AUTOMATIC POETRY CLASSIFICATION USING NATURAL LANGUAGE PROCESSING

BY

VAIBHAV KESARWANI

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulllment of the requirements
for the Master of Science in Computer Science

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

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. Metaphor is a part of diction indeed, due to the complexity involved in its detection, we dedicate an entire chapter to it.

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 did a comparative analysis with GloVe embeddings.

Previous work on metaphor detection relies on either rule-based or statistical models, none of them applied to poetry. Our method focuses on metaphor detection in a poetry corpus, but we test on non-poetry data as well. It combines rule-based and statistical models (word embeddings) to develop a new classification system. Our system has achieved a precision of

0.759 and a recall of 0.804 in identifying one type of metaphor in poetry by building a machine learning model. We also build a deep learning model for metaphor detection that achieves a precision of 0.831 and a recall of 0.836 in identifying one type of metaphor in poetry. 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.

A 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 Outline 3

CHAPTER 2 BACKGROUND 5
- 2.1 Natural Language Processing 5
- 2.2 Feature Engineering for Textual Data 7
- 2.3 Important NLP Tasks 12
- 2.4 Machine Learning 15
- 2.5 Deep Learning 18
- 2.6 Word Embeddings 22

CHAPTER 3 RELATED WORK ON POETRY COMPUTATIONAL ANALYSIS 25

CHAPTER 4 RHYME CLASSIFICATION 28
- 4.1 Introduction 28
- 4.2 Related Work 29
- 4.3 Background 30
- 4.4 Methodology 33
- 4.5 Results 35
- 4.6 Web Application for Rhyme Visualization 37
- 4.7 Conclusion 40
- 4.8 Future Work 41

CHAPTER 5 DICTION ANALYSIS 42
5.1 Introduction 42
5.2 Related Work 43
5.3 Method 43
5.4 Results 51
5.5 Web Application for Diction Analysis 53
5.6 Conclusion and Future Work 55

CHAPTER 6 METAPHOR DETECTION 56
6.1 Introduction 56
6.2 Related Work 57
6.3 Method 58
6.4 Results 63
6.5 Error Analysis 66
6.6 Deep Learning Classification 67
6.7 Web Application for Generic Metaphor Detection 69
6.8 Conclusions and Future Work 73

CHAPTER 7 CONCLUSION AND FUTURE WORK 75
7.1 Conclusion 75
7.2 Future Work 75

REFERENCES 77

LIST OF TABLES

2.1 Alphabetical list of part-of-speech tags used in the Penn Treebank Project. 9

4.1 ARPAbet phoneme set. Ph denotes phonemes, Ex denotes example words and Tr denotes translations. 31
4.2 Full Rhyme Example. 32
4.3 Slant Rhyme Example. 32
4.4 Rich Rhyme Example. 33
4.5 Mosaic Rhyme Example. 33
4.6 Rhyme Example. 34
4.7 Rhyme Example. 34
4.8 Rhyme Weights. 35
4.9 Dataset statistics and classification results. 36
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 RIVER in GloVe model 51
5.4 RIVER in PoFo model 51

6.1 Results for the class *metaphor* 62
6.2 Results for classifiers trained on PoFo+TroFi+Shutova data, and tested on the 487 poetry sentences 64
6.3 Results for the class *non-metaphor* 64
6.4 Results of the direct comparison with related work (Rule+Stat = rule-based and statistical) 65
6.5 A selection of incorrectly predicted PoFo sentences (L = literal, M = metaphorical) 67
6.6 Range of parameters tested. 68

6.7 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. 69

LIST OF FIGURES

2.1 Stanford basic dependencies example 8
2.2 A natural language classifier 13
2.3 Layers in a convolutional neural network. 19
2.4 An unrolled recurrent neural network. 20
2.5 Schematic diagram of the CBOW model. 23
2.6 Architecture diagram of the Skip-gram model. 24

4.1 A poem converted to its phonetic transcription. Title: Summer begins to have the look. Poet: Emily Dickinson . . . 32
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 . 39
4.5 On clicking a poem node, that specific poem displayed with rhyme types in color legend. 40

5.1 WordNet hypernym hierarchy for **happiness** (abstract) and {textbfshark (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. 68
6.2 Screenshot of the single-line metaphor detection web application. 71
6.3 Screenshot of the multi-line metaphor detection web application. The full text is not captured in this screenshot. 71

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 Intelligent 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. In turn, 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 intriguing. 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, punctuations, end-of-line do not provide the complete picture, as is the case with other genre of text like news or books. 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 differentiate a poem from each other and help to interpret and understand them.

Further, it has been observed that the field of computational linguistics has not permeated much to the classification and analysis of poetry. Poetry being such a complex sub-genre of literature demands that we indeed apply these novel techniques to their analysis. There have been studies to apply them to poetry but most of them limit to a specific genre or to a certain time period. This dissertation tries to focus 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 are done for this dissertation or we wish to do in the future, have never been done. This in itself is the biggest motivation of all.

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, visualise, 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 and enjambments) (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 rhyme classification system with rhyme types and rhyme sub-types.
3. We develop a diction classification system with multiple features like verbal density, inflection ratio, etc.
4. We introduce a metaphor poetry dataset. This dataset is for a single type of metaphor and was annotated by us.
5. We develop a metaphor classification system based on our dataset and some third-party ones.

1.4 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 give future directions 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

2.1 Natural Language Processing

Natural Language Processing (NLP) is at the intersection of Computer Science, Artificial Intelligence, and Linguistics and it consists of systematic processes for analyzing, understanding, and deriving information from text data in a smart and efficient manner. By utilizing NLP and its components, one can organize massive chunks of text data, perform numerous automated tasks, and solve a wide range of problems such as automatic summarization, machine translation, named entity recognition, relation extraction, sentiment analysis, speech recognition, and topic segmentation.[1]

Since text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as *text preprocessing*.

Text preprocessing is predominantly comprised of three steps[1]:

- Noise Removal
- Lexicon Normalization
- Object Standardization

2.1.1 Noise Removal

Any piece of text which is not relevant to the context of the data and the processing goal can be considered noise.

[1]https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

Some examples of noise are stopwords (frequently used words of a language such as *is, am, the, of, in,*), URLs or links, social media entities (mentions, hashtags), punctuation, and markup. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noise entities, and iterate trough the text's tokens in order to eliminate those tokens which are present in the noise dictionary. Another approach is to use regular expressions to detect special patterns of noise.

2.1.2 Lexicon Normalization

Another type of preprocessing detects multiple representations exhibited by single word.

For example, *play, player, played, plays* and *playing* are the different variations of the word *play*. This step converts all the inflected forms of a word into their base form (also known as lemma). Normalization is a pivotal step for feature engineering for text data, as it converts the high dimensional features (N) to a lower dimensional space (that has only one feature for each base form), which is better for machine learning.

The most common lexicon normalization practices are :

- Stemming: Stemming is a rudimentary rule-based process of stripping the suffixes (*ing, ly, es, s*, etc.) from a word.

- Lemmatization: Lemmatization, on the other hand, is an organized and step-by-step procedure of obtaining the base form of the word; it uses vocabulary (dictionary of words) and morphological analysis (word structure and grammar relations).

2.1.3 Object Standardization

Text data often contains words or phrases which are not present in any standard lexical dictionaries. These pieces are not recognized by search engines and models.

Some examples are acronyms, hashtags, and colloquial slangs. With the help of regular expressions and manually-prepared dictionaries, this type of noise can be removed.

2.2 Feature Engineering for Textual Data

To analyse a preprocessed data, it needs to be converted into features. Depending upon the usage, text features can be constructed using assorted techniques[2]:

- Syntactic parsing
- Entities / n-grams / word-based features
- Statistical features
- Word embeddings

2.2.1 Syntactic Parsing

Syntactic parsing involves the analysis of words in the sentence for grammar and their arrangement in a manner that shows the relations among the words. Dependency relations and part-of-speech tags are the important attributes of syntax.

Dependency Trees: Sentences are composed of some words sequenced together. The relationship among the words in a sentence is determined by the basic dependency grammar. A dependency grammar is a class of syntactic analysis that deals with (labeled) asymmetrical binary relations between two lexical items (words). Every relation can be represented in the form of a triplet (relation, governor, dependent). For example[3]: consider the sentence *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.* The relation among the words can be observed in the form of a tree representation as shown in Figure 2.1.

