EAPOEM: RULE-BASED LEARNING OF POETIC FORMS

ESZTER FODOR
5873320

Supervisors:
WOUTER BEEK
KARINA VAN DALEN-OSKAM

Bachelor Thesis Artiffical Intelligence
Credits: 18 EC

Faculty of Science
University of Amsterdam
Science Park 904

28 June 2013 – Version 1.0

ABSTRACT

This thesis describes the research that focused on creating a computer program that is able to automatically learn the distinguishing characteristics of various poetic forms, using a training set of poems and some background knowledge. The primary poetic forms that were targeted were sonnets and sestinas since these forms have a rather rigid structure. The data for this project was acquired from websites on the Internet and preprocessed using (amongst other things) rhyme dictionaries, which were created based on pronouncing dictionaries available on-line. The actual learning program was created using Inductive Logic Programming (ILP) on the data, which created hypotheses on which characteristics of a certain poem form are essential for distinguishing it from other forms. The resulting hypotheses proved to be accurate, but not complete. However, the constructed methods during this research show approaches that can prove useful for future researches regarding poetic forms.

Once upon a midnight dreary, while I pondered weak and weary,
Over many a quaint and curious volume of forgotten lore,
While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
'Tis some visitor,' I muttered, 'tapping at my chamber door -
Only this, and nothing more.'

— The Raven, Edgar Allan Poe

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Wouter Beek and Karina van Dalen-Oskam, for their help and encouragement during my research. Also many thanks to my friends Sharon and Melissa for reading through my thesis and helping me to improve it.

CONTENTS

1 INTRODUCTION 1
1.1 Motivation and Research Question 1
1.2 Data and approach 2
1.3 Existing work 2

2 APPROACH 4
2.1 The Used Data 4
2.2 Rhyme annotation 5
2.3 Hypothesis forming 9
2.3.1 Inductive Logic Programming 9
2.3.2 Constructed hypotheses 13
2.3.3 Hypothesis validation/falsification 14
2.4 Software 14

3 CONCLUSION, DISCUSSION AND FUTURE WORK 15
3.1 Conclusion and Discussion 15
3.2 Future work 16

A APPENDIX 17

BIBLIOGRAPHY 19

1

INTRODUCTION

This thesis reports the steps that were taken and the results that were achieved during the research which focuses on the distinction between different poetic forms based on their characteristic properties, using Inductive Logic Programming (ILP).

In this chapter the motivation behind the research will be described, followed by a brief introduction to the data used and an outline of the approach to the problem.

1.1 MOTIVATION AND RESEARCH QUESTION

Poetry forms a large proportion of literature all over the world and it is well preserved. The invention of certain technologies (e.g. computers, database management systems) during the past few decades provided scientists with the opportunity to use the preserved texts for complex researches. Scientists (mainly linguists) have been and still remain interested in the origin and evolution of languages and their components (e.g. grammar, semantics) which have resolved in various subfield within linguistics. These vary from evolutionary linguistics, which investigates the origin and growth of languages, to neurolinguistics, which explores language processing in the brain, to the study of language meaning.

Another branch of linguistic research is computational linguistics, which is concerned with the modeling of natural language from a computational perspective. Many researchers within the field of Artificial Intelligence (AI) have been exploring this area, approaching it from different angles. Some are more concerned with information retrieval or machine translation, while others focus on classification of texts.

It was the challenge of text classification involving poems that formed the basis for launching this project and to conduct research on poetic forms. During the last decade researchers did focus on automatically distinguishing prose from poetry, but only a few concentrated on discriminating between different poetic forms. Those who did (e.g. *D. M. Kaplan, 2006* [7] in his thesis) considered only one language which limited the usability of their software. Although the primary focus of this research was English poems, the approach of rule-based learning should be able to resolve the interlingual obstacles.

The research question that was set to be answered during the project is: *How accurately can a computer program distinguish between different poetic forms?* The subquestions that need to be answered for this are: What are the properties of the chosen forms? What are the properties that distinguish them from one another? How should the data be processed? In which methodological framework will the machine learning task be performed? And how can the program be evaluated? All these questions will be answered in the following chapters.

1.2 DATA AND APPROACH

Due to the fact that there are many different forms of poetry and taking the duration of the project (only two months) into account, a choice had to be made to limit the number of poetic forms that form the basis of the research. The two primary forms that were chosen are sonnets and sestinas. These kind of poems have a fairly rigid structure which makes them great candidates for creating the first version of a rule-based learning software for poems.

Acquiring the data is made simple by the fact that suitable databases are accessible on the Internet. However, it would be difficult for a software to work with plain text without any annotation or metadata. Therefore the acquired data has had to be processed into a usable format. How this was done will be explained in Section 2.1: The Used Data.

