DOI: 10.14750/ME.2015.009

University of Miskolc

Doctoral Thesis

Efficiency Analysis of Inflection Rule Generation

Author:
Zsolt Tóth

Supervisor:
Dr. habil. László Kovács

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Jozsef Hatvany Doctoral School
Faculty of Mechanical Engineering and Informatics
University of Miskolc

December 19, 2014

Declaration of Authorship

I, Zsolt Tóth, declare that this thesis titled, 'Efficiency Analysis of Inflection Rule Generation' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

"Deadlines refine the mind. They remove variables like exotic materials and processes that take too long. The closer the deadline, the more likely you'll start thinking waaay outside the box."

Adam Savage

Recommendation

Zsolt Tóth has started his studies at the University of Miskolc in 2006. He has got a Bachelor Degree in Information Technology and he was specialized to Web Technologies in 2010. Then he got his Master Degree in Computer Science in Application Development specification in 2011. After his master studies he successfully applied to the József Hatvany Doctoral School. He finished his doctoral studied and started work as Assistant Lecturer at the Department of the Information Science at the University of Miskolc. He teaches programming software testing and database management. During his work he contributed in the writing of 5 journal papers, a book chapter and 14 conference papers.

December 19, 2014

Dr. habil. László Kovács

UNIVERSITY OF MISKOLC

Abstract

Faculty of Mechanical Engineering and Informatics
Department of Information Technology

Doctor of Philosophy

Efficiency Analysis of Inflection Rule Generation

by Zsolt Tóth

This paper focuses on the learning of inflection rules and grammar induction. Inflection algorithms usually require the numerical representation of the words. The existing mappings do not consider the phonetic features of the letters. Thus a method has been developed to crate a phonetic features based alphabet. The yielded alphabet has been shown superior to the ASCII code table and the traditional alphabet based encodings.

The induction of inflection rules is considered as a classification problem. To create an efficient inflecting algorithm, the problem domain has been analyzed. My analysis showed that there are numerous linear non separable cluster pairs in the training set of 54.000 (stem, inflected form) pairs. The high number of the linear non–separable clusters could yield decrease the precision of the standard classification methods. To reduce the number of the linear non–separable cluster pairs, I have proposed a novel inflection algorithm. The algorithm uses classifier to determine the inflection rule of the regular and untrained words and it uses Associative memory to store the irregular words. An inflection rule is considered irregular if its frequency is low. The proposed method is compared with standard inflection algorithms.

In the topic of grammar induction the efficiency of the existing grammar induction methods were measured. To compare these algorithm I have designed a grammar induction and text mining framework. However there are several machine learning frameworks but none of them focus on the modeling of formal grammars and the grammar induction. My framework provides the necessary classes to model any kind of formal grammars in Chomsky hierarchy and it defines interfaces for grammar induction and parsing algorithms. Some context–free grammar induction methods was implemented in the framework. Experimental results show that the framework is able to model formal grammars.

Acknowledgements

This dissertation contains the results of many years of work that would not have been realized without the support of others.

First and foremost I would like to thank to my supervisor Dr. habil. László KOVÁCS for his continuous support and help during my work. I would like to thank to the members of the Department of the Information Technology for their help, remarks and advises.

I am thankful to my parents who made me possible to learn and supported me from the beginnings.

I also would like to thank to Dóra Pászinczki for her support and patience.

Contents

List of Figures

List of Tables

Abbreviations

SVO	**S**ubject **V**erb **O**object
CFG	**C**ontext–**F**ree **G**rammar
PCFG	**P**robabilistic **C**ontext–**F**ree **G**rammar
GA	**G**enetic **A**lgorithm
NL	**N**atural **L**anguage
NLP	**N**atural **L**anguage **P**rocessing
META	**M**iskolc **E**nvironment for `T`ext `A`nalysis
NP	**N**on–deterministic **P**olynomial–time
XOR	Exclusive `OR`
TM	**T**ext **M**ining
DM	**D**ata **M**ining
OASIS	**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards
UIMA	**U**nstructured **I**nformation **M**anagement **A**rchitecture
GUI	**G**raphical **U**ser **I**nterface
`abcd`	computer program related word
URL	**U**niform **R**esource **L**ocator
XML	e**X**tensible **M**arkup **L**anguage
JAXB	**J**ava **A**PI for **X**ML **B**inding
API	**A**pplication **P**rogramming **I**nterface

Symbols

ω	sentence
ω_i	ith word of a sentence
$\mathcal{L}$	formal language
$\mathcal{G}$	formal grammar
$\mathcal{L}_\mathcal{G}$	language generated by $\mathcal{G}$
$\mathcal{T}$	set of terminal symbols
$\mathcal{N}$	set of non–terminal symbols
$\mathcal{P}$	set of production rules
$\mathcal{S}$	set of sentence symbols
$a, b, \ldots$	terminal symbols
$A, B, \ldots$	non–terminal symbols
$\alpha, \beta, \ldots$	symbol sequence
$\mid \alpha \mid$	length of symbol sequence
$A \rightarrow \alpha$	replacement rule
$S \rightarrow^*_\mathcal{G} \omega$	deduction in n steps
l	letter
$l_{i,k}$	kth feature of letter l_i
$d(l_i, l_j)$	distance of l_i and l_j
$\mathbf{A}, \mathbf{B}, \ldots$	matrices
$\mathbf{a}, \mathbf{b}, \ldots$	vectors
X, Y	random variables
$E(X)$	expected value of E
$\overline{x}$	average of X
$\sigma(X)$	deviation of X
$cov(X, Y)$	covariance of X and Y

Σ	covariance matrix
f_i	feature (random variable)
λ	eigenvalue
Λ	vector of eigenvalues
$\mathbf{v}$	eigenvector
$\mathbf{V}$	vector of eigenvectors (matrix)
$[0, 1, \ldots]$	row vector
$[0, 1, \ldots]^T$	column vector
$\mathbf{I}$	identity matrix
$\mathcal{X}$	cluster of objects
$P(\mathbf{W}, t)$	hyperplane
$\mathcal{X}_\rangle \parallel \mathcal{X}_|$	$\mathcal{X}_\rangle$ and $\mathcal{X}_|$ are linear separable
$\mathcal{A}, \mathcal{B}, \ldots$	sets
$O()$	order of function
π_p	partition p
$T(\omega)$	parse table of ω
U	samples
U_+	positive samples
U_-	negative samples
$\mathcal{G}_\omega$	sub–grammar related to ω

To my beloved family . . .

Chapter 1

Introduction

Natural Language Processing is an active research area of computer science and knowledge engineering. There are also text books in the topic of text mining [Dom07] and natural language processing [Kor08] too. The applications [Cho03] and researches are usually focus on a specific natural language. The English is the target language of most of these applications and researches. On the other hand efforts have been made in other languages such as Czech [PŽ10] Portuguese [BCM+08] The "Szószablya" [RG06, HKN+04] may be the most famous Hungarian project in this field. The Hungarian Academy of Science has developed a plagiarism checker [Pat06a, Pat06b]. Moreover it is an actively researched in the Doctoral School too. For example there are works on the knowledge extraction from text or speech [Bar13], on the modeling of the extracted knowledge [Var11] and on the text generation [Bed12].

Learning of inflection rules have an important role in many natural language processing tasks. The presented results connect to the previous works of our Department in various ways. My work focuses on the learning of inflection rules which belongs to the natural language processing topic. The proper inflecting is crucial for text analysis and generation methods. Inflection usually modifies slightly the meaning of the base word and an inappropriate inflected word could make the sentence hard–to–understand or change its meaning. The methods were tested on Hungarian text and an effective method has been developed. A framework has been designed and implemented to test the efficiency of methods.

1.1 Research Goals

The main goal of my researches was to analyze the different inflection rule generation methods and to develop an efficient algorithm for Hungarian language. The word representation has a key role to develop an efficient algorithms.The words are usually represented in a vector space because it allows to perform well–known methods and operations. There are many different approaches to convert the words into real vectors. These conversions require to define a real value for each letter in the alphabet. In the current solutions have some drawbacks. For example the phonetic features of the letters are not considered or the distance of the words is usually constant. Hence one of my goals were to create an mapping which is based on the phonetic features and the distance of two letter represent their similarities.

Inflection can be considered as a string transformation which transforms the base word into its inflected form. The transformation can be determined for each word pairs and it can be described by a transformation string. The induction of the inflection rules can be converted into a classification task where the inputs are the base words and the transformation strings are the categories. My goal is to develop an efficient methods to learn the inflection rules as string transformations. A training set of about 54.000 samples are used to measure the performance of the developed algorithm.

Formal Grammars is a mathematical formalism to model the grammatical structures of both natural or artificial languages. Formal Grammars can be defined manually or automatically. The manual generation requires an expert of the given language and it is a time–consuming and costly job. Context–Free Grammars widely used because they are powerful enough to model many features of natural languages. On the other hand there are arguments against the context–freeness of natural languages [Shi87]. Analysis and comparison of Context–Free Grammar induction methods is an important goal of my researches. To measure the performance of Context–Free Grammar induction methods a novel framework is designed.

1.2 Dissertation Guide

The rest of this paper is organized as follows.

Chapter 2 gives a brief overview of the theoretical background of the used formalism and terminology in the paper. A few different classifications of the formal languages is presented and the properties of Hungarian are emphasized. Then it gives a short overview of the text mining and natural language processing tasks. Inflection is presented here as inverse of the stemming which is a well–studied text mining task. Next the mathematical bases of the formal languages are presented. The Chomsky hierarchy is reviewed which is the most–well–known classification of formal grammar and it is based on the form of the production rules. Context–Free Grammars are detailed due to their importance in grammar modeling. Finally some open questions are presented.

Chapter 3 presents a novel method to create a mapping of letters to numerical values based on their phonetic features. This chapter gives a short introduction of the phonetic features of the letters. Then it details how the letters can be converted into real vectors based on their phonetic features. This representation allows the usage of principal component analysis which is also reviewed. Next a MatLab application is presented which were developed to create a phonetic alphabets. A resulted phonetic alphabet is evaluated and it is compared with ASCII and traditional alphabet based encodings. The experimental results showed the phonetic alphabet based encoding superior. When the induction of inflection rules is considered as a classification task, then the neural networks show poor precision. The high number of non linear separable clusters can be a reason of the poor performance. A linear programming based approach is chosen to test linear separability of the clusters and its computational cost is estimated. The test were performed on a training set of about 54.000 samples with alphabetical and phonetic encoding. These results also showed the phonetic encoding superior.

Chapter 4 introduces a new classification based inflection algorithm enhanced with associative memory. There is a brief overview of the algorithms of computational morphology. These works mainly focused on morphological analysis and stemming, moreover only a few works are focused on Hungarian. Then the inflection as a classification problem is detailed. Next the standard classifiers were tested to solve the problem. The measurement are focused on the precision, the learning cost and the size of the classification structure. Then the proposed inflection method is detailed. The novel algorithm is tested with different size of the associative memory. Finally the experimental results are evaluated and the novel methods is compared with the traditional algorithms.

Chapter 5 presents the META which is a novel grammar induction and text mining framework. The related data and text mining frameworks and works are reviewed. The overview shows that there is no common environment to model formal grammar. Purpose of the development is to provide a framework to model formal grammars and define interfaces for induction and parsing methods. The architecture and the main components of the framework are detailed and the formal grammar module is emphasized. The design and the important decisions are also presented. Next there is a brief overview of some well–known context–free grammar induction methods. These methods are implemented in the framework and their time cost is measured. Our experimental results are also summarized. So the ability of META framework to model formal grammars has been confirmed.

Chapter 6 sums up the new scientific results of my work and presents the future tasks.

Chapter 2

Theoretical Background

This chapter gives a brief overview of the theoretical background of Natural Langauge Processing and the mathematical formalism of Formal Grammars. Natural languages can be classified based on their aspects such as their history, word order or morphology. The certain properties of the Hungarian language are emphasized during the overview. Natural languages often use inflection to slightly modify the meaning of word. The inflection rules are abstract and they are formalized for humans. The automatic generation of inflection rules is an open question in computer science and the most of current works are focused on other languages [Pin91, Mol01, GAH04].

2.1 Categorization of Natural Languages

Natural languages can be classified by various aspects such as their history, word order or morphology. In historical linguistics the languages are organized into language families. There is an ancestor descendant relationship within the families, so a language families are often represented as a tree. The root of the tree is called proto–language such as Indo-European, Uralic, Caucasian, American, Austroasiatic, and Sino-Tibetan languages. A proto–langauge is the latest common ancestor of a language family. The family tree has multiple levels, the nodes are subcategories and the leaves are the languages. These families contain other subgroups. For example the Indo-European languages can be classified into Anatolian, Tocharian, Germanic, Italic, Celtic, Armenian, Balto-Slavic, Hellenic, Indo-Iranian and Albanian subgroups [GI90]. These subgroups also can be divided into subcategories. Most of the spoken languages in Europe belong to the Indo-European family. For example English and German are Germanic, Spanish, France and Italian are Italic, and Slovakian and Polish are Balto-Slavic languages. However Hungarian is a Uralic language [Hel03].

In linguistic typology the languages are classified by their structural features such as word order. From the point of view of typology subject, object and verb are distinguished and the languages are classified by their order. There are nine typological classes plus a category for free word order languages. The vast majority of the languages belong to Subject-Object-Verb or Subject-Verb-Object category [Mey10].

In morphology the languages are classified by how the words are formed from morphemes. Isolating languages and synthetic languages are the main groups in morphology. In isolating languages the morpheme per word ratio is low. The syntax of the words depends on their position and auxiliary words are used to express complex concepts. Chinese and English are examples for isolating languages. In case of synthetic languages the morpheme per language ratio is high. Words are formed by affixing the stem morpheme. Synthetic languages are often subdivided into polysynthetic, fusional, and agglutinative languages. In polysynthetic languages the word are long and their meaning could be a whole sentence in other languages. Some Native American languages are polysynthetic. Fusional languages have some common aspects with analytic languages, but their ancestors were synthetic languages. Many Indo-European languages belong to the group of fusional languages. Int agglutinative languages, such as Japanese, Hungarian or Esperanto, the words are formed by combination of the stem with phonetically unchangeable affixes. The main difference between agglutinative and fusional languages is that there are more affixes in agglutinative languages. Moreover the stem and the affixes can be separated easily in agglutinative languages.

2.1.1 Characteristics of Hungarian Language

Hungarian is an agglutinative language and belongs to the Uralic language family. The word order is not strict and it is mostly used to emphasize the content. Inflection is used to define the grammatical role of the words. An inflected form of a word is called a case. There are 17-28 different [Mor03]. A chain of inflection has been evolved in Hungarian because of two reasons. Firstly the affixes are put in the end of the word in most of the cases. Secondly an inflected word can be the base word of another inflection. For instance in the sentence “Peter has refused our calls” the “our calls” is the object and it is plural. This sentence in Hungarian is “Péter visszautasította a hívásainkat” where the objects is „hívásainkat”. The affixes can be separated as “(hívás)(aink)(at)” where „hívás” is the stem, „aink” means „our” that the word is in plural and „at” denotes it is an object.

2.1.2 Comparison of European Languages

Table 2.1 shows some of the above detailed aspects of some European languages. It shows also that Hungarian differs from the major European languages. Most of these languages belong to the West–Germanic or Italic language families, but Hungarian is an Uralic languages. These languages usually have the same word order but it is not given for Hungarian. Finally Hungarian is an Agglutinative language while the others are mostly fusional.

Table 2.1: Characteristics of Some European Languages

Language	Family	Word Order	Morphology
English	West–Germanic	SVO	Isolating
French	Italic	SVO	Fusional
German	West–Germanic	Flexible	Fusional
Hungarian	Uralic	Free	Agglutinative
Italian	Italic	SVO	Fusional
Russian	Balto–Slavic	Free	Synthetic
Spanish	Italic	SVO	Fusional

2.2 Text Mining

Text mining is an umbrella term of analytical methods to extract information from text or unstructured documents. It has a strong relationship with Natural Language Processing which is an actively investigated field of computer science. However its theoretical backgrounds were laid in the middle of the last century [Cho56, Cho59, Lev66, Nav01], but the computational capacity was not enough to deal with natural language processing tasks and solve real world problems. The invention of the World Wide Web and the widely available Internet connection resulted the rapid proliferation of digital documents. Today about 80 per cent of the information is stored in text based documents [Dom07]. At the dawn of this century the natural language processing became a popular topic again [DFG11, Por01, GAH04]. This paper focuses only on the induction of inflection rules and formal grammars.

2.2.1 Natural Language Processing Projects for Hungarian

However there are serious efforts taken in the field of the text mining and natural language processing in general, but there are only a few works which focus on Hungarian language. Moreover the most of these works focuses on Natural Language Processing tasks such as stemming [HKN+04, RG06] or learning morphemes [Dud06]. For example the Szószablya is Hungarian stemming algorithm or the KOPI [Pat06a, Pat06b] plagiarism search engine of the Hungarian Academy of Science. On the other hand my researches were focused on not the stemming, but the inflection of Hungarian words. Inflection can be considered as a string transformation which is used in many natural languages and it also plays an important role in Hungarian too.

In the text books there are abstract inflection rules which are marked out for human understanding. To generate complex text in natural languages, these abstract inflection rules have to be converted into a more specific form which can be used by computer programs. This conversion is a difficult and costly task. There are a few researches on the generation of inflection rules, but these works are focused on other languages [Con98, RDM65].

2.3 Formal Languages

My researches were also focused on the induction of Formal Grammars which are often used to model the structures natural languages. The theory of formal grammars was laid in the mid 1950s by Noam Chomsky [Cho56] [Cho59]. The most popular classification of formal grammars is called Chomksian–hierarchy and it distinguishes the following four categories: *Regular*, *Context–Free*, *Context–Sensitive* and *Recursive Enumerable Grammars*. Although many grammar processing methods were developed in the 1950s and 1960s, they were not used in real world applications due to their computational complexity. Since the edge of the century many grammar induction algorithm have been developed, implemented and tested.

In the literature, there are only a few comparison of Context–Free Grammar induction methods. A common environment was required to compare the different induction algorithms. However there are some data mining frameworks [HDW94, HFH$^+$09, WFT$^+$99, FHH$^+$05], tools [Min14], only few standards [OAS14, Apa14] and educational softwares [Gru06] focus on text mining and grammar processing.

2.3.1 Formal Grammars

Formal languages and formal grammars are often used to model grammatical structures of the natural languages. A formal language is a set of valid sentences $\mathcal{L} = \{\omega\}$ where $\mathcal{L}$ denotes the language and ω stands for a sentence. Formal languages $\mathcal{L}$ can be given as a set of sentences, but it is very costly. Thus a formal language is often given by a formal grammar $\mathcal{G}$. A $\mathcal{G}$ grammar generates a $\mathcal{L}$ language if each sentence of the language can be derived by the grammar and it is denoted by $\mathcal{L}_{\mathcal{G}}$. The $\mathcal{G}$ grammar is defined as a quadruplet $\langle\mathcal{T},\mathcal{N},\mathcal{P},\mathcal{S}\rangle$ where

$\mathcal{T}$ is a set of the terminal symbols $(a, b, c \cdots \in \mathcal{T})$,

$\mathcal{N}$ is a set of the non terminal symbols $(A, B, C \cdots \in \mathcal{N})$,

$\mathcal{P}$ is a set of production rules $\mathcal{P} \subseteq \alpha A \beta \times \gamma$ where and $\alpha, \beta, \gamma, \dots \in \{\mathcal{T} \cup \mathcal{N}\}^*$

$\mathcal{S}$ is a set of sentence symbols $\mathcal{S} \subseteq \mathcal{N}$.

The $\mathcal{T}$, $\mathcal{N}$, $\mathcal{S}$ cannot be empty set and there is at least one sentence symbol in every grammar which is a non–terminal symbol. There are definition where $\mathcal{S}$ is not a set but a single non–terminal symbol. Moreover $\mathcal{S}$ denotes a set of sentence symbols $(\mathcal{S} = \{S_1, S_2, \dots S_n\})$. The replacement is given by the production rules. In an arbitrary formal grammar any sequence of symbols which contains at least one non–terminal symbol can be replaced by an other sequence of symbols. But a sequence of terminal symbols cannot be replaced. The replacements are denoted by $A \Rightarrow a$. A complex formal grammar contains many production rules and the deduction of a sentence requires multiple replacements. Deduction is a process which derive ω sentence from a sentence symbol based on the production rules. The chain of rules can be written as $S \Rightarrow \alpha A \beta \Rightarrow \cdots \Rightarrow \omega$, but this notation is quite long and contains too many details. So $S \Rightarrow^*_{\mathcal{G}} \omega$ denotes that sentence ω can be generated from sentence symbol S in finite steps based on grammar $\mathcal{G}$.

2.3.1.1 Chomsky–Hierarchy

Formal grammar are often classified on the basis of their production rules. A well-known classification of formal grammar was made by Noam Chomsky [Cho56, Cho59] which is called Chomsky hierarchy and it is showed in Figure 2.1 The form of the rules are summarized in Table 2.2. The most specific class are called regular grammars where a non terminal symbol always can be replaced by terminal symbol or a terminal and a nonterminal symbols. Regular grammar can be divided into two subclasses: left regular grammar and right regular grammars. Left regular grammars contains rules in form of $A \rightarrow a$ and $A \rightarrow Ba$ while the rules of right regular grammars are in form of $A \rightarrow a$ and $A \rightarrow aB$. From the point of view of practice in a left regular grammar the word is written from left to right, in a right regular grammar it is written from right to left.

In context–free grammars a non terminal symbol can be replaced by an arbitrary sequence of terminal and non terminal symbols. Context–free grammars are often used to model natural languages due to two reasons. First there are many efficient algorithms to process context–free grammars. Secondly they can capture many aspects of natural languages.

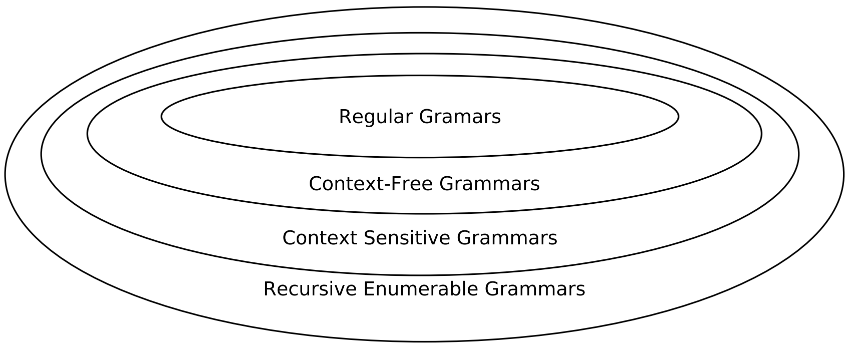


Figure 2.1: Chomsky Hierarchy

In context sensitive grammars the replacement of a non terminal symbol depends on its context. If the length of the context is zero ($| \alpha |=| \beta |= 0$) then the grammar is context–free. So the context–free grammars are special context sensitive grammars. The most general class is obviously the recursively enumerable grammars because it allows arbitrary replacements.

Table 2.2: The Chomksy hierarchy

Class	Production rules
Recursively Enumerable	$\alpha \rightarrow \beta$
Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Context Free	$A \rightarrow \gamma$
Regular	$A \rightarrow a$ and $A \rightarrow Ba$ or $A \rightarrow aB$

In the literature, there are many algorithms to process or induce regular grammars. In the field of natural language processing the context–free and context sensitive grammars are widely used to model natural languages. However both of these models are used, there is no accordance in the question where the natural languages are belong to [Shi87]. While English can be well modeled by Context–free grammars, some other languages such as Spanish, Russian or Hungarian have context–sensitive aspects. For example in Spanish the nouns have gender and the adjective has to match to its noun. In Russian the words nouns have gender and the words are inflected and the meaning of the word depends on its case. Although there are no genders in Hungarian, the 17 different cases of a word make the language complex.

The different grammar classes can be implemented by automatons. Regular grammars can be modeled by finite state automatons. The context–free grammars can be implemented by non–deterministic push–down automaton. The context–sensitive grammars can be modeled by linear–bounded non–deterministic Turing machine. The importance of recursively enumerable languages is mostly theoretical. This is the most general grammar where an arbitrary string of symbols can be replaced by another symbol sequence. Thus it can be implemented by the most general automaton, the Turing machine.

2.3.1.2 Context–Free Grammars

The context–free grammars are enough efficient tools, moreover they can well represent some significant feature of the natural and artificial languages. Thats is why context–free grammars are often used in natural language processing tasks.

Parsing of Context–Free Grammars The parsing algorithms decide a sentence ω is in the language generated by a grammar $\mathcal{L}_\mathcal{G}$ or not. The parsing is a quite important task and it is used in many induction algorithms. Thus the efficiency of parsing algorithm is essential. Mathematically the parsing algorithms can be defined as the following function.

$$parse : \{\omega\} \times \{\mathcal{G}\} \rightarrow \{true, false\} \tag{2.1}$$

$$parse(\omega, \mathcal{G}) = \left\{ \begin{array}{ll} true & \exists S \in \mathcal{S}_\mathcal{G}, S \rightarrow^*_\mathcal{G} \omega \\ false & otherwise \end{array} \right\} \tag{2.2}$$

Induction of context–free grammars Induction of Context–Free Grammars requires a training set which contains positive and negative samples. While it is easy to find positive sentences, the generation of negative samples is a hard task. Probabilistic Context–Free Grammars can be generated from only positive sentences. There are top–down [NI00, NM02] and bottom–up [SK99, Sak05, UJ07, OU09] methods for this task. The generation of Context–Free Grammar from samples is a $\mathcal{NP}$–hard problem. However the performance of these algorithms is often measured one by one on different training sets, but there is no common framework to implement, test and compare these algorithms.