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 is a itself a dependency tree with relations such as (*Bills – ports* by *proposition* relation), (*ports – immigration* by *conjugation* relation).

This type of tree, when parsed recursively in top-down manner gives grammar relation triplets as output which can be used as features for many NLP

[2]https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

[3]https://nlp.stanford.edu/software/stanford-dependencies.shtml

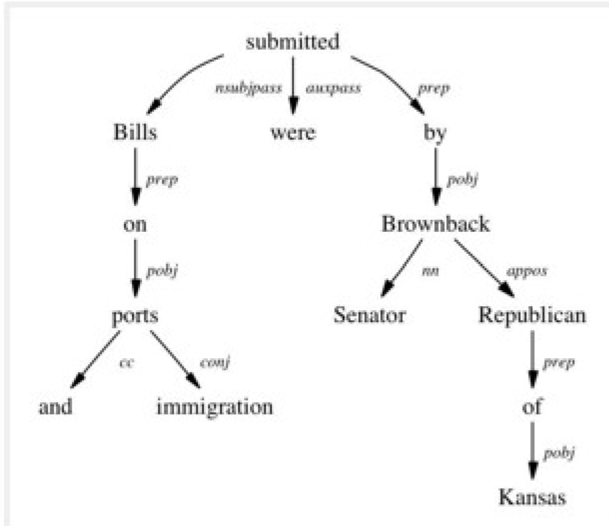


Figure 2.1: Stanford basic dependencies example

problems like entity wise sentiment analysis, actor and entity identification, and text classification. StanfordCoreNLP (Manning et al., 2014) (from the Stanford NLP Group) and NLTK (Bird et al., 2009) dependency parsers can be used to generate dependency trees.

Part-of-speech tagging: Apart from the grammar relations, every word in a sentence is also associated with a part of speech (POS) tag like nouns, verbs, adjectives, adverbs, etc. The POS tag defines the usage and function of a word in the sentence. Table 2.1 shows the list of all possible POS tags defined by the Penn TreeBank Project.

Part of Speech tagging is used for many important purposes in NLP:

Word sense disambiguation: Some language words have multiple meanings according to their usage. 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 different context, 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.

Tag	**Description**
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
VBN	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.

Improving word-based features: A learning model could learn different contexts of a word when using words as features; however, if POS tags are also used as features, the context is preserved, thus allowing for stronger features.

Normalization and Lemmatization: POS tags are the basis of the lemmatization process for converting a word to its base form (lemma).

Efficient stopword removal: POS tags are also useful in efficient removal of stopwords. There are some tags which always define the low frequency or function words of a language. For example, *IN* that denotes Preposition or subordinating conjunction in the Penn Treebank tag list.

2.2.2 Entities / N-grams / Word-based Features

Entities are defined as the most important chunks of a sentence, such as noun phrases and verb phrases. Entity Detection algorithms are generally ensemble models of rule-based parsing, dictionary lookups, POS tagging, and dependency parsing. The applicability of entity detection can be seen in automated chat bots, content analyzers, etc.

The three key entity detection methods in NLP are:

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

Named Entity Recognition: The process of detecting the named entities such as person names, location names, company names, etc. from the text is called as Named Entity Recognition (NER). For example:

Sergey Brin, the manager of Google Inc. is walking on the streets of New York.

Named Entities : *person : Sergey Brin, org : Google Inc., location : New York*

A typical NER model consists of three blocks:

- Noun phrase identification: This step deals with extracting all the noun phrases from a text using dependency parsing and POS tagging.

- Phrase classification: This is the classification step in which all the extracted noun phrases are classified into respective categories (locations, names, etc.). The open databases from DBPedia or Wikipedia can be used to identify person names or company names. Apart from this, one can curate the lookup tables and dictionaries by combining information from different sources.

- Entity disambiguation: Sometimes it is possible that entities are misclassified; therefore, creating a validation layer on top of the results is useful. Use of knowledge graphs can be exploited for this purposes. The popular knowledge graphs are DBpedia and Wikidata.

Topic Modeling: Topic modeling is a process of automatically identifying the topics present in a text corpus; it derives the hidden patterns among the words in the corpus in an unsupervised manner. Topics are defined as *a repeating pattern of co-occurring terms in a corpus.* A good topic model results in *health, doctor, patient, hospital* for a topic *Healthcare*, and *farm, crops, wheat* for a topic *Farming.* Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is the most popular topic modeling technique.

N-Grams as Features: A combination of N words together are called N-Grams. N grams (N >1) are generally more informative as compared to words (unigrams) as features. Also, bigrams (N = 2) are considered as a very important feature.

2.2.3 Statistical Features

Text data can also be quantified directly into numbers using several techniques described in this section:

Term Frequency–Inverse Document Frequency (TF-IDF): TF-IDF is a weighted model commonly used for information retrieval problems. It aims to convert the text documents into vector models on the basis of occurrence of words in the documents without taking considering the exact ordering. For Example, 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 logarithm of 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:

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

$$\mathrm{tf}_{\mathrm{i,\ j}} = \text{number of occurrences of i in j}$$
$$\mathrm{df}_{\mathrm{i}} = \text{number of documents containing i}$$
$$\mathrm{N} = \text{total number of documents}$$

2.2.4 Word Embeddings

This will be explained in detail in Section 2.6.

2.3 Important NLP Tasks

This section talks about different use cases and problems in the field of natural language processing[4].

2.3.1 Text Classification

Text classification is one of the classical problems of NLP. Text classification is defined as a technique to systematically classify a text object (document or sentence) in one of a fixed number of categories. It works well when the amount of data is large, especially for organizing texts, information filtering, and storage purposes. Some examples include email spam detection, topic classification of news, sentiment classification, etc.

[4]https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

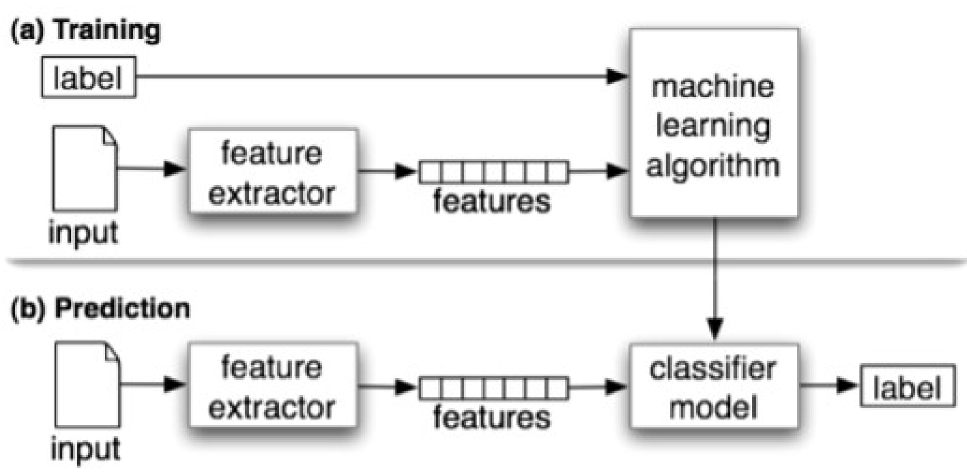


Figure 2.2: A natural language classifier

A typical 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 created. The machine learning model then learns how these features associate with categories, and the model can be used to make predictions on a new text.

The text classification models are heavily dependent upon the quality of the features. While applying any machine learning model, it is always a good practice to include more a lot of training data (and enough relevant training data for all categories) to improve the predictions.

2.3.2 Text Similarity

One of the important areas of NLP is the matching of text objects to find similarities. Important applications of text matching includes semantic analysis, data de-duplication, etc. A number of text matching techniques are available depending upon the requirements. Some important techniques for text similarity are:

- Levenshtein Distance
- Phonetic Matching
- Cosine Similarity
- Block Distance

- Euclidean distance
- Latent Semantic Analysis

Levenshtein Distance : The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character.

Phonetic Matching : A Phonetic matching algorithm takes a keyword as input (persons name, location name etc) and produces a character string that identifies a set of words that are (roughly) phonetically similar. It is very useful for searching large text corpora, correcting spelling errors and matching relevant names. Soundex and Metaphone are two main phonetic algorithms used for this purpose.

Cosine Similarity : When the text is represented as vector notation, a general cosine similarity can also be applied in order to measure vectorized similarity.

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 which called singular value decomposition (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 is a process of finding relational links among the words (or phrases) within the sentences. Consider an example sentence:

Donald went to John's office to see the new table. He looked at it for an hour.

