

It was annotated *s* because the full poetic context was lacking. Here the first head word is *youth* and the second head is *lords* if we consider the Type 1 tag sequence. If we consider the whole sentence, then *children* should be the first head word instead of *youth*. Since, Type 1 sequence is restrictive, we later consider generic metaphor detection (explained in section 6.7).

Sometimes we cannot ignore words that are not in the POS tag sequence. For example:

*for there christ is the king 's attorney , @3@y* (Raleigh, 1895)



Here *christ* is the first head word. If we consider the POS tag sequence, then *king* ought to be the second head, but it does not complete the phrase. Therefore, the whole phrase *king's attorney* is considered while annotating.

Another borderline example, in which the fragment *tree were a tree* can be either metaphorical or literal, depending on the context:

*that is , if tree were a tree . @5@n* (Baker, 1994)

Cases like these were very difficult to annotate. Most of them had to be forwarded to the third annotator for a final vote. Such cases were responsible for the rather low value of kappa, the inter-annotator agreement.

When the annotation process was concluded, we checked for the distribution of classes. Metaphor turned out to be present in 49.8% instances. Non-metaphor accounted for 44.8%, and 5.4% examples were skipped. We had an almost balanced dataset, so we did not need to apply any re-sampling in our classification. The sentences with skipped annotation were removed from our data. The final dataset contained 680 sentences.<sup>1</sup>

### 6.3.3 Rule-based Metaphor Detection

First, we applied rule-based methods to our poetry dataset. We used the Abstract-Concrete (Turney et al., 2011) and Concrete Category Overlap rules (Assaf et al., 2013). The Abstract-Concrete rule needs the hypernym class of each noun; we find that in WordNet (Miller, 1995). We got all hypernyms of head nouns and checked for each parent till we reached the hypernym *abstract entity* or *physical entity*.

Apart from the above rules, we used a feature based on ConceptNet (Speer and Havasi, 2012). For each noun in our sentence, we extracted the corresponding SurfaceText from ConceptNet. A SurfaceText contains some associations between the specific word and real-world knowledge. For example, *car* gives the following associations:

- *drive* is related to *car*
- You are likely to find *a car* in *the city*

---

<sup>1</sup>The data can be found at [http://www.eecs.uottawa.ca/~diana/resources/metaphor/type1\\_metaphor\\_annotated.txt](http://www.eecs.uottawa.ca/~diana/resources/metaphor/type1_metaphor_annotated.txt).

and so on.

The entities are already highlighted in the SurfaceTexts. We parsed these associations and extracted all the entities. There can be action associations as well:

- *a car can crash*
- *a car can slow down*

and so on.

These entities and actions were used to establish an overlap in the head nouns of the sentences in the poems. We call this method ConceptNet Overlap. We assigned *true* if there was an overlap and *false* otherwise. This was used as one of the features in our rule-based model.

#### 6.3.4 Statistical-based Metaphor Detection

To capture the distortion of the context that a metaphor causes to a sentence, we computed the vector difference between the vectors for the head words. The underlying idea is this: the smaller the difference, the more connected the words would be. Conversely, a significant difference implies disconnected words and hence very likely a metaphor. We rendered this difference by means of a 100-dimensional vector representation, and we set it as our first statistical feature. Later we tested with 200 dimensions as well, to observe the effect on our task.

To get the word vectors of head words, we used the GloVe vectors pre-trained on the English Gigaword corpus (Pennington et al., 2014). Earlier, we had used a custom-trained model based on the British National Corpus (Clear, 1993) but we switched to GloVe to test on a larger corpus. Another reason why we tested on two different corpora was to remove any bias that may be perpetuated due to the presence of common-speech metaphor in the corpus. We did not use the available pre-trained word2vec vectors (Mikolov et al., 2013a), because the GloVe vectors had been shown to work better for many lexical-semantic tasks (Pennington et al., 2014).

We trained word embeddings on the PoFo poems, but we did not use that for metaphor detection because the corpus was not large enough. Moreover, we needed a corpus that had as few metaphor occurrences as possible,

and poetry was obviously not an ideal choice. Training on a poetry corpus would generate word embeddings suited to poems in general, and might miss metaphor instances commonly occurring in poetry. In this task, we were more concerned with the detection of all types of metaphor, not just poetic metaphor. In effect, distinguishing between common-speech and poetic metaphor has been left for our future work.

We computed the cosine similarity for all word vector pairs, and made it another feature of our model. We also added a feature based on Pointwise Mutual Information in order to measure if a word pair is a collocation:

$$\ln \frac{C(x,y).N}{C(x)C(y)}$$

where N is the size of the corpus, C(x,y) is the frequency of x and y together, C(x) and C(y) are the frequencies of x and y in the corpus, respectively.

## 6.4 Results

We applied our method to the sentences extracted from the 12,830 PoFo poems and annotated manually (see section 6.3.2). For training data, we used a combination of the datasets such as TroFi (Birke and Sarkar, 2006) and Shutova (Mohammad et al., 2016) with our own poetry dataset. We included other datasets annotated for metaphor, in addition to poetry, in order to increase the training set and thus get better classification predictions. We report all results explicitly for the test set throughout this chapter.