After the poems had been adjusted and extended by annotations, they were suitable for use by the actual learning software. This was an inductive logical program, called *ALEPH* [1], which constructed general theories for the specific poetic forms, based on positive and negative examples, along with some background knowledge. How exactly this algorithm works and how the results were acquired will be elaborated in Section 2.3.1: Inductive Logic Programming.

1.3 EXISTING WORK

Rhymes, rhythm and meter are the key components of poems and are therefore the primary aspects that need attention when conducting research regarding poetry. Analyzing rhymes presents a number of problems for researchers due to the fact that natural language is very adaptable and poets are creative when it comes to writing.

D. M. Kaplan [7] proposed in his thesis that there are four basic types of rhyme; *identity rhyme* (same words), *perfect rhyme* (different words, perfect rhyme), *semirhyme* (length of words differ, but they rhyme perfectly) and *slant rhyme* (stress vowels or the phoneme strings following the stressed vowels are identical). He analyzed acquired rhymes in poems by inspecting four line endings at a time, more specifically considering the ending sequence of each line, where *"the sequences start with the consonants before the final stressed vowel phoneme of a line (or simply the final vowel for lines ending with monosyllabic words) and continue to the end of the line of text"* (Kaplan 2006, p. 41). This approach can most likely be considered as universal, since the definition of rhyme is: "Rhyme is the identity in sound, of the accented vowels of words, usually the last one accented, and of all consonantal and vowel sounds following, with a difference in the sound of the consonants immediately preceding the accented vowels." (Maxey Brooke, 1976)[2]

The task of representing rhythm in poems was approached by *Greene, Bodrumlu and Knight* [3] as an unsupervised learning task. By mapping the known syllable-stress pattern to each word in each of the lines within poems with a known meter (such as the sonnets of Shakespeare written in *iambic pentameter*) and assigning probabilities to stress, given the corresponding word, they were able to generate poems, based on the user's input of desired meter. Using the same approach, the researchers also implemented a program that was able to

translate a short Italian poem to English. Though it was not a perfect translation or a great poem, the ideas that Green et al propose as future work seem rather promising.

Since this research focuses on natural text and tries to process it by computer programs, the texts need to be prepared before the programs are able to process it. Rhymes must be annotated to recognize rhyme schemes, reappearing words need to be tagged due to the fact that for some poetic forms reappearing words are important characteristics. A very useful tool for this kind of task is GATE [4]. This framework enables users to process textual data as desired. GATE contains a great number of built-in applications which can be used for tagging of the dataset. It is also possible for the user to build and include new applications within GATE in order to expand the data processing possibilities. The description provided by Cunningam et al [4] helps every new user to understand this framework and makes it easier for them to start a project using GATE.

Following the preprocessing of the dataset the main task of this project can be conducted: i.e. implementing an *inductive logical program (ILP)* which composes a hypothesis about the characteristic attributes of a particular type of poem. The book *Inductive Logic Programming: Techniques and Applications* written by N. Lavrač and S. Džeroski [9] will form a solid basis during the implementation considering the fact that I do not have any experience with ILP's. Another source for guide lines will be the bachelor thesis of Sander Jetten [13] which also contains information about the application of ILP's and how these should be implemented.

2

APPROACH

This chapter describes the data that was used during the research, along with the steps that were taken to acquire it. Furthermore, the necessary approach of processing the data is discussed, followed by a description of the methods used and the results of hypothesis forming using ILP.

2.1 THE USED DATA

Due to the fact that are many different poetic forms, as mentioned in Section 1.2: Data and approach, a choice had to be made in order to limit the amount of considered forms during the implementation of this first version of the software. The two chosen forms are sonnets and sestinas, considering the fact that these sorts of poems have a fairly rigid structure.

As explained by Paul Fussel in his book *Poetic Meter and Poetic Form* (1965, p.128) [12], the two most common sonnet forms are the Shakespearean and the Petrarchan sonnets, both consisting of fourteen lines but with different rhyme schemes. Shakespearean sonnets consist of three *quatrains* each existing of four lines, followed by a *couplet* consisting of two lines: `abab cdcd efef gg`. The letters here represent the line endings in order of their occurrence in the poem where the matching letters correspond to the rhyming lines. Using the same representation, the structure of the Peptrarchan sonnets is: `abba cddc efefef`. Though the structure of quatrains is present in the first eight lines, they are referred to as an *octave* and the last six lines as a *sestet*.