Humans can quickly figure out that *he* denotes Donald (and not John), and that *it* denotes the table (and not John's office). Coreference resolution is the component of NLP that does this job automatically. It is used in document summarization, question answering, and information extraction.

2.4 Machine Learning

Machine learning (ML) is a subset of Artificial Intelligence that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output value within an acceptable range[5]. There are broadly three types of ML algorithms[6]:

- Supervised Learning : These algorithms consist of a target / category (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples: Regression, Decision Tree, Random Forest, KNN, Logistic Regression, etc.

- Unsupervised Learning : In these algorithms, we do not have any target or category to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples : Apriori algorithm, K-means.

[5] http://whatis.techtarget.com/definition/machine-learning

[6] https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

- Reinforcement Learning : Using this algorithm, the machine is trained to make specific decisions. The machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example : Markov Decision Process.

List of commonly used machine learning algorithms are[6]:

2.4.1 Linear Regression

Linear Regression is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationships between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y = a*X + b.

2.4.2 Logistic Regression

Logistic Regression is used to estimate discrete values (binary values like 0/1, yes/no, true/false) based on a given set of independent variable(s). In other words, it predicts the probability of the occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

2.4.3 Decision Tree

Decision Tree (Quinlan, 2014) is a type of supervised learning algorithm that is mostly used for classification problems. It works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/independent variables to make as distinct groups as possible. To split the population into different heterogeneous groups, it uses various techniques like Gini, Information Gain, Chi-square, entropy, etc.

2.4.4 Support Vector Machines

Support Vector Machines (SVM) (Boser et al., 1992) is a classification method in which we plot each data item as a point in an n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we have two features like height and weight of an individual, we first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as Support Vectors). Now, we find a line (discriminant function) that splits the data between these two differently classified groups of data. This line is chosen in such a way that the distances between the closest point in each of the two groups is maximized.

2.4.5 Naive Bayes

Naive Bayes (John and Langley, 1995) is a classification technique based on Bayes theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes classifier would consider all of these properties to independently contribute to the probability of the classification, in order to simplify the computation.

2.4.6 KNN (K-Nearest Neighbors)

KNN (Aha et al., 1991) can be used for both classification and regression problems. However, it is more widely used in classification problems. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class, that is the most common amongst its K nearest neighbors and measured by a distance function. These distance functions can be Euclidean, Manhattan, Minkowski, or Hamming. The first three are used for continuous functions and the fourth one for categorical variables.

2.4.7 K-Means Clustering

K-Means (Arthur and Vassilvitskii, 2007) is a type of unsupervised algorithm that addresses the clustering problem. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable k. The algorithm works iteratively to assign each data point to one of k groups based on the features that are provided [7].

2.4.8 Random Forest

Random Forest (Breiman, 2001) is a term used for an ensemble of decision trees. In a Random Forest, we have a collection of decision trees known as a *forest*. To classify a new object based on attributes, each tree gives a classification and we say the tree *votes* for that class. The forest chooses the classification having the most votes over all the trees in the forest.

2.5 Deep Learning

Deep learning is a class of machine learning algorithms that[8]:

- 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) manners.
- 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.

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-

[7] http://stanford.edu/ cpiech/cs221/handouts/kmeans.html
[8] https://en.wikipedia.org/wiki/Deep_learning

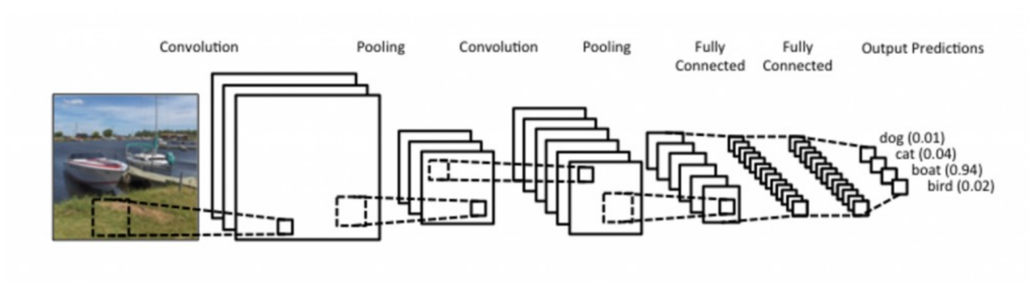


Figure 2.3: Layers in a convolutional neural network.

vation functions like ReLU (Nair and Hinton, 2010) or tanh applied to the results. In a traditional feedforward neural network, each input neuron is connected to each output neuron in the next layer. This is also called a fully connected layer, or affine layer. In CNNs, instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands, and combines their results. During the training phase, a CNN automatically learns the values of its filters based on the task. For example, in image classification, a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features[9].

Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents 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 10 word sentence using a 100-dimensional embedding we would have a 10x100 matrix as our input[9].

[9]http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

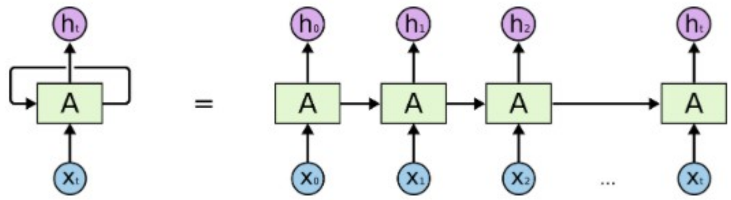


Figure 2.4: An unrolled recurrent neural network.

2.5.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are used for sequential data where the previous output is used to predict the next one and hence these networks have loops within them. The loops within the hidden neurons gives them the capability to store information about the previous words for some time, to be able to predict the output. The output of the hidden layer is sent again to the hidden layer for t time stamps. The unfolded neuron is shown in figure 2.4. The output of the recurrent neuron goes to the next layer only after completing all the time stamps. The output sent is more generalized and the previous information is retained for a longer period[10].

Some commonly used terms in deep learning are explained below[11] [12]:

- Activation Function: To allow neural networks to learn complex decision boundaries, we apply a nonlinear activation function to some of its layers. Commonly used functions include sigmoid, tanh, ReLU (Rectified Linear Unit) and variants of these.
- Attention Mechanism: These are inspired by human visual attention, the ability to focus on specific parts of an image. Attention mechanisms can be incorporated in both language processing and image recognition architectures to help the network learn what to *focus* on when making predictions.

[10]https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/

[11]http://www.wildml.com/deep-learning-glossary/

[12]https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/

- Backpropagation: It is an algorithm to efficiently calculate the gradients in a Neural Network, or more generally, a feedforward computational graph. It boils down to applying the chain rule of differentiation starting from the network output and propagating the gradients backward.

- Batches: While training a neural network, instead of sending the entire input in one go, we divide in input into several chunks of equal size randomly. Training the data on batches makes the model more generalized as compared to the model built when the entire data set is fed to the network in one go.

- Dropout: It is a regularization technique for neural networks that prevents overfitting (Srivastava et al., 2014). It prevents neurons from co-adapting by randomly setting a fraction of them to 0 at each training iteration. Dropout can be interpreted in various ways, such as randomly sampling from an exponential number of different networks.

- Epochs: An epoch is defined as a single training iteration of all batches in both forward and back propagation. One epoch is a single forward and backward pass of the entire input data.

- 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 units to calculate the hidden state in an RNN, we help the network to efficiently propagate gradients and learn long-range dependencies.

- Pooling: It is common to periodically introduce pooling layers in between the convolution layers. This is basically done to reduce a number of parameters and prevent over-fitting. The most common type of pooling is a pooling layer of filter size(2,2) using the MAX operation.

- ReLU: Short for Rectified Linear Unit(s). ReLUs 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 functions like tanh include that they tend to be sparse (their activation is easy 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)=1/(1+eˆ-x). This is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear.
- Softmax: The softmax function is typically used to convert a vector of raw scores into class probabilities at the output layer of a neural network used for classification. It normalizes the scores by exponentiating and dividing by a normalization constant.

2.6 Word Embeddings

Word embeddings are a novel way of representing words as vectors. The aim of word embedding is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are widely used in deep learning models such as Convolutional Neural Networks and Recurrent Neural Networks.

Word2Vec (Mikolov et al., 2013a) and GloVe (Pennington et al., 2014) are the two popular models to create word embeddings of a text corpus. These models take a text corpus as input and produce the word vectors as output.