2.4 Conclusions

In this chapter the theoretical background of Linguistic,Text Mining, Natural Language Processing and Formal Grammars were briefly summarized. The natural languages were classified based on historical, topological and morphological approaches. The properties of the Hungarian language were emphasized because my researches were focused on it.

Then the current state of text mining and its subtask the stemming were reviewed. There is a lack on the efforts in learning the Hungarian inflection rules. Finally there was an overview of the mathematical formalism of the formal grammars which are widely used to model natural languages.

Chapter 3

Complexity Analysis of the Inflection Induction Problem

The complexity of a problem can be caused by many factors such as computational complexity, time cost, or size and so on. These factors are usually analyzed separately. In algorithm theory, the computation complexity is the most important measurement. There are classes of complexity and algorithms are distinguished belongs to one of these classes and it is usually given with big O notation [Knu76]. For example bubble sort is a polynomial algorithm because its time cost is $O(n^2)$ and binary search is a logarithmic algorithm with $O(log(n))$ cost.

The determination of the computational complexity requires theoretical analysis which could be a hard task for advanced algorithms and methods with many parameters such as heuristic searches, artificial neural networks, genetic algorithms and so on. In this case the time cost of the algorithms is measured directly with fixed parameters. Based on the experimental results the time cost can be estimated as a function of the different parameters.

There are problems which are hard to solve even with efficient algorithms because of the size of the data. The huge amount of data usually is a great challenge in data processing and storage tasks. For example multiple levels of triggers are used [TG05] for data processing in the Large Hadron Collider of CERN to filter the events. By the filtering these triggers reduces the size of the collected data about the measurements, thus it is possible to process and store the data. Huge amount of data is also a challenge in data storage and there are some solutions such as the Bigtable of Google [CDG+08].

The complexity of the inflection rule induction depends on many aspects of natural languages. There can be thousands of words in the vocabulary of a natural language.

Moreover the vocabulary changes over time, old–fashioned words extinct and new words are coined or the meaning of the words changes. For example keyboard was a part of the piano about 60 years ago, but these days keyboard is usually associated with computers. In language text books, the words belong to different inflection classes [MLCL04]. Linear separability of the clusters is vital to create efficient classifiers so the complexity of learning inflection rules is measured as the linear separability of the different inflection classes. Linear separability depends on the representation of the words. Vector space representation of words were used during my research because it is widely used in Natural Language Processing tasks. To measure the linear separability a simplex method based testing algorithm was implemented. The tests were performed on a traditional alphabet based encoding and a novel phonetic features based alphabet.

3.1 Linear Separability

Complex objects in any problem domain are represented by a set of features. These observations are usually converted into numerical vectors. Thus each observation can be described as a feature vector $\mathbf{v} \in R^d$ in a d dimensional space where d is the number of features. This representation allows to handle the observations in a formal way. There are many analysis on the vector space model to provide an efficient feature representation.

The goal of classification in vector space model is to assign a category value for the vectors based on their feature values. The binary classification problem is related to the case when two category values are defined in the domain. The clusters of elements which are belonging to class $\mathcal{X}_1$ and $\mathcal{X}_2$, can be disjoint or overlapped in general case. This paper focuses on the case when the two clusters are linear separable. In this case there are many efficient algorithm for the classification problem.

In the case of linear separable clusters, there is a hyperplane $P(\mathbf{w}, t) = \{\mathbf{x} \mid \mathbf{x}\mathbf{w}^T + t = 0\}$ which separates them. We use the notation $\mathcal{X}_1 \parallel \mathcal{X}_2(P)$ or simply $\mathcal{X}_1 \parallel \mathcal{X}_2$ if the elements of $\mathcal{X}_1$ are on one side of hyperplane P and the elements of $\mathcal{X}_2$ are on the other side.

Linear separability is important feature for many machine learning tasks such as classification with Artificial Neural Networks [Yeg09, Zha00] or Support Vector Machines [SV99]. For example Perceptron network is able to learn basic logical functions like AND, OR, NOT. But it cannot learn XOR function because the XOR problem is not linear separable. Multilayer Perceptron is improved by hidden layers thus it can learn the XOR problem, but the determination of number and the size of hidden layers is not an easy task.

3.1.1 Testing Linear Separability with Linear Programming

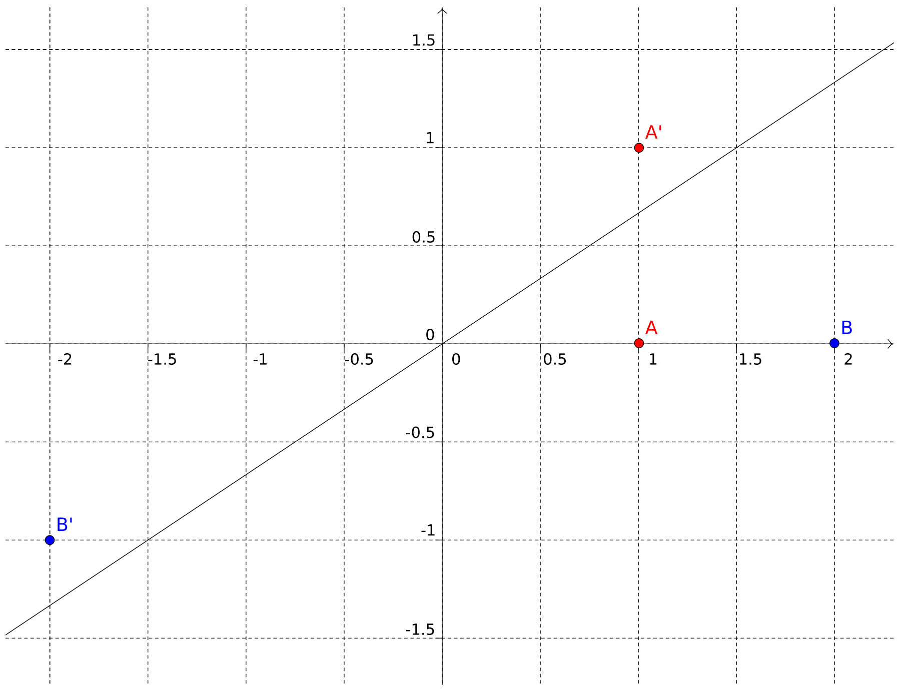
Due to the importance of linear separability, there are many different methods to test it. These methods can be categorized [Eli06] based on the applied mathematical technique. Although these algorithms provide exact solution, they require many computation. Testing based on convex hulls is quite intuitive and easy to understand [Sha75]. If the convex hulls of the sets have intersection then the sets cannot be separated linearly. The drawback of this method is the significant computational cost of the convex hull determination process.

The methods based on linear programming are easy to understand and there are many efficient solvers .Testing based on linear separability transforms the d dimensional points into $d+1$ dimensional points. The new dimension value is initially 1 for each point. Next, the elements of $\mathcal{X}_2$ cluster are mirrored to the Origo. Clusters $\mathcal{X}_1$ and $\mathcal{X}_2$ are linear separable if and only if each of the transformed points are on the same side of the hyperplane which crosses the Origo. Formally $\mathcal{X}_1 \parallel \mathcal{X}_2$ if $\exists P(\mathbf{w}, t)$ where $Origo \in P(\mathbf{w}, t)$ and $p < P(\mathbf{w}, t)$ or $p > P(\mathbf{w}, t)$ where $\forall p \in \{\mathcal{X}_1 \cup \mathcal{X}_2\}$. This transformation defines an inequality system.

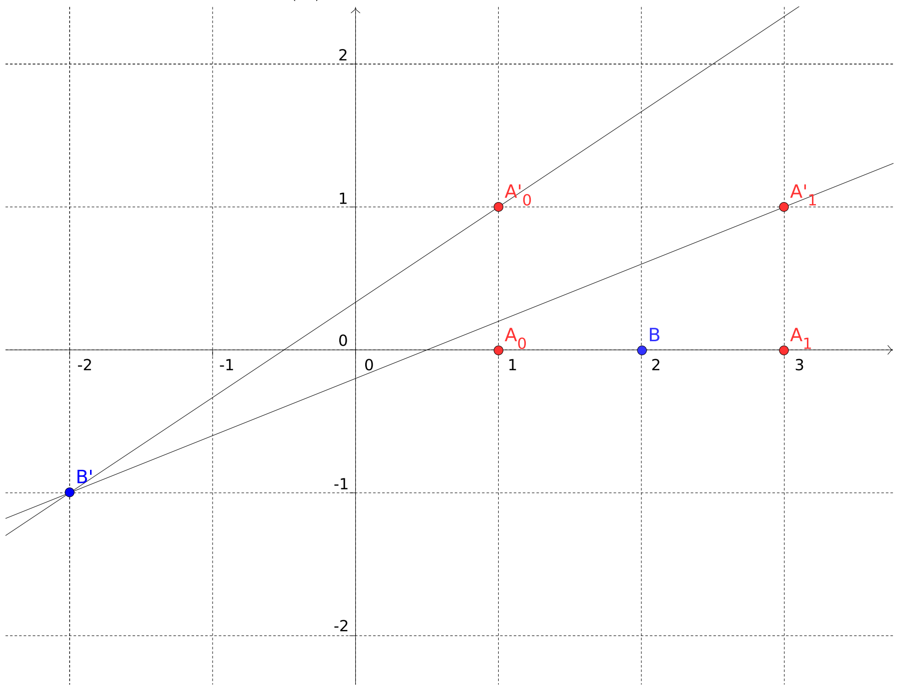
Figure 3.1a presents a separable case, there are two clusters where $\mathcal{A} = \{(1)\}$ and $\mathcal{B} = \{(2)\}$. The transformed clusters are $\mathcal{A}' = \{(1,1)\}$ and $\mathcal{B}' = \{(-2,-1)\}$. The transformed points lay on the same side of hyperplane $P(2/3, 0)$, so these two sets are linear separable. Figure 3.1b shows a non linear separable case. Cluster $\mathcal{A} = \{(1),(3)\}$ and cluster $\mathcal{B} = \{(2)\}$ is transformed into $\mathcal{A}' = \{(1,1),(3,1)\}$ and $\mathcal{B}' = \{(-2,-1)\}$. There is no line which crosses the Origo and both of the points are its same side.

Simplex method [NM65] is one of the best known and most widely used algorithm to solve linear programming tasks and linear separability test. It is based on the pivot operation and many software packages contains linear programming solvers such as MS Excel, MatLab or SciLab. It can be used to solve both maximum or minimum optimization tasks but the problem has to be in standard form. To test linear separability, the standard normal form is used which is shown in Equation 3.1.

$$\begin{aligned} \mathbf{yb} &\to min \\ \mathbf{yA} &\geq \mathbf{c} \\ \mathbf{y} &\geq 0 \end{aligned} \tag{3.1}$$



(A) Linear Separable Points



(B) Non Linear Separable Points

FIGURE 3.1: Examples for linear separability

where $\mathbf{y}$ is a n dimensional real vector of free variables, $\mathbf{yb}$ is the object function and $\mathbf{A} \in R^{m \times n}$ matrix which describes the coefficients in constraints, where m is the number of constraints.

The constrains are inequalities and they are given by the transformation of feature vectors. Let X_1 denote the $R^{s_1 \times d}$ matrix which represents $\mathcal{X}_1$ where d is the number of features and $s_1 =| \mathcal{X}_1 |$ and $X_2 \in R^{s_2 \times d}$ where $s_2 =| \mathcal{X}_2 |$. Using matrix formalism the transformation yields X_1' and X_2' matrices as following $X_1' = [X_1, 1]$ and $X_2' = -[X_2, 1]$. Matrix $\mathbf{A}$ is the concatenation of X_1 and X_2 i.e. $\mathbf{A} = [X_1'; X_2']$ so $\mathbf{A} \in R^{(s_1+s_2)\times d}$.

Testing of linear separability can be considered as a minimum problem. There can be many separator plane between $\mathcal{X}_1$ and $\mathcal{X}_2$. The presented linear program determines only one of these planes. The plane $P(\mathbf{w}, t)$ is chosen by the object function which can be formalized as $\sum_{i=1}^{d} c_i w_i + t \to min$ where c_i is the ith coefficient and w_i is the ith value of the normal vector $\mathbf{w}$ of the plane. In the current experiments, these coefficients were chosen to 1, thus the object function was $w_1 + w_2 + \cdots + t \to min$. Equation 3.2 shows the linear programming task which is used to test the linear separably of cluster $\mathcal{X}_1$ and $\mathcal{X}_2$.

$$\begin{gathered} w_1 + w_2 + \cdots + t \to min \\ [\mathbf{w}, t]^T \begin{bmatrix} X' \\ Y' \end{bmatrix} > 0 \end{gathered} \tag{3.2}$$

To use simplex methods to solve the given linear programming task, it has to be converted into standard form. By the following substitutions $w_i = w_{i\_aux\_1} - w_{i\_aux\_2}$ and $t = t_1 - t_2$ the inequalities are transformed into grater equal relationship, $[\mathbf{w}, t]_{trans}$ stands for the transformed $[\mathbf{w}, t]$. Moreover new constrains are defined, because the auxiliary variables have to be non negative. Let $A_{trans} \in R^{(s_1+s_2)\times(2*(d+1))}$ denote the transformed $\mathbf{A}$ matrix. Equation 3.3 shows $\mathbf{A}_{trans}$ with auxiliary variables and equation 3.4 shows the same matrix with values of $\mathbf{A}$ matrix.

$$\begin{bmatrix} a_{1,1\_aux\_1} & -a_{1,1\_aux\_2} & a_{1,2\_aux\_1} & -a_{1,2\_aux\_2} & \cdots & a_{1,d+1\_aux\_1} & -a_{1,d+1\_aux\_2} \\ a_{2,1\_aux\_1} & -a_{2,1\_aux\_2} & a_{2,2\_aux\_1} & -a_{2,2\_aux\_2} & \cdots & a_{2,d+1\_aux\_1} & -a_{2,d+1\_aux\_2} \\ a_{3,1\_aux\_1} & -a_{3,1\_aux\_2} & a_{3,2\_aux\_1} & -a_{3,2\_aux\_2} & \cdots & a_{3,d+1\_aux\_1} & -a_{3,d+1\_aux\_2} \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \end{bmatrix} \tag{3.3}$$

$$\begin{bmatrix} \mathbf{A}(1,1) & -\mathbf{A}(1,1) & \mathbf{A}(1,1) & -\mathbf{A}(1,1) & \ldots & \mathbf{A}(1,d+1) & -\mathbf{A}(1,d+1) \\ \mathbf{A}(2,1) & -\mathbf{A}(2,1) & \mathbf{A}(2,1) & -\mathbf{A}(2,1) & \ldots & \mathbf{A}(2,d+1) & -\mathbf{A}(2,d+1) \\ \mathbf{A}(3,1) & -\mathbf{A}(3,1) & \mathbf{A}(3,1) & -\mathbf{A}(3,1) & \ldots & \mathbf{A}(3,d+1) & -\mathbf{A}(3,d+1) \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \end{bmatrix} \tag{3.4}$$

$$\begin{aligned} w_{1\_aux\_1} - w_{1\_aux\_2} + w_{2\_aux\_1} - w_{2\_aux\_2} + \cdots + t_1 - t_2 \rightarrow min \\ [\mathbf{w}, t]^T_{trans} \left[\mathbf{A}_{trans}\right] \geq 1 \\ w_{i\_aux\_1}, w_{i\_aux\_2} \geq 0 \\ t_1, t_2 \geq 0 \end{aligned} \tag{3.5}$$

This transformation allows to solve the optimization task so that to test the linear separability. Equation 3.5 shows the transformed minimization problem.

3.1.2 Multiple Sets

In real applications usually there are more than two classes. The above detailed method can be used to decide if every pair of classes are linear separable or not. If each class can be separated linearly from the others then these classes are piecewise linear separable. In this case the sets can be separated by conical hulls [BM94]. Thus if the classes are piecewise linear separable then there are conical hulls which separate the points of class from others. This separation makes possible to create efficient classification of the points. To sum up, the separability of multiple sets can be decided by the above detailed method.

3.1.3 Cost Analysis of the Simplex Method based Linear Separability Testing

The above reviewed linear separability testing method, based on Simplex method [NM65], has been implemented in Java in the META Grammar Induction and Text Mining framework [5, 6, 9]. The Apache Commons Mathematical Library was used to solve the optimization task because it is a well tested, reliable open source and free library. The testing method is implemented into the `meta.core.linsep` package because it is not related directly to any grammar induction task. The result of testing is a logical value which is true if $X \| Y$ and false otherwise. It also provides a way to check piecewise linear separability. Experimental results shows that the simplex method has $O(dp^2)$ complexity where d is the number of the dimensions and p stands for the number of the points.

3.1.3.1 Simplex Method

The selection of the pivot element plays a very important role in Simplex algorithm. There are different selection rules but none of them is a silver bullet. The time cost of the simplex method based testing algorithm was measured as a function of the number of dimensions and the number of points. There were two additional case distinguished, whether the sets are linear separable or not. It was necessary because the algorithm throws an exception when it find the problem unfeasible. Due to this behavior of used library there can be big differences in the time cost if the sets are linear separable. The linear separability was tested only between cluster pairs.

The implementation was tested with two different cases. In the first case the members of classes were generated randomly. Thus there were only a few linearly separable classes. Moreover if two classes are not linear separable then the simplex method is not feasible so the algorithm throws and exception so it stops. In the second case, linear separable classes were generated to measure the maximum time cost of the algorithm. Figure 3.2 shows the average measured time costs as a surface where the x axis stands for the number of points, axis y denotes the number of dimensions and axis z is the measured time cost.

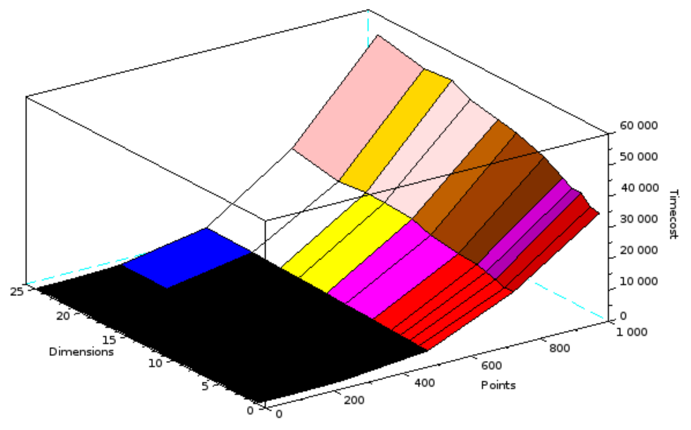


FIGURE 3.2: Time cost of Simplex method based linear separability testing

3.1.3.2 Random Case

The experimental results show that the time cost of the simplex method is less if the sets are non linear separable. The method was tested with different dimension number from 1 up to 25. For each dimension count,the linear separability between two sets, whose size was the same (5 to 1000), was tested . Figure 3.3 shows how the time cost depends on the number of dimensions. The time cost grows with the number of dimensions linearly thus it is not significant. The lines shows how many point were in a set.

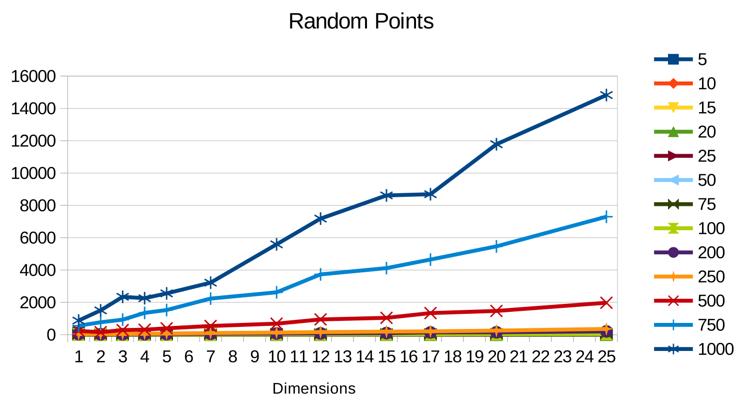


FIGURE 3.3: Time cost depends on dimension number with random points

Figure 3.4 shows how time cost depends on the size of sets. It shows clearly that the time cost does not grow linearly with the size of sets. Based on these measurements a polynomial connection between the time cost and the number of points can be assumed. Thus the number of points in the sets has a higher impact on the time cost of the testing algorithm.

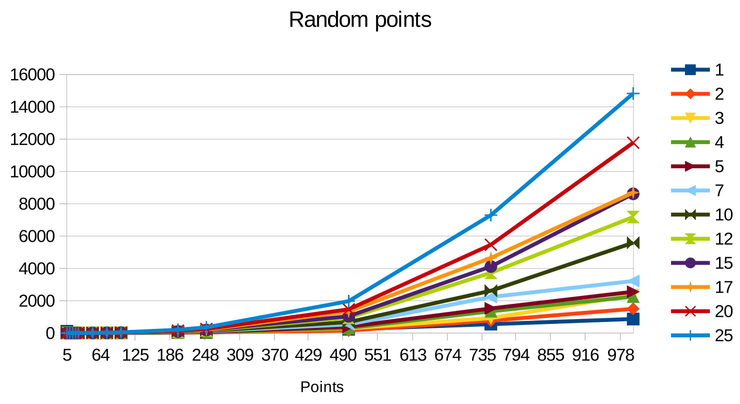


FIGURE 3.4: Time cost depends on set size with random points

3.1.3.3 Separable Case

The applied optimization API throws exceptions if the tasks cannot be solved and stops which improves the performance of the library. These exception were handled as $\mathcal{X} \nparallel \mathcal{Y}$. So if the clusters are not linear separable than the API stops the calculation. To measure the cost of the method the linear separable case were tested too. The results show if two sets are linear separable then its time cost is about 5 times of the linearly non separable case. To compare the separable case with the random case, the measurement was run with same dimension and size parameters, but in separable case the point sets were generated in different intervals.

Figure 3.5 shows the time cost as a function of the number of the dimensions. It can be compared with Figure 3.3 which shows the random case. It can be seen that the time cost grows linear with the number of dimensions.

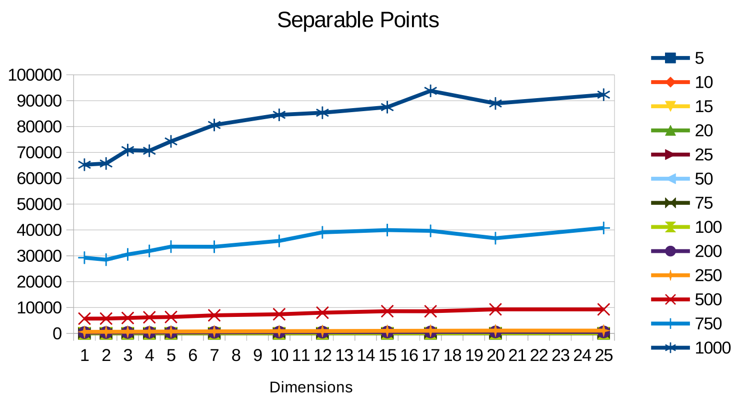


FIGURE 3.5: Time cost depends on the number of dimensions with separable point sets

The connection of time cost and the number of points is shown in Figure 3.6. It shows clearly the time cost also grows polynomial with the number of points. The biggest set contained 1000 points and it took about 3 minutes to test linear separability. The polynomial growth makes the calculation slow. Regression function $f_r(p) = ap^2 + bp + c$ was calculated by linear regression with $R^2 = 0.9946$. Based on this function the run time of the algorithm was estimated like weeks. Time cost of testing for the two largest point sets was estimated about 10 hours.

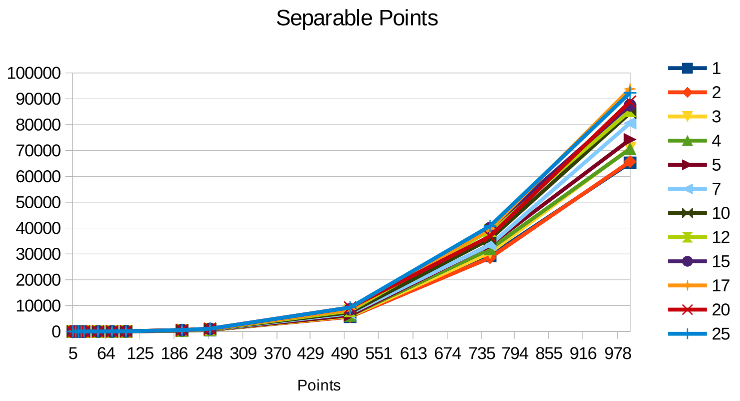


FIGURE 3.6: Time cost depends on the size of point sets in separable case

3.2 Phonetics

Written text can be considered as an encoding of speech with symbols called letters i.e. the sounds are denoted by letters. The smallest parts of the speech are the sounds which can be distinguished by.their phonetic features. Because letters usually denotes the same or similar sound, the phonetic features of the letters can be assigned by linguists. Natural languages can pronounce a letter differently even a word can have multiple pronunciation such as the word "neither".

3.2.1 International Phonetic Alphabet

International Phonetic Association created a system of phonetic notation for oral languages which is called International Phonetic Alphabet [Ass99]. The notation of the International Phonetic Alphabet is based on the latin alphabet and it uses other symbols to denote sounds. This notation system is widely used by lexicons, dictionaries, languages learners and teachers, translators, actors and so on. Beside the letters, diacritics are also used to compose symbols. The diacritics slightly modify the sound of the previous letter. The latest version was released in 2005 and it contains 107 different letters and 52 diacritics.