The structure of a sestina is more complex, though very determined. Strand and Boland describe it in *The Making of a Poem* (2000, p. 21) [10] as a poem of thirty-nine lines, six stanzas (verses) of six lines each, followed by an *envoi* (short stanza) of three lines. Furthermore, in the stanzas the same six end-words must occur, but in a changing order that follow a set pattern (see Figure **??**) and the envoi must deploy the six end-words.

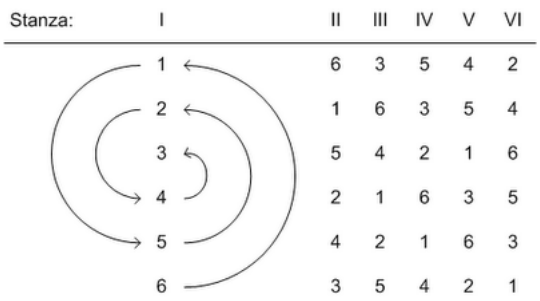

Figure 1: The order of the end-words in a sestina

During the project sonnets were used as the primary poetic forms since they are less complex than sestinas, thus can be more easily used for implementing and testing the first version of the program. The used sonnets (the hundred and fifty-four sonnets of Shakespeare) were acquired from the website `http://poetry.eserver.org/sonnets/` using a web crawler written in Python, which stored each poem in an individual xml file. Unuseful characters such as comma's, brackets etc. were removed from the texts.

The next step in processing the data was annotating the rhyming words in each poem. The steps that were taken will be explained in the next section.

2.2 RHYME ANNOTATION

Natural language is one of the most complicated concepts within science. It is prone for change and is easily adaptable in every day use. Languages change during the course of time; adopting new words, replacing old ones, changing pronunciation of words or even the grammar of the language. Furthermore, every language has a various amount of dialects which makes it even more difficult for linguists to conduct their researches.

Linguistic researches cope with these problems regularly, especially when the research involves corpora which originate from previous centuries. Even more challenging are poems, since poetry has constraints which need to be met. For this reason, poets are resourceful when it comes to writing. Their choice of words and the fact that they occasionally leave out vowels from words, thus creating new words (called *pseudowords* in this thesis), in order to meet the constraints of the form of poem they try to write, leave linguists and other scientist with obstacles that need to be overcome.

Remaining at the topic of poems and poetry, but zooming in on the aspect of rhymes, there are some features that require attention when a research is conducted. The most important aspect of rhymes is phonetics, since by definition two words are rhyming if their pronunciation is similar. However, the complications mentioned above, regarding the flexibility of language, ought to be taken into account. William Shakespeare's first sonnet, *Sonnet I*, is a good example to illustrate these complications:

```
From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.
```

There are two important irregularities that can interfere with the analysis of this sonnet:

1. **Pseudowords:** The words *feed'st* and *mak'st* are the result of the authors way of 'cheating' by eliminating a vowel, thus creating words that are one syllable shorter than they otherwise would have been. The words they are derived from are *feedest* and *makest*, the archaic [1] second-person singular simple present form of *feed* and *make*, respectively. These words do not occur in modern day dictionaries.

2. **Irregular rhyme:** Shakespearean sonnets are known to have a fixed rhyme scheme, namely: *a,b,a,b,c,d,c,d,e,f,e,f,g,g*. This means that the words 'die' and 'memory' are ought to rhyme. However, according to the modern English pronunciation of these words, they do not sound similar at all. Considering the fact that the other ten lines of the poem do follow the rhyme scheme, one can conclude that either one of the words *die* and *memory* were pronounced differently in sixteenth century England.

These kind of deviations can greatly interfere with any kind of research regarding poetry. Many linguists are familiar with the differences between contemporary English and the English used by the well-known poets and writers of the past centuries. Nowadays, however, a great deal of researches are done by employing computers and computer programs, which do not have this kind of knowledge.

A computer program needs a great amount of information to be able to perform calculations on a corpus of natural language. Considering this particular research, which involves poems, annotating rhyme is essential. Thus, the information that the program needs to acquire is "What is the definition of rhyme?". As mentioned above, two words rhyme if their ending is similar. Therefore a way of representing how words sound is required. This task seemed rather simple since there are various phonetic dictionaries available on the Internet. These dictionaries consist of generally used words and their phonetic transcriptions, including lexical stress. However, as the research progressed various complications, involving these dictionaries, occurred which will be elaborated in this paragraph.

One well-known phonetic dictionary is that of the *Carnegie Mellon University*, the CMU pronouncing dictionary [8]. It consists of more than 125.000 English words represented as word-phonetics pairs of the form: **POETRY P OW1 AH0 T R IY0**, where 0 represents no stress, 1 represents primary and (occurring in other words) 2 represents secondary stress. This dictionary is fully open source and was downloaded to serve as basis for creating a *rhyme dictionary*. Another dictionary that was created with the same purpose used the lexicons of the Max Planck Institute for Psycholinguistics (called Celex) [11]. This dictionary is based on the same principle as the CMU version, only with another form of word-phonetics representation: **poetry\'p5-I-trI**, where the apostrophe indicates the lexical stress.