Experiments	Train	Test	Precision	Recall	F-score
Rules (CA+CCO+CN)	340 PoFo	340 PoFo	0.615	0.507	0.555
PoFo poetry data	340 PoFo	340 PoFo	0.662	0.675	0.669
TroFi data	1771 Tr	1771 Tr	0.797	0.860	0.827
Shutova data	323 Sh	323 Sh	0.747	0.814	0.779
PoFo + TroFi + Shutova	4383 All	487 PoFo	0.759	0.804	0.781

Table 6.1: Results for the class *metaphor*

Table 6.1 shows the results for the class *metaphor*. For rule-based experiments, we included Concrete-Abstract, Concrete-Class-Overlap and ConceptNet features (CA, CCO, and CN). Training was done on 340 PoFo poem

Classifier	<i>metaphor</i>			<i>literal</i>		
	Precision	Recall	F-score	Precision	Recall	F-score
J48	0.710	0.615	0.659	0.574	0.675	0.620
Naive Bayes	0.663	0.665	0.664	0.564	0.561	0.563
Bayes Net	0.695	0.662	0.678	0.587	0.623	0.604
JRip	0.635	0.745	0.686	0.573	0.443	0.500
SVM (linear poly.)	0.656	0.742	0.696	0.597	0.495	0.541
SVM (norm. poly.)	0.657	0.767	0.708	0.614	0.481	0.540
ZeroR Baseline	0.565	1.000	0.722	0.000	0.000	0.000
Adaboost	0.760	0.713	0.735	0.655	0.708	0.680
Multilayer Perceptron	0.772	0.713	0.741	0.661	0.726	0.692
KNN	<b>0.782</b>	0.756	0.769	0.697	<b>0.726</b>	<b>0.711</b>
Random Forest	0.741	<b>0.822</b>	0.779	<b>0.731</b>	0.627	0.675
SVM (Puk)	0.759	0.804	<b>0.781</b>	0.724	0.670	0.696

Table 6.2: Results for classifiers trained on PoFo+TroFi+Shutova data, and tested on the 487 poetry sentences (sorted on *metaphor* F-score)

sentences, and testing on the remaining 340 sentences. For PoFo data, training and testing were the same, but with the word vector feature set instead of rules. For the TroFi data, training and testing was done on 1771 instances, each with the same feature set as PoFo. For Shutova’s data, training was done on 323 instances and testing on the other 323. Lastly, all the above datasets were aggregated as training data, in order to build a model and to test it on 487 PoFo sentences. Training for this aggregated set was done on 3543 TroFi instances, 647 Shutova instances, and the remaining 193 PoFo instances.

When analyzing the results, one can observe that the TroFi data give the best values overall. Still, a comparison of the PoFo results with the aggregate results shows that the values of all three metrics have drastically increased when the training data volume grew. The precision on isolated PoFo data is 0.662, whereas on aggregate data it is 0.759. This also establishes that in detecting metaphor in poetry non-poetry data are as helpful as poetry data.

It can be argued that the recall which we report is not the recall of metaphor throughout the whole poem. Instead, it is the recall of the specific POS tag sequence extracted by our algorithm. There can indeed be sentences that are metaphorical in nature, but are missed due to a different POS tag sequence. We agree with this argument, and therefore we work on a type-independent metaphor identification algorithm (explained in detail in

Classifier	p value	p value range	Statistically significant
J48			
Naive Bayes	0.2683	p>0.05	No
Bayes Net	0.0277	p<0.05	Yes
JRip	0.2881	p>0.05	No
SVM (linear poly.)	0.0317	p<0.05	Yes
SVM (norm. poly.)	0.2981	p>0.05	No
ZeroR Baseline	0.141	p>0.05	No
Adaboost	0.1127	p>0.05	No
Multilayer Perceptron	0.0172	p<0.05	Yes
KNN	0.0186	p<0.05	Yes
Random Forest	0.6693	p>0.05	No
SVM (Puk)	0.3227	p>0.05	No

Table 6.3: Statistical significance tests for the classification results given in Table 6.2. We compare each row with the previous row via t-tests, based on the F-score for the *metaphor* class.

Experiments	Train	Test	Precision	Recall	F-score
Rules (CA+CCO+CN)	340 PoFo	340 PoFo	0.462	0.408	0.433
PoFo poetry data	340 PoFo	340 PoFo	0.585	0.570	0.577
TroFi data	1771 Tr	1771 Tr	0.782	0.697	0.737
Shutova data	323 Sh	323 Sh	0.810	0.743	0.775
PoFo + TroFi + Shutova	4383 All	487 PoFo	0.724	0.670	0.696

Table 6.4: Results for the class *non-metaphor*

section 6.7) to handle such missing cases.