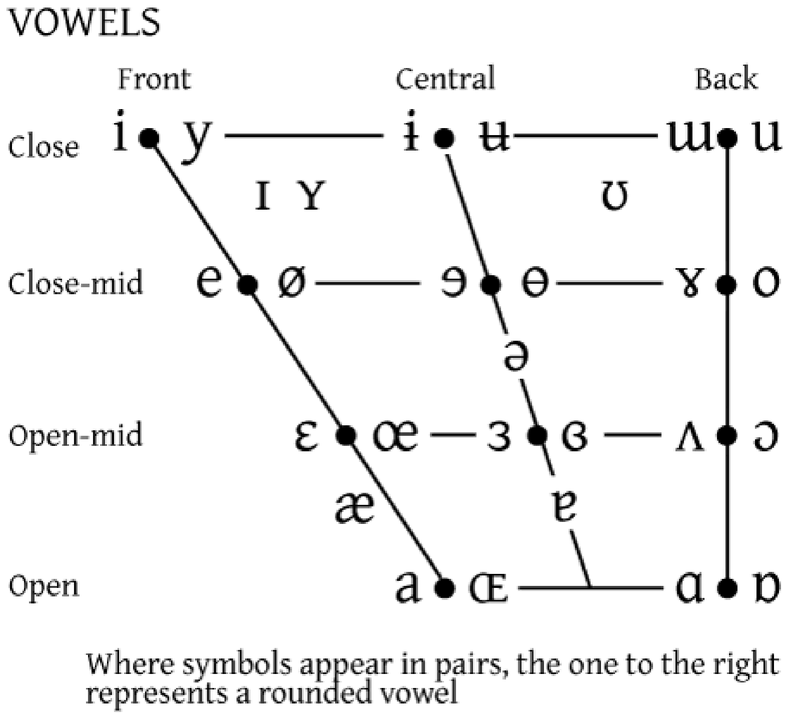
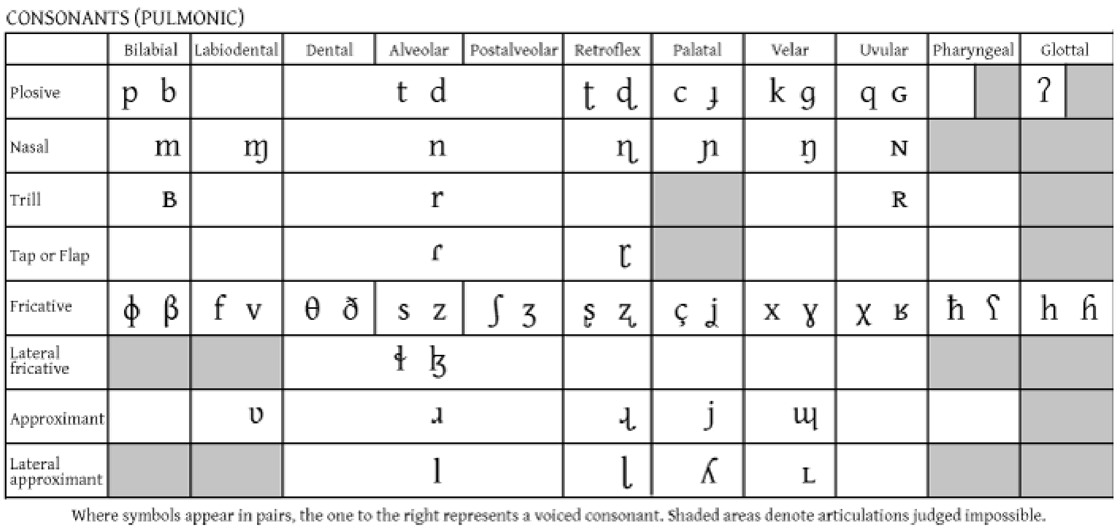


Figure 3.7: International Phonetic Alphabet – Vowels [Ass99]

3.2.2 Phonetic Features of Hungarian

The International Phonetic Alphabet allows the notation of any sound of spoken languages, but the natural languages usually use only a subset of these sounds. The Hungarian alphabet contains only 26 consonants and 14 vowels and these letters usually denote



CONSONANTS (PULMONIC)

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ	▨	ʔ ▨
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ	▨	▨
Trill	ʙ			r			▨		ʀ		▨
Tap or Flap				ɾ		ɽ	▨				▨
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative	▨	▨		ɬ ɮ						▨	▨
Approximant		ʋ		ɹ		ɻ	j	ɰ			▨
Lateral approximant	▨	▨		l		ɭ	ʎ	ʟ		▨	▨

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

FIGURE 3.8: International Phonetic Alphabet – Consonants [Ass99]

only one sound so only a part of the International Phonetic Alphabet is used to describe Hungarian words. Because my research is focused on the induction of inflection rules of Hungarian, the phonetic features of letters of Hungarian alphabet are presented below.

The vowels and consonants have different phonetic features. The presented model distinguishes the shape of the lips, the position of the tongue and the pitch in the case of vowels and the voice, the way and the place of the production in the case of the consonants. Table 3.1 shows the phonetic features of the vowels. The rows denote the position of the tongue, the voice is defined by the columns and the italic letters stand for the rounded vowels and the plain letter denote the non rounded ones. The phonetic features of the consonants are summarized in Table 3.2. The rows show the different places of the sound production. The columns show the way of the sound production and within these categories the voice is also distinguished. Because the pronunciation of the letters can slightly differ in natural languages, their phonetic features can be found in language specific text books [ASA01, Zsu97].

TABLE 3.1: Phonetic Features of Vowels

	Front	Back
Close	*u*,*ú*	*ü*,*ű*,i,í
Mid	*o*,*ó*	*ö*,*ő*,é
Open	a,*á*	e

Phonetic features have ordinal or categorical values. However linguists defined the categories for phonetic features neither ordering nor numerical values are defined for these features. On the other hand ordering can be defined for some features, such as the pitch can be low or high, the position of the tongue can be upper, middle or lower. Many well–known data mining method [Jol05] can work only with numerical features. The proper

Table 3.2: Phonetic Features of Consonants

Way of Production	Polsive		Nasal		Fricative		Lateral approx.		Lateral frica-tive		Trill	
Place of Production	Voiced	Unvoiced	Voiced	Unvoiced	Voiced	Unvoiced	Voiced	Unvoiced	Voiced	Unvoiced	Voiced	Unvoiced
Bilabial	b	p	m									
Labio–dental					v	f						
Dental / Alveolar	d	t	n		z	sz	l		dz	c	r	
Dental / Postalveolar					zs	s			dzs	cs		
Palatal	gy	ty	ny		j							
Velar	g	k										
Glottal						h						

conversion of phonetic categories into numerical values requires an expert of linguistics. Next it is assumed that each phonetic feature is converted to real values so letters can be modeled as real vectors.

To sum up a sound can is determined by its phonetic features. In my work following six features and their categorical values are distinguished:

Consonants

- Way of Production
  - Polsive
  - Nasal
  - Fricative
  - Lateral approx.
  - Lateral fricative
  - Trill
- Place of Production
  - Bilabial
  - Labio–Dental
  - Dental / Alveolar
  - Dental / Postalveolar
  - Palatal
  - Velar
  - Glottal

Vowels

- Position of the tongue
  - Front
  - Back
- Shape of the lips
  - Rounded
  - non–rounded
- Pitch
  - Close
  - Mid
  - Open

- Voice
  - Voiced
  - Unvoiced

3.3 Vector Space Representation

Letters can be identified by their phonetic features so they can be assigned to tuples of phonetic features. Both vowels and consonants are defined by different three–three features, so any letter can be represented in a six dimensional space. Because the phonetic features are converted to real values, this space is a vector space, $l \in R^6$ where l stands for a letter. If a letter does not have a given feature then its value is 0 in the corresponding dimension. There are many different method in the literature to handle data in vector space.

To define the similarities between the letters, their distances can be calculated in Euclidean space. The distance of a letter pair can be calculated as $d(l_i, l_j) = \sqrt{\sum_k (l_{i,k} - l_{j,k})^2}$ where $l_{i,k}$ denotes the kth feature of the l_i word. The distance is smaller for similar letters. Although in mathematics an alphabet is a set of letters, in traditional alphabets the positions are well–defined and fixed. To create an phonetic features based traditional alphabet the letters have to be mapped into a one dimensional space where each letter has a corresponding one dimensional point and the distances of the original and the mapped points remains similar. The dimensionality reduction usually yields errors in the distance ration of the resulted point set.

Figure 3.9 shows a mapping of a two dimensional space into a one dimensional. The points in the two dimensional space are A, B,, C and D, E and their images in the one dimensional space are A, B, C', D' and E'. Because A and B points are on the X axis and the point are projected into the X axis they are their own images. The C' point splits the AB into two parts AC' and BC' which pairs length is proportional with the length of AC and BC. It can be seen in the ABD and ABE triangles too. In the case of C D and E point the distances of their images are not proportional with the distances of original points. It can be seen the distance of DE and $D'E'$ remained but the $C'D'$ and $C'E'$ distances are got shorter. So the dimensionality reduction usually yields error.

There are many different methods in the literature [Fod02] to map a n dimensional vector space into a m dimensional where $n \geq m$. The Principal Component Analysis [Jol05, WEG87] and the Multidimensional Scaling [Kru64] are classical dimensionality reduction techniques. There are application specific researches on the methods of dimensionality reduction techniques [KHP05]. There are also recent results on the non–linear

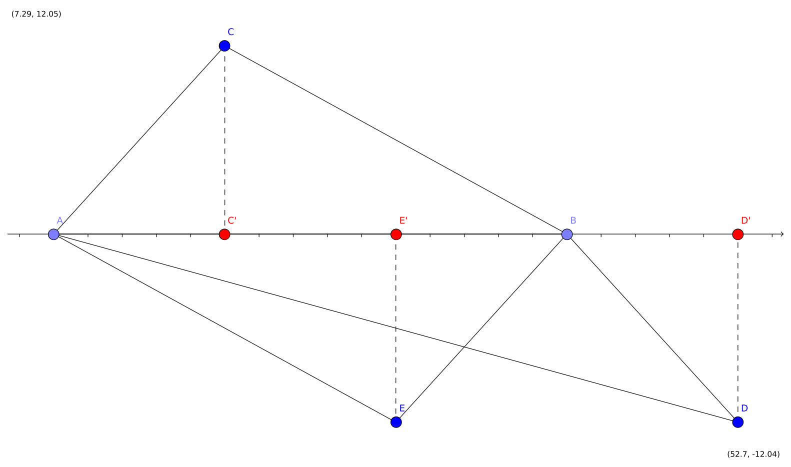


FIGURE 3.9: Dimensionality reduction

dimensionality reduction [TDSL00]. To create the phonetic alphabet we used the Principal Component Analysis because it is a classical, well–studied and widely used technique.

3.3.1 Dimension Reduction based on Principal Component Analysis

Principal Component Analysis is based on the eigenvalue decomposition of the covariance matrix of the samples. The objects have to be represented in a vector space and each feature has to be numerical. Hence the dataset can be converted into a real matrix. Let $\mathbf{L}$ denote the matrix which represents the dataset of the letters. Its size is determined by the number of the letters in the alphabet and the number of the phonetic features, $\mathbf{L} \in R^{44 \times 6}$ for the Hungarian language.

The covariance measures how two random variables change together. The sign of the covariance shows the tendency in the linear relationship. The value of the covariance is not so easy to interpret. If it is zero, then the two random variables are independent. In statistics its normalized version, the correlation is used often. The covariance is defined in equation 3.6 where X and Y are the random variables and $E(X)$ denotes the expected values of X. Based on the definition $cov(X, X) = \sigma^2(X)$ where $\sigma(X)$ is the variance of X. The empirical covariance can be calculated with the equation 3.7 where $\overline{x}$ is the average of X and $\overline{y}$ is the average of Y.

$$cov(X, Y) = E(E(x - E(X))E(y - E(y))) \tag{3.6}$$

$$cov(X, Y) = \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \tag{3.7}$$

Covariances between n random variable can be organized into a $n \times n$ matrix form. Covariance matrix Σ is a symmetric square matrix where each the rows and columns denote features in a fix order. Equation 3.8 shows the structure of an arbitrary covariance matrix, where f_i stands for the ith random variable.

$$\Sigma = \begin{bmatrix} cov(f_1, f_1) & cov(f_1, f_2) & \dots & cov(f_1, f_n) \\ cov(f_2, f_1) & cov(f_2, f_2) & \dots & cov(f_2, f_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(f_n, f_1) & cov(f_n, f_2) & \dots & cov(f_n, f_n) \end{bmatrix} \tag{3.8}$$

Eigenvalues $\mathbf{v}$ and eigenvectors λ of a covariance matrix can be determined. The eigenvalue and the eigenvector satisfy the $\mathbf{\Sigma v} = \lambda \mathbf{v}$ equation. This equation can be transformed into a linear equation system $(\mathbf{\Sigma} - \lambda \mathbf{I})\mathbf{v} = 0$. The equation system has multiple solution and each solution gives an eigenvalue Let $\mathbf{V}$ denote a column vector of eigenvectors $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_n}]^T$ where $\mathbf{v_i}$ is the ith eigenvector. And $\mathbf{\Lambda}$ stands for the column vector of eigenvalues $\mathbf{\Lambda} = [\lambda_1, \lambda_2, \dots, \lambda_n]^T$ where λ_i is the ith eigenvalue. The most significant eigenvector has the highest corresponding eigenvalue so the eigenvectors are ordered by their eigenvalues. It means the point spread across this vector in a biggest range. The higher eigenvector means higher spread across the corresponding eigenvector. Figure 3.10 shows principal components of a dataset.

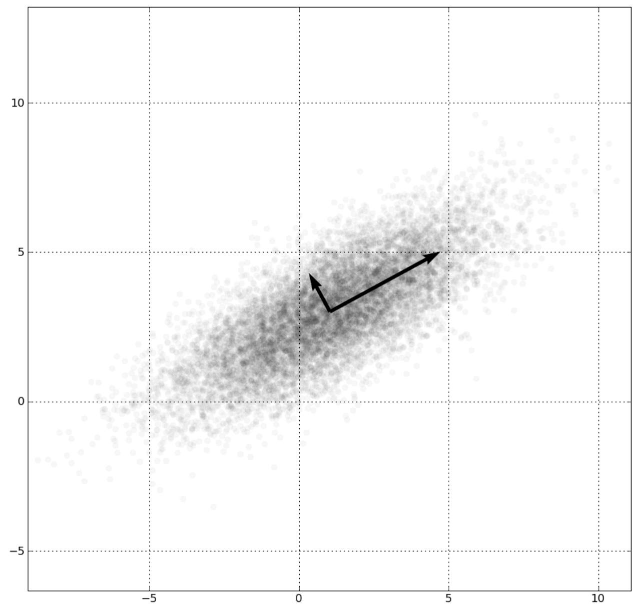


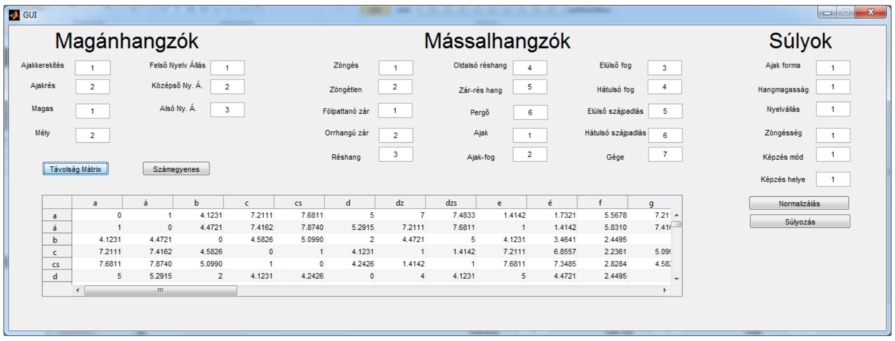
FIGURE 3.10: Principal Component Analysis Example [Wik14]

To reduce the dimensionality of the dataset the eigenvectors with less eigenvalue can be omitted. The reduced matrix of eigenvectors is denoted by $\mathbf{V'}$ and $\mathbf{\Lambda'}$ stands for the reduced vector of eigenvalues. This omission yields error in the reduced dataset as it was shown above, but this error is minimized in principal component analysis. Because the goal is to define a phonetic features based traditional alphabet every eigenvector except the most significant are omitted. In other words only the most significant eigenvector is used to create the phonetic alphabet.

The reduced dataset $\mathbf{L_r}$ is yielded by a linear transformation and a transposition. The chosen eigenvectors are ordered by their eigenvalue gives the first component of the product $\mathbf{V'}$. The second component $\mathbf{L}^*$is the mean–adjusted and transposed dataset i.e. $\mathbf{L}^* = (\mathbf{L} - [\overline{\mathbf{L_{(:,1)}}}, \overline{\mathbf{L_{(:,2)}}}, \ldots, \overline{\mathbf{L_{(:,m)}}}])^T$ where $\overline{\mathbf{L_{(:,i)}}}$ denotes the average of the ith column or feature of dataset $\mathbf{L}$ and the $[\overline{\mathbf{L_{(:,1)}}}, \overline{\mathbf{L_{(:,2)}}}, \ldots, \overline{\mathbf{L_{(:,m)}}}]$ vector has as many rows as $\mathbf{L}$. The result of the multiplication contains the objects in the columns and the features are encoded by the rows. It can be transposed in order to bring the objects into the original format. Formally the reduced dataset is given as $\mathbf{L_r} = (\mathbf{V'} \times \mathbf{L}^*)^T$ and its size if $n \times m'$ where $m' \leq m$ is the number of the chosen eigenvectors.

3.4 Generation of Phonetic Alphabet

A MatLab application was developed to create phonetic features based alphabet for Hungarian. Appendix B details how the source code can be obtained. MatLab supports to perform matrix operation on complex n dimensional matrices. We used the neural networks toolbox to show the proposed phonetic alphabet based encoding is superior the standard alphabet based one. Moreover MatLab as a programming language allows to create graphical user interface for applications which can use the MatLab functions. With its built–in graphical user interface editor it is easy to create simple applications for data processing or other purposes.



Magánhangzók

Ajakkerekítés	1	Felső Nyelv Állás	1
Ajakrés	2	Középső Ny. Á.	2
Magas	1	Alsó Ny. Á.	3
Mély	2		

Mássalhangzók

Zöngés	1	Oldalsó réshang	4	Elülső fog	3
Zöngétlen	2	Zár-rés hang	5	Hátulsó fog	4
Fölpattanó zár	1	Pergő	6	Elülső szájpadlás	5
Orrhangú zár	2	Ajak	1	Hátulsó szájpadlás	6
Réshang	3	Ajak-fog	2	Gége	7

Súlyok

Ajak forma	1
Hangmagasság	1
Nyelvállás	1
Zöngésség	1
Képzés mód	1
Képzés helye	1

	a	á	b	c	cs	d	dz	dzs	e	é	f	g
a	0	1	4.1231	7.2111	7.6811	5	7	7.4833	1.4142	1.7321	5.5678	7.21
á	1	0	4.4721	7.4162	7.8740	5.2915	7.2111	7.6811	1	1.4142	5.8310	7.41
b	4.1231	4.4721	0	4.5826	5.0990	2	4.4721	5	4.1231	3.4641	2.4495	
c	7.2111	7.4162	4.5826	0	1	4.1231	1	1.4142	7.2111	6.8557	2.2361	5.09
cs	7.6811	7.8740	5.0990	1	0	4.2426	1.4142	1	7.6811	7.3485	2.8284	4.58
d	5	5.2915	2	4.1231	4.2426	0	4	4.1231	5	4.4721	2.4495	

FIGURE 3.11: User Interface of the developed MatLab Application for Hungarian

The user interface of the developed application is shown in Figure 3.11 (and Appendix A). It consists of four parts. In the first part the user can define the numerical values for the features of the vowels. The features of the consonants can be mapped into numerical values in the second part of the user interface. The weights of the different features and the buttons are placed in the third part. The weights allows to set the relative significance of the features. If a feature has a greater weight then it is more important. The program allows to turn off the weights in this case the features have the same weight. It also supports the normalization of the features. Finally the fourth part of the user interface shows a table which contains the distance matrix of the letters. Its value is computed based on the parameters set in the first and second panel. The yielded alphabet can be visualized in chart which is shown in a separate window (See Figure 3.12).

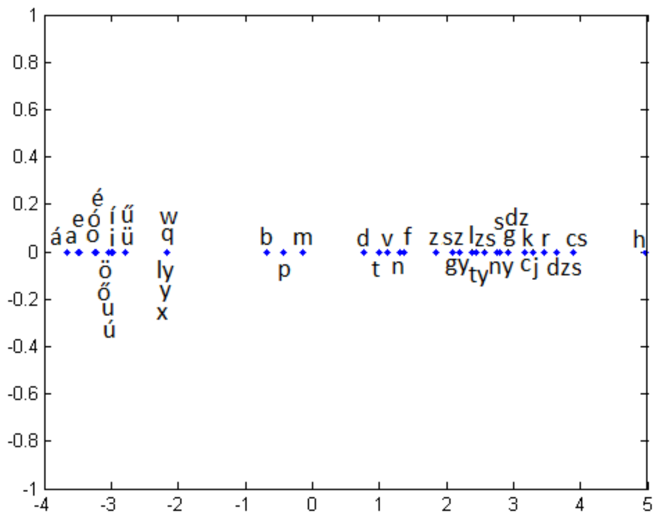


FIGURE 3.12: Phonetic features based Hungarian alphabet

The above detailed method to generate phonetic features based alphabet was tested with a different parameters. To generate this phonetic alphabet, the categories of the different features were encoded by their indexes in the textbook [Zsu97]. The weights of the features were set to 1 for each feature and the category values were normalized.

The generated alphabet is visualized in Figure 3.12 and the numeric values of the letters are in Table 3.3. The numerical values in the Table were limited to four digits, but the MatLab stored their values more precisely.

Six clusters of similar letters can be identified in Figure 3.12. The first cluster contains the vowels. The vowel `o` and `ó` has the same value in Table 3.3 their values but difference is too small to visualize. The `w`, `q`, `ly`, `y`, `x` letters belong to the second cluster and have the same numerical value. Its reason is that the phonetic features were not defined [ASA01, Zsu97] so their feature vectors were chosen to equally zero. The third cluster

TABLE 3.3: Generated Phonetic Alphabet

Letter	Value	Letter	Value	Letter	Value	Letter	Value	Letter	Value
á	-3.6677	i	-2.9822	p	-0.4340	l	2.3863	r	3.4672
a	-3.4845	í	-2.9822	m	-0.1346	ty	2.4458	dzs	3.6467
e	-3.4746	ü	-2.7990	d	0.7649	zs	2.5658	cs	3.8877
o	-3.2384	ű	-2.7990	t	1.0059	ny	2.7453	h	4.9666
ó	-3.2384	ly	-2.1764	v	1.1258	s	2.8068		
é	-3.2284	q	-2.1764	n	1.3053	g	2.9248		
ö	-3.0452	w	-2.1764	f	1.3668	dz	2.9267		
ő	-3.0452	x	-2.1764	z	1.8458	k	3.1658		
u	-2.9922	y	-2.1764	sz	2.0868	c	3.1677		
ú	-2.9922	b	-0.6750	gy	2.2.048	j	3.2857		

contains the `b`, `p` and `m` letters which are Bilabials moreover (`b`,`p`) is a voiced–unvoiced pair. The `d`, `v`, `f`, `n` and `t` belong to the fourth cluster. These letters are both Labio–dental or Dental words and in addition (`d`,`t`) and (`v`,`f`) are also voiced–unvoiced pairs. The fifth cluster is the biggest and it the dental letter are placed in the lower part, the palatal letters are in the middle and velar letters are on the upper part of the cluster. Finally the last cluster contains only the `h` letter which is the only glottal letter in the Hungarian alphabet.

The clusters and the distances of the letters shows that the similar ones are close together in the yielded alphabet. This property can improve the fault tolerance of systems. For example the `u` and `ú` letters are mapped into the same numerical value based on Table 3.3 and they has the same pronunciation in the word "áru" and "kúp".

However the resulted alphabet has many advantages compared with the traditional one, but there are still some errors. For example the letter `s` and `g` are close together in the alphabet, but they have no common phonetic feature. Moreover the letters `w`, `q`, `ly`, `y`, `x` were mapped without the proper definition of the phonetic features. On the other hand the letters `w`, `q`, `y`, `x` do not occur in Hungarian words except names or foreign words and the letter `ly` denotes the same sound as `j`.

3.5 Linear Separability of Inflection Rules in Hungarian

Inflection can be considered as a string transformation which describes how can a word converted into its inflected form. These transformation can be encoded by a transformational string which describes the transformation. Transformation string can be derived from the stem – inflected form pairs. A category of words is defined by each transformation string.

Classification is a process which decides that a given object in which predefined class belongs to. Most of the classification algorithms require vector space representation of the objects. There can be a cluster of objects defined for each category. A cluster is the smallest convex hull in the space which contains each point of given category . These clusters can be disjoint or overlapped. Efficient and accurate classification algorithms can be created if each clusters are disjoint. If two clusters are disjoint, than they are linear separable.

Testing of linear separability based on linear programming is reviewed [Eli06]. The optimization task is formalized from the $\mathcal{X}_1$ and $\mathcal{X}_2$ clusters of the objects. The time cost of the method was evaluated on generated data set. Measurements showed that the time cost grows polynomially with the number of points.