Word2Vec model is composed of 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). It first constructs a vocabulary from the training corpus and then learns word embedding representations from them. They can be used as feature vectors for ML models, for measuring text similarity using cosine similarity techniques, for clustering, and other text classification tasks[13].

2.6.1 CBOW (Continuous Bag of Words)

The way CBOW works is that it tends to predict the probability of a word given a context. A context may be a single word or a group of words. The matrix with the one-hot encoded input is sent into a shallow neural network

[13] https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

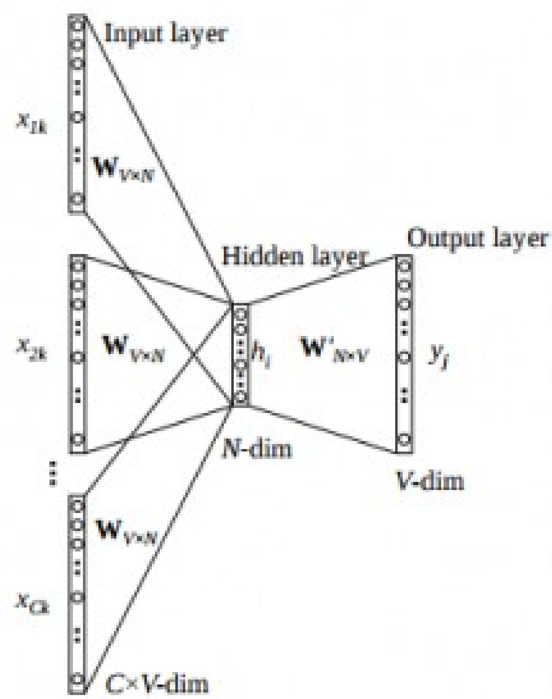


Figure 2.5: Schematic diagram of the CBOW model.

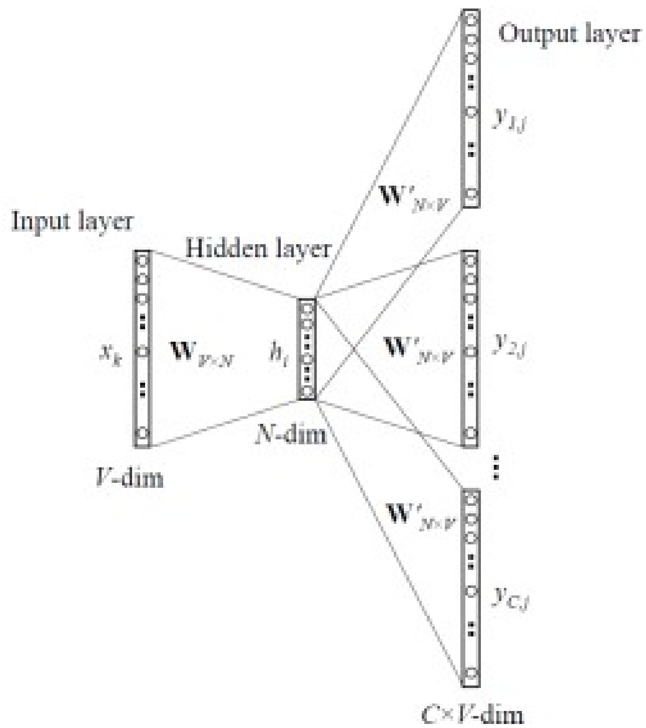


Figure 2.6: Architecture diagram of the Skip-gram model.

with three layers: an input layer, a hidden layer and an output layer. The output layer is a softmax layer which is used to sum the probabilities obtained in the output layer to 1. Figure 2.5 shows a schematic diagram of the CBOW model.

2.6.2 Skip-Gram model

The skip-gram model follows the same topology as CBOW, but it has the reverse architecture of CBOW. The aim of skip-gram is to predict the context given a word. The input vector for skip-gram is going to be similar to a 1-context CBOW model. The weights between the input and the hidden layer are taken as the word vector representation after training. The loss function or the objective is of the same type as in the CBOW model. Figure 2.6 shows the architecture diagram of the skip-gram model.

CHAPTER 3

RELATED WORK ON POETRY COMPUTATIONAL ANALYSIS

There have been numerous works on computational poetry in the past that have aimed to visualize 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 (Name Entity Recognition) tools, and taggers. In addition the system adds syntactic and semantic structural analysis and prosodic modeling. They analyse 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 do semantic and syntactic analysis of poetry, but also do visualization on a graph with connected edges. Delmonte (2015) generates three relational views of a poem: phonetic, poetic and semantic. Each of these views show 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 a way that it aims to visualize sonic elements of a poem primarily rhyme types as a path view diagram. While we consider sonic feature an important factor for poem analysis, we do not consider it a skeleton but a leg in multi-faceted poem analysis and hence we consider a poem as an extension of natural text. For the same reason, we delve into metaphor detection (explained in Chapter 6) considering it a semantic exercise 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 natural text.

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 phoneme. The major difference with 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 show 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 length. User can also perform unweighted and weighted search on the poem content. We have also applied keyword search in our diction application, but we not just show 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, instead as a group and connect them graphically 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. 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) that are very computationally expensive techniques. Instead, we build word embeddings by 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 tremendous 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 a RNN decoder. 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 RCM (Recurrent Context model) and RGM (Recurrent Generation model). Though we do not directly do poetry generation, but nevertheless we keep a scanner on the field to see which metrics are the most critical for agreeableness of poems, and we make sure to include them in our reverse task of poetry analysis.

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

Earlier, a similar effort was done for building a system named Aoidos (Mittmann et al., 2016) for Portuguese poetry. The notable differences with our system are that no classification is done on the rhyme sub-types and the absence of multiple pronunciations for a word. Portuguese being a purely phonetic based language, it does not pose the same flexibility while transcribing words as seen for English. Further, testing for Aoidos does not involve any classifiers and based on human judges giving acceptable/unacceptable ratings for rhyme detected by the system. Our system aims to take care of all the intermediate error cases as well that go undetected in fully supervised techniques.

RhymeDesign (McCurdy et al., 2015) is another 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 focuses 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) uses rhyme quantification methodology similar to ours, but the usage context is different. They use quantification with a generative model for machine translation of the poems to another language and maintain the rhyme and metrical integrity in the process. Our focus is on the classification aspects instead, to augment the recommendation of poems based on their rhyme similarity.

SPARSAR, another system for automatic poetry analysis by Delmonte (2013) detects meter, rhyme and semantic structures of poetry. This system does not use any classifiers for classification purposes; instead, it uses rule-based patterns (AABBCC, AABCDD, etc.). Similarly, an earlier program named AnalysePoems by Plamondon (2006) also used rule-based patterns for classifying rhymes in poetry. This system also handles a large corpus of poems, just like ours.

The analysis of poetic style by Kao and Jurafsky (2015) uses rhyme sub-types like 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 is more inclusive of a larger number of poems from various periods. In our case, the patterns extracted, if any, will not be manual, but automatically-detected.

4.3 Background

To detect rhyme, we need phonetic transcription of the poem. We use CMU-Dict (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 overall contextual setup and not by the individual word itself. Lastly, archaic words are very poorly represented in the phonetic dictionary. For all

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.

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 with its phonetic transcription.