For data preprocessing, we have performed attribute selection by various algorithms, including Pearson’s, Infogain and Gain ratio (Yang and Pedersen, 1997). We report the results for the highest accuracy among these algorithms. For classification, we have used the following classifiers: Random Forest, JRip, J48, K-Nearest Neighbor, SVM (Linear Polynomial Kernel), SVM (Normalized Polynomial Kernel), SVM (Pearson Universal Kernel), Naïve Bayes, Bayes Net and Multilayer Perceptron. We have experimented with almost all classifiers available in the Weka software suite (Hall et al., 2009); we report the 10 best results.

Table 6.2 shows a comparison of the results for all classifiers that we tested on the PoFo+TroFi+Shutova data, keeping the training and test set exactly the same. The results are reported on the 487 poetry test data points, as

Experiments	Method	Precision	Recall	F-score
TroFi (our method)	Rule+Stat	0.797	0.860	<b>0.827</b>
TroFi Birke and Sarkar (2006)	Active Learning	N/A	N/A	0.649
Shutova (our method)	Rule+Stat	0.747	0.814	<b>0.779</b>
Shutova Shutova et al. (2016)	MIXLATE	0.650	0.870	0.750

Table 6.5: Results of the direct comparison with related work (Rule+Stat = rule-based and statistical)

noted before and sorted on *metaphor* F-score column. The ZeroR classifier is used as a baseline; it puts all the instances in the metaphor class, because it is the larger class with 56% of the instances. Table 6.3 shows the statistical significance of the classification results from Table 6.2. The p value is computed on the basis of the F-score for the *metaphor* class. Each row is compared with the previous row and the corresponding p value and the statistical significance is given. It can be observed that Adaboost is not better than the ZeroR baseline. The Multilayer Perceptron is significantly better than the baseline and Adaboost. KNN is significantly better than them. While SVM with the Puk kernel is the best, it is not significantly different from Random Forest and KNN.

For the results in Tables 6.1 and 6.4, the SVM classifier with the PUK kernel was used because it gave the best F-score for the metaphor class (as compared to other classifiers and to SVM with other types of kernels). For attribute selection, we used the Gain ratio evaluator.

Metaphor detection is our prime task, but we cannot ignore the *non-metaphor* class. We need to have an acceptable F-score for that as well, so as to maintain the credibility of our classification. Table 6.4 shows the results for the class *non-metaphor*. The precision values of the *metaphor* and *non-metaphor* classes are almost equal. On the other hand, the recall of the *non-metaphor* class is lower at 0.670 than for the class *metaphor* at 0.804. Error analysis (see section 6.5) showed that these *skipped* cases were mostly archaic words or poetic terms that do not have word vector representations. Still, we observe that the statistical method scored better than the rule-based method for all metrics.

Table 6.5 shows a direct comparison between our method – rule-based and statistical – and the methods of Shutova et al. (2016) and Birke and Sarkar

(2006) on their test data (not poetry). Our method performed better than the best-performing method MIXLATE (Shutova et al., 2016) on Mohammad et al.’s metaphor data (Mohammad et al., 2016). Our method also performed better than the Active Learning method of Birke and Sarkar (2006) on the TroFi dataset.

We also tested on 200-dimensional word vectors in order to investigate the effect of increasing the number of dimensions from 100 to 200 on accuracy metrics. Results showed that the accuracy dropped by 1%, along with a slight decline in the values of other metrics.

## 6.5 Error Analysis

#	PoFo sentence	Original class	Predicted class
1	my father ’s <b>farm</b> is an apple <b>blossomer</b> .	L	M
2	what is the answer ? the <b>answer</b> is the <b>world</b> .	L	M
3	long ago , this <b>desert</b> was an inland <b>sea</b> . in the mountains	L	M
4	so utterly absorbed that <b>love</b> is a <b>distraction</b> ; even	L	M
5	the <b>interviewer</b> was a <b>poet</b> . mann offered him no coffee , and	L	M
6	the body and the material things of the <b>world</b> are the <b>key</b> to any	L	M
7	though <b>beauty</b> be the <b>mark</b> of praise ,	L	M
8	strephon , who found the <b>room</b> was <b>void</b> ,	L	M
9	where <b>people</b> were <b>days</b> becoming months and years .	M	L
10	the <b>law</b> was <b>move</b> or die . lively from tigers	M	L
11	my <b>name</b> is a household <b>word</b> , writes the hid teacher	M	L
12	that the hot <b>wind</b> is <b>friend</b> , lifter of stones , trembler of heavy	M	L
13	<b>brilliance</b> is a <b>carcass</b>	M	L
14	to thee , whose <b>temple</b> is all <b>space</b> ,	M	L
15	<b>age</b> is <b>naught</b> but sorrow .	M	L

Table 6.6: A selection of incorrectly predicted PoFo sentences (L = literal, M = metaphorical)

Table 6.6 shows selected PoFo sentences that were predicted incorrectly by the classifier. We did error analysis on the PoFo test set to find the cause of these errors. The major cause was the absence of word vectors for certain

poetic words: *blossomer*, *fadere*, *hell-drivn*, and so on. Another significant cause was the presence of multi-word expressions not identified correctly by the parser, for example *household word* (#11).