1 No longer in general use, but still found in contemporary texts.

The purpose of a rhyme dictionary would be to enable the computer program to automatically annotate rhymes and by using such a dictionary the time that it takes to accomplish this would be more limited. Based on the phonetic dictionaries described above a few rhyme dictionaries were created for testing. These dictionaries were constructed by following the steps described in the pseudocode below:

Algorithm 1: Creating a rhyme dictionary

```
Data: File containing words and their phonetic transcription
forall line in file do
    create tuple with the word and its phonetic transcription (starting
    from the last stressed vowel until the end of the word)
end
forall tuples do
    search for matching phonetics within the other tuples
    if match found then
        add the found word to the list with rhyming words
    end
end
return list containing a primary word and a a list with all the words that
rhyme with it
```

To provide an example, a line returned by this algorithm and stored in the rhyme dictionary look like:

```
['POETRY', ['GROCHOWSKI', 'SHEPPY', 'GENADY', 'KARWOSKI', 'EMBRY',
'WINNIE', ... ]]
```

Representing rhyming words as a list of lists, with one keyword as first element and the list of words that rhyme with the keyword as the second element, has one big advantage: while annotating rhymes, the program only has to parse through the first element of the lists to find the word that it must annotate (i.e. 'increase'). After finding that word, the program can compare the other end-words of the poem with the elements of the list with rhyming words to be able to conclude whether the two words do or do not rhyme.

However, as mentioned before, there are some difficulties that arise while working with poems and rhymes, for which a rhyme dictionary, created as described by the pseudocode, can not provide a solution:

Historical aspect:
Poems that were not written in the near past contain words that are either *a*) unknown words in the modern day language (e.g. 'niggarding'), *b*) distorted (e.g. 'feed'st'), or *c*) pronounced differently in the time they were written (e.g. 'die' or 'memory').

Phonetic representation:
Different phonetic representations lead to different rhyme dictionaries. One of the best examples to illustrate the difference between the dictionaries created by using the CMU and Celex pronouncing dictionaries is the list with words that rhyme with 'wet'. Where the Celex based dictionary

only comes up with a list of two words: `['WET', ['WHET', 'WET']]`, the CMU based version constructs a list containing 359 words. Therefore, the CMU based dictionary is more useful, however the created file has a size of multiple gigabytes, which makes it time consuming to use for annotation.

Though automatic annotation of poems proved to be time consuming, it did result in fairly accurate annotations of the sonnets. That is, if the words exist in the dictionary and are pronounced similar according to the phonetic transcriptions. Taking Shakespeare's first sonnet as an example again, the automatic annotation returned the following annotated poem (the colors correspond to the rhyming words, also indicated by the tags):

```
From fairest creatures we desire <a>increase</a>,
That thereby beauty's rose might never die,
But as the riper should by time <a>decease</a>,
His tender heir might bear his memory:
But thou contracted to thine own bright <c>eyes</c>,
Feed'st thy light's flame with self-substantial <d>fuel</d>,
Making a famine where abundance <c>lies</c>,
Thy self thy foe, to thy sweet self too <d>cruel</d>:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton <g>be</g>,
To eat the world's due, by the grave and <g>thee</g>.
```

This sonnet still demonstrates the difficulties that need to be overcome before a computer program can work with poems without manual help from people during the preprocessing phase of a research:

1. The first two words which are not annotated ('die' and 'memory') have been discussed previously; they do not get annotated because of the fact that their pronunciation is not similar.

2. The words 'ornament' and 'content' also ought to rhyme, however they are not annotated by the program. The reason for this is that according to the CMU pronunciation dictionary the syllable *'ent'* in both words *almost* have the same phonetics, though not precisely. This syllable corresponds in 'ornament' with `AHO N T` and in 'content' with `EHO N T`. These rhymes are called *slant rhymes*.

3. The words 'spring' and 'niggarding' also lack annotation, due to the fact that 'niggarding' is not a used word in modern English, therefore it is not present in the CMU dictionary.

There are some possible solutions for these inconsistencies, however these still would not guarantee a perfect rhyme annotator. To solve the problem of words that were pronounced differently in the past, a database with more possible pronunciations of those words could provide an answer. To be able to annotate slant rhymes, rules that ensure that those words are recognized as rhyming can be

added to the program. The problem of the words that are not present in existing pronunciation dictionaries has two possible solutions; 1) the dictionaries could be manually extended or 2) the annotator program could search for known words which are similar to the unknown words, thus assigning a similar pronunciation for the ending of the unknown word.