Rhyme is the similarity of phonemes and stress fall is given by 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 line pause and the last word of the line 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 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 nature of similarity of phonemes (Ros, 2000). Full rhyme refers to a perfect match:

In the above example, stress fall is exactly in the similar region of phonemes,

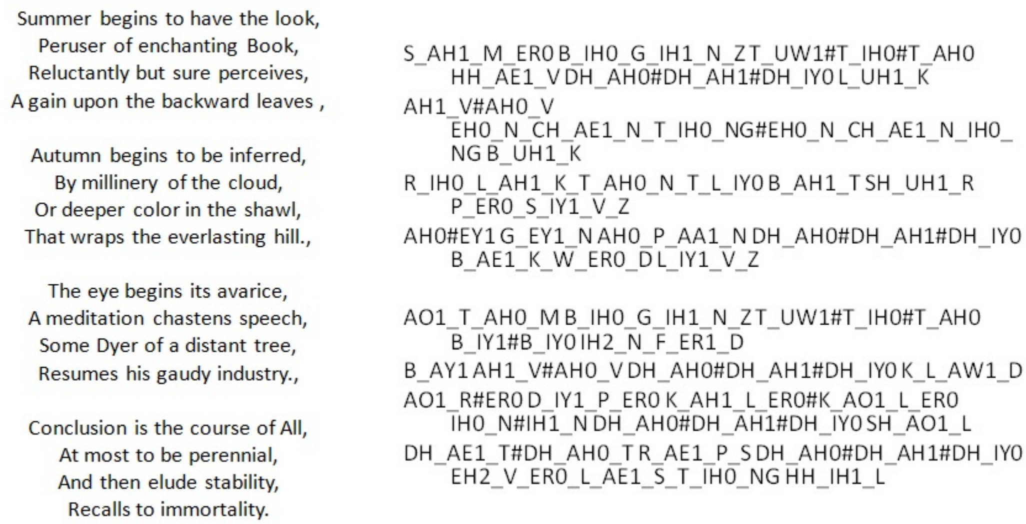


Figure 4.1: A poem converted to its phonetic transcription. Title: Summer begins to have the look. Poet: Emily Dickinson

True	T	R	UW1
Blue	B	L	UW1

Table 4.2: Full Rhyme Example.

hence we have a Full rhyme. Slant rhyme refers to a non-perfect match:

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:

In the above case, the two words have exactly the same phonemes. For all the above rhyme sub-types, the similarity is done by phonetic transcription. 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

Blue	B	L	UW1
Blew	B	L	UW1

Table 4.4: Rich Rhyme Example.

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:

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 in converted to its phonetic transcription. A list is generated for all the end words and another for all the line pause or caesura words. To detect the end rhyme, the list with 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 normalised to denote rhyming at poem level. Each rhyme type has its own RhymeScore.

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

As stated earlier, a word may have different pronunciations. For example, the word *process* has two pronunciations:

Thus all possible phoneme combinations are tried and all possible RhymeScore$_{\text{word}}$ are calculated for that word pair and the maximum of all scores is taken as

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

Table 4.6: Rhyme Example.

the final $\text{RhymeScore}_{\text{word}}$. 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:

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

Just matching each phoneme with the corresponding phoneme of another word is not enough. Rhyme also depends on the positioning of each phoneme in a word. For example, we may have different vowels on a specific position in two words, they may not match, but the stress may be the same. Therefore, we cannot term it as zero rhyme phoneme. There has to be some intermediate scoring so as to give these not-perfect matches some weight. Here is 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.

*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:

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

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

Table 4.8: Rhyme Weights.

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

To make this score independent of the length of the poem, it is normalised by a Normalisation factor $\mathrm{norm_f}$.

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

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

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

For alternate end rhyming, it is equal to:

$$\mathrm{norm_f} = \text{Number of lines in poem - 2}$$

For Internal rhyme, it is equal to:

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

After calculating RhymeScores for all rhyme types and sub-types for all poems, we can have a similarity classification.

4.5 Results

Our input set is a corpus of 12,831 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 establish the accuracy of our classification. These poems have been annotated by two human judges, with

Title	**Dev Set**	**Test Set**
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.

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.

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 little End rhyme. Internal rhyme is present; this is rare, as it appears in very few poems. It possesses 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 a very high score for Full rhyme. 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. We used 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:

- pan in all four directions
- zoom in and out

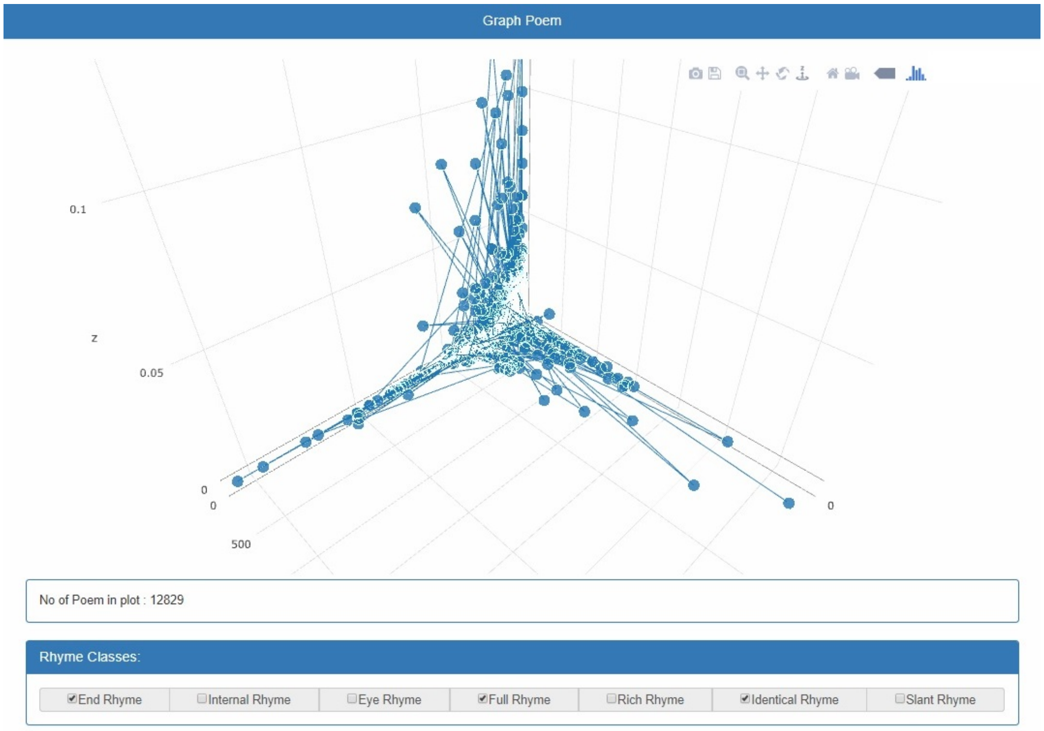


Figure 4.2: A 3D scatter plot of PoFo poems

- orbital rotation in a 360 degree plane to rotate all the axes
- 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 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 4.4 shows a small region that was selected in the 2D scatter plot. In both the 2D and 3D plots, if the mouse is hovered on a poem node, a