Multiple word senses were also responsible for some of the errors, such as *key* in #6. There were also borderline cases which even human annotators found difficult to annotate (e.g., #2). Finally, quite a few errors were caused by the absence of compositionality while choosing word pairs. For example, *temple* and *space* in #14 are not enough to express a metaphor. There should be a composition of *all* and *space* as well, to capture the holistic meaning of the phrase. We aim to handle errors of those types in our future work in order to improve our classification.

## 6.6 Deep Learning Classification

The first requirement for deep learning classification is lots of examples / data points (by the thousands). Therefore, when we were able to collect more data (around 4870 instances) with metaphor (poetry and non-poetry), we experimented with Convolutional Neural Network (CNN) (Kim, 2014) to examine whether we can get any gains in F-score when compared to the standard machine learning classifiers. Figure 6.1 shows the details of the CNN text classifier schema.

We used the Keras<sup>2</sup> (Chollet et al., 2015) deep learning framework with a Tensorflow<sup>3</sup> (Abadi et al., 2015) backend and used a local Graphics Processing Unit (GPU) to accelerate the training process. The parameters that we tested on are given in Table 6.7.

The results of our experiments are given in Table 6.8. The best result, i.e., F-score 0.833 for metaphor and F-score 0.744 for the non-metaphor class was seen with epochs 300, batch 70, neurons 206 and inputs 103. Though we tested on hundreds of combinations of hyper-parameters, only the top results are being reported here for brevity. We also did significance testing to inspect whether the improvement due to optimizing parameters in Table 6.8 is statistically significant. We compared the *metaphor* F-score for the first and the last row of the table and the p value for the t-test was 0.0203

---

<sup>2</sup><https://keras.io/>

<sup>3</sup><https://www.tensorflow.org/>



Parameter	Range
Inputs	103 - 106
Input activation function	ReLU, TANH
Hidden layers	1 - 4
Neurons in 1st layer	6 - 306
Output activation function	Softmax, Sigmoid
Dropout	0 - 0.9
Outputs	2
Epochs	20 - 1000
Loss function	Categorical Cross Entropy, Binary Cross Entropy
Optimizer	ADAM
Batch size	20 - 200

Table 6.7: Ranges of parameters tested.

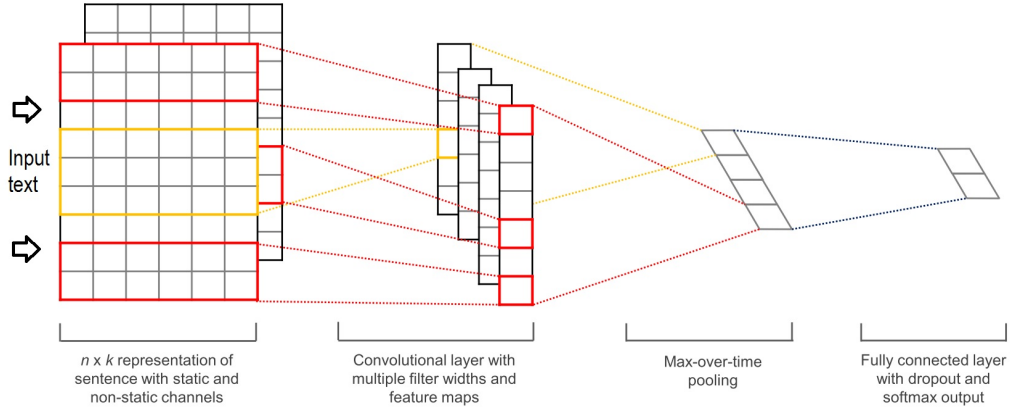


Figure 6.1: Schema diagram of the CNN text classifier.

( $p < 0.05$ ), hence the improvement is statistically significant.

It can be observed that with the same training and test set, CNN performed significantly better than SVM and other machine learning algorithms. The best F-score for metaphor class was 0.781, seen with SVM with Puk kernel. For CNN, we get a gain of 5.2% and we get a high F-score of 0.833. For the non-metaphor class, KNN obtained an F-score of 0.711. For CNN, we get a gain of 3.3% and we get a higher F-score of 0.744. For both classes, the performance is better with CNN.

The major drawback for the CNN classification (when compared to the other machine learning algorithms) is that a lot of data points are needed for training. Consequently, though we got better results for PoFo+TroFi+Shutova

Parameters	<i>metaphor</i>			<i>literal</i>		
	Precision	Recall	F-score	Precision	Recall	F-score
e:200 b:5 n:202 i:102	0.812	0.795	0.804	0.698	0.720	0.709
e:200 b:50 n:202 i:102	0.810	0.826	0.818	0.727	0.704	0.715
e:100 b:150 n:202 i:102	0.805	0.833	0.819	0.732	0.694	0.712
e:100 b:70 n:202 i:102	0.811	0.850	0.830	0.754	0.699	0.725
e:100 b:70 n:206 i:103	0.823	0.840	0.831	0.748	0.725	0.736
e:250 b:70 n:206 i:103	0.837	0.823	0.830	0.737	0.756	0.746
e:300 b:70 n:206 i:103	0.831	0.836	<b>0.833</b>	0.748	0.740	<b>0.744</b>