Though Steve Hanov has created an on-line rhyming dictionary, Rhyme Brain [15], which finds rhyming words even for non-existing words and thus could be seen as solution number 3 above, the resulting words for 'niggarding' still do not include 'spring'. Therefore, a combination of the mentioned solution would be required to solve all the problems with rhyme annotation.

As one can see, there are various obstacles that need to be overcome in order to be able to automatically process and annotate poems. After concluding that it would be more time consuming to automatically annotate rhymes than assumed and considering the fact that rhyme annotations were necessary for the progress of the research, a choice was made to manually annotate the rhymes in Shakespeare's sonnets.

2.3 HYPOTHESIS FORMING

The next step in the research, after preprocessing the data, was to construct hypotheses about what the defining properties of a poetic form is, in this case that of a sonnet. This section describes the steps that were taken to accomplish this and it analyzes the constructed hypotheses.

2.3.1 Inductive Logic Programming

Inductive Logic Programming (ILP) is a subfield of Machine Learning (ML), which in contradiction to other forms of ML, such as Support Vector Machines (SVM), learns the properties that define the trainings data rather than learning the class the data belongs to, given the *known* properties. More elaborately, the purpose of an ILP is to define an unknown relation (i.e. the target predicate) from the examples, given the known relations from the background knowledge (Lavrač and Džeroski, 1994)[9]. As the name implies, ILP-systems use logic as formalism, more specifically first-order logic, therefore these systems are mostly written in Prolog (I. Bratko, 2001)[5]. The training examples (positive as well as negative) are represented as ground facts of the target predicate and most often the background knowledge is restricted to have the same form. Hence, given the correct positive and negative examples, as well as the appropriate background knowledge, an ILP system can derive the relation *parent(X,Y)*, given the relations *mother(X,Y)* and *father(X,Y)* which would result in the predicates:

```
parent(X, Y):-
        father(X, Y).
parent(X, Y):-
        mother(X, Y).
```

These predicates are constructed by following the definition of Inductive concept learning given bij Lavrač and Džeroski [9]:

Inductive concept learning: Given a set of examples $\mathcal{E}$ of positive and negative examples of a concept $\mathcal{C}$, find a hypothesis $\mathcal{H}$, expressed in a given concept description language $\mathcal{L}$, such that:

- Every positive example $e \in \mathcal{E}^+$ is covered by $\mathcal{H}$ and
- No negative example $e \in \mathcal{E}^-$ is covered by $\mathcal{H}$.

2.3.1.1 ILP System

There are various ILP systems, which have been implemented and are available for general use. The best known systems are FOIL (Quinlan & Cameron-Jones, 1995)[6], PROGOL (Muggleton, 1995)[14] and Aleph (Ashwin Srinivasan, 2007)[1]. Due to the fact that within Aleph the background knowledge as well as the hypotheses are expressed as Prolog clauses (i.e. subset of first-order predicate logic), which results in a broad hypothesis space this system was chosen as learning environment.

The Aleph algorithm requires three files as input: two files containing the trainings examples and one the background knowledge about the domain. The contents of these files are explained in the following sections.

The algorithm follows the steps below to construct the theory that satisfies the constraints of Inductive concept learning (Ashwin Srinivasan, 2007)[1]:

1. **Select example:** Select a positive example to be generalized, if all have been already selected previously by the algorithm, stop. Otherwise proceed to next step.

2. **Build most-specific-clause:** Construct a "bottom clause" that entails the chosen example. Bottom clauses generally contain many literals. This steps follows *inverse entailment* described by Stephen Muggleton (1995)[14]

3. **Search:** Based on a subset of literals with the "best" score within the bottom clause find a more general clause. The score of the literals is defined by the positive and negative examples they entail.

4. **Remove redundant:** The best clause is added to the theory, the used positive examples entailed by the theory are removed and the algorithm returns to step one.

2.3.1.2 Background Knowledge Base

The background knowledge contains information about the domain, the hypothesis space and language restrictions. These informations are given as Prolog clauses which determine the predicates and rules that the algorithm is required to learn. The most basic restrictions are *mode, type* and *determination* declarations.