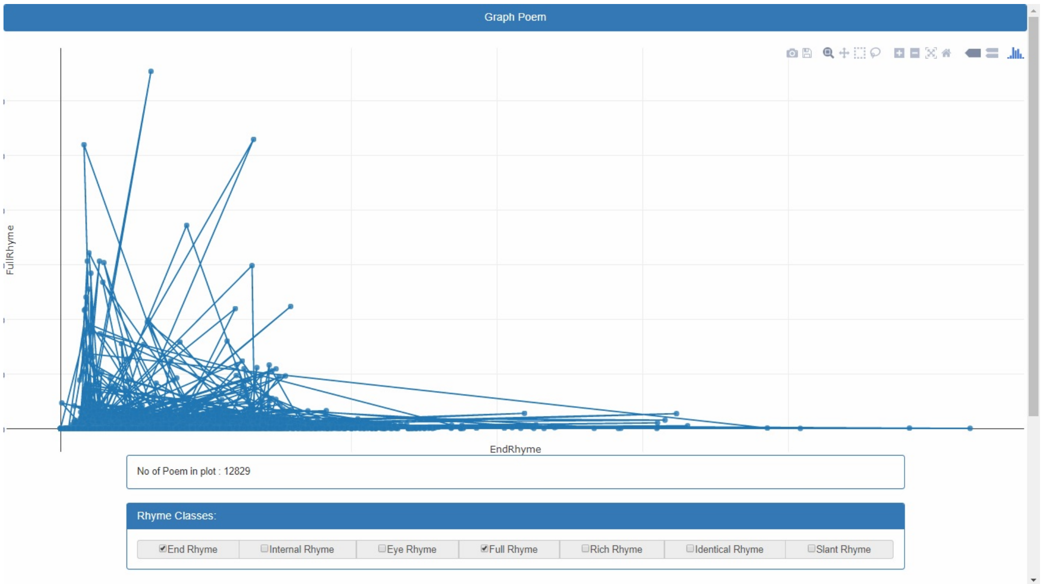


Figure 4.3: A 2D scatter plot of PoFo poems

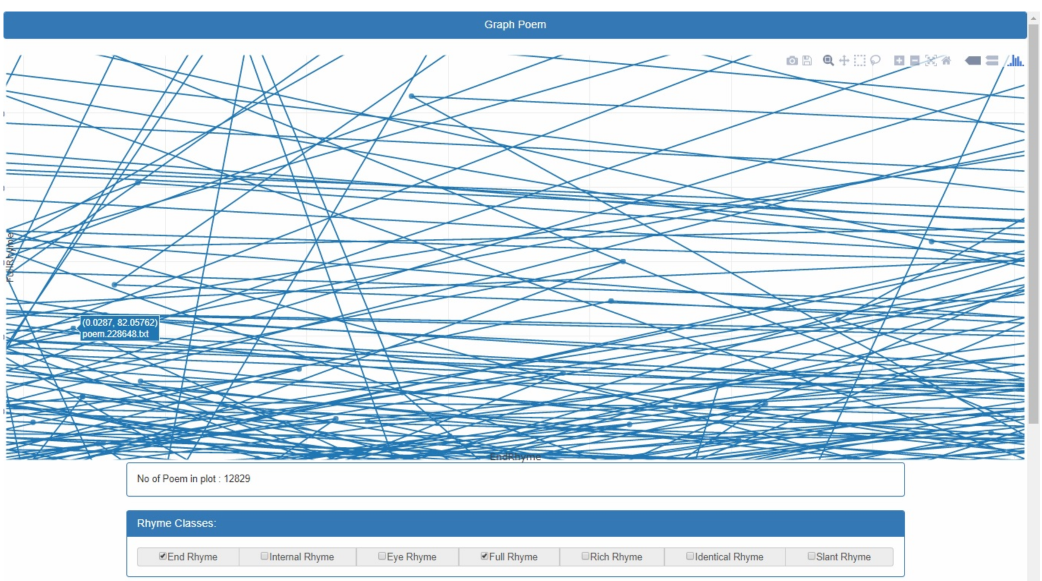


Figure 4.4: Poem RhymeScore displayed on mouse over event in the plot

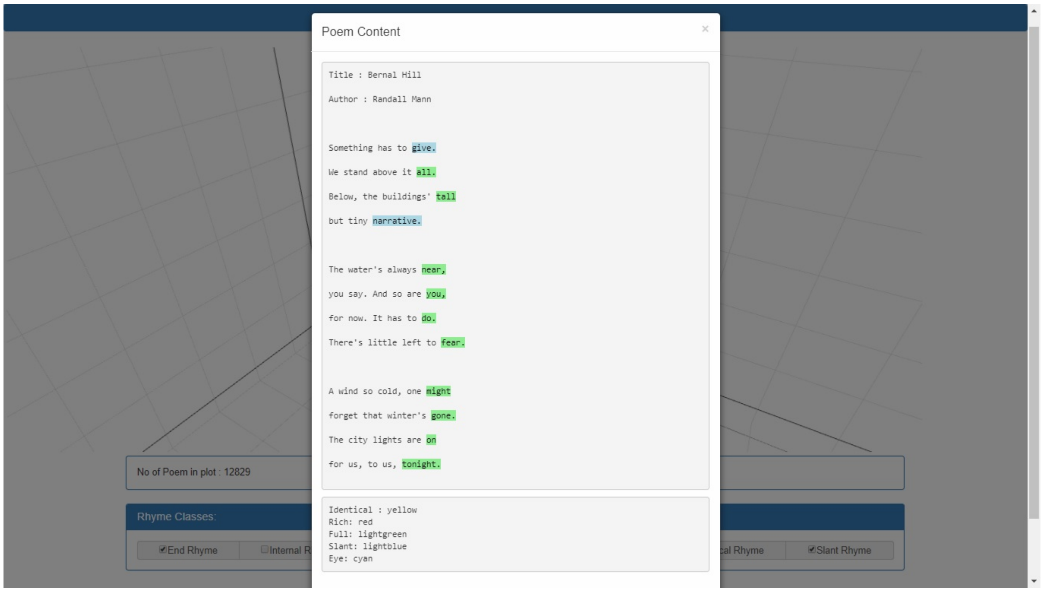


Figure 4.5: On clicking a poem node, that specific poem displayed with rhyme types in color legend.

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

We have built a rhyme classifier based on CMUDict phoneme dictionary that classifies poem lines into rhyme types and subtypes. Then we quantified rhymes into a normalized RhymeScore for each poem so that we may visualize a poem graphically on the basis of its rhyme footprint. For visualization, we built a web application for generating a scatter plot based on the metrics obtained.

4.8 Future Work

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.

In the Alternate rhyme type, we are currently considering alternation with a line difference of 1. There are cases where rhyme is present even far apart. To consider those cases, we need to check for rhyme in an alt-n pattern, where n is an integer greater than 1 and smaller than length of poem minus 2.

Further, we also aim to increase our corpus size in future. We are considering including more poems from Representative Poetry Online repository by Plamondon (2006). This would increase the richness of our analysis. Inclusion of more rhyme types like Mosaic, Assonant, Consonant, Broken, etc., is also possible in 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 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 linguistic features like 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 object(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. Our features closely resemble theirs and 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 diction analysis of poetry that can be visualized on a graphical model. Further, 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.

5.3 Method

5.3.1 Abstract/Concrete ratio

The Abstract/Concrete ratio is the ratio of 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 is employed to get this categorization. In WordNet, all nouns possess a hypernym hierarchy and the root is always the class *entity*. The

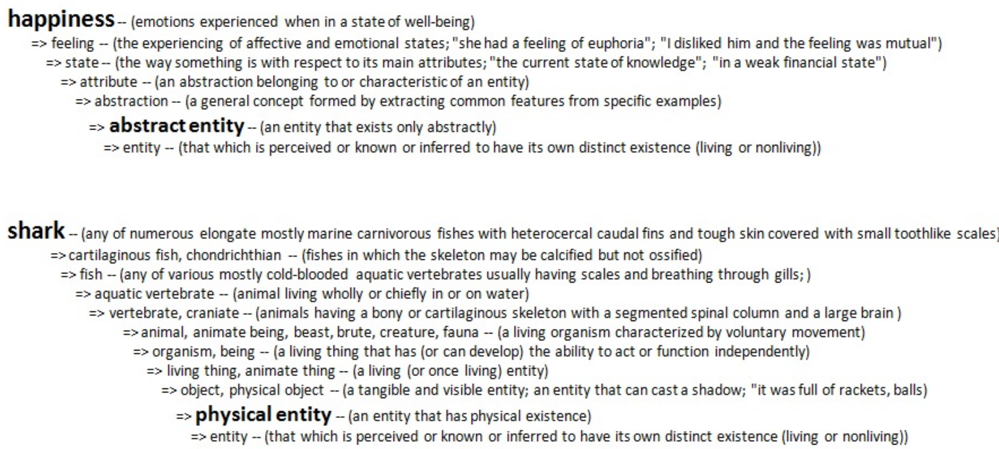


Figure 5.1: WordNet hypernym hierarchy for **happiness** (abstract) and {textbfshark (concrete).

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.

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 verbs to the total words in a poem. The total token count does not include any punctuation or numbers and contains only words. Verb denotes all types of verbs like :

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

5.3.4 Noun density

Noun density is the ratio of total nouns to the total words in a poem. Total token count does not include any punctuations or numbers and contain only words. Noun denotes all types of nouns like:

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

5.3.5 Adjective density

The adjective density is the ratio of total adjectives to the total number of words in a poem. The total token count does not include any punctuation or numbers and contains only words. Adjective denotes all types of adjectives like:

- 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 types of pronouns like:

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

Noun denotes all types of nouns like:

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

For finding the POS tags for all the above metrics, the Spacy library (Honnibal and Johnson, 2015) is employed.

5.3.7 Concordance

Concordance refers to the retrieval of a queried term and all the related terms in the context of a poem. 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 *siblings* or related words to the stem word. For example, *ocean* is a sibling 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[1] is used to get all occurrences of the words in our set. Retrieval is done on an already populated index of all PoFo poems. 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 Highligter is used to highlight the search terms in the poem. The highlighted terms are shown in cyan color.

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 equivalance relations (word associations). Word embeddings are the very first step 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

[1]https://lucene.apache.org/core/

Gigaword corpus. We observed that though good for detecting common speech, we could not use the already available models for building word associations for poetry. The prime reason for this observation was 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 above hypothesis, 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 *river* 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 *river* with better objects (like *lake*, *ocean*, *bank*, etc.). The GloVe model captures associations more related to definition or dictionary-like aspect with objects like *tributary*, *estuary*, *reservoir*, etc.