3.5.1 Linear Separability of Inflection

Linear separability in inflection was tested with alphabetical and phonetic encoding. The vectors were expanded to length of the longest word which has 22 characters. With both encoding we used left and right adjust to create the real vector. Left adjust means the vector starts with the numeric representation of the word. In the case of right adjust, the vector ends with the numeric values. Empty characters were represented by zeros.

The input contained about 54.000 word pairs which can be categorized into about 160 different categories in both encoding. Approximately 98% of the training set belongs to one of the largest 15 clusters. Thus there are a few large classes and lot of small ones. From the point of view of time cost it means some of testing could require hours or days while others only seconds.

The maximum size of a cluster were limited to 2000 samples in order to reduce the time cost of the testing. As it is shown above the time cost of the applied simplex method grows polynomially with the number of the points.

The clusters which are smaller than the limit were unchanged. The bigger clusters were reduced by the omission of the elements. The elements were ordered by their base words, then 2000 elements were chosen equidistantly based on their indexes. Due to the ordering, if an element is omitted, then there can be a similar one in the reduced cluster.

The omission affects on the results too because it could yield false positive tests. Two clusters can become linear separable by the omission of the overlapping points. Hence the true test results can be false positives. This omission has no effect on the false test because if two clusters are not linear separable, then the will not become linear separable by adding new points to one of the clusters i.e. there are no false negative tests. This trade–off was necessary due to the huge time cost of the testing algorithm.

Table 3.4 shows the how many non linear separable cluster pairs are in the tested cases. The encodings are stored in the columns. The first row shows how many linear non separable cluster pairs were found during the test. The second row shows the ratio of the non separable cluster pairs with the total number of cluster pairs. It can be seen that less non separable cluster pairs were found in the case of Phonetic alphabet than in the case of Alphabetical encoding. And there are about twice as mans as non separable cluster pairs with right encoding than with left encoding.

Table 3.4: Linear Separability Ratio of Tested Representation for Hungarian Accusative Case

Encoding	Alphabetical		Phonetic	
Adjustment	Left	Right	Left	Right
Non Separable	405	972	350	853
Ratio	3.0675%	7.3620%	2.6509%	6.4607%

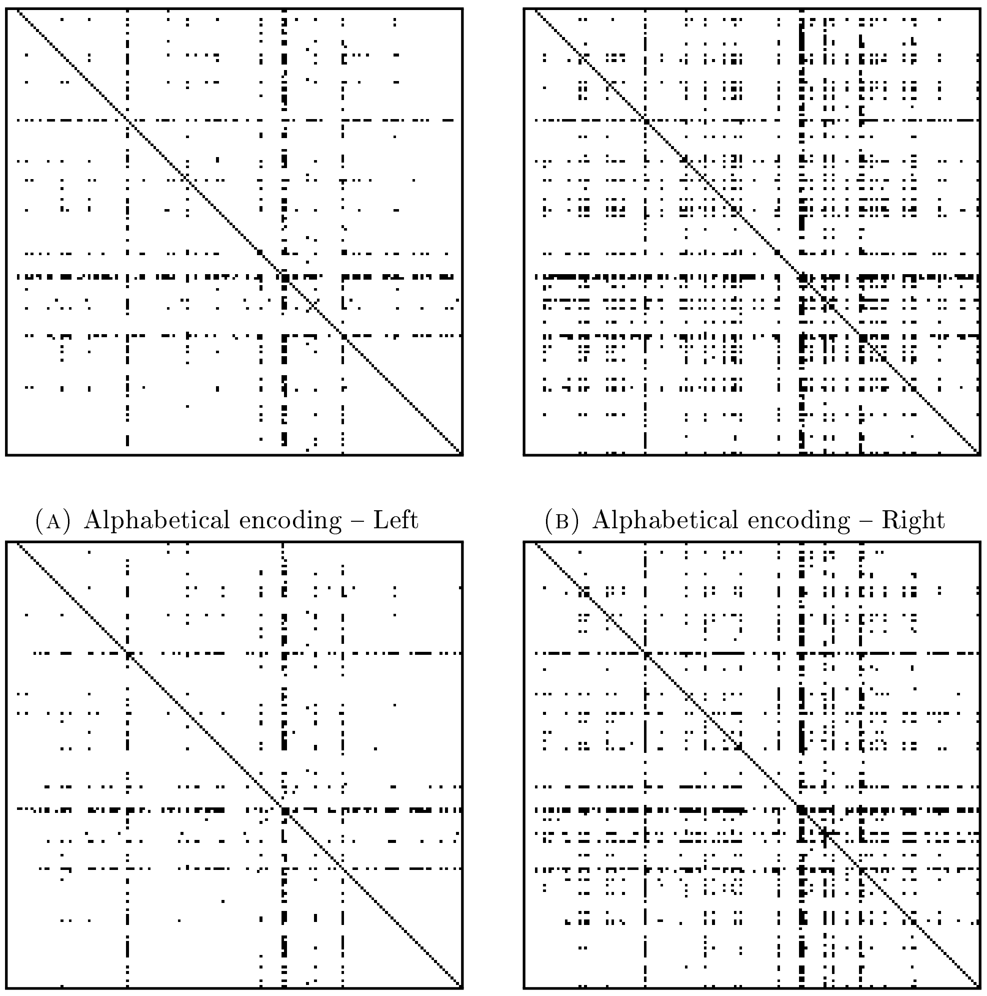
3.5.1.1 Visualization

There were about 160 categories in both encoding so the results are visualized in an bitmap image. The categories define the size of the image. A white dots stand for a separable cluster pair and a black dots denote a non separable ones. One cluster cannot be separated from itself, thus if each cluster are separable from other then there is only a black diagonal line in the white square. Because the separability is symmetric, the dots are also symmetric onto the diagonal.

The result bitmap images are in Figure 3.13. Figure 3.13a shows that there are a few non linear separable clusters in the case of alphabetical encoding and left adjust. Using same encoding with right adjust, there are more non linear separable clusters which can be seen in Figure 3.13b. The results of the phonetic encoding are shown in Figure 3.13c and Figure 3.13d. Left adjust yielded more separable classes in the current training set, than the right adjust. Figures shows also that, there are more separable clusters with phonetic encoding with left adjust than in the case of alphabetical representation. On the other hand the worst case was given by phonetic encoding with right adjust.

3.6 Conclusions

To sum up, a method has been presented to generate phonetic features based alphabet which maps the letters into real values. Although this representation is one dimensional,


(A) Alphabetical encoding – Left (B) Alphabetical encoding – Right

(C) Phonetic encoding – Left (D) Phonetic encoding – Right

Figure 3.13: Linear Separability with Alphabetical and Phonetic encoding

it is similar to the traditional alphabet because it defines an ordering and the distances of the letters represent their phonetic similarities. There are clusters of similar letters in the yielded alphabet which confirms that the phonetic alphabet fulfills its initial goals.

Linear separability is an important property of clusters which allows to create efficient classification methods. Traditional and phonetic alphabets were used to encode a training set of about 56,000 samples stem – accusative case word pairs in Hungarian. Category keys were determined by Levenshtien distance of the words. The linear separability testing method has $O(dp^2)$ time cost where d denote the number of the letters in the longest word and p stands for the size of the clusters. Although phonetic alphabet shown superior to the traditional alphabet in experimental results, the experiment showed that there are non linear separable cluster pairs with both tested representations.

Thesis 1.

I have measured the complexity of inflection rules induction by the ration of the linear separable clusters pairs in the vector space. I have introduced a methods to create phonetic features based alphabet. The created phonetic alphabet based encoding was shown superior to traditional alphabet based encoding.

Related Publications: [8], [9]

Chapter 4

Induction of Inflection Rules with Classification and Associative Memory

Inflection is a vital element to express semantic in synthetic languages. Proper inflection is crucial for text generation and reporting systems. The induction of inflection rules is an open question in computational linguistics. The existing solutions use dictionary, transformation rules or statistical observations to inflect a stem. These methods have drawbacks either in precision and cost efficiency. A novel inflection method, called AMC, has been developed which is based on classification and associative memory. The words which belong to non-frequent categories are stored in the associative memory thus the classification process can be performed faster. The transformations for the regular words are determined by the classifier. Precision, size and time cost of the algorithm are measured with different sized associative memory. The precision of the proposed algorithm can exceed the 90 per cent based on the experimental results. The AMC method is compared with standard classification based inflection methods.

4.1 Computational Morphology

Computational Linguistics aims at capturing the aspects of the natural languages by rule-based and statistical models. Computational linguistics provides solution for various tasks such as morphological analysis, stemming or inflection. The algorithms of computational linguistics are widely used in Natural Language Processing solutions. For example stemmers and morphological analyzers are used in information retrieval systems.

Algorithms on inflection are used in text generation and machine translation applications, too.

The first systems were focusing on simpler morphological problems. For example, the inflection of the past tense of English was analyzed in [Pin91]. It is assumed that there are different modules in human mind which are responsible for the inflection. There is a rule based process to inflect the regular verbs which allow the inflection of even unknown words. But its drawback is the over regularization which transforms irregular verbs into incorrect form. For example children make grammatical errors when they say “comed” or “breaked” [Mar96]. The learning of irregular verbs is similar to an associative memory. On the other hand there can be similarities found between irregular verbs which lead to the theory of rule-associative-memory.

Inflection is the inverse function of stemming which is well-studied in text mining. Stemming algorithms are based on various approaches such as dictionary, rules or statistics. Although dictionary based methods provide the highest accuracy, they cannot generalize nor handle untrained words. Moreover the building of the dictionary is costly, time-consuming and requires language experts. Rule based stemming methods has a trade-off between accuracy and cost. The Porter stemmer is one of the most popular rule based stemming algorithms [Por80, Por01]. Statistical methods requires no language experts, but they can have a low accuracy. An unsupervised statistical stemming method is presented in [GAH04] which transforms the induction task into an optimization task. Although it has promising results for fusional languages, it was not tested with agglutinative languages which have more suffixes and more complex inflection rules.

SMOR is a morphological analyzer [SFH04] for German inflection rules based on Finite State Transducers. Stochastic transducers are also used to learn morphology [Cla02]. The rules were implemented in Stuttgart Finite State Transducer tools and SMOR uses a lexicon which only stores the properties of the stems. SMOR has rules for prefix, suffixes, derivation, composition and inflection. In the experiments the precision of the SMOR were above 95 per cent in general and the precision depends on the frequency of the word.

Bayesian approach was used to perform morphological analysis in [NG09]. It assumes that the spelling rules occur at the end of the word. The $P(c, t, f, y, r \mid w)$ model is used to define the stem for the word where w is the word, c is the class of the word, t is the stem, f is the suffix, y is the type of the spelling rule and r is the transformation. During the inference a standard Markov Chain Monte Carlo technique was used. Their experiments showed that the accuracy of the stem and suffix recognition depends on the context. The accuracy of stem recognition is about 65 per cent and the accuracy of suffix recognition is about 78 per cent. Although this method is no as precise as the above mentioned rule based algorithms, it does not require human experts and a priori knowledge about the grammatical rules of the language.

The endings of the words are considered as classes in [MLCL04] because the language learners often learn induction tables where a cell denotes an inflection class. Based on their endings the words are organized into candidate inflection classes. These classes can be organized into a lattice. The authors tested five different reduction algorithms from the point of view of precision and recall. The tests were evaluated with both English and Spanish languages.

4.1.1 Hungarian Solutions

Although most of the researches are focused on English or other major language, there are solutions for languages with only a few million native speakers such as Hungarian. The "Szószablya" project [RG06, HKN$^+$04] provides a morphological analyzer for Hungarian. The KOPI is a plagiarism [Pat06a, Pat06b] checker developed by the Hungarian Academy of Sciences. These projects are focused on the information extraction.

Text generation is another branch of natural language processing which aims to generate natural language texts such as reports or questions. For Hungarian, László Bednarik created a system to generate questions for exam from annotated text [Bed12].

4.2 Dataset

The induction of the inflection rules for the accusative case of Hungarian was tested based on a training set of 54.000 (stem,inflected form) pairs. The training set contains almost every Hungarian noun and the inflection was made by native speakers. The training set was converted into an structure which contains a category key and a numeric array for each pairs. The efficiency of the classification depends among others on the representation of the items. According to our previous investigations on the different word representation methods [8, 9], the phonetic alphabet representation method is superior to the standard methods which are usually based on the external parameters of the objects. Phonetic alphabet considers the phonetic features of the sounds, denoted by the letters. The letters can be mapped into real values based on the phonetic alphabet and the letter with similar sound will be closer together than different letters. Due to the different length of the words the shorter words was augmented by spaces to the length of the longest word so each word had the same length. These words was converted to real vectors by the substitution of the letter with real values. The test measure the time cost of the learning, the precision and the serialized size of the classifier structure. These parameters usually depend on the size of the training set.

4.2.1 Categories

The categories are denoted by the transformation string which is determined based on the stem, inflected form pair. Thus each category denotes an inflection rule. Our preliminary assumption is that there are only a few or maybe a dozen inflection rules for a case. The experimental results shows that there are approximately 160 categories in the training set. On the other hand most of the words belong to on of the the big classes. Hence the high number of the categories can be explained with the irregular words.

4.2.1.1 Transformation String

The transformation rule can be characterized by the corresponding string edit distance which measures the minimum cost of the transformation which transforms a string into another form. Calculation of the Levenshtein distance [Lev66, Nav01] is based on dynamic programming. The insertion, deletion, and substitution costs are parameters of the algorithm. These values were set to 1 in the current tests and in the example shown in Table 4.1 which shows the matrix created to determine the Levenshtein distance of "alma" and "almát" words. The result is 2 because it requires a substitution and an insertion to transform the base word "alma" into its inflected form "almát". The bold numbers in the table show minimum path which determines the transformation.

TABLE 4.1: Levenshtein distance of "alma" and "almát" words

	*	a	l	m	á	t
*	**0**	1	2	3	4	5
a	1	**0**	1	2	3	4
l	2	1	**0**	1	2	3
m	3	2	1	**0**	1	2
a	4	3	2	1	**1**	**2**

The optimum transformation path is encoded by the transformation string which is built from the following symbols:

* no transformation, multiple times

- empty character

(transformation starts

/ transformation separator

) transformation ends

Based on this encoding the `***(a/á)(-/t)` transformation string can be read from Table 4.1. In order to be independent from the invariant parts of the words, sequence of `*` symbol is replaced with a single `*`. This generalized transformation strings were used as category identifiers.

4.2.1.2 Evaluation of Problem Domain

Approximately 160 categories were distinguished based on the transformation string. In spite of the great number of the categories, only a few category contains many word pairs and most of the categories describe a special case. It shows that there is a general rule of the inflection of accusative case in Hungarian, there are a few exception and there are many irregular words. This results confirm our preliminary assumptions about the size of the clusters and the number of the big clusters.

TABLE 4.2: Number of transformations in categories

Transformation	# of word pairs	% of training set
`*(-/t)`	28239	52.31
`*(-/o)(-/t)`	7399	13.71
`*(-/e)(-/t)`	6731	12.47
`*(a/á)(-/t)`	5975	11.07
`*(-/a)(-/t)`	2041	3.78
`*(e/é)(-/t)`	969	1.80
`*(é/e)*(-/e)(-/t)`	454	0.84
`*(-/ö)(-/t)`	399	0.74
`*(á/a)*(-/a)(-/t)`	191	0.35
`*(o/m)(m/a)(-/t)`	177	0.33
`*(õ/ö)(-/v)(-/e)(-/t)`	138	0.26
`*(ú/u)*(-/a)(-/t)`	123	0.23
`*(e/m)(m/e)*`	109	0.20
`*(-/v)(-/e)(-/t)`	96	0.18
`*(ũ/ü)(-/v)(-/e)(-/t)`	89	0.16
Sum	53130	98.43

Table 4.2 shows the 15 largest categories and their size. The first column shows the transformation rule. Next is the number of the pairs which belong to the category. Last two columns show their integrated size which means these categories gives about 98.5% of the training set.

4.3 Standard Classification Methods

Classification is a process to determine the unknown category feature of instances based on their known features. It is a widely used technique in data mining and there are numerous classification methods in the literature. Induction of inflection rules is considered as a classification problem. As standard classifiers, the following algorithms were used to solve the classification task: Naive Bayes [JL95], K* [CT$^+$95] and Multilayer Perceptron [Zha00]. The selected algorithms were evaluated from the point of view of precision, learning cost and size of the classifier structure.

Naive Bayes classifier is considered the punching bag of the classifiers because other classification methods are usually compared with it. It is based on the Bayes Theorem for conditional probability of independent random variables. The independence of the random variables is assumed in Naive Bayes classifier, but this assumption is not fulfilled for real problems.

K* classifier is an instance–based learner and it uses an instance-database. Classification is based on distance calculation and minimum search in instance–based classifiers which is a costly task. The minimum search is a linear algorithm $O(n)$ which could slow the classification process in the case of big training set. There are various distances defined between real vectors or sets. For real vectors the Minkowsky distance is applied usually which special cases are the Manhattan and the Euclidean distance. Own distance function has to be defined for complex structures with ordinal or categorical variable which is not an obvious task.

Artificial Neural Networks are considered as universal function approximation methods. For classification task the feed-forward neural networks which is called Multilayer Perceptron are applied. The input of the network is the known feature and the output denotes the category. Neural networks can handle only numerical features because it is based on matrix operations. Thus the non numeric values have to be converted and this conversion could require and expert. The size of the network depends on the number and the size of the layers. The time cost of the classification is considered to be constant.

4.3.1 Precision

Precision is one of the most important measures of classifiers. It shows how many percentage of the items are classified correctly. The following figures show how the precision of the different algorithms depending on the size of the training set in each representation. The X-axis shows the number of the samples and the Y-axis shows the precision.

Figure 4.1 shows that the precision of the Naive Bayes classifier decreases with then size of the training set. The best precision is less than 30% which is a poor result. The measurement shows that the left adjustment is superior to the right adjustment. This phenomena can be explained with the properties the inflection of accusative case in Hungarian.

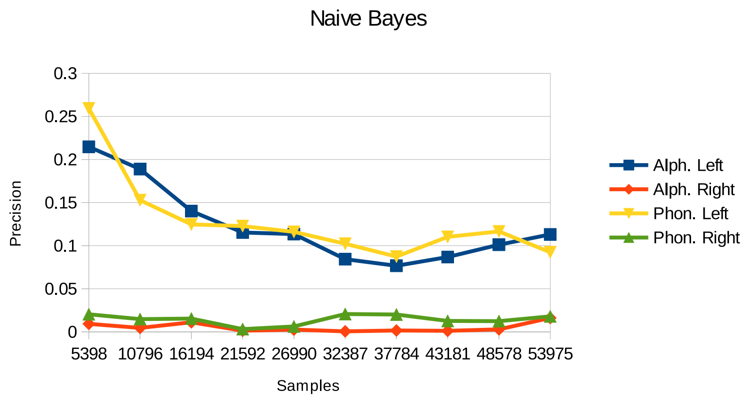


FIGURE 4.1: Precision of Naive Bayes Classifier

Figure 4.2 shows the precision of the K* classification algorithm. The precision slowly increases with the size of the training set and it is around 70%. This phenomena can be explained with that there are only few exceptions in the training set and most of the words belong to one of the few big categories. It also shows that the left adjustment is superior to the right one and the alphabetical encoding suits better for this algorithm.

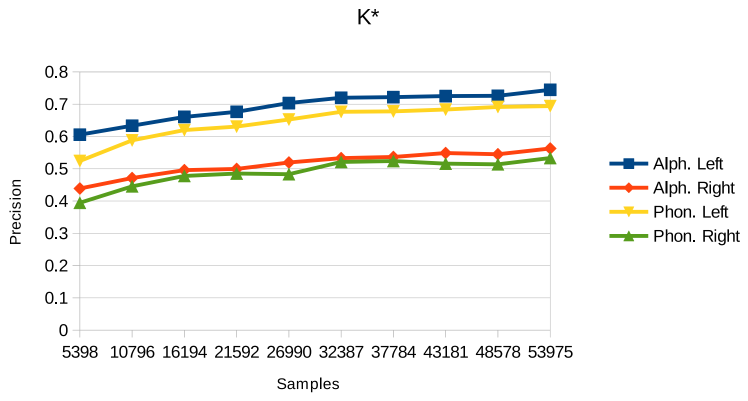
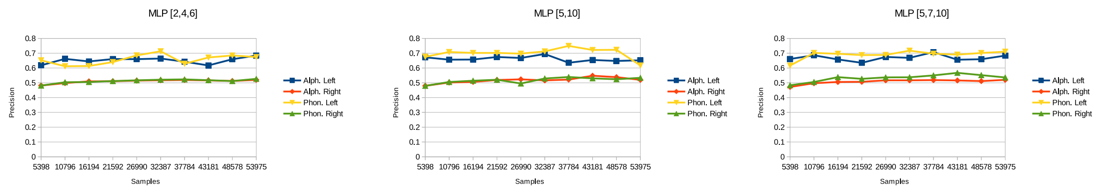


FIGURE 4.2: Precision of K* Classifier

Figure 4.3 shows how the precision of the Multilayer Perceptron classifier with different hidden layers. The size of the hidden layers are 2, 4, 6 in Figure 4.3a, 5, 10 in Figure 4.3b and 5, 7, 10 in Figure 4.3c (see Appendix A for enlarged figures). The left adjustment was also shown superior in the measurements. The precision of the neural network is around

70% so the Multilayer Perceptron classifier is considered as precise as the K* classifier. In the measurements, the phonetic encoding was found better for neural network.



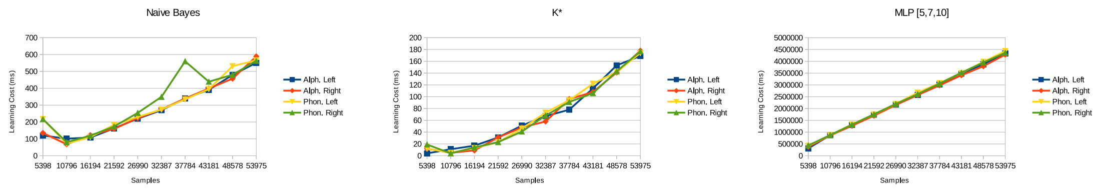
(A) [2, 4, 6] (B) [5, 10] (C) [5, 7, 10]

FIGURE 4.3: Precision of Multilayer Perceptron Classifier with different hidden layers

4.3.2 Learning Cost

Applicability of classification methods for real life applications is limited by their huge training costs. Natural languages have tens of thousands of base words and these words can have multiple inflected forms, so training sets can be of very huge size. The size of the training set is shown on the X–axis and the Y–axis shows the cost in millisecond.

Figure 4.4 shows how the learning cost of the tested classifiers depends on the size of the training set. The enlarged version of these figures can be found in Appendix A. Our experiences show that the learning costs of the tested algorithms grow linear with the size of the training set. The learning cost Naive Bayes and the K* algorithm is similar. The Multilayer Perceptron has the most significant time cost because its training requires approximately an hour with about 54.000 samples. Moreover the learning cost of the Multilayer Perceptron depends on the number and the size of the hidden layers too. The conclusion of these measurement is that the learning cost is not a significant limit of these classifiers.

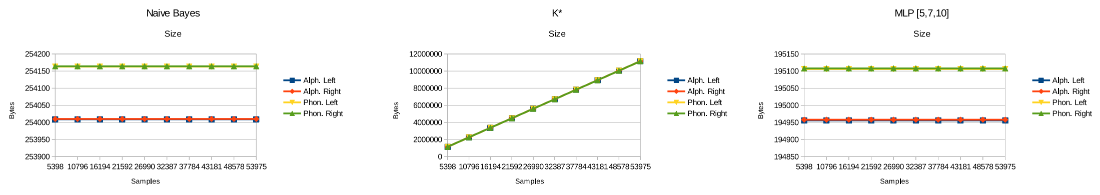


(A) Naive Bayes (B) K* (C) Multilayer Perceptron

FIGURE 4.4: Learning Cost of the Tested Classifiers

4.3.3 Size of the Classifier Structure

The size of the classifier structure is crucial from the point of performance. If the classifier fits in the memory it can work fast, otherwise the classification requires more memory swapping which decreases the efficiency. Figure 4.5 shows how the size of the classier objects depends on the size of the training set. The X–axis shows the size of the training set and the Y–axis shows the size of the serialized classifier object in bytes. Naive Bayes and Multilayer Perceptron classifiers have similar size which is between 100 and 300 MB. The size of the K* algorithm grows linear with the number of the samples because it stores the instance–database. The size increased from 1GB to 10GB. This huge size of the classifier limits the usage of this method.



(A) Naive Bayes (B) K* (C) Multilayer Perceptron

FIGURE 4.5: Size of the Tested Classifiers

4.3.4 Evaluation

Experimental results showed that the left adjustment in the encoding is superior to the right one. From the point of view of precisions the Naive Bayes classifier had poor results and the K* and the Multilayer Perceptron has similar performance. The phonetic encoding suits better for Naive Bayes and Multilayer Perceptron classifiers and the alphabetical encoding was better for K* classifier. The learning cost increased linearly with the size of the training set. Although the Multilayer Perceptron had significant learning cost, it has no significant effect on its usability for the learning of inflection rules. The size of the classifier objects was constant for the Naive Bayes and the Multilayer Perceptron classifiers, but the size of the K* classifier grows linearly with the size of the training set. Hence the usage of the K* algorithm has significant time cost for classification which limits its usability.