Table 6.8: Top results for CNN classification tested on variable hyper-parameters where e denotes epochs, b denotes batch size, n denotes the number of neurons in 1st layer and i denotes the number of inputs. All other hyper-parameter remain constant, input activation : ReLU and output activation : SOFTMAX. The difference between the first and the last row is statistically significant ( $p < 0.05$ ).

dataset collectively, results on the rest of the datasets (Table 6.1) individually were worse than the SVM results for both the classes. We empirically observed that, in our case, anything less than 3,000 instances was insufficient for training the CNN and the results are much worse (close to the baseline of 50%). If we are able to overcome this soft threshold of 3,000 data points, deep learning classification works appreciably better.

## 6.7 Web Application for Generic Metaphor Detection

Earlier we chose the potential word pairs to be checked for metaphoric intent based on POS tag sequence. For Type I metaphors, this sequence was noun-verb-noun or noun-verb-det-noun. But we found this to be too restrictive and hence we expanded the dependency checking we did for rectification of word pair detection into a full-fledged word pair choosing algorithm. Now this new method is not based on POS tag sequence checks, instead it works entirely on Stanford dependency parsing (de Marneffe et al., 2006). We termed this as Generic Metaphor detection. By applying generic metaphor detection, we were able to detect metaphors that earlier went undetected.

We used our best performing machine learning model (SVM with F-score 0.781) and developed a web application for:

1. Single line metaphor detection

## 2. Multi-line metaphor detection

These two web applications work with the generic metaphor detection algorithm and work for not just poetry, but also for any natural language text. We use the following dependencies to extract the potential word pairs:

- nsubj : nominal subject
- dobj : passive subject
- nsubjpass : direct object
- acl:relcl : relative clause modifier

Moreover, the pre-trained model (SVM with F-score 0.781) used for prediction in these two applications is serialized to decrease the execution time. It normally takes 10 - 12 seconds for execution. If serialization is not used, execution time can be as high as 40 seconds.

### 6.7.1 Single line metaphor detection

This application accepts single-line text and outputs the result of the analysis by enumerating the prediction of each potential word pair. Figure 6.2 shows a screenshot of the web application. The full text of the analysis is given below:

Sentence : I went to the classroom to absorb knowledge .

Processing went : i

Prediction : literal

Processing absorb : knowledge

Prediction : metaphor

Time taken : 9.889 secs

## Metaphor Detection in Natural Language

---

Enter a sentence:

I went to the classroom to absorb knowledge

Process

Sentence : i went to the classroom to absorb knowledge .

Processing went : i  
Prediction : literal

Processing absorb : knowledge  
Prediction : metaphor

Time taken : 9.889 secs

Figure 6.2: Screenshot of the single-line metaphor detection web application.

## Metaphor Detection in Natural Language

---

Enter text:

A woman measures her life's damage  
my eyes are caves, chunks of etched rock  
tied to the ghost of a black boy  
whistling  
crying and frightened  
her tow-headed children cluster  
like little mirrors of despair  
their father's hands upon them  
and soundlessly  
a woman begins to weep.

Process

Line : A woman measures her life's damage

Processing measures : woman  
Prediction : metaphor

Processing measures : damage  
Prediction : metaphor

Line : my eyes are caves, chunks of etched rock

Processing caves : eyes  
Prediction : metaphor

Figure 6.3: Screenshot of the multi-line metaphor detection web application. The full text is not captured in this screenshot.

### 6.7.2 Multi-line metaphor detection

This application accepts multi-line text and outputs line-by-line result for the analysis. There can be multiple word-pairs for each line that are analyzed for metaphoric intent. Figure 6.3 shows a screenshot of the web application. The poem (excerpt) Lorde (2000) entered in the application is given below:

Poem Title : Afterimages (excerpt)

Author : Audre Lorde

A woman measures her life's damage  
my eyes are caves, chunks of etched rock  
tied to the ghost of a black boy  
whistling  
crying and frightened  
her tow-headed children cluster  
like little mirrors of despair  
their father's hands upon them  
and soundlessly  
a woman begins to weep.

The complete result from the application is given below:

Line : A woman measures her life's damage

Processing measures : woman

Prediction : metaphor

Processing measures : damage

Prediction : metaphor

Line : my eyes are caves, chunks of etched rock

Processing caves : eyes

Prediction : metaphor

Line : her tow-headed children cluster

Processing cluster : children

Prediction : metaphor

Line : like little mirrors of despair

Processing like : mirrors

Prediction : metaphor

Line : their father’s hands upon them

Processing hands : father

Prediction : literal

Line : a woman begins to weep.

Processing begins : woman

Prediction : literal

Time taken : 15.814 secs

## 6.8 Conclusion and Future Work

To the best of our knowledge, this is the first work on the computational analysis of poetic metaphor. The preliminary results with Type I metaphor encourage us to continue, and to apply more methods. We are already working on dataset preparation for generic metaphor identification to increase the recall of our analysis. When it comes to rule-based methods, we could work on context overlap in order to remove the ambiguity between various senses that a word may have. This may increase the classification accuracy.