Table 5.3: RIVER in GloVe model

Word	**Score**
rivers	0.796
creek	0.746
lake	0.743
valley	0.713
danube	0.704
basin	0.695
tributary	0.693
canal	0.689
flows	0.684
estuary	0.677
reservoir	0.674
stream	0.671
crossing	0.668
yangtze	0.666
lakes	0.661
along	0.658
tributaries	0.652
bridge	0.646
watershed	0.645
nile	0.635

Table 5.4: RIVER in PoFo model

Word	**Score**
lake	0.880
road	0.870
ocean	0.844
bridge	0.838
mountain	0.836
valley	0.827
bank	0.824
stream	0.822
ridge	0.816
hill	0.815
horizon	0.801
path	0.801
shore	0.800
desert	0.798
forest	0.796
sea	0.793
meadow	0.792
east	0.781
wood	0.780
west	0.778

5.4 Results

To illustrate Concordance, an example highlighting for a poem by the Lucene Highlighter is given below:

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. In the next chapter, we will explain about our diction web application where we can do comparative analysis of diction features on a poem.

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 PoFo corpus. This application incorporates all the linguistic quantitative features explained in the previous chapters. For semantic features, we have build applications for metaphor detection that will be explained in detail in a later chapter.

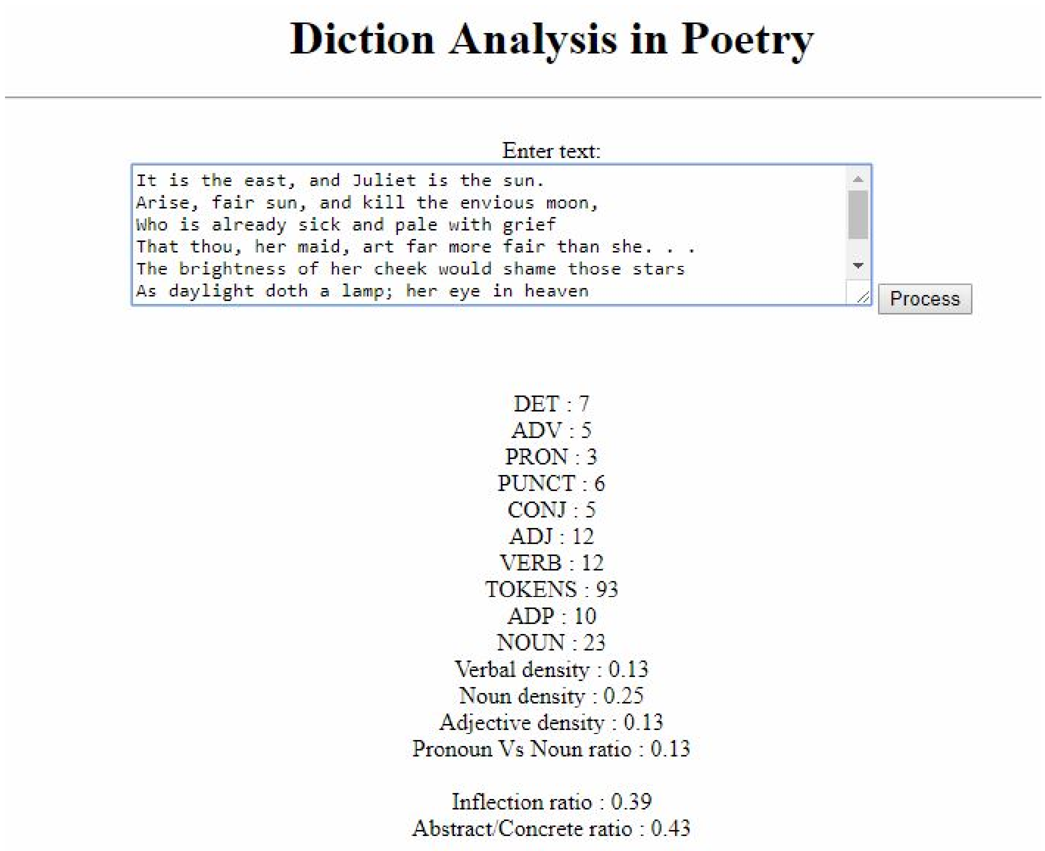


Figure 5.2: Screenshot of the diction analysis web application that quantifies linguistic quantitative features.

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. 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 may expand it to index more poems as well.

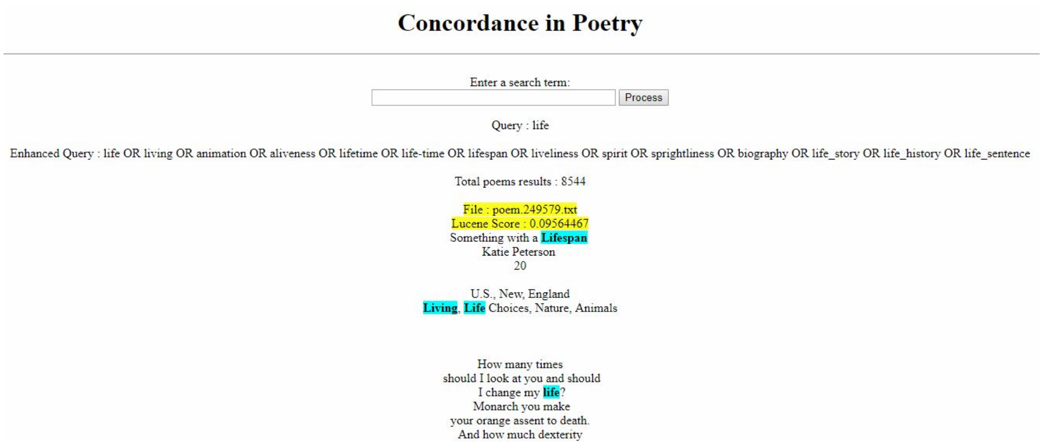


Figure 5.3: Screenshot of the concordance web application that uses semantic query expansion to retrieve poems from PoFo corpus.

5.6 Conclusion and Future Work

We have attempted to quantify all 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 for 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 implemented 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 in 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. 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.

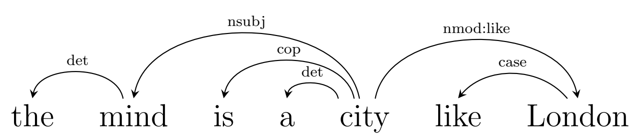
Type I metaphor has a POS tag sequence of Noun-Verb-Noun where the verb is a copula (Neuman et al., 2013). We have extended this to include the tag sequence Noun-Verb-Det-Noun, since we have found that many instances were skipped due to the presence of a determiner. Type II has a tag sequence of Noun-Verb-Noun 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, currently, we are also working on a method independent of POS tag sequences. We employ a dependency parser (de Marneffe et al., 2006) to identify all associations in a sentence. We will 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 will be 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.

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 annotate 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 full poetic context was lacking. Here the first head word is *youth* and the second head is *lords*.

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 (Ralegh, 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.[1]

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*

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*

[1]The data can be found at `http://www.eecs.uottawa.ca/~diana/resources/metaphor/type1_metaphor_annotated.txt`.

- *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.

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*

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 do 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 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.

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 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

	metaphor			*literal*		
Classifier	Precision	Recall	F-score	Precision	Recall	F-score
ZeroR	0.565	1.000	0.722	0.000	0.000	0.000
Random Forest	0.741	**0.822**	0.779	**0.731**	0.627	0.675
JRip	0.635	0.745	0.686	0.573	0.443	0.500
J48	0.71	0.615	0.659	0.574	0.675	0.620
KNN	**0.782**	0.756	0.769	0.697	**0.726**	**0.711**
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
SVM (Puk)	0.759	0.804	**0.781**	0.724	0.670	0.696
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
Adaboost (RF)	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

Table 6.2: Results for classifiers trained on PoFo+TroFi+Shutova data, and tested on the 487 poetry sentences

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 6.3: Results for the class *non-metaphor*

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 are therefore working on a type-independent metaphor identification algorithm to handle such

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

Table 6.4: Results of the direct comparison with related work (Rule+Stat = rule-based and statistical)

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 noted before. In the case of ZeroR, the classifier just keeps all the instances in the metaphor class, because it is the larger class with 56% of the instances.

For the results in Tables 6.1 and 6.3, the SVM classifier (with 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 6.3 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 6.4 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

Table 6.5 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.

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

Table 6.5: A selection of incorrectly predicted PoFo sentences (L = literal, M = metaphorical)

6.6 Deep Learning Classification

The first requirement for deep learning classification is lots of examples / data points (in thousands). Therefore, when we were able to collect more data (around 4870 instances) with metaphor (poetry and non-poetry), we experimented on 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 (Chollet et al., 2015) deep learning framework with a Tensorflow (Abadi et al., 2015) backend and used a local GPU to accelerate the training process. The parameters that we tested on are given in Table 6.6.