4.4 Proposed AMC Method

The algorithms of Computational Linguistics usually have a common model which can be seen in Figure 4.6. Morphological analyzers, stemmers and inflection systems usually have two core parts. It contains an engine to perform the transformation on the input word and to produce the output word. The engine has no direct knowledge about the language. The morphological rules are stored in a separate rule set. The structure of the rule set depends on the inflection algorithm. For example Snowball [Por01] is a language to describe stemming rules for Porter stemmer [Por80]. Rules of the SMOR [SFH04] morphological analyzer are given by the Stuttgart Finite State Tools and the engine is realized as Finite State Transducer. Classification based inflection algorithms can use the category to encode the transformation.

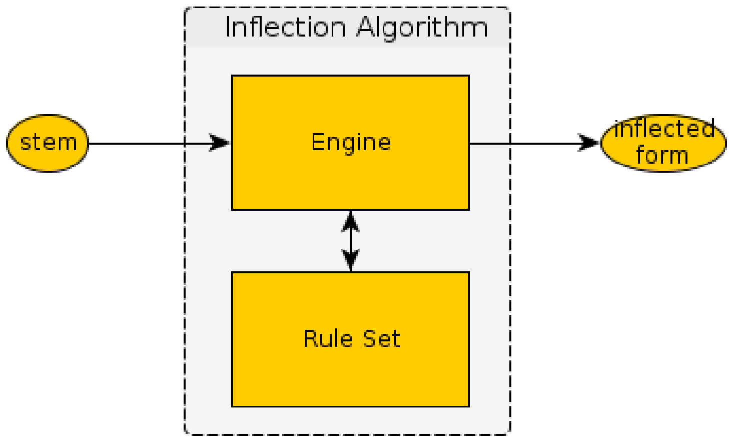


FIGURE 4.6: Common Model of Inflection Algorithms

The model of the presented inflection algorithm is showed in Figure 4.7. The implemented rule set consists of two parts: an associative memory and a classifier. The associative memory stores the transformation rules of irregular words. A word transformation class is considered as irregular, if there are only few words belonging to that class. Formally the $t = \{w\}$ category contains irregular words if $| t | \leq \epsilon$ where w is a word which belongs to class t and ϵ is a cardinality threshold. Although the associative memory can retrieve the exact transformation string for each learned stem it cannot be used to determine the transformation string of untrained words. The classifier is used to capture the frequent inflection rules. It can perform generalization thus untrained words can be processed. The generalization may easily fal on exception. Considering the classifier systems, they have lower precision than associative memory and the precision usually depends on the training set too. Secondly it has a more difficult learning algorithm which acquires a significant additional learning time cost. In some cases, the classification also can have a significant time cost for each word. The instance based classifiers such as k-NN classifier [CMBT] determines the k most similar object to the classified instance from an instance database. The distance calculation and the search also can be costly thus the inflection algorithm can be slow.

The rule set determines the behavior of the inflection algorithm so the precision of the algorithm depends on the rule set. During the learning process the rules set is defined as pairs of stem and inflected form. Transformation string can be determined for each word pairs with the Levenshtein distance algorithm [Nav01]. The transformation strings

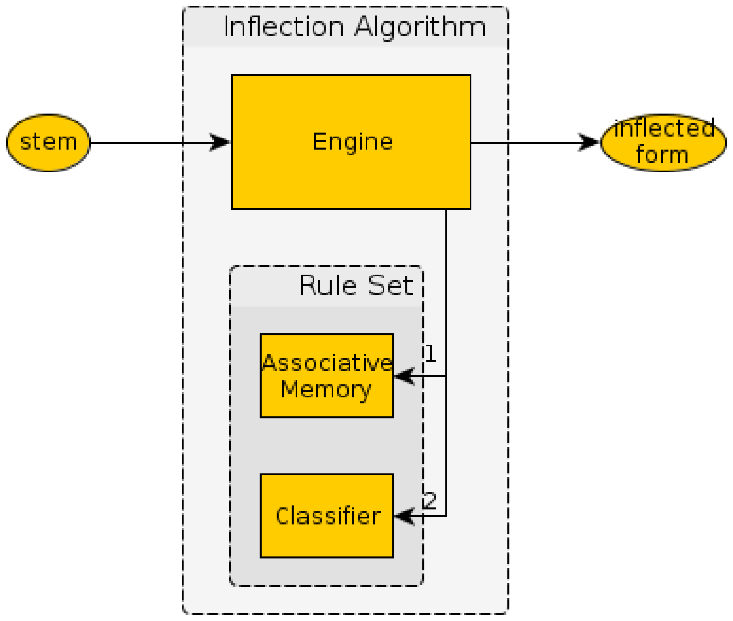


FIGURE 4.7: Model of the presented Inflection Algorithm

are considered as categories of stems. Naive Bayes [JL95] and Multilayer Perceptron [JMM96] classifiers were used to solve this classification problem.

The representation form of the words affects the efficiency of the classification process, too. The words are usually converted into real vectors by mapping the letters into real values. This mapping can be based on code tables such as ASCII or traditional alphabet. These mappings have numerous drawbacks. For example the distance of the letters is constant and the phonetic features of the letters are not considered. A phonetic features based alphabet was presented in [8] for Hungarian. The phonetic alphabet based encoding was shown superior to the traditional alphabet and the ASCII code table based encodings. Phonetic and traditional alphabet based encoding were used during the tests.

The presented inflection algorithm uses both classifier and an associative memory to learn inflection rules. Regular words are classified by the classifier and the irregular words are stored in the associative memory. The size of the associative memory is a parameter of the method. The algorithm looks for the word in the associative memory. If the word is not found, then the transformation string is determined by the classifier.

The learning phase has two main steps. In the first step the categories are ordered by their size. Then the associative memory is populated with the irregular words. If a word is put into the associative memory, then it is also removed from the training set. The population of the associative memory is based on a greedy approach i.e. if the size of the associative memory is bigger than the training set, then the training set is put into the associative memory. Then in the second step, the classifier is built based on the rest of the training set. This training depends on the chosen classification method.

The presented method is evaluated from the point of the view of precision, size and time cost. The precision is the ratio of the correctly inflected words and the total number of the words. The size of the rule set is measured with the length of the serialized classifier structure. Finally the time cost is measured as the required time for learning in milliseconds. The method was evaluated with alphabetical and phonetic alphabet based letter encodings and Naive Bayes and Multi-Layer Perceptron classifiers and different sizes for the associative memory.

4.5 Experimental Results

The experimental measurements were implemented on a training set of 54,000 pairs (stem, inflected word) of the accusative case of Hungarian. The inflection algorithm was implemented as a module of the META framework in Java. Alphabetical and phonetic encodings were used in the tests. The Weka data mining and machine learning framework was used for classification. Naive Bayes and Multilayer Perceptron classifiers were used to learn the inflection rules. In the learning phase, the 75, 90 and 100 percent of the training set was used to train the algorithm. But the entire training set was used during the testing. Thus the behavior of the algorithm with untrained input can be examined. The measurements were done with both fixed and relative associative memory sizes. The fix measurements were done with small associative memory sizes because it was assumed that there are only a few irregular words. The relative sizes were set to every 10 per cent of the size of the training set.

Hungarian is an agglutinative language where are more than 17 cases of words [Mor03]. Only the inflection of the accusative case of the singular nouns was used in the tests. On the other hand inflection can be also applied on an inflected word with some restrictions thus the grammar allows chain of inflections.

4.5.1 Precision

Figure 4.8 and Figure 4.9 show how the precision changes in the function of the size of the associative memory with alphabetical and phonetic alphabet based encodings with Naive Bayes classifier. The X axis shows the size of the associative memory and the Y axis shows the precision. If the size of the associative memory is zero then the algorithm uses only the classifier to determine the transformation strings. In this case the precision of the algorithm is equivalent with the precision of the classifier. The precision of the algorithm increases with the size of the associative memory. The precision reaches the top around 20.000 which is approximately the 40 per cent of the whole training set. The precision has a break down when the entire training set fits into the associative memory. It can be explained with that in this case the algorithm do not use the classifier so that it cannot generalize. In these case the precision is the ratio of the training set and the validating set.

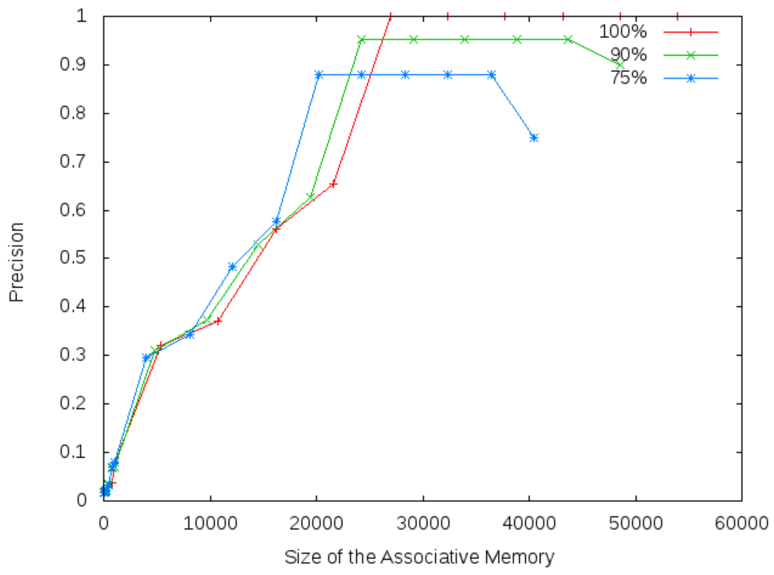


FIGURE 4.8: Precision with Naive Bayes Classifier and Alphabetical Encoding

The precision increases more quickly with the reduced training sets in the cases of both encodings. Although the precision is peak at the same level with both alphabetical and phonetic encodings, the precision increases more quickly with phonetic encoding. This phenomena can be explained with the learning algorithm of the classifier. Because the irregular cases are placed in the associative memory, the number of the categories is reduced. If a category, which is not linear separable from other clusters, is put into the associative memory, then the number of the linear non separable cluster pairs decreases. Hence the usage of the associative memory can reduce the number of the linear non separable cluster pairs. This reduction could yield the increase of the precision of the inflection algorithm.

Multilayer Perceptron shows better precision than Naive Bayes classifier even without associative memory. In this case the precision increases steadily and tops around 20.000 similar to the Naive Bayes classifier. Figure 4.10 shows how the precision depends on the size of the associative memory in the case of the Multilayer Perceptron with alphabetical encoding. Figure 4.11 shows how the precision changes with the size of the associative

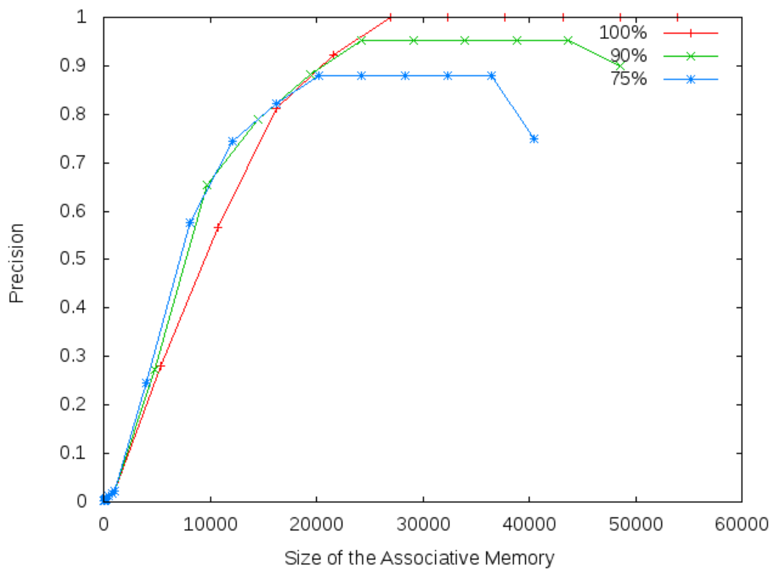


FIGURE 4.9: Precision with Naive Bayes Classifier and Phonetic Encoding

memory in the case of phonetic encoding. In this case, regarding the Multilayer Perceptron classifier there is no significant difference between the two encoding unlike with the Naive Bayes classifier.

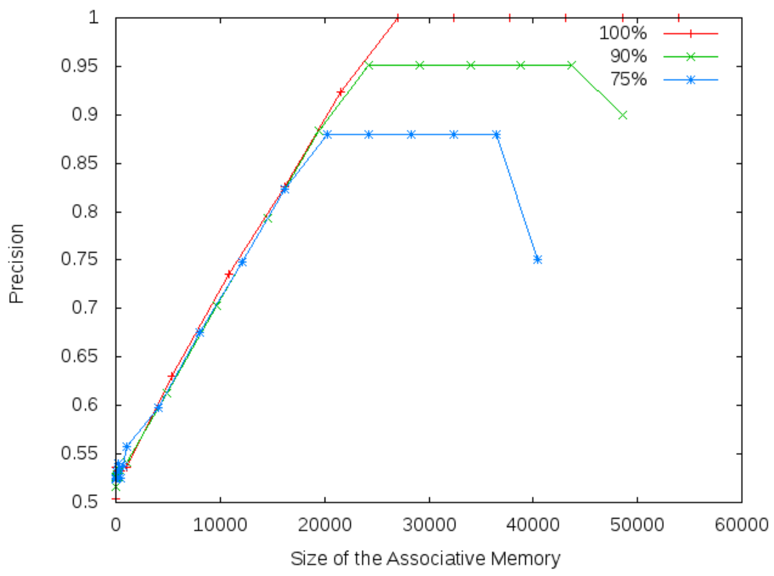


FIGURE 4.10: Precision with Multilayer Perceptron Classifier and Alphabetical Encoding

4.5.2 Size of the data structure

The size of the learning algorithm was measured as the size of the serialized classification structure in bytes. The serialization was possible with the Java API because the Classifier class of Weka implements the `Serializable` interface. Because the Naive Bayes and the

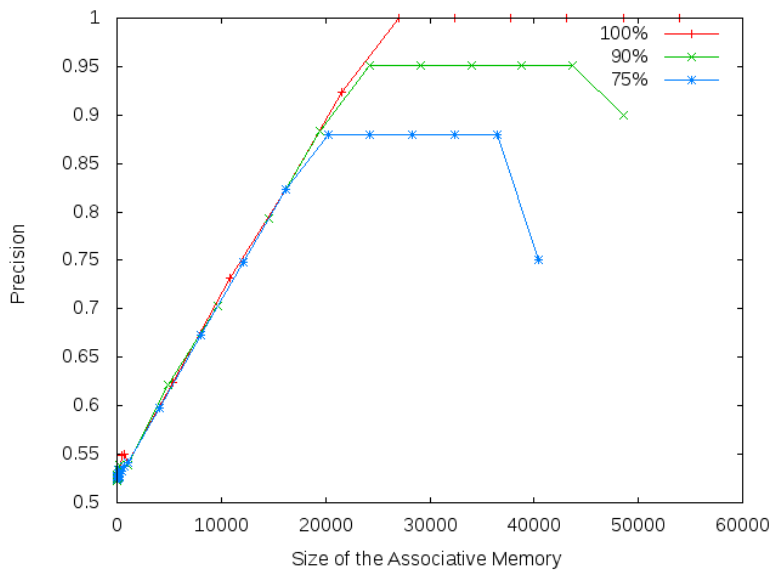


FIGURE 4.11: Precision with Multilayer Perceptron Classifier and Phonetic Encoding

Multilayer Perceptron classifiers had similar behavior with both encodings. Figure 4.12 shows how the size of the algorithm depends on the size of the associative memory. The measurements showed that the size of the classifier has no significant effect on the size of the inflecting algorithm. The X axis shows the size of the associative memory and the Y axis shows the size of the serialized object of the algorithm in bytes.

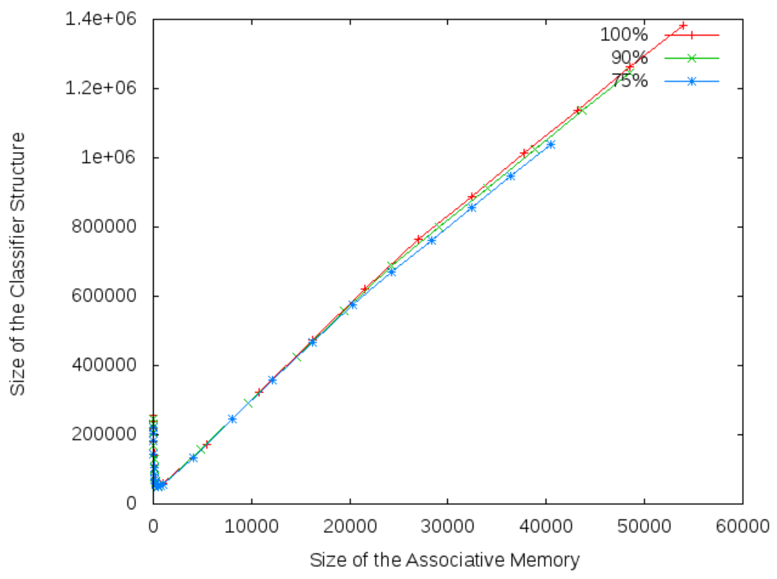


FIGURE 4.12: Size of the data structure for the inflection algorithm

The size of the algorithm decreases quickly until approximately 1000 then it starts to increase steadily. The size increases linearly because the size of the associative memory also increases linearly. So linear connection between the size of the algorithm and the size of the associative memory can be assumed. Figure 4.13 shows how the size of the algorithm changes between 0 and 3000. It shows that the size drops until about 200 then it has a minimum between 500 and 1000. Finally it starts to increase. The simplification of the classification method can be the reason of this fall. Because the least frequent category is put into the associative memory first, many of the categories can be eliminated from the training set even with a small associative memory. This elimination yields a simplified classifier which requires less memory.

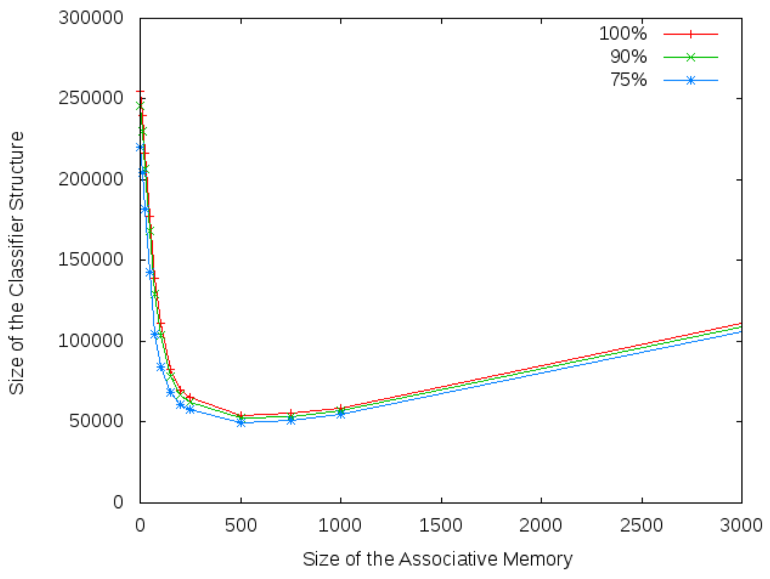


FIGURE 4.13: Size of the data structure for the inflection algorithm

4.5.3 Time Cost

The time cost of the algorithm is the time which is require for training. Although this cost occurs only in the learning phase it makes the tuning of the algorithm slower. Moreover the learning cost can limit the applicability of the algorithm if it is too high. For example if the time cost would grow exponential with the number of samples then it could be applied with only small training sets.

The measurements showed that the learning cost of the algorithm significantly depends on the classifier. Figure 4.14 shows how the learning cost decreases with the increase the size of the associative memory. It also shows a short transient phase up to 3.000. Then it decreases steadily and there is a fall around 20.000.

Figure 4.15 shows the time cost in the case of the Multilayer Perceptron classifier. Although the training cost of the neural network is much more higher than the Bayesian classifier the time cost function is similar. The time cost can be reduced with the application of a small associative memory. Then it decreases slowly and it also has a break down around 20.000.

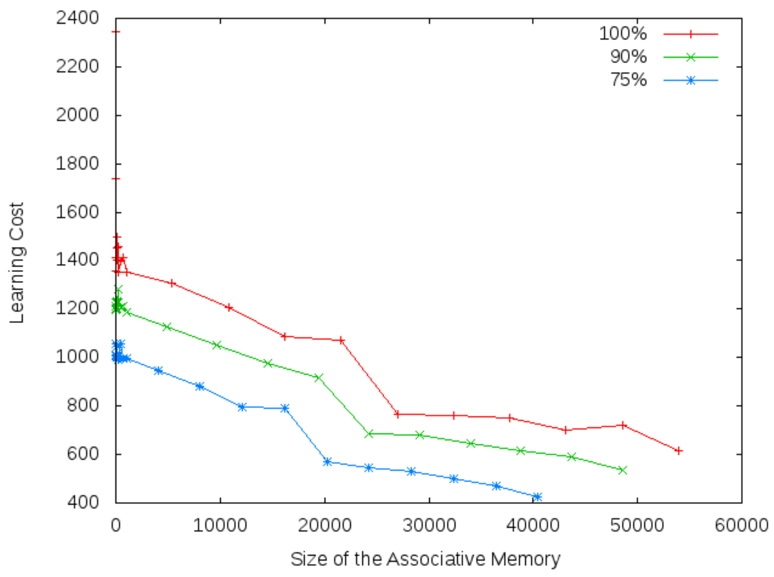


FIGURE 4.14: Time Cost with Naive Bayes Classifier

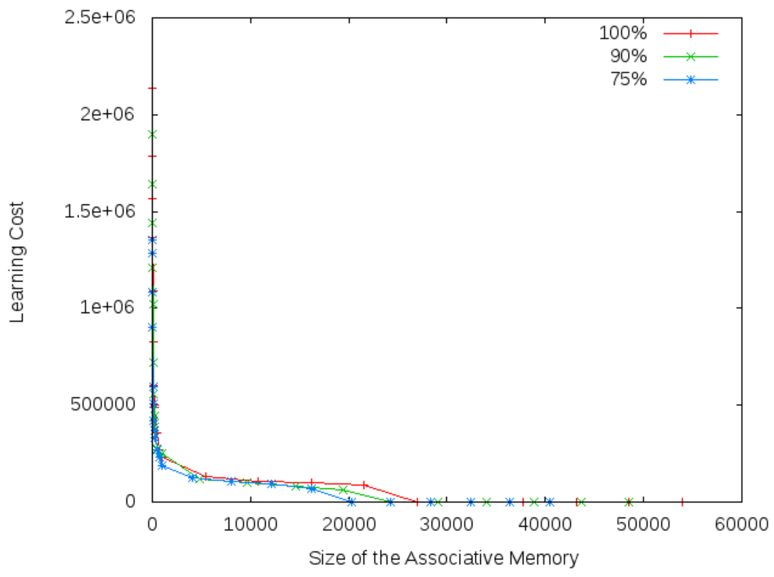


FIGURE 4.15: Time Cost with Multilayer Perceptron Classifier

The measurements showed that the precision of the classification based inflection algorithms can be increased with the usage of associative memory. But above a certain size of associative memory the precision will not increase. Moreover if the associative memory is too big then the precision of the algorithm can decrease.

Measurements showed that the size of the algorithm depends on the size of the associative memory. Above a certain size of associative memory there is a linear connection between the algorithm and the associative memory. The learning cost depends on the classification method. However the different classification methods had different learning cost the time cost decreased similar with both classifiers

4.6 Evaluation

The proposed AMC method has been compared with the standard classification methods from the point of view of precision, learning cost and size of the classification structure. The algorithms were implemented in Java with Weka framework. The measurements were preformed with a training set of 54.0000 samples which contains stem and accusative case pairs.

Table 4.3 shows the performance of the AMC and the standard methods. The experimental results of the proposed AMC method is shown with different size of the associative memory. The size of the associative memory is given as a percentage of the training set. Due to the big differences the learning cost and the size are given in standard form. The precision and the size of the classification structure increase with the size of the associative memory. On the other hand the learning cost decreases.

TABLE 4.3: Comparison of Standard and AMC Methods

Method	Precision (%)	Learning Cost (ms)	Size (byte)
Standard			
Naive Bayes	11.3%	$5.5 * 10^2$	$2.5 * 10^5$
K*	74.45%	$1.6 * 10^2$	$1.1 * 10^7$
MLP	68.45%	$4.1 * 10^6$	$1.9 * 10^5$
AMC (10%)			
Naive Bayes	28.00%	$1.3 * 10^3$	$1.7 * 10^5$
MLP	62.34%	$1.3 * 10^5$	$1.8 * 10^5$
AMC (20%)			
Naive Bayes	56.60%	$1.2 * 10^3$	$3.2 * 10^5$
MLP	73.20%	$1.0 * 10^5$	$3.3 * 10^5$
AMC (40%)			
Naive Bayes	92.16%	$1.0 * 10^3$	$6.2 * 10^5$
MLP	92.32%	$6.9 * 10^4$	$6.3 * 10^5$
AMC (90%)			
Naive Bayes	100.0%	$6.2 * 10^2$	$1.2 * 10^6$
MLP	100.0%	$7.3 * 10^2$	$1.2 * 10^6$
AMC (100%)			
Naive Bayes	100.0%	$6.1 * 10^2$	$1.3 * 10^6$
MLP	100.0%	$6.4 * 10^2$	$1.4 * 10^6$

Table 4.4 shows how the precision changes with the relative size of the associative memory. Without associative memory the performance of the algorithm is determined by the precision of the applied classification method. Naive Bayes and Multilayer Perceptron classifiers were used during the test. The K* algorithm was omitted due to the huge size of the classification structure. Each classifier was trained with 75%, 90% and 100% of the training set and was validated with the entire training set. It can be seen that if

the algorithm uses only associative memory, than the precision of the algorithm is equal with the ratio of the trained samples and training set.