There are many statistical methods to look into. To begin with, in future work, we can analyze phrase compositionality (Mikolov et al., 2013b) in order to handle multi-word expressions and phrases better. Since we are identifying metaphor in word pairs rather than in the whole sentence, the accuracy of the vector representation for those words is crucial. If a word pair extracted by the algorithm does not represent the whole phrasal meaning, then the classification that follows may obviously prove inaccurate.

We are considering variations of CNN classifiers as well, so as to improve the F-score for the metaphor class further. Variations include LSTM (Hochreiter and Schmidhuber, 1997), Bi-LSTM (Graves and Schmidhuber, 2005) and C-LSTM for text classification (Zhou et al., 2015).

Next, we plan to distinguish between poetic and common-speech metaphor, a rather major undertaking. Finally, we plan to explore ways of quantifying commonalities and hierarchies between metaphor occurrences and thus develop metrics for metaphor quantification. Eventually, such a metric can be used in the graph rendering, in visualization, and in the analysis of poetry corpora.

The recent advances in natural language processing invite new and more consistent automatic approaches to the study of poetry. We intend to establish that poetry is amenable to computational methods. We also want to demonstrate that the statistical features which this research examines can indeed contribute significantly to the field of digital literary studies, and to academic poetry criticism and poetics in general. A case in point is our observation that non-poetry data are as helpful as poetry data in the task of metaphor detection in poetry.

So far, we have built on types of metaphor already defined by NLP scholars, and added two types we identified. Those types are based on parts of speech and syntactic structure.

In a perspective more explicitly informed by Digital Humanities, we will also explore the applicability of both established and unconventional approaches to metaphor in the humanities. It will therefore be interesting, for example, to look into the computability of metaphor as strictly POS-based (nominal, verbal, etc.) as a general framework, alongside marginal but intriguing concepts such as that of prepositional metaphor (Lakoff and Johnson, 2003). The latter has a not insignificant following in contemporary linguistics and stylistics (Goatly, 2011).

# CHAPTER 7

## CONCLUSIONS AND FUTURE WORK

### 7.1 Conclusions

The GraphPoem project addresses the following aspects of poetry analysis:

1. Topic classification (Lou et al., 2015)
2. Meter analysis (Tanasescu et al., 2016)
3. Rhyme detection
4. Diction analysis
5. Metaphor detection (Kesarwani et al., 2017)

This dissertation covers the last three aspects.

For rhyme analysis, we explained the detection methodology for rhyme types and subtypes. Then we explained how the rhyme is quantified to a score called RhymeScore and also detailed the alt-n rhyme. Our detection of alt-n rhyme pattern is quite novel. We did comparative analysis of various poems based on the RhymeScore values and finally illustrated the application for rhyme visualization.

For diction analysis, we explained all the linguistic and semantic features and their detection methodology. We compared our custom trained word embeddings for the PoFo model with the GloVe model. Then we gave an example involving quantified diction scores and concluded the chapter by describing the web application for diction analysis.

For metaphor analysis, we explained the methodology for the annotated poetry metaphor corpus that we created for Type I metaphors. Then we detailed the rule-based and statistical metaphor detection algorithms along with their respective results on our corpus and two other popular corpora.



Then we detailed our deep learning classifier for metaphor detection that utilizes CNN network. Lastly, we illustrated the generic metaphor detection algorithm and the web application that utilizes it for single and multi-line metaphor detection.

In conclusion, we have built classifiers for rhyme, diction and metaphor and quantified rhyme and diction for graph visualization. We are planning to do quantification for metaphor as well in the future.

We have built a comprehensive framework that tries to encompass all the facets of poetry analysis and we hope to continue refining on that in the future.

The research hypotheses that we started with in this dissertation have been proven correct. We did accomplish rhyme analysis by quantification and visualization. For diction, we were able to quantify the various quantitative features and analyzed the semantic ones. And lastly, for metaphor detection, we were able to successfully detect metaphor in poetry.

To conclude this dissertation, here is a quote (Bonta, 1985):

*The true poem rests between the words. - Vanna Bonta*

This dissertation is just a first step towards reading between the words of poetry.

## 7.2 Future Work

With the advent and percolation of deep learning into the field of NLP, it would be a perhaps naive thing to say that computational poetry and digital humanities as a whole will not be influenced by this advancement. We have tried to foresee this dramatic change and therefore, included elements of deep learning into our work (see sections 5.3.8, 6.6 above). We were quite encouraged by the results that we obtained and we hope to continue working in this direction in the future. We hope to include other deep learning models like RNN and LSTM (Hochreiter and Schmidhuber, 1997) to improve our classification tasks.