Mode declarations determine the predicates that can appear in the hypothesis proposed by the system. The general form is:

```
:- mode(RecallNumber, PredicateMode).
```

Where `RecallNumber` restrains the number of times a predicate is allowed to be called. It can either be *a*) a positive integer, specifying the number of calls, or *b*) '*' declaring the the number of calls to be undetermined. `PredicateMode` determines the variable types that the predicate expects as input arguments, adopting the general form:

```
predicate(ModeType, ModeType, ...).
```

Using the example of the relation *parent(X, Y)* introduced above, the mode declaration would be of the form:

```
:- modeh(*,parent(+person,-person)).
:- modeb(1,father(+person,-person)).
:- modeb(1,mother(+person,-person)).
```

Here the first line declares the head of the predicate (hence the 'h' after 'mode'), which is the relation that is required to be learned. The other two lines declare the body of the predicate, each stating that a person can only have one father and one mother. The instantiation pattern '+' defines that the argument must be fully instantiated (i.e. "input" variable), where '-' defines that the argument must unbound (i.e. "output" variable). There is a third type of instantiation pattern, namely '#', which determines that the argument must be a constant.

The third form of restriction statements is *determination*, which declares the predicates that can be used for constructing the hypothesis, adopting the general form:

```
:- determination(TargetP/Arity, BodyP/Arity).
```

Which for the example of parents would be:

```
:- determination(parent/2, father/2).
:- determination(parent/2, mother/2).
```

During the research the choice was made to initially focus only on implementing the learning algorithm for Shakespearean sonnets, since these are short poems with a rigid structure. As explained in Section 2.1 these sonnets are constructed of three quatrains and one couplet, where the quatrains consist of four lines, each with the rhyme scheme `abab` and the couplet of two rhyming lines. These features determined the primary predicates that the ILP system was required to learn. The background knowledge base (BgKB 1) therefore contained declarations for `couplet/2, quatrain/4` and `sonnet/14`, where the input arguments consisted of the tags for the rhyme annotations, extended with the line numbers (e.g. `d-6` assigning line number 6 to tag `d`). [2]

2 For complete background knowledge see Listing A.1 in the appendix.

However, after concluding that the constructed hypotheses, based on this background knowledge, were not satisfactory, another knowledge base (BgKB 2) have been constructed, which contained less restrictions for the predicate 'sonnet'. More elaboratly; it only difinied the predicate `sonnet/1`, where input argument of consisted of a list containing the fourteen rhyme tags. [3]

The obtained results based on both BgKB's will be explained in Section 2.3.2: Constructed hypotheses.

2.3.1.3 Examples

As described above, the Aleph system needs two individual example files, one for the positive and an other for the negative examples.

Positive exampes:
Evidently, these examples are the positive cases for the poetic form that needs to be learned, in this case sonnet. The examples are constructed in the form that is defined by the predicate mode in the background knowledge base.

Negative examples:
Consequently, this file contains examples that are *not* a sonnet. The form in which these examples are represented must be compatible with that of the positive examples.

Due to the fact the ILP system was required to learn multiple predicates to satisfy the restrictions of the BgKB 1, the example files needed to contain cases for all of these predicates, namely `couplet/2, quatrain/4` and `sonnet/14`. The examples adopted the form:

```
couplet(g-13,g-14). % Positive example
couplet(a-1,b-4). % Negative example

quatrain(a-1,b-2,a-3,b-4). % Positive example
quatrain(a-1,b-2,c-5,d-6). % Negative example

% Positive example of sonnet
sonnet(a-1,b-2,a-3,b-4,
c-5,d-6,c-7,d-8,
e-9,f-10,e-11,f-12,
g-13,g-14).

% Negative example of sonnet
sonnet(a-1,b-2,c-5,d-6,
e-9,f-10,g-13,a-3,
b-4,c-7,d-8,e-11,
f-12,g-14).
```

Whereas in the case of BgKB 2 only the predicate `sonnet/1` had to be learned, thus the algorithm only required examples of this form.

3 For complete background knowledge see Listing A.2. in the appendix.

2.3.2 Constructed hypotheses

As previously mentioned, the hypotheses that were created were correct, however not conclusive. Based on BgKB 1 Aleph flawlessly learned the predicates `couplet/2` and `quatrain/4`, however the hypothesis that it constructed for `sonnet/14` only contained one predicate in the body, namely `quatrain/4`, which was also not completely correct.
The following example demonstrates the flaws in the theory:

```
[theory]

[Rule 1][Pos cover = 3 Neg cover = 0]
couplet(A,B):-
   rhyme(A,B).

[Rule 2][Pos cover = 4 Neg cover = 0]
quatrain(A,B,C,D):-
   rhyme(B,D).

[Rule 3][Pos cover = 5 Neg cover = 0]
sonnet(A,B,C,D,E,F,G,H,I,J,K,L,M,N):-
   quatrain(N,B,N,D).
```

As claimed above, the clauses couplet and quatrain are learned correctly. However a note has to be made concerning the quatrain clause; the reader could expect that the theory would consist of two body clauses for both `rhyme(A, C)` and `rhyme(B, D)`, but due to the fact that Aleph learns the most general rule, the constructed clause for quatrain is sufficient.