Table 4.4: Precision of AMC Algorithm

Classifier	Naive Bayes			MLP		
Associative Memory	75%	90%	100%	75%	90%	100%
0%	0.29%	0.029%	0.31%	52.60%	52.51%	52.32%
10%	24.64%	27.41%	28.00%	59.82%	62.18%	62.34%
20%	57.76%	65.37%	56.60%	67.32%	70.32%	73.20%
30%	74.42%	78.95%	81.25%	74.82%	79.32%	82.32%
40%	82.32%	88.32%	92.16%	82.32%	88.32%	92.32%
50%	88.02%	95.17%	100%	88.03%	95.17%	100%
60%	88.02%	95.17%	100%	88.03%	95.17%	100%
70%	88.02%	95.17%	100%	88.03%	95.17%	100%
80%	88.02%	95.17%	100%	88.03%	95.17%	100%
90%	88.02%	95.17%	100%	88.03%	95.17%	100%
100%	75.00%	90.00%	100%	75.00%	90.00%	100%

The words had same weight during the testing. But the different words have different frequency. Moreover the frequent words are often irregular [Pin91]. Thus the word frequency could be used to wight the words during the testing and it could give a more accurate results. Because the AMC method puts the irregular words into the associative memory thus they have high precision. This testing requires further works.

Based on the results the proposed AMC methods is superior to the standard classification based methods. The size of the associative memory allows the tuning of the algorithm. High precision could have been achieved with an associative memory which size is about 40% of the training set. On the other hand the algorithm can lose its generalization ability, if the size of the associative memory is too big. Another advantage of the usage of the associative memory is the drop of the learning cost which allows the application of the method with huge training sets.

4.7 Conclusion

A novel inflection algorithm has been presented which uses classification method enhanced with associative memory. The method uses associative memory to learn the irregular words and the exceptions. The classifier is used to capture the regular transformation rules. The algorithm looks for the rules first in the associative memory. If it does not find the rule, then the classifier is used to determine the transformation rule. These two phases allow to achieve high precision and compact size.

The experimental measurements were focused on the precision, the size of the data structure of the algorithm and the learning cost. Results showed that precision increases fast with the size of the associative memory. The maximum precision was achieved with an approximately 20.000 sized associative memory, for a training set of 54.000 samples. Phonetic alphabet based encoding showed better results with Naive Bayes classifier and the encoding had no significant effect on the Multilayer Perceptron classifier. Measurements on the size of the data structure of the algorithm showed that the size grows linearly with the size of the associative memory and the classifier do not modify the size significantly. Experimental results show that the learning cost of the algorithm depends on classifier. The learning cost decreased similarly with both tested classifiers although the order of the cost function was different.

The presented inflection algorithm is capable to handle irregular words and to determine the transformation string for the regular ones with high precision. The algorithm could achieve approximately 90 per cent precision with incomplete training sets. The presented AMC method was compared with standard classification methods and has been showed superior. The size of the associative memory is a vital parameter of the method. The proper chose of this parameter requires tuning or examination of the training set.

Thesis 2.

I have presented a classification based inflection method enhanced with associative memory. The algorithm has been shown superior to the standard classification based inflection algorithms from the point of view of precision.

Related Publications: [10], [11], [12]

Chapter 5

META Framework

Rewriting systems is a mathematical formalism used to describe string transformations. There are many different rewriting systems in the literature such as L–systems [RS80], Markov algorithm and formal grammars [Kor08]. In Natural Language Processing tasks formal grammars are often used to model the grammatical structures of sentences so there are many different formal grammar processing algorithm in the literature [DFG11]. However these algorithms are well documented, there is a lack of their comparison and there are implemented in various programming languages on different platforms. The META framework aims to provide the environment to implement and test grammar and text processing algorithms. Appendix B details how the META framework and its documentation can be obtained.

5.1 Data Mining and Text Processing Frameworks

Data mining (DM) is a process to extract hidden, non-trivial knowledge from huge data sets. One of its special part is text mining (TM) which processes and analyzes unstructured text document sets. Natural language processing is closely related to text mining because it can increase the performance of text mining methods, but natural language processing is more general. Formal grammars are used in natural language processing to describe and represent the grammatical structure of natural or artificial languages. The formal grammars can be built manually which is a costly and slow process or automatically from a training set or samples which called grammar induction however it is an $\mathcal{NP}$–hard task.

However grammar induction and text mining are popular and actively researched areas of computer science, there are only a few framework for these purposes. Weka is one of the most famous of data mining and machine learning framework [HDW94, HFH+09]. For educational purpose, Zeph Grundchlag implemented a few demo application to demonstrate CFG processing algorithms at the University of Columbia. And OASIS has a standard for unstructured document processing called Unstructured Information Management Applications which is implemented by Apache and IBM too. There are many natural language processing frameworks such as GATE [CMBT], TectoMT [PŽ10] or Zemberek [AA07].

5.1.1 Weka

Weka is a data mining and machine learning framework [HDW94, HFH+09, WFT+99, FHH+05] implemented in Java and it has some text mining methods, utilities. It can be used as a desktop application with its own Graphical Use Interface or as a class library for application development. It has tools for preprocessing, classification, clustering, regression, visualization, etc. for data mining and has some text processing utilities for example stemmers and tokenizers. Text mining and data mining have some common methods and processes like classification but the usage of these methods for text mining and data mining differ from each other.

Weka is popular because of its extensibility, GUI and it is well documented and also has a lots of applications and references. It orders the different methods into a package hierarchy in Java, so it has a logical set-up and it is easy to extend.

5.1.2 RapidMiner

RapidMiner [Mie13, Min14] is also an open source data mining software. It is a desktop application and has a well structured, nice GUI and a bunch of methods for different purposes. It can extract data from various sources called repositories for example Excel files or databases. In the design view, the user can determine the analysis process by means of a directed graph. The roots of this graph are the selected repositories, the other nodes are transformations and the terminations are the different results.

5.1.3 cfgrep and JavaCFG

There are some educational applications for formal grammar processing and Context–Free Grammars induction. One of these is made by Zeph Grunschlag at University of Columbia [Gru06]. He wrote two educational software for Context–Free Grammar processing and representation.

The first one is called `cfgrep`, it is a simple Java application to parse sentences based on a Context–Free Grammar rule set. It has to optional flags and requires two parameters. The first parameter describes the grammar the other is a file path which contains the sentences. The given result depends on the flags.

The `JavaCFG` is a parse tree generator for CFGs based on the Early's parsing algorithm. It is also console based application, but it visualizes the result in a `JTree` object moreover it generates a TeX file which contains the parse tree too.

5.1.4 UIMA Standards

UIMA stands for Unstructured Information Management Applications, which project is managed by IBM and Apache [Apa14], and it has Java and C++ implementations and it is based on OASIS standard [OAS14]. Watson, the computer won Jeopardy, uses UIMA. Its goal is to provide a framework to develop platform-independent reusable (distributed) analysis modules which can process different unstructured data sources.

The Apache UIMA framework's basic components called `Analysis Engines` are composed to process and extract knowledge from different unstructured data sources.

5.1.5 GATE Framework

GATE [CMBT] is a component based framework. Although the components can be implemented in various programming languages, but there must be a Java class to represent the component. It provides both an API and a graphical user interface for developers. The framework distinguishes three types of resources> *language*, *processing* and *visual*. The resources can be stored in database or serialized into files in binary or XML formats. GATE also support various data formats such as XML, RTF, HTML. It uses Unicode characters so multiple languages can processed by the framework. The *processing* resources are reusable algorithms and methods. GATE provides many natural language processing methods such as tokeniser, sentence splitter, semantic tagger or co–referencer.

5.1.6 TectoMT

TectoMT [PŽ10] is a multi–purpose open source natural language processing framework. It provides various functionalities such as sentence segmentation, tokenization, syntax parsing and so on. It is used in a English–Czech machine translation system and it provides modules for other languages like German, Russian or Arabic. The different algorithms, methods and functions are represented by blocks in TectoMT where each block has a well–defined input and output. Hence TectoMT provides the modularity on the level of blocks and it includes over 400 block [PŽ10]. The language processing tasks are built from blocks and they are called scenarios. A language has four different layers of description: *word*, *morphological*, *shallow–syntax* and *deep–syntax*. The documents are stored in files and they store a sequence of sentences. Each sentence is represented by a tree which is called bundle. TectoMT is implemented in Perl on Linux operation system

TectoMT The tasks are represented in TectoMT as sequence of block where TectoMT provides reusable modules which are

5.1.7 Zemberek Framework

Zemberek framework [AA07] is an open source, extensible, general purpose Natural Language Processing Library for Turkic languages. Although the Turkic languages have own alphabets and letters, but the framework supports only ASCII characters. Multiple Turkic languages can be handled by the framework, but the language specific aspects have to be implemented. The language implementations consist of various settings files sud as alphabet file, suffix file (XML) or root word dictionary. Zemberek framework also provides basic natural language processing tools like morphological parsers or spell checkers. The framework is implemented in Java and it is used in real world application like OpenOffice or Turkish Linux Distribution Pardus.

5.2 Purpose of the Framework

META aims to give a well defined, extensible grammar induction and text mining framework which can be applied both in research and education. META provides some interface for programmers to extends the framework with their own algorithms, modules and process document sets. Analyzers are designed to use the implemented algorithm to analyze document sets and visualize the given results by available modules. Thus META has to be general and it has to provide necessary classes and interfaces to training set handling, data preprocessing, machine learning, analyzing and visualization. Other important goal is to be well documented because documentation helps the understanding and usage of the framework.

META was used to test and analyze different Context–Free Grammar induction methods such as TBL[SK99, Sak05], ITBL[UJ07, OU09] and inductive CYK[NI00, NM02]. META provides a good way to analyze these algorithms in a common environment. Moreover, META defines some interfaces for educational purposes like stemming, tokening, and so on.

Table 5.1 summarizes some main characteristics of the above reviewed frameworks and applications. The developed META framework is also shown in the framework. The first column shows the framework. The main purpose of the framework is in the 2nd column. The next column shows how the framework supports natural language processing tasks. The 4th column shows whether the framework support multiple languages or not. The 5th column stands for the formal grammar modeling functions. Finally the last column shows that the framework in which language can be extended, programmed.

Table 5.1: Comparision of Frameworks

Framework	Purpose	NLP	Multi–Lang.	FG	Prog. Lang.
Weka	DM, ML	limited			Java
Rapid Miner	DM	limited			Visual
cfgrep, JavaCFG	Education			X	Java
UIMA	Standard	X	X		C++, Java
GATE	NLP	X	X		Java, any
TectoMT	NLP, Translation	X	X		Perl
Zemberek	NLP, Turkic	X	Turkic		Java
META	NLP, FG	X	X	X	Java

5.3 Components of META

META is designed to provide a method collection for grammar induction and text mining [2, 5] and to provide some interfaces for programmers and developers to implement own methods. Both grammar induction and text mining have a bunch of methods and algorithms so META categorizes these algorithms into modules. The hierarchies and dependencies of these modules defines the structure of the framework. Figure 5.1 shows the basic logical architecture of META.

Grammar processing and text mining functions are distinguished in the architecture. The common classes are in the `Core` module. The `Filter` and `Analyzer` modules are related to text mining. They contain only interfaces for the different processing tasks, but these functions are not implemented yet. This part of the framework can be used for educational purposes.

Grammar processing related functions are placed into the `Grammar` module. It was used during my researches to implement and compare different grammar induction methods. META defines the main classes to model formal grammars. However it was used to handle

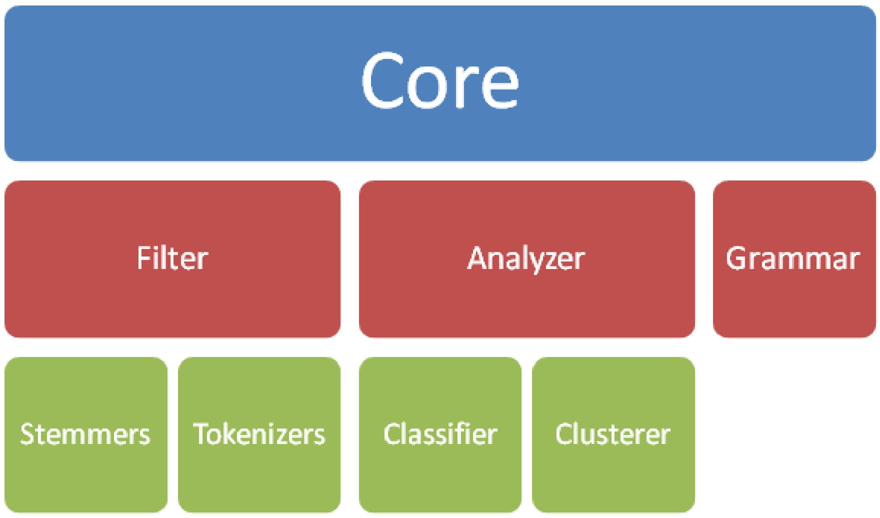


FIGURE 5.1: Architecture of META framework

context–free grammars, it uses a general representation of formal grammars. Moreover it provides a way to handle probabilistic grammars too.

The framework is implemented in Java as a Maven project which has `framework` and `grammar.processing` Maven modules. The modules of META are realized as packages in Java. The interfaces and common classes are defined in the `framework` Maven module. The concrete, implemented algorithms are placed into the `grammar.processing` Maven module. Thus the framework is separated from the implementation of the algorithms. Moreover the dependency handling was solved by Maven too so there is no need to copy 3rd party libraries into the project because they are contained in the Maven central repository.

5.3.1 `core` module

Common classes of META framework are implemented in `core` package. These classes represent the different data sources, document sets and training sets. Some functions are organized into other packages under the `core` package such as `core.linsep` or `core.markov` packages. The `core.linsep` package contains the interfaces for linear separability testing methods. There is an implementation of Markov Algorithm, which is a rewriting system, in the `core.markov` package.

Figure 5.2 shows some main classes of the `core` package of META framework. The framework collects methods for computing unstructured data sets. This package contains the representation of different data sources for example various text documents such as emails, HTML pages, word docs or different voice documents like speech, song, or images. The current version of META framework supports only text based documents.

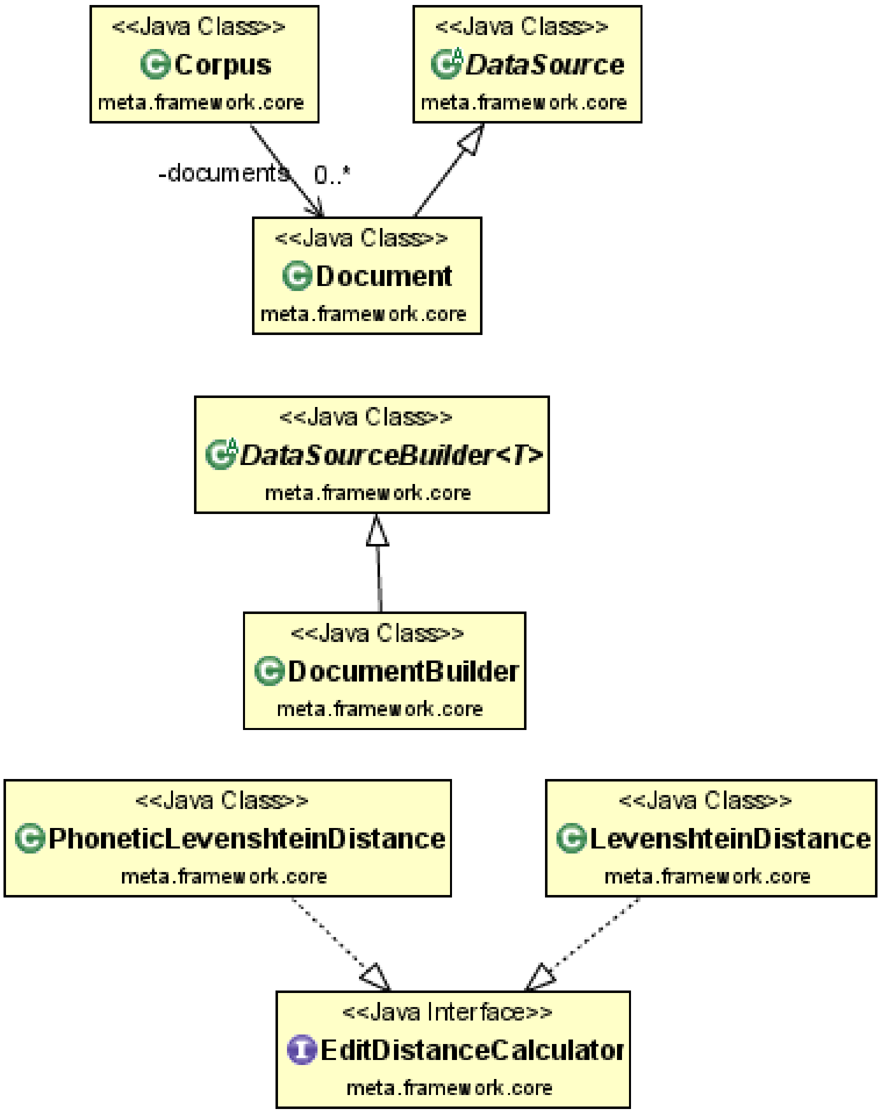


FIGURE 5.2: core package of META framework

The `DataSource` class is an abstract superclass for each unstructured document class. It has an URL attribute which identifies the document, and a map to contain the meta data of the data source. The most common keys for the map of meta data are contained in the `MetaData` class. `Document` class is inherited from `DataSource`, and it represents an unstructured text document and it has a list of string which represents the text of the document. The `DateSource` and its descendants has no public constructor, because they are built by the corresponding builder. The builders can be structured into a hierarchy similar to the hierarchy of `DataSource` objects. The `Corpus` is a collection of `Document` objects. Users can add, remove or get documents to the corpus and corpus also could be processed by different filters and analyzers.

The `core` module also contains implementation of common subtasks like calculation of edit distance of string. The `EditDistanceCalculator` defines an interface for the concrete algorithms. There are two implementation of this interface in the `core` module which are the `LevenshteinDistance` and the `PhoneticLevenshteinDistance` algorithms. `LevenshteinDistance` class implements the Levenshtein algorithm [Nav01] to calculate the edit distance of two strings and insertion, deletion and substitution costs can be set by the constructor. Based on the phonetic alphabet, a modified Levenshtein algorithm was implemented in `PhoneticLevenshteinDistance` class. It uses a phonetic alphabet which determines the insertion, deletion and substitution costs.

5.3.2 `filter` and `analyzer` modules

Interfaces for different text filter methods, such as stemmers and tokenizers, are contained in this package. The filters can modify the `DataSource` object in a well defined way, and they must implement the `Filter` interface. `Filter` interface has no method but it is required for filter classes to be able to modify the given `DataSource` object. The current version of the framework supports only the processing of text based documents, so it has only a `text` package and this package contains the `Stemmer` and `Tokenizer` super classes.

The `analyze` package contains super classes for analyzer objects. The classifiers and clustering method returns a list of set of `Document` objects. Programmers should derive their methods from the corresponding superclass. The current version of the framework contains only the specification of analyzer methods and classes.

5.3.3 `grammar` module

The `grammar` module contains classes to model and handle formal grammars. The building elements like terminal, non–terminal symbols, production rules and probabilistic rules are represented as classes. There are interfaces defined in this package for parsing and induction methods. The formal grammars are general so any class of Chomsky hierarchy [Cho56, Cho59] can be handled by this grammar. The `Grammar` class represent a formal grammar. XML serialization of `Grammar` objects is realized by JAXB 3rd party library.

5.3.3.1 Symbol Representation

Formal grammars are built from terminal, and non–terminal symbols. These symbols are distinguished by META Framework. The `Terminal` class represent a terminal, the `NonTermina` represents a non–terminal symbols. The `Symbol` is an abstract class which was created in order to handle the common aspects of `Terminal` and `NonTerminal` classes.

The Symbol class implements the Clonable interface because it is required to create copy of Grammar objects. Figure 5.3 shows these classes on a class diagram.

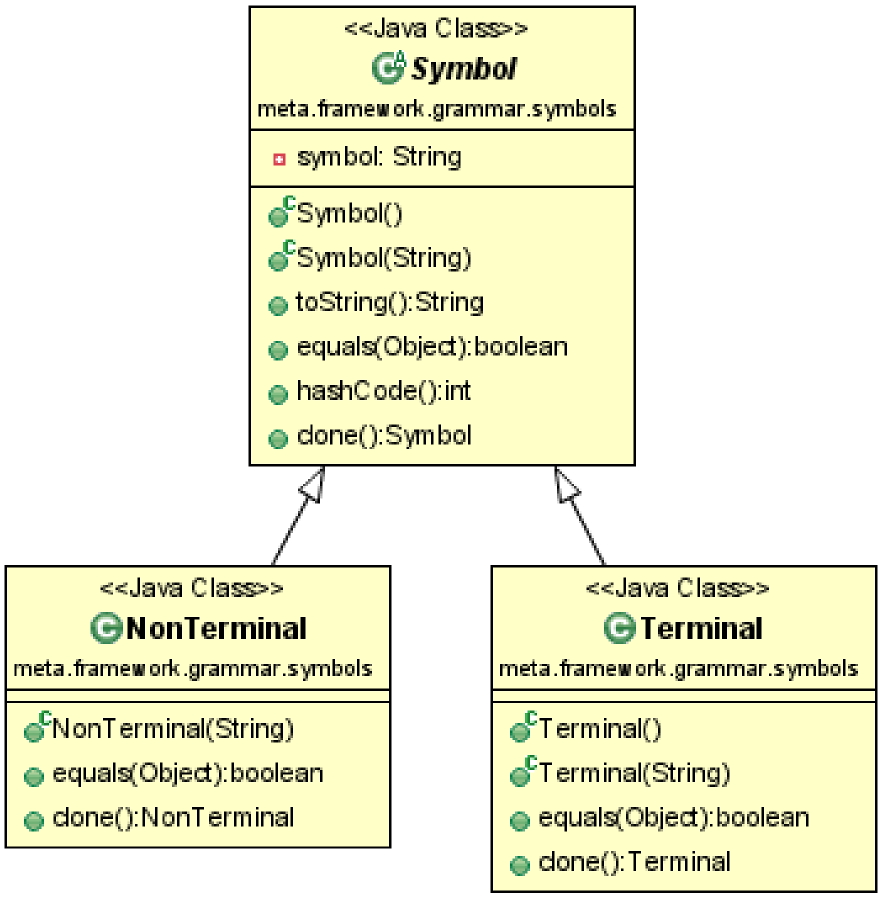


FIGURE 5.3: Hierarchy of Symbol classes.

5.3.3.2 Rule Representation

Probabilistic and deterministic formal grammars can be distinguished in META framework. Grammatical rules describe a substitution by defines which symbol sequence can be replaced by an another one. The first symbol sequence is called the head or left side of the rule, and the second is the tail or right side. The head is common in both production rules and probabilistic rules. There are only one tail for a head in the case of production rules. Probabilistic rules allows more tails for a head and each tail has a probability of occurrence.

Figure 5.4 shows the class hierarchy of grammatical rules and their connection to the Grammar class. The GrammaticalRule class is an abstract super class which represents both production and probabilistic rules and it defines that, each rule has to have a head. The ProductionRule represents production rules and it has only one tail. Probabilistic

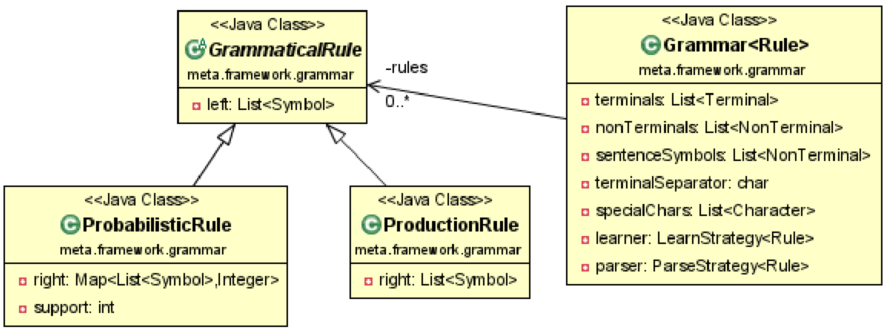


FIGURE 5.4: Class hierarchy of `GrammaticalRule` classes.

rules are represented by `ProbabilisticRule` class which uses collections to handle multiple tails for a head. This class hierarchy makes it possible to distinguish the different rules and to handle them on a common way in the `Grammar` class.

Formal grammars are represented by the `Grammar` class. It has attributes to store `Terminal`, `NonTerminal`, `GrammaticalRule` objects. It is a generic class and its generic type has to be a `GrammaticalRule` so the `Grammar` class can handle `ProductionRule` or `ProbabilisticRule` objects. If a `Grammar` object is instantiated with a `ProductionRule` parameter, then it can handle only `ProductionRule` rules.

5.3.3.3 Parsing and Learning of Grammars

META Framework is easy to extend with new grammar processing algorithms because it uses the strategy pattern [GHJV94] to change the behavior of the class. There are `ParseStrategy` and `LearnStrategy` interfaces defined for parser and induction algorithms. The `Grammar` class can use the implementation of these interfaces. The `ParseStrategy` and `LearnStrategy` interfaces are also generic and their generic type has to match with the generic type of the `Grammar` object. Thus if a grammar has `ProductionRule` rules, then the processing method has to work with the same kind of rules. The mixing of the types yields error during the development and the compilation.