The ultimate objective of the GraphPoem project is not just to detect or classify, but also to visualize corpora as poem graphs and analyze them for

graph-theory-related features. Hence, in the future we will refine our poetic word embeddings and will try some other algorithms like t-SNE (Maaten and Hinton, 2008) to visualize poetry in a vector space. The resulting graphs will be analyzed for graph theory features like connectivity, cliques, cut nodes, percolation, etc. We will likely prefer a multiplex network (or multigraph) over a plain graph as our model is a particular case of network of networks (NoN). We also plan to refine, update and include in the web-based app the work on poetic topic, meter and syntax done by the other contributors to the GraphPoem project, while also expanding our work on poetic style by developing classifiers for other figures of speech besides metaphor.

## REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Abdul-Rahman, A., Lein, J., Coles, K., Maguire, E., Meyer, M., Wynne, M., Johnson, C. R., Trefethen, A., and Chen, M. (2013). Rule-based visual mappings—with a case study on poetry visualization. In *Computer Graphics Forum*, volume 32, pages 381–390. Wiley Online Library.
- Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning*, 6(1):37–66.
- Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.
- Assaf, D., Neuman, Y., Cohen, Y., Argamon, S., Howard, N., Last, M., Frieder, O., and Koppel, M. (2013). Why “dark thoughts” aren’t really dark: A novel algorithm for metaphor identification. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2013 IEEE Symposium on*, pages 60–65. IEEE.
- Baker, D. (1994). Murder. In *After the Reunion*. University of Arkansas Press.
- Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. ” O’Reilly Media, Inc.”.
- Birke, J. and Sarkar, A. (2006). A Clustering Approach for Nearly Unsupervised Recognition of Nonliteral Language. In *Proc. EACL*, pages 329–336.

- Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.
- Bonta, V. (1985). *Shades of the World*. Dora Books.
- Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.
- Chollet, F. et al. (2015). Keras. <https://github.com/fchollet/keras>.
- Clear, J. H. (1993). The British National Corpus. In *The digital word*, pages 163–187. MIT Press.
- CMU, S. (2007). Logios lexicon tool.
- Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242.
- Crabbe, G. (1950). *The village*. University Tutorial Press.
- Crane, H. (2006). *Complete Poems and Selected Letters*. Library of America.
- De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000). A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278.
- de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proc. LREC*, pages 449–454.
- Delmonte, R. (2013). Computing poetry style. In *ESSEM@ AI\* IA*, pages 148–155.
- Delmonte, R. (2015). Visualizing poetry with sparsar-visual maps from poetic content. In *CLfL@ NAACL-HLT*, pages 68–78.
- Delmonte, R. and Prati, A. M. (2014). Sparsar: An expressive poetry reader. In *EACL*, pages 73–76.
- Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.
- Eliot, T. S. (1915). The Love Song of J. Alfred Prufrock. *Poetry*, 6(3):130–135.
- Frost, R. (1962). The Road Not Taken. In *The Poetry of Robert Frost*. Holt, Rinehart & Winston.

- Ghazvininejad, M., Shi, X., Priyadarshi, J., and Knight, K. (2017). Hafez: an interactive poetry generation system. *Proceedings of ACL 2017, System Demonstrations*, pages 43–48.
- Goatly, A. (2011). *The Language of Metaphors*. Routledge.
- Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13).
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Honnibal, M. and Johnson, M. (2015). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.
- John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc.
- Kantosalo, A. and Riihiäho, S. (2014). Let’s play the feedback game. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, pages 943–946. ACM.
- Kao, J. T. and Jurafsky, D. (2015). A computational analysis of poetic style. *LiLT (Linguistic Issues in Language Technology)*, 12.
- Kaplan, D. M. and Blei, D. M. (2007). A computational approach to style in american poetry. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 553–558. IEEE.
- Keats, J. (2009). *Bright Star: Love Letters and Poems of John Keats to Fanny Brawne*. Penguin.
- Kesarwani, V., Inkpen, D., Szpakowicz, S., and Tanasescu, C. (2017). Metaphor detection in a poetry corpus. In *Proceedings of the Joint SIGHUM Workshop at Association for Computational Linguistics*, pages 1–9.

- Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.
- Koch, K. (1962). *Thank You, and other Poems*. Grove Press.
- Krause, E. F. (1975). *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation.
- Kutner, M. H., Nachtsheim, C., and Neter, J. (2004). *Applied linear regression models*. McGraw-Hill/Irwin.
- Lakoff, G. and Johnson, M. (2003). *Metaphors we live by*. University of Chicago Press.
- Landauer, T. K. and Dumais, S. T. (1997). A solution to plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26. ACM.
- Levin, E., Pieraccini, R., and Eckert, W. (1998). Using markov decision process for learning dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 201–204. IEEE.
- Lorde, A. (2000). *The collected poems of Audre Lorde*. WW Norton & Company.
- Lou, A., Inkpen, D., and Tanasescu, C. (2015). Multilabel Subject-Based Classification of Poetry. In *Proc. FLAIRS*, pages 187–192.
- Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.
- Madnani, N. (2005). Emily: A tool for visual poetry analysis. See: <http://www.umiacs.umd.edu/~nmadnani/emily/emily.pdf>.
- Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