The results of our experiments are given in Table 6.7. 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

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 6.6: Range of parameters tested.

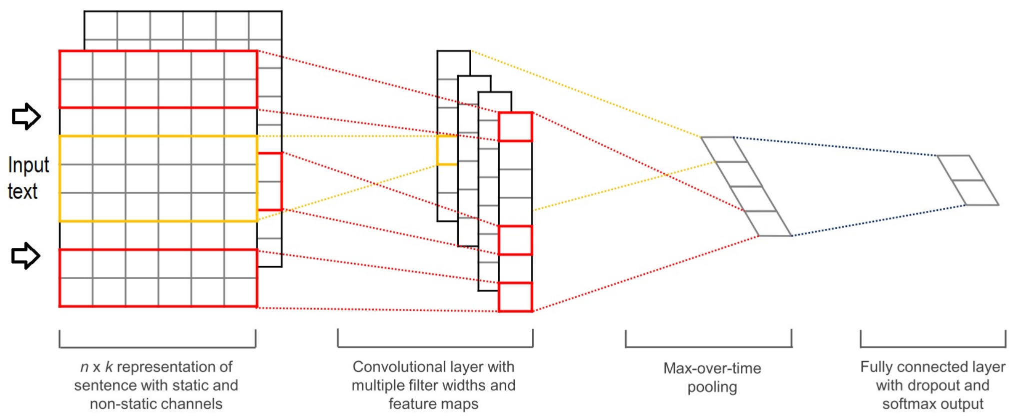


Figure 6.1: Schema diagram of the CNN text classifier.

tested on hundreds of combinations of hyper-parameters, only the top results are being reported here for brevity.

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 a 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

	metaphor			*literal*		
Parameters	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	**0.833**	0.748	0.740	**0.744**

Table 6.7: 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.

training. Consequently, though we got better results for PoFo+TroFi+Shutova 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

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

This web application works for not just poetry, but also for any natural language text. For this application, we have not used POS tag sequence for Type 1 metaphor, instead we use Stanford NLP parser dependencies given below to extract the potential word pairs:

- nsubj

- dobj
- nsubjpass
- acl:relcl

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

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 analysed 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

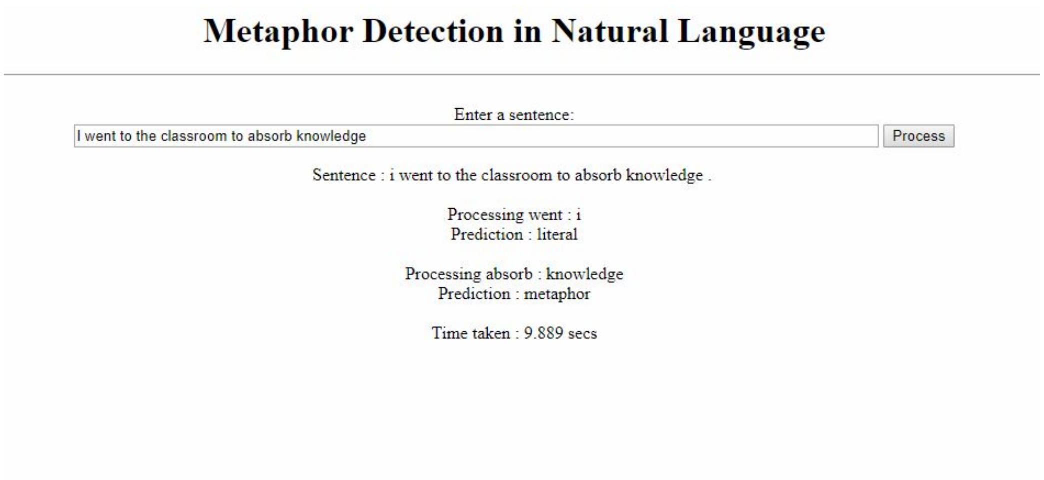


Figure 6.2: Screenshot of the single-line metaphor detection web application.

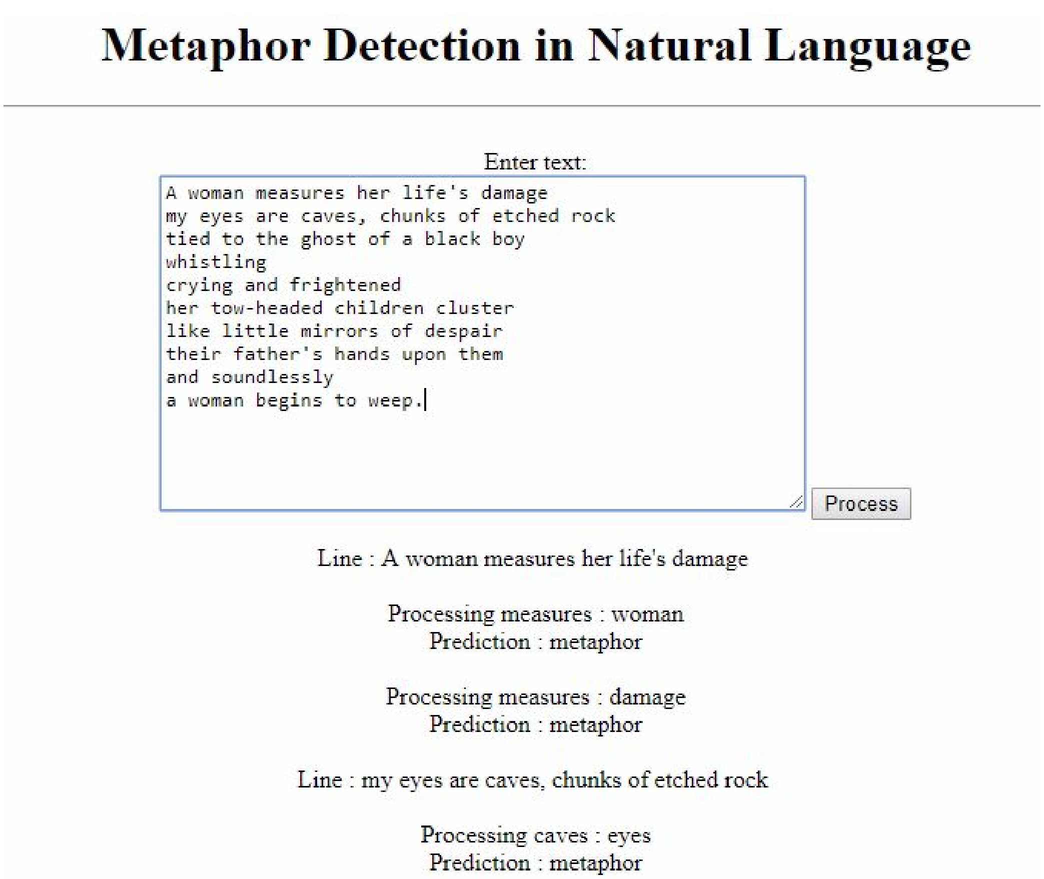


Figure 6.3: Screenshot of the multi-line metaphor detection web application. The full text is not captured in this screenshot.

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 Conclusions and Future Work

To the best of our knowledge, this is the first dissertation 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

CONCLUSION AND FUTURE WORK

7.1 Conclusion

The GraphPoem project addresses the following sides 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 parts.

In conclusion, we have built classifiers for rhyme, diction and metaphor and quantified rhyme and diction for graph visualization. We may 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.

7.2 Future Work

With the advent and penetration of deep learning in the field of NLP, it would be sheer ignorance to say that computational poetry and digital humanities as a whole will not get influenced by this advancement. We have tried to foresee this drastic change and in turn inculcated elements of deep learning into our work. 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, 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 and to *graphicalize* poetry. Hence, we may in the future, refine our poetic word embeddings and may try some other visualizations like t-SNE (Maaten and Hinton, 2008) to visualize poetry in a vector space.

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.

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.

Crabbe, G. (1950). *The village*. University Tutorial Press.

Crane, H. (2006). *Complete Poems and Selected Letters*. Library of America.

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.

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.

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.

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.

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.

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.

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.

Musaoğlu, 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.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. In *Proc. EMNLP*, volume 14, pages 1532–1543.

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.

Ralegh, 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.

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.

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.

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*.