The concrete processing methods can be set anytime during the life cycle of a `Grammar` object. There are no default processing algorithms defined so they are initialized with `null`. So `StrategyNotSpecifiedException` is defined which is thrown by `parse` and `learn` methods of a `Grammar` object, if the given strategy is not set.

META framework can be extended with parsing and induction algorithms. This algorithms are implemented in derived classes of `ParseStrategy` or `LearnStrategy` classes. Figure 5.5 shows the connection of the `Grammar` class with the strategies and some implementation. The implemented algorithm are not generic unlike their interfaces. Because

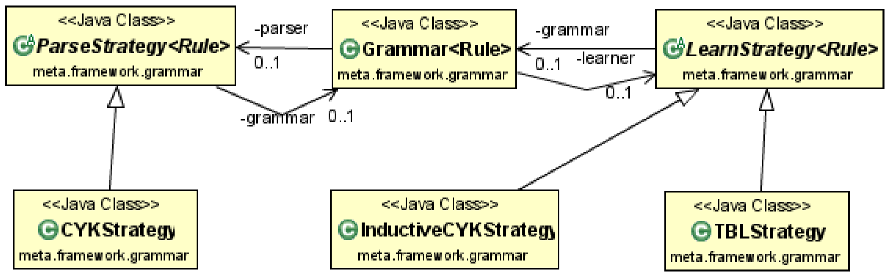


Figure 5.5: Parse and Learn Strategies

the derived class has to define the generic type in its definition and an algorithm works with production rules or probabilistic rules.

5.3.4 Context–Free Grammar Extension

Grammar induction of context free languages is an actively researched area of computer science. There are some algorithm, based on different approaches, to induce context–free grammars. Some well–known Context–Free Grammar processing methods have been implemented, tested and compared in the META Framework. These methods are placed in the `grammar.processing` Maven module, so they are separated from the framework but it depends on it.

There is no extra class for CFG representation and this extension contains only methods for CFG parsing and learning methods [7]. These methods are subclasses of `ParseStrategy` or `LearningStrategy` classes so they can be set as a corresponding strategy to a common `Grammar` object. We have implemented a parsing method, (`CYKStrategy`), and three learning methods and their modifications (`TBLStrategy`, `ITBLStrategy` and `ICYKStrategy`).

5.4 Implemented Methods

The `grammar.processing` Maven module contains the implementation of some well–known context–free grammar processing algorithms. It uses the `framework` Maven module. For parsing, the CYK parser has been implemented. Induction algorithms such as the TBL[SK99, Sak05], the ITBL[UJ07, OU09], the inductive CYK[NI00, NM02] algorithm, were implemented and compared. These algorithms were chosen because they are well–well known and well–documented.

5.4.1 Parsers

Parser algorithms are constructed to solve the membership problem of formal grammars, in other words they are means to decide that a given sentence ω is element of the language $\mathcal{L}_\mathcal{G}$ generated by grammar $\mathcal{G}$ or not. Sentence ω is element of language $\mathcal{L}_\mathcal{G}$ if there are at least one chain of production rules which starts with a sentence symbol $S \in \mathcal{S}$ and ends with a sequence of terminal symbols of ω It can be defined with the following mathematical formalism $\omega \in \mathcal{L}_\mathcal{G} \Leftrightarrow \exists S \Longrightarrow^*_\mathcal{G} \omega$. The Early and the CYK parsers are the most well–known and widely used algorithms to solve the membership problem of Context–Free Grammars. The CYK algorithm has been implemented in the META framework.

5.4.1.1 CYK Algorithm

CYK parser is a bottom-up parsing algorithm to solve the membership problem of CFGs in $O(n^3)$ time. The CYK algorithm works on grammars in Chomkian Normal Form (CNF) and it increases the cost of computation. It uses a table like data structure ($\mathbf{T}$) to determine which non terminal symbol set can generate a given part of the sentence. Data is structured into a lower triangle matrix which cells contain set of non terminal symbols and if $N \in \mathcal{N}$ and $N \in \mathbf{T}_{i,j}$, then $N \Longrightarrow^*_\mathcal{G} \omega_i\omega_{i+1}\ldots\omega_{i+j}$. Figure 5.6 shows the pseudo code of the CYK algorithm. Sentence ω is in the language $\mathcal{L}_\mathcal{G}$ if there is at least one non terminal symbol at the box $\mathbf{T}_{r,0}$.

```
for $c = 0 \rightarrow \mid \omega \mid$ do
    $\mathbf{T}_{0,c} := \{N \mid N \in \mathcal{N} \text{ and } N \rightarrow \omega_c \in \mathcal{P}\}$
end for
for $r = 1 \rightarrow \mid \omega \mid$ do
    for $c = 1 \rightarrow \mid \omega \mid -r$ do
        for $k = 0 \rightarrow r - 1$ do
            $\mathbf{T}_{r,c} := \mathbf{T}_{r,c} \cup \{A \mid A \in \mathcal{N},\ B \in \mathbf{T}_{k,c},\ C \in \mathbf{T}_{r-k-1,c+r-k},\ A \rightarrow BC \in \mathcal{P}\}$
        end for
    end for
end for
```

Figure 5.6: CYK Algorithm

The CYK algorithm is implemented in `CYKStrategy`, and it can be used by its `parse` method. It is invoked by the `Grammar` object `parse` method if its `parser` attribute is a `CYKStrategy` object. During the definition of the class, `ProductionRule` is defined as generic type of the `ParseStrategy` so the implemented algorithm only can work with `Grammar` object with `ProductionRule` generic parameter.

5.4.2 CFG Induction

The induction of Context–Free Grammars is an actively researched area of computer science and there are many different methods for this purpose. The induction methods differ from each other in their approach, applied techniques, performance and computational and time cost. The `grammar.processing` module contains the implementation of some induction methods. These methods are briefly reviewed in the followings.

5.4.2.1 TBL Algorithm

TBL algorithm is based on genetic algorithm [SK99][Sak05]. It requires positive and negative samples to induce Context–Free Grammars. It uses a table like data structure, similar to the data structure of CYK algorithm, to determine each possible unique parse tree for each given sentences. These parse trees determines a so called primitive grammar which exactly contains every possible deduction tree for each positive sentences. Hence the primitive grammar accepts each positive sentences, but it contains about $O(n^5)$ production rules for each positive sentences where n stands for the length of the word. Genetic algorithm is applied to reduce the primitive grammar. The reduction is based on the partitioning of the set of non–terminal symbols. If two non–terminal symbols belongs to the same partition, they are in equivalence relationship so they are interchangeable. Hence the non–terminal symbols can be substituted by one symbol from the partition they belong to. This substitution results the reduction of the non–terminal symbols which yields the merging of production rules too.

TBL algorithm consists of the following three steps:

1. Generation of primitive grammars ($G^{\omega}(T(\omega))$) for each positive sentence ($\omega \in U_+$)
2. Composing the union of primitive grammar ($G(U_+) = \cup G^{\omega}(T(\omega))$)
3. Reducing the result grammar with a genetic algorithm

The essence of TBL algorithm is the applied genetic algorithm which assigns the non–terminal symbols into partitions. It uses group number encoding to represent the solutions. A temporary grammar is created for each candidate solution to evaluate the fitness value. The fitness function is the following with C_1, C_2 parameters:

$$f_1(p) = \frac{\mid \{\omega \in U_+ \mid G(T(U_+))/\pi_p\} \mid}{\mid U_+ \mid} \tag{5.1}$$

$$f_2(p) = \frac{1}{\mid \pi_p \mid} \tag{5.2}$$

$$f(p) = \begin{cases} 0 & \text{if } \exists \omega \in U_- \text{ can be generated by} \\ & \omega \in L(G(T(U_+))/\pi_p) \\ \frac{C_1 f_1(p) + C_2 f_2(p)}{C_1 + C_2} & \text{otherwise} \end{cases} \tag{5.3}$$

Although the TBL algorithm seems to be a good way to induce Context–Free Grammars, its time cost is very high because of the fitness function of the applied genetic algorithm. The candidate solutions which accept any negative sentence are discarded. The determination of a grammar for a solution is a complex and costly process and the calculation of fitness value requires parsing for each positive and negative sentences which has $O(n^2)$ time cost.

The `TBLStrategy` implements this method and it can be invoked via the `Grammar` object `learn` method.

5.4.2.2 ITBL Algorithm

ITBL algorithm[UJ07][OU09] is a modification of base TBL algorithm which has lower time cost than the base TBL algorithm. It differs from the base TBL only the applied genetic algorithm and the fitness function.

The fitness function of ITBL algorithm with C_1, C_2, C_3, C_4 parameters is the following.

$$f_3(p) = \frac{\mid \{\omega \in U_- \mid G(T(U_+))/\pi_p\} \mid}{\mid U_- \mid} \tag{5.4}$$

$$f(p) = \begin{cases} \frac{-(C_3 f_3 + C_4(1 - f_1))}{C_3 + C_4} & \exists \omega \in U_- \text{ can be generated} \\ & \text{by } \omega \in L(G(T(U_+))/\pi_p) \\ \frac{C_1 f_1(p) + C_2 f_2(p)}{C_1 + C_2} & \text{otherwise} \end{cases} \tag{5.5}$$

Besides the modified fitness function, ITBL algorithm uses a new special mutation operations. These operations delete a non terminal symbol or a symbol block with a low probability.

As result of these modifications, the ITBL algorithm makes faster more compact grammars. On the other hand, it needs plenty of time to produce the grammar and its result depends on the parameters of the applied genetic algorithm. Another drawback of TBL and ITBL algorithms is that it can not work iteratively so a learned grammar can not be extended with these methods. From that reason that the users have to complete the training set with the new sentences, then run the method again on the new training set.

5.4.2.3 Heuristical pre-reduction with TBL and ITBL Algorithm

A heuristical pre-reduction [3, 4] method can make faster the TBL and ITBL algorithm. The approach of this method is that the TBL and ITBL methods produce primitive grammars which have $O(n^3)$ non terminal symbols and production rules. We suppose that the most costly part of these methods is the applied genetic algorithm. The time cost of the genetic algorithm depends on the length of chromosome which is given by the number of the non terminal symbols. A pre-reduction method to reduce the number of non terminal symbols was proposed so it makes faster the applied genetic algorithm.

The suggested pre–reduction distinguishes the following two steps:

1. Merge non terminal symbols which point to the same terminal symbol.
2. Merge rules with the same right side.

The order of these steps is important because of the production of primitive grammars. The first step reduce the number of $A \rightarrow a$ type rules (see Figure 5.7). It can perform a great reduction on the set of $A \rightarrow a$ type rules because $\mid \mathcal{T} \mid \leq \sum_{\omega \in TS_+} \mid \omega \mid$ where TS stands for training set. Unfortunately, the set of $A \rightarrow a$ type rules is a small subset of $\mathcal{P}$, hence we proposed a second reduction step (see Figure 5.8.) which merges the $A \rightarrow BC$ type rules. It is a greedy step because it works iteratively until it can not merge any non terminal symbol. It applies inverted rule lists to determine which non terminal symbols can deduce same BC, then it merges into same non terminal symbol.

```
for $r \in \mathcal{R}$ do
    if r like $A \rightarrow a$ then
        r.left := UpperCase(r.right)
    end if
end for
```

FIGURE 5.7: Heuristical pre-reductions first step

The `HTBLStrategy` and `HITBLStrategy` apply this pre-reduction method before the genetic algorithm of TBL or ITBL algorithms. This pre-reduction method makes more faster the above reviewed algorithm but the cost function remains still high.

```
while wasMerge do
    wasMerge := false
    for r in $\mathcal{R}$ do
        iverseMap.put(r.right,r.left)
    end for
    for key in inverseMap do
        if inverseMap.get(key).size $>$ 1 then
            merge(inverseMap.get(key))
            wasMerge := true
        end if
    end for
end while
```

FIGURE 5.8: Heuristical pre-reductions second step

5.4.2.4 Inductive CYK algorithm

The inductive CYK algorithm [NI00, NM02] differs from the previous two methods because it has different approach and works differently. However it uses a similar table like data structure, it does not build primitive grammars. It is based on CYK parsing algorithm. It adds new non–terminal rules and production rules stochastically. It uses backtracking to avoid the learning of negative grammars.

Inductive CYK algorithm learns the sentences separately and works iteratively. For each sentence (ω) it produces the parse table ($\mathbf{T}_\omega$) which is a minimum subset of the grammar ($\mathcal{G}_\omega = \langle \mathcal{T}_\omega, \mathcal{N}_\omega, \mathcal{P}_\omega, \mathcal{S}_\omega \rangle$) which can generate the sentence or parts of it.Because it completes this $\mathcal{G}_\omega$, the modifications remain locally and it also helps to trace back the faults. It stochastically adds non terminal symbols N_i and production rules $r_i = N_i \rightarrow BC$ where $B, C \in \mathcal{N}_\omega$ to the grammar $\mathcal{G}_\omega$. After each modification it tests $\mathcal{G}_\omega$ with all negative sentences. If it accepts at least one negative sentence it undoes the most recently modifications, else it keeps the modifications and tests with ω. It it accepts ω then it finalizes the necessary modifications. Because $\mathcal{G}_\omega \subseteq \mathcal{G}$ these modifications affect on $\mathcal{G}$.

The Figure 5.9 shows a brief pseudo code of inductive CYK algorithm, the base method contains R and R_{max} parameters where $R \leq R_{max}$ to control its work.

The `InductiveCYKStrategy` class implement this algorithm. This `learn` method returns `true` if and only if it learned each positive sentence successfully.

```
$\mathbf{T} := \mathrm{cyk}(\omega)$
**if** $\mathbf{T}_{|\omega|,0} \cap \mathcal{N} \neq \emptyset$ **then**
    **return**
**end if**
**for** $c := 0 \rightarrow \mid \omega \mid$ **do**
    **if** $\mathbf{T}_{0,c} = \emptyset$ **then**
        $\mathcal{N}_\omega := \mathcal{N}_\omega \cup A$
        $\mathcal{R}_\omega := \mathcal{R}_\omega \cup \{A \rightarrow \omega_c\}$
        $\mathbf{T}_{0,c} := \mathbf{T}_{0,c} \cup A$
    **end if**
**end for**
**for** $r := 1 \rightarrow \mid \omega \mid$ **do**
    **for** $c := 0 \rightarrow \mid \omega \mid -r$ **do**
        **if** $\mathbf{T}_{r,c} \equiv \emptyset$ **then**
            **for** $k := 0 \rightarrow r$ **do**
                **for** $B \in \mathbf{T}_{k,c}$ **do**
                    **for** $C \in \mathbf{T}_{r-k,c+k}$ **do**
                        *nondeterministical step*
                        $\mathcal{N}_\omega := \mathcal{N}_\omega \cup A$
                        $\mathcal{R}_\omega := \mathcal{R}_\omega \cup \{A \rightarrow \omega_c\}$
                        $\mathbf{T}_{r,c} := \mathbf{T}_{0,c} \cup A$
                        *check*
                    **end for**
                **end for**
            **end for**
        **end if**
    **end for**
**end for**
```

Figure 5.9: Inductive CYK algorithm

5.5 Experimental Results

The TBL, ITBL, and Inductive CYK algorithms were implemented and tested in the META Framework. The measurements focus on the time cost of these algorithms. There were two training set created for the comparison.

5.5.1 Measurements

There were two training set generated for the testing. The first training set (TS_{abc}) contains words of a language like $\mathcal{L} = \{a^n b^m c^o \mid n, m, o \in \mathbb{N}^+ \cup \{0\}\}$ and the other training set (TS_{SQL}) contains SQL like sentences. The methods were tested with different size of these training sets. It allowed to measure how the algorithms behave with the increase of the number of samples. The roman numbers denote the size of the training sets where the I is the smallest and V is the largest training set. The `TBLStrategy`, `ITBLStrategy` and `InductiveCYKStrategy` classes were used in the META Framework during the tests.

5.5.2 Results of TS_{abc}

The first language contains sentences of arbitrary length of sequence of a, b and c characters and the characters are divided by `space`. Table 5.2 summarizes some parameter of TS_{abc} and the roman numbers denote different subset of training set TS_{abc}, $\mid TS_{abc} \mid$ denotes the count of positive sentences, $\overline{\mid \omega \mid}$ stands for the average length of positive sentence and $\mid \mathcal{T} \mid$ denotes the number of terminal symbols (different word) in the positive sentences. The algorithms were tested with relatively small training sets because of their time complexity.

TABLE 5.2: Characteristics of TS_{abc}

TS_{abc}	**I**	**II**	**III**	**IV**	**V**
$\mid TS_{abc} \mid$	2	4	5	6	7
$\overline{\mid \omega \mid}$	3	3	3	3.5	3.8
$\mid \mathcal{T} \mid$	3	3	3	3	3

Each method were tested three times with each training set and their results were aggregated. The average time costs of these algorithms are summed up in Table 5.3 and visualized in Figure 5.10. In the table, row denoted by TBL contains the results given by `TBLStrategy`, the ITBL refers to `ITBLStrategy` and IndCYK denotes `InductiveCYKStrategy` and the values are in milliseconds. In Figure 5.10, the vertical axis denotes the required average runtime for the algorithms in milliseconds. The line of `ITBLStrategy` overlaps the line of `TBLStrategy` and the x axis overlaps the line of `InductiveCYKStrategy`.

TABLE 5.3: Results on TS_{abc}

TS_{abc}	**I**	**II**	**III**	**IV**	**V**
TBL	737.33	2797.66	5278.33	106281.3	459550.3
ITBL	515.66	2772	5667.66	102435.3	450272
IndCYK	3	5	6.33	33.33	53

5.5.3 Results of TS_{SQL}

The second language contains SQL queries on a scheme with two tables ($t1$ and $t2$) with four-four fields ($f1$-$f4$ and $f5$-$f8$). Table 5.4 sums up the characteristics of training set TS_{SQL} where the different subsets are denoted by roman numbers and $\mid TS_{SQL} \mid$ stands for the number of positive sentences, $\overline{\mid \omega \mid}$ stands for the average length of positive sentences and $\mid \mathcal{T} \mid$ denotes the number of terminal symbols.

We have tested the algorithms three times with these training sets and analyzed the given results. The average runtime in milliseconds are summed up in Table 5.5. The results

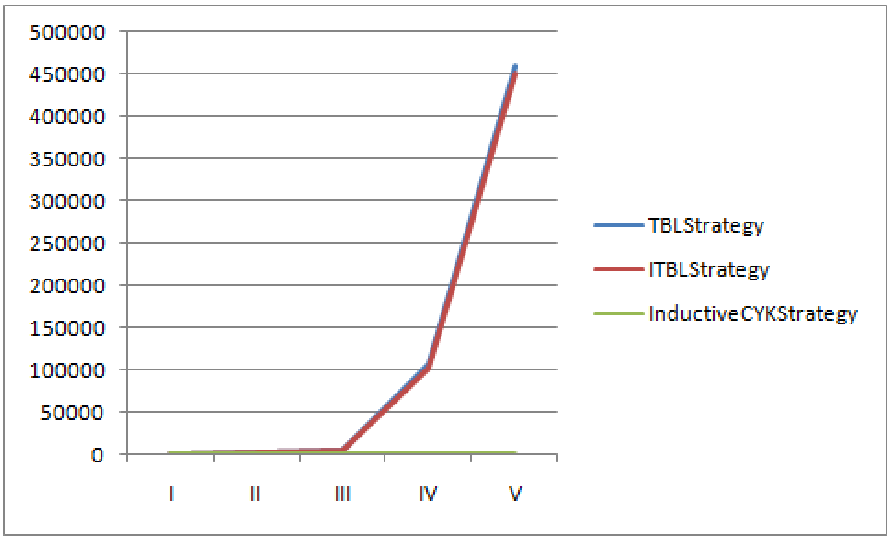


FIGURE 5.10: Time costs on TS_{abc}

TABLE 5.4: Characteristics of TS_{SQL}

TS_{SQL}	I	II	III	IV	V
$\mid TS_{SQL} \mid$	2	4	6	8	10
$\overline{\mid \omega \mid}$	4	4.75	4.5	4.75	4.8
$\mid \mathcal{T} \mid$	5	8	9	12	13

are visualized in Figure 5.11 where the y axis denotes the required time in milliseconds. The line of `ITBLStrategy` overlaps the line of `TBLStrategy` and the x axis overlaps the line of `InductiveCYKStrategy`.

TABLE 5.5: Results on TS_{SQL} (ms)

TS_{SQL}	I	II	III	IV	V
TBL	1952,33	47133.66	86925.66	304511.3	636345.3
ITBL	1697	48356.66	87380.66	303317.3	635760.3
IndCYK	247.66	218	232.66	289	256

In this case, we found that the `InductiveCYKStrategy` often stuck and did not give any response. To solve this problem, we added a counter to it to break the method if it do too many iteration. If it could not learn the grammar, its runtime was negative because it was unsuccessful. We think that this problem is the reason why we got these results.

5.6 Conclusions

This chapter presents the META which is a novel grammar induction and text processing framework. It is similar to Weka because it is well structured and implemented in Java, but it is focused on grammar induction and processing tasks instead of data mining and

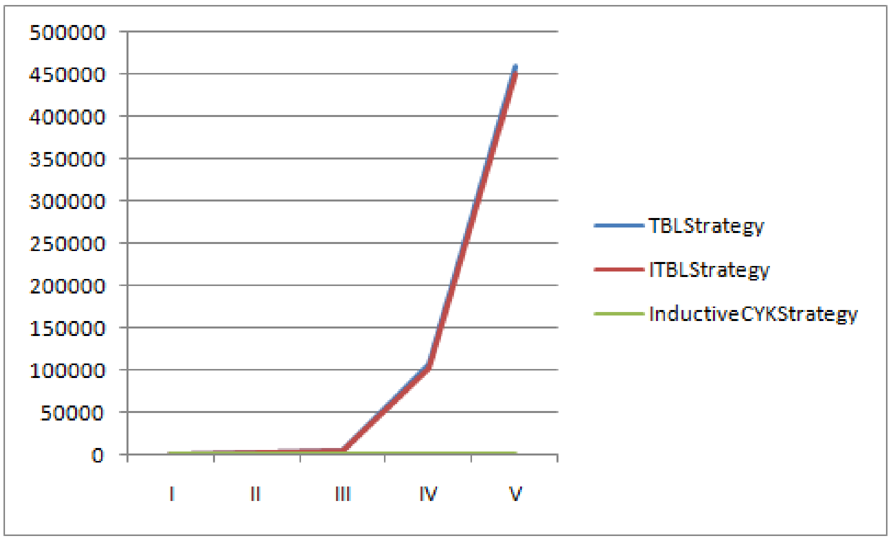


FIGURE 5.11: Time costs on TS_{abc}

machine learning. Compared with other grammar induction systems, META provides a common environment and interfaces to implement own induction and parsing methods. The architecture and the main components were presented and the formal grammar modeling module were detailed. To demonstrate the ability of META the CYK paring and the TBL, ITBL, Heuristic TBL and Inductive CYK induction algorithms were implemented and compared. The experiments were focused on the time cost of the chosen methods. Inductive CYK algorithm was shown superior to TBL and ITBL algorithms. Thus the presented META framework provides a common environment to implement, test and compare formal grammar induction and processing methods.

Thesis 3.

I have designed and developed the META framework to provide an environment for implementation, testing and comparison of different natural language processing methods. My experiments showed that both inflection and formal grammar processing methods can be implemented in the META framework

Related Publications: [2], [3], [4], [5], [6], [7]

Chapter 6

Summary

6.1 Contribution

There were three new scientific results in two major categories presented in this paper. The first topic is the learning of the inflection rules. The main goal was to create an universal inflecting algorithm which can learn any kind of inflection which was considered as string transformation. The problem domain was analyzed and the linear separability of the classes was used to measure the complexity. Then a novel inflection algorithm was introduced which is based on classification and it also uses associative memory. The proposed algorithm was compared with standard classifiers and it has been shown superior. The second topic is focused on the induction of formal grammar especially on the induction of Context–Free Grammars. META Framework was created to provide a common environment for grammar processing and induction methods.

To measure the complexity of the inflection rule induction, I have tested the linear separability of inflection classes in vector space. This property depends on the representation so I have introduced a method to create phonetic features based alphabet. This methods put the letters into a vector space based on their phonetic features, then reduces the space into one dimension. While the traditional alphabet defines only an ordering of letters, the phonetic alphabet maps the letters into real values where the position and the distances are based on the phonetic features. Linear separability was tested on a training set of 54.000 samples which contains stem and accusative form of Hungarian nouns. Experimental results showed the phonetic alphabet based encoding superior to the traditional alphabet based encoding .

I have proposed a classification based inflection algorithm which uses associative memory to store the exceptions. The inflection rules for the regular words are determined by the classifier thus the algorithm can generalize and inflect untrained words. The irregular words are stored are stored in the associative memory thus the algorithm can handle the exceptions with high precision. The size of the associative memory is a parameter of the algorithm. Experimental results showed that, the 90% precision can be exceed with an associative memory of 20.000 samples which is approximately the 40% of the total training set. The proposed algorithm was compared with standard classification method. The comparison showed that the algorithm has higher precision and lower learning cost but the size of the classification structure increases with the associative memory.