The critical issue with this hypothesis lies within the body clause of `sonnet/14`, which consists of one, not entirely correct clause. Within a quatrain the second and fourth variables are supposed to be rhyming, precisely as represented by B and D in `quatrain(N, B, N, D)`. However, the variable N can not be a part of the same quatrain as B and D, since it represents the ending of the last line of the sonnet, where B represents the ending of the second line and D that of the fourth line.

To improve these results tests were run with adjusted instantiation patterns and recall numbers within the background knowledge base, however these adjustments did not constructed the desired outcome. Therefore, BgKB 2 was created in order to test whether another form of background knowledge and less information about sonnets would ensure better results. Though, the hypotheses that were created based on BgKB 2 are more accurate, these are still not conclusive:

```
[theory]

[Rule 1][Pos cover = 5 Neg cover = 0]
sonnet([A,B,C,D,E,F,G,H,I,J,K,L,M,N]):-
   rhyme(B,D), rhyme(F,H).
```

An important remark that is to be made, is the fact that the bottom clause that is constructed as the first step while running the algorithm, does uses all rhyme combinations, which forms a good basis for future research. This will also be elaborated in Section 3.2: Future work.

2.3.3 Hypothesis validation/falsification

The created hypotheses were evaluated manually by comparing the constructed clauses with the actual examples. For this poetic form this is not a time consuming process. However, this task will be required to be performed automatically during more exhaustive research.

2.4 SOFTWARE

There were multiple softwares used during the research. There were some experiments conducted with GATE during the first period of the research, however these were not used afterwards. Nevertheless, it can be a useful software to use for future work, this will be elaborated in Section 3.2. Python was used for creating a web crawler in order to acquire the data from the Internet and store it in individual XML files. Furthermore, it was used for constructing and testing the rhyme dictionaries from the downloaded pronunciation dictionaries. The reason that python was chosen for this task is that it is a fairly easy to use programming language and it also supports functional programming styles. Lastly, as mentioned previously, Prolog was used during the Inductive Logic Programming phase.

3

CONCLUSION, DISCUSSION AND FUTURE WORK

This chapter reviews the conclusion that can be drawn based on the so far completed work. Section 3.1 discusses the results of the program and Section 3.2 describes the work that can be conducted in the future as a follow up on this research.

3.1 CONCLUSION AND DISCUSSION

There are separate conclusion that can be drawn based on the research so far:

Dictionaries:
The CMU pronunciation dictionary proved to form a proper basis for creating a rhyme dictionary. Its representation of the pronunciation of words and the amount of words it contains is sufficient enough to be able to create an accurate rhyme dictionary. The constructed dictionary based on CMU performs really well as a source for a rhyme annotation system, though exclusively on words that are present in the pronunciation dictionary. Therefore, there is some room for improvement regarding the rhyme dictionary, which will be elaborated in Section 3.2.

ILP:
Though the hypotheses created by the ILP system Aleph, based on the information it was provided, were correct, they were not conclusive enough to be usable yet. There are multiple possibilities for improving the results which will also be elaborated in the next section.

The research as a whole so far:
The results regarding the task of automatically annotating the rhymes of a poem are satisfactory, since a percentage of 96 of correctly annotated rhymes is considerably high. The constructed rhyme dictionary therefore will form a solid basis for the continuation of the research.

The Inductive Logic Programming part of the research resulted in less adequate outcome, since the created hypotheses after many tests were not conclusive enough to be usable. However, the results were not incorrect, only incomplete. Therefore, the constructed background knowledge and example files can be used as foundation for either constructing new files or adjusting the input information.

3.2 FUTURE WORK

Many suggestions for the continuation of the research had been addressed in the previous sections. This section will elaborate these suggestions, dividing them into three categories:

Preprocessing of the data:

As mentioned in Section 2.4, some tests were conducted using the software GATE during the first stage of the research, however it was not used afterwards. Nevertheless, it can be a very useful tool for annotation of the sonnets. By adding Part-Of-Speech (POS) tags to the poems and considering them as properties of it, the information that forms the input of the ILP system can be expanded, thus mayhap improving the results of the system.

Another and even more important addition for the preprocessing is the tagging of the meter of the poems. Rhythm is a one of the essential elements of poetry and it is distinguishing for the different forms.