- Manurung, H. (2004). An evolutionary algorithm approach to poetry generation.
- MARGENTO (2012). *NOMADOSOPHY*. Max Blecher Press.
- McCurdy, N., Lein, J., Coles, K., and Meyer, M. (2016). Poemage: Visualizing the sonic topology of a poem. *IEEE transactions on visualization and computer graphics*, 22(1):439–448.
- McCurdy, N., Srikumar, V., and Meyer, M. (2015). Rhymedesign: A tool for analyzing sonic devices in poetry. In *Proceedings of the Fourth Workshop on Computational Linguistics for Literature*, pages 12–22.
- Meneses, L. and Furuta, R. (2015). Visualizing poetry: Tools for critical analysis. *paj: The Journal of the Initiative for Digital Humanities, Media, and Culture*, 3(1).
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*, pages 3111–3119.
- Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.
- Milton, L. and Lu, C. Versevis: Visualization of spoken features in poetry.
- Mittmann, A., von Wangenheim, A., and Dos Santos, A. (2016). Aoidos: A system for the automatic scansion of poetry written in portuguese. In *17th International Conference on Intelligent Text Processing and Computational Linguistics*.
- Mohammad, S. M., Shutova, E., and Turney, P. D. (2016). Metaphor as a Medium for Emotion: An Empirical Study. In *Proc. \*SEM*, pages 23–33.
- Musaoglu, O., Dağ, Ö., Küçükakça, Ö., Salah, A. A. A., and Salah, A. A. (2017). A generic tool for visualizing patterns in poetry.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.
- Neuman, Y., Assaf, D., Cohen, Y., Last, M., Argamon, S., Howard, N., and Frieder, O. (2013). Metaphor Identification in Large Texts Corpora. *PLOS ONE*, 8(4):1–9.

- Odell, M. and Russell, R. (1918). The soundex coding system. *US Patents*, 1261167.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. In *Proc. EMNLP*, volume 14, pages 1532–1543.
- Philips, L. (2000). The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43.
- Plamondon, M. R. (2006). Virtual verse analysis: Analysing patterns in poetry. *Literary and Linguistic Computing*, 21(suppl.1):127–141.
- Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.
- Raleigh, S. W. (1895). The Passionate Man’s Pilgrimage. In Schelling, F. E., editor, *A Book of Elizabethan Lyrics*, pages 129–131. Ginn and Company.
- Reddy, S. and Knight, K. (2011). Unsupervised discovery of rhyme schemes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 77–82. Association for Computational Linguistics.
- Ritchie, L. D. (2013). *Metaphor (Key Topics in Semantics and Pragmatics)*. Cambridge University Press.
- Ryan, K., Young, D., Curdy, A., Stallings, A., Moritz, A., Collins, B., and Pound, E. (2005). And fit audience find. *Poetry*, 186(1):60–70.
- Ros, A. (2000). Glossary of rhymes.
- Schwartz, D. (1989). *Last & Lost Poems*, volume 673. New Directions Publishing.
- Shakespeare, W. (1904). *The Tragedy of Hamlet*. Cambridge University Press.
- Shutova, E., Kiela, D., and Maillard, J. (2016). Black Holes and White Rabbits: Metaphor Identification with Visual Features. In *Proc. 2016 NAACL: HLT*, pages 160–170.
- Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, 27(4):521–544.
- Speer, R. and Havasi, C. (2012). Representing General Relational Knowledge in ConceptNet 5. In *Proc. LREC*, pages 3679–3686.



- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.
- Stamatatos, E., Fakotakis, N., and Kokkinakis, G. (2000). Automatic text categorization in terms of genre and author. *Computational linguistics*, 26(4):471–495.
- Tanasescu, C., Inkpen, D., Kesarwani, V., and Paget, B. (2017). Access(ed) Poetry. The Graph Poem Project and the Place of Poetry in Digital Humanities. In *Proceedings of the DH2017 (Digital Humanities 2017) at McGill University*.
- Tanasescu, C., Paget, B., and Inkpen, D. (2016). Automatic Classification of Poetry by Meter and Rhyme. In *The Twenty-Ninth International Flairs Conference*.
- Turney, P. D., Neuman, Y., Assaf, D., and Cohen, Y. (2011). Literal and Metaphorical Sense Identification through Concrete and Abstract Context. In *Proc. EMNLP*, pages 680–690. Association for Computational Linguistics.
- Weide, R. (2005). The cmu pronouncing dictionary [cmudict. 0.7].
- Wright, J. (1958). At the Executed Murderer’s Grave. *Poetry*, pages 277–279.
- Yan, R. (2016). i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *IJCAI*, pages 2238–2244.
- Yang, Y. and Pedersen, J. O. (1997). A Comparative Study on Feature Selection in Text Categorization. In *Proc. ICML*, volume 97, pages 412–420.
- Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095.
- Zhang, X. and Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. In *EMNLP*.
- Zhou, C., Sun, C., Liu, Z., and Lau, F. (2015). A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.
- Zobel, J. and Dart, P. (1996). Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172. ACM.