Formal Grammars are widely used to model the grammatical structures of the natural languages. The theory of formal grammars were laid in the 1950's – 1960's. These algorithms have significant computational complexity and the capacities were not enough. At the dawn of this century these algorithms have been realized and many context–free grammar induction method was developed. However the performance of these methods were analyzed separately, but there is no common environment to model formal grammars and to implement, test and compare grammar induction and processing methods. META Framework was designed to handle formal grammars. The abilities of META Framework were tested with well–known context–free grammar induction algorithms such as TBL, ITBL or Inductive CYK methods. Experimental results showed that the META is capable to model formal grammars and provides the sufficient interfaces for grammar processing methods.

These scientific results were summarized in the following three thesis.

Thesis 1.

I have measured the complexity of inflection rules induction by the ration of the linear separable clusters pairs in the vector space. I have introduced a methods to create phonetic features based alphabet. The created phonetic alphabet based encoding was shown superior to traditional alphabet based encoding.

Related Publications: [8], [9]

Thesis 2.

I have presented a classification based inflection method enhanced with associative memory. The algorithm has been shown superior to the standard classification based inflection algorithms from the point of view of precision.

Related Publications: [10], [11], [12]

Thesis 3.
I have designed and developed the META framework to provide an environment for implementation, testing and comparison of different natural language processing methods. My experiments showed that both inflection and formal grammar processing methods can be implemented in the META framework
Related Publications: [2], [3], [4], [5], [6], [7]

6.2 Future Works

The presented topics are still active research areas. The presented phonetic alphabet generation method was tested with indexes of the categories. In the future the performance of different phonetic alphabet could be measured where the parameter values should be defined by a linguist. The representation of the letters also could be refined by the encoding of a letter with real vector.

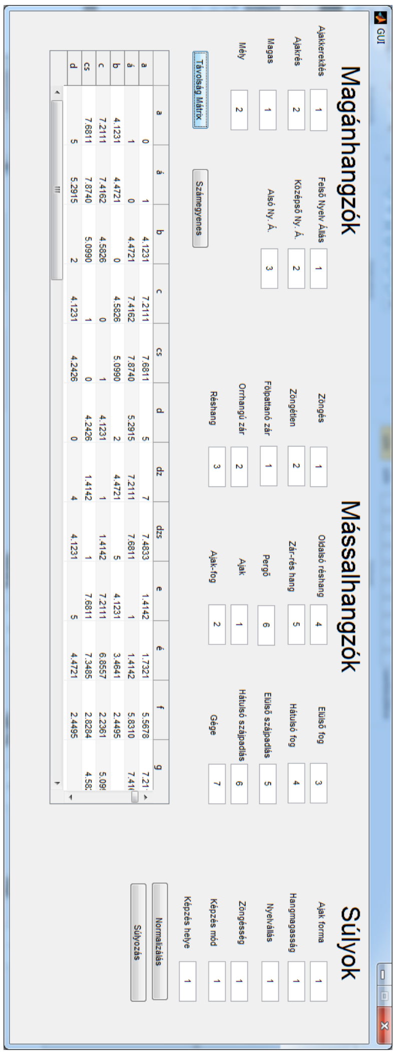
The testing of linear separability showed why the linear classifiers have poor performance. Hence different non–linear and other classification methods will be probed to learn the inflection rules. The research will be also focused on the applicability of final state transducers and concept latices for learning inflection rules.

The current version of META Framework contains the implementation of only a few well–known context–free grammar induction algorithms however it was designed to model any kind of formal grammars. It should be extended with other context–free and probabilistic context–free grammar induction methods. Moreover there are many different regular processing algorithms in the literature which could be also implemented in the framework.

Appendix A

Zoomed Figures

This appendix contains the zoomed version of Figure 3.11, Figure 4.3 and Figure 4.4.



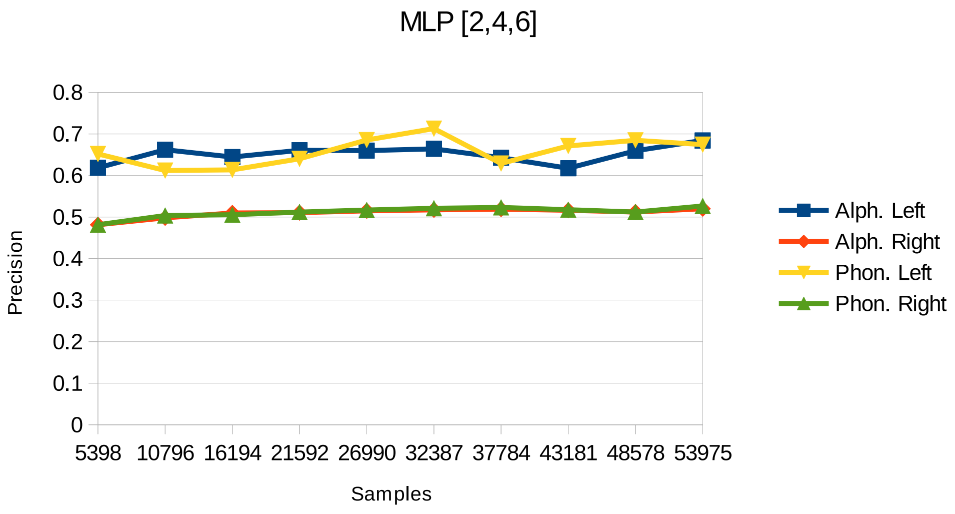
Ajakkerekítés 1
Ajakrés 2
Magas 1
Mély 2
Felső Nyelv Állás 1
Középső Ny. Á. 2
Alsó Ny. Á. 3

Zöngés 1
Zöngétlen 2
Fölpattanó zár 1
Orrhangú zár 2
Réshang 3
Oldalsó réshang 4
Zár-rés hang 5
Pergő 6
Ajak 1
Ajak-fog 2
Elülső fog 3
Hátulsó fog 4
Elülső szájpadlás 5
Hátulsó szájpadlás 6
Gége 7

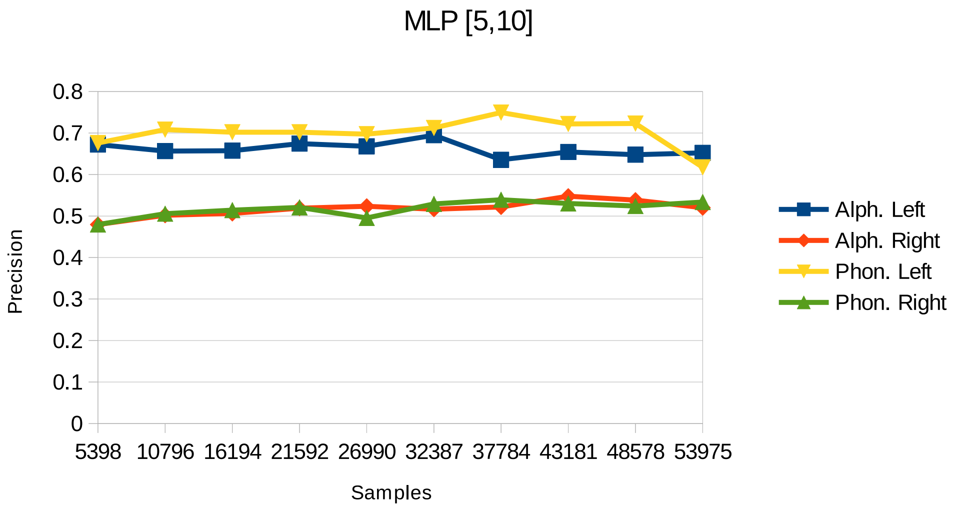
Ajak forma 1
Hangmagasság 1
Nyelvállás 1
Zöngésség 1
Képzés mód 1
Képzés helye 1

	a	á	b	c	cs	d	dz	dzs	e	é	f	g
a	0	1	4.1231	7.2111	7.6811	5	7	7.4833	1.4142	1.7321	5.5678	7.21
á	1	0	4.4721	7.4162	7.8740	5.2915	7.2111	7.6811	1	1.4142	5.8310	7.41
b	4.1231	4.4721	0	4.5826	5.0990	2	4.4721	5	4.1231	3.4641	2.4495	
c	7.2111	7.4162	4.5826	0	1	4.1231	1	1.4142	7.2111	6.8557	2.2361	5.09
cs	7.6811	7.8740	5.0990	1	0	4.2426	1.4142	1	7.6811	7.3485	2.8284	4.58
d	5	5.2915	2	4.1231	4.2426	0	4	4.1231	5	4.4721	2.4495	

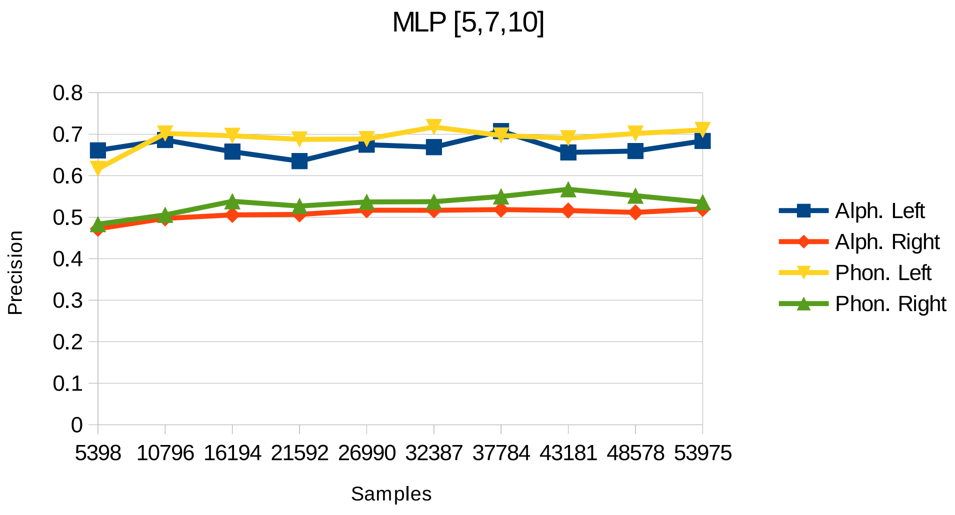
Figure A.1: User Interface of the developed MatLab Application for Hungarian



(A) [2, 4, 6]

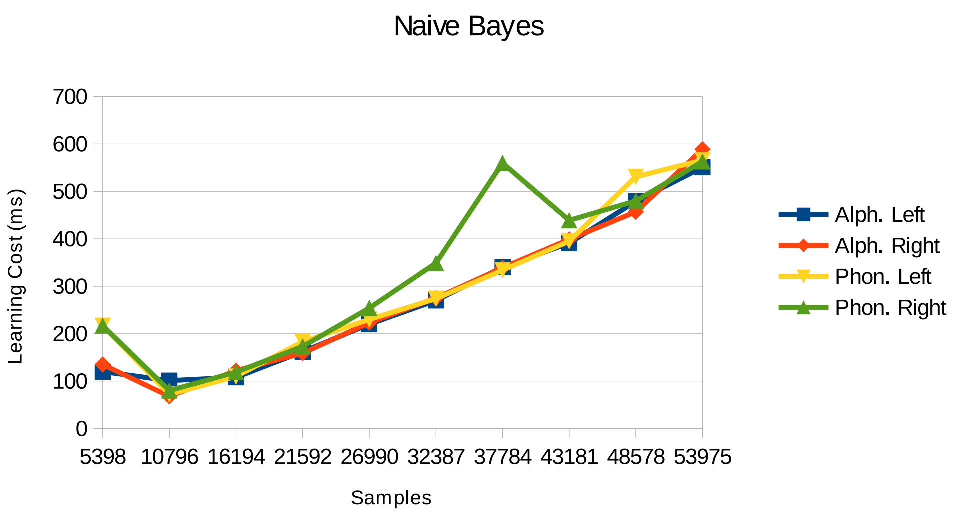


(B) [5, 10]

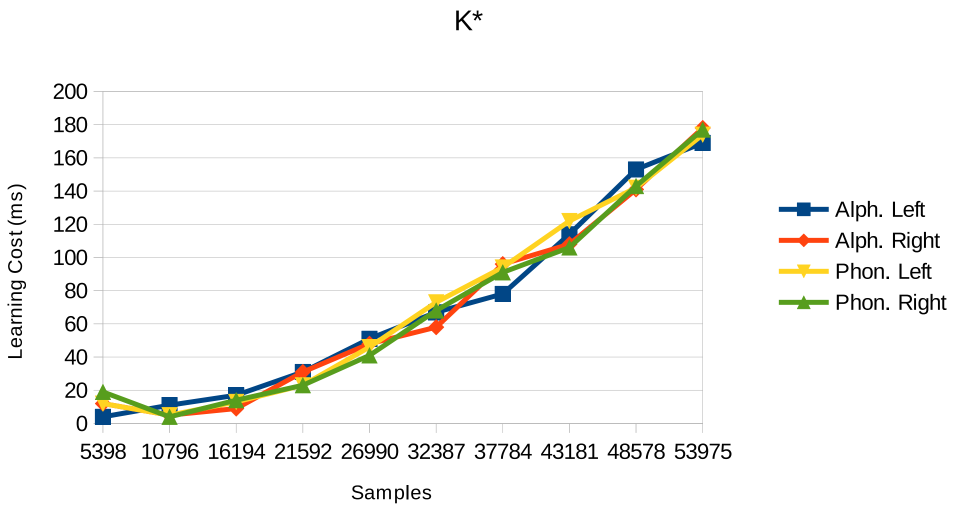


(C) [5, 7, 10]

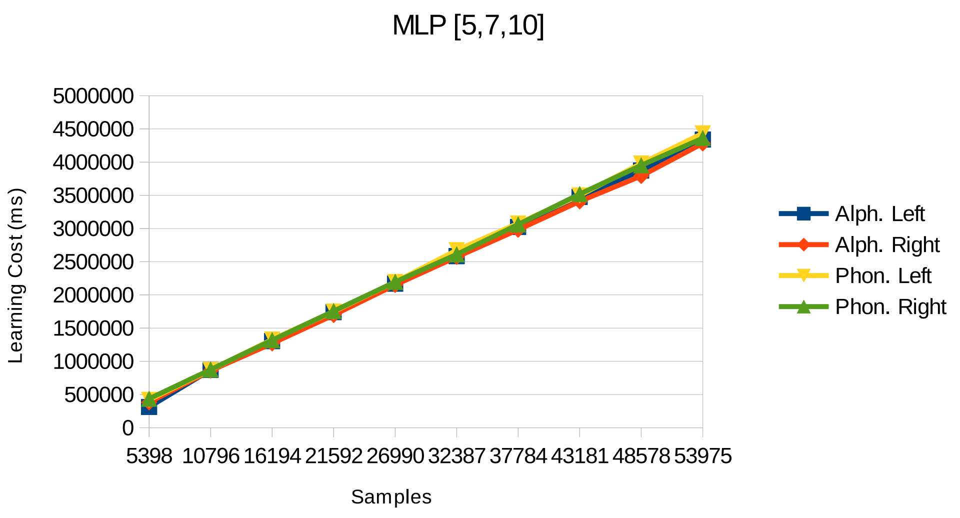
FIGURE A.2: Precision of Multilayer Perceptron Classifier with different hidden layers



(A) Naive Bayes



(B) K*



(C) Multilayer Perceptron

Figure A.3: Learning Cost of the Tested Classifiers

Appendix B

Source Codes

B.1 MatLab Application

The MatLab script files can be downloaded as a zip archive from http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/tothzs/research/phoneticAlphabet/phoneticAlphabet.zip

B.2 META Framework

B.2.1 Framework

The META Framework can be downloaded as a JAR file from
http://www.iit.uni-miskolc.hu/iitweb/opencms/users/tothzs/research/meta/meta-0.9.0-SNAPSHOT.jar

B.2.2 Documentation

The documentation is found at
http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/tothzs/research/meta/site.zip

Bibliography

[AA07] Ahmet Afsin Akın and Mehmet Dündar Akın. Zemberek, an open source nlp framework for turkic languages. *Structure*, 10, 2007.

[Apa14] Apahce. Uima, 2014. http://uima.apache.org/index.html.

[ASA01] Raátz Judit Antalné Szabó Ágnes. *Magyar nyelv és kommunikáció Tankönyv 5-6. évfolyam számára.* Nemzeti Tankönyvkiadó, 2001.

[Ass99] International Phonetic Association. *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet.* Cambridge University Press, 1999.

[Bar13] Péter Barabás. *Domain– and Language–Adaptable Natural Language Controlling Framework.* PhD thesis, University of Miskolc, József Hatvany Doctoral School of Information Science, Engineering and Technology, 2013.

[BCM$^+$08] António Branco, Francisco Costa, Pedro Martins, Filipe Nunes, João Silva, and Sara Silveira. Lx-service: Web services of language technology for portuguese. In *LREC*, 2008.

[Bed12] László Bednarik. *Automatizált Kérdésgenerállás Annotált Szövegből.* PhD thesis, University of Miskolc, József Hatvany Doctoral School of Information Science, Engineering and Technology, 2012.

[BM94] Kristin P Bennett and Olvi L Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3(1-3):27–39, 1994.

[CDG$^+$08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[Cho56] Noam Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, 1956.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

[Cho03] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

[Cla02] Alexander Clark. Memory-based learning of morphology with stochastic transducers. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 513–520. Association for Computational Linguistics, 2002.

[CMBT] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: an architecture for development of robust hlt applications hamish cunningham, diana maynard, kalina bontcheva, valentin tablan department of computer science university of sheffield.

[Con98] DM Conway. An algorithmic approach to english pluralization. In *Proceedings of the Second Annual Perl Conference. C. Salzenberg. San Jose, CA, O'Reilly*, 1998.

[CT$^+$95] John G Cleary, Leonard E Trigg, et al. K^*: An instance-based learner using an entropic distance measure. In *ICML*, pages 108–114, 1995.

[DFG11] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. A survey of grammatical inference methods for natural language learning. *Artif. Intell. Rev.*, pages 1–27, 2011.

[Dom07] Tikk Domokos. *Szövegbányászat.* Typotex Kft, 2007.

[Dud06] László Dudás. Morfémák megtanulása szövegből. In *MicroCAD 2006 International Scientific Conference*, pages 61–66, Miskolc, 2006. University of Miskolc.

[Eli06] David Elizondo. The linear separability problem: Some testing methods. *Neural Networks, IEEE Transactions on*, 17(2):330–344, 2006.

[FHH$^+$05] Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H Witten, and Len Trigg. Weka. In *Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[Fod02] Imola K Fodor. A survey of dimension reduction techniques, 2002.

[GAH04] Alexander Gelbukh, Mikhail Alexandrov, and Sang-Yong Han. Detecting inflection patterns in natural language by minimization of morphological model. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 432–438. Springer, 2004.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[GI90] Thomas V Gamkrelidze and VV Ivanov. The early history of indo-european languages. *Scientific American*, 262(3):110–116, 1990.

[Gru06] Zeph Grunschlag. Authored software. webpage, 2006. http://www.cs.columbia.edu/~zeph/software.html.

[HDW94] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[Hel03] Eugene Helimski. Areal groupings (sprachbünde) within and across the borders of the uralic language family: A survey. *Nyelvtudományi közlemények*, 100:156–167, 2003.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[HKN+04] Péter Halácsy, András Kornai, László Németh, András Rung, István Szakadát, and Viktor Trón. Creating open language resources for hungarian. In *LREC*, 2004.

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[JMM96] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[Jol05] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[KHP05] Hyunsoo Kim, Peg Howland, and Haesun Park. Dimension reduction in text classification with support vector machines. In *Journal of Machine Learning Research*, pages 37–53, 2005.

[Knu76] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.

[Kor08] András Kornai. *Mathematical linguistics*. Springer, 2008.

[Kru64] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[Lev66] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[Mar96] Gary F Marcus. Why do children say" breaked"? *Current Directions in Psychological Science*, pages 81–85, 1996.

[Mey10] Charles F Meyer. *Introducing English Linguistics International Student Edition*. Cambridge University Press, 2010.

[Mie13] Ingo Mierswa. Getting used to rapidminer. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*, page 19, 2013.

[Min14] Rapid Miner. Home page, 2014. http://rapid-i.com.

[MLCL04] Christian Monson, Alon Lavie, Jaime Carbonell, and Lori Levin. Unsupervised induction of natural language morphology inflection classes. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 52–61. Association for Computational Linguistics, 2004.

[Mol01] Raymond A Molnar. *"Generalize and Sift" as a Model of Inflection Acquisition*. PhD thesis, Massachusetts Institute of Technology, 2001.

[Mor03] Edith Moravcsik. Inflectional morphology in the hungarian noun phrase: A typological assessment. *Noun phrase structure in the languages of Europe*, 20, 2003.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[NG09] Jason Naradowsky and Sharon Goldwater. Improving morphology induction by learning spelling rules. In *IJCAI*, pages 1531–1536, 2009.

[NI00] Katsuhiko Nakamura and Takashi Ishiwata. Synthesizing context free grammars from sample strings based on inductive cyk algorithm. 2000.

[NM65] John A Nelder and Roger Mead. A simplex method for function minimization. *Computer journal*, 7(4):308–313, 1965.

[NM02] Katsuhiko Nakamura and Masashi Matsumoto. Incremental learning of context free grammars. 2002.

[OAS14] OASIS. Unstructured information management architecture, 2014. http://docs.oasis-open.org/uima/v1.0/uima-v1.0.html.

[OU09] Marcin Jaworski Olgierd Unold. Learning context-free grammar using improved tabular representation. *Applied Soft Computing*, 2009.

[Pat06a] Máté Pataki. Distributed similarity and plagiarism search. 2006.

[Pat06b] Máté Pataki. Plagiarism search within one document. 2006.

[Pin91] Steven Pinker. Rules of language. *Science*, 253(5019):530–535, 1991.

[Por80] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[Por01] Martin Porter. Snowball: A language for stemming algorithms, 2001.

[PŽ10] Martin Popel and Zdeněk Žabokrtskỳ. Tectomt: modular nlp framework. In *Advances in Natural Language Processing*, pages 293–304. Springer, 2010.

[RDM65] HL Resnikoff, JL Dolby, and Lockheed Missiles. The nature of affixing in written english. *Mechanical Translation*, 8(3):4, 1965.

[RG06] Farkas Richárd and Szarvas György. Statisztikai alapú tulajdonnév-felismerő magyar nyelvre. 2006.

[RS80] Grzegorz Rozenberg and Arto Salomaa. *The mathematical theory of L systems*, volume 90. Academic press, 1980.

[Sak05] Yasubumi Sakakibara. Learning context-free grammar using tabular representation. *Pattern Recognition*, 2005.

[SFH04] Helmut Schmid, Arne Fitschen, and Ulrich Heid. Smor: A german computational morphology covering derivation, composition and inflection. In *LREC*, 2004.

[Sha75] Michael Ian Shamos. Geometric complexity. In *Proceedings of seventh annual ACM symposium on Theory of computing*, pages 224–233. ACM, 1975.

[Shi87] Stuart M Shieber. *Evidence against the context-freeness of natural language.* Springer, 1987.

[SK99] Yasubumi Sakakibara and Mitsuhiro Kondo. Ga-based learning of context-free grammars using tabular representation. 1999.

[SV99] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[TG05] The CMS Trigger and Data Acquisition Group. The cms high level trigger. *arXiv preprint hep-ex/0512077*, 2005.

[UJ07] Olgierd Unold and Marcin Jaworski. Improved tbl algorithm for learning context-free grammar. 2007.

[Var11] Erika Baksáné Varga. *Ontology–based Semantic Annotation and Knowledge Representation in a Gramar Induction System.* PhD thesis, University of Miskolc, József Hatvany Doctoral School of Information Science, Engineering and Technology, 2011.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52, 1987.

[WFT$^+$99] Ian H Witten, Eibe Frank, Leonard E Trigg, Mark A Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[Wik14] Wikipedia. Principal component analysis @ONLINE, 2014.

[Yeg09] B Yegnanarayana. *Artificial neural networks.* PHI Learning Pvt. Ltd., 2009.

[Zha00] Guoqiang Peter Zhang. Neural networks for classification: a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30(4):451–462, 2000.

[Zsu97] Hajas Zsuzsa. *Magyar nyelv 9. osztály.* Pedellus Kiadó, 1997.

Author's Publications

[1] László Kovács, László Grigger and Zsolt Tóth, *Induction of PCFG trees*, microCAD International Scientific Conference, 2010

[2] Zsolt Tóth, László Kovács *Cost Analysis of Grammar Induction with CFG* , XXV. microCAD International Scientific Conference 2011.

[3] Zsolt Tóth, László Kovács *Előredukció alkalmazása a TBL algoritmus időköltségének csökkentésére* , Miskolci Egyetem, Doktoranduszok Fóruma 2011.

[4] Zsolt Tóth, László Kovács *Applying Prereduction to Reduce the Tim Cost of TBL Algorithm* , 12th IEEE International Symphosium on Computational Intelligence and Informatics, pp. 544-546, 2011.

[5] Zsolt Tóth, László Kovács *META a novel grammar induction and text mining framework* , XXVI. microCAD, 2012.

[6] Zsolt Tóth, László Kovács *CFG Extension for META Framework* , 16th International Conference on Intelligent Engineering Systems, pp. 495-500 2012.

[7] Zsolt Tóth *Formális nyelvtani modul a META keretrendszer számára* , GÉP: A Gépipari Tudományos Egyesület folyóirata, 2012. volume 5, pp. 99-102

[8] Tóth Zsolt, Kovács László *Fonetikai tulajdonságok alapú abc készítése* , Multidiszciplináris tudományok, 2013. volume 3 pp. 317-326

[9] Zsolt Tóth, László Kovács *Testing Linear Separability in Classification of Inflection Rules* , SISY 2014 IEEE 