Improving the rhyme dictionary:

There are three main issues that need to be addressed in order to improve the performance of the dictionary:

1. **Expansion with 'old' words:** Words that are not used in modern English can not be found in the rhyme dictionary as it is now. This obstacle can be solved by either *a*) adding known words that are frequently used in classical poetry manually, or *b*) creating a learning system which uses already (completely and correctly) annotated poems to recognize words that are not present in the dictionary and adds them to the correct list of rhyming words.
2. **Recognition of *slant rhymes*:** Some rhymes, which are less obvious than others, which is very common in poetry, are not recognized by the dictionary. This can be resolved by a more detailed processing of the pronunciation dictionary.
3. **Performance:** The rhyme annotation proved to be a very time consuming process, which is the result of either the incorrect way of how the dictionary is constructed or the inappropriate implementation of the annotation algorithm. Resolving this issue important for the continuation of the research.

Improving the results of the ILP system:

The hypotheses that are created as far do not prove to be very sufficient. This part of the research needs the most improvement before it can be declared finished. Exhaustive testing will be required on different poetic forms as well as on diverse input information.

APPENDIX

```
        :- modeh(*, couplet(+end, +end)).
        :- modeb(1, rhyme(+end,+end)).

        :- modeh(*, quatrain(+end, +end, +end, +end)).
        :- modeb(2, rhyme(-end,-end)).

        :- modeh(*, sonnet(+end,+end,+end,+end,+end,+end,+
           end,+end,+end,+end,+end,+end,+end)).
        :- modeb(*, quatrain(-end,-end, -end, -end)).
        :- modeb(*, couplet(-end, -end)).

        :- determination(couplet/2, rhyme/2).
        :- determination(couplet/2, not_the_same_line/2).
        :- determination(quatrain/4, rhyme/2).
        :- determination(sonnet/14, quatrain/4).
        :- determination(sonnet/14, couplet/2).

        not_the_same_line(X, Y):-
                X \== Y.

%======Tags of line endings=====%
end(a-1).
end(b-2).
end(a-3).
end(b-4).
end(c-5).
end(d-6).
end(c-7).
end(d-8).
end(e-9).
end(f-10).
end(e-11).
end(f-12).
end(g-13).
end(g-14).

%======Rhyming tags=====%
rhyme(a-1,a-3).
rhyme(b-2,b-4).
rhyme(c-5,c-7).
rhyme(d-6,d-8).
rhyme(e-9,e-11).
rhyme(f-10,f-12).
rhyme(g-13,g-14).
```

Listing A.1: Primary background knowledge base (BgKB 1)

```
        :- modeh(*, sonnet([+end,+end,+end,+end, +end,+end
            ,+end,+end,+end,+end,+end,+end,+end])).
        :- modeb(*, rhyme(-end, -end)).

        :- determination(sonnet/1, rhyme/2).

%=========Rhyming tags========%
rhyme(a-1,a-3).
rhyme(b-2,b-4).
rhyme(c-5,c-7).
rhyme(d-6,d-8).
rhyme(e-9,e-11).
rhyme(f-10,f-12).
rhyme(g-13,g-14).

rhyme(a-3,a-1).
rhyme(b-4,b-2).
rhyme(c-7,c-5).
rhyme(d-6,d-7).
rhyme(e-11,e-9).
rhyme(f-12,f-10).
rhyme(g-14,g-13).
```

Listing A.2: Less restricted background knowledge base (BgKB 2)

BIBLIOGRAPHY

[1] Ashwin Srinivasan. The Aleph Manual. `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html`, 2007.

[2] Brooke, Maxey. Rhyme and Rine. *Word Ways*, 9(1):17, 1976.

[3] K. K. E. Greene, T. Bodrumlu. Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation. In *EMNLP '10 Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2012.

[4] K. B. . V. T. H. Cunningham, D. Maynard. GATE: an Architecture for Development of Robust HLT Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[5] Ivan Bratko. *Prolog Programming for Artifficial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[6] J. R. Quinlan and R. M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing*, 13:287 – 312, 1995.

[7] D. M. Kaplan. Computational Analysis and Visualized Comparison of Style in American Poetry. Master's thesis, Princeton University, 2006.

[8] Kevin Lenzo. The CMU Pronouncing Dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.

[9] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellise Horwood, 1994.

[10] Mark Strand and Eavan Boland. *The Making of a Poem*. W.W. Norton & Company, Inc., 2000.

[11] Max Planck Institute for Psycholinguistics. WebCelex. `http://celex.mpi.nl`.

[12] Paul Fussell, Jr. *Poetic Meter and Poetic Form*. Random House, Inc., 1965.

[13] Sander K. Jetten. Automated Modelling of Qualitative Data. Bachelor thesis, University of Amsterdam, 2012.

[14] Stephen Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 1995.

[15] Steve Hanov. RhymeBrain. `http://rhymebrain.com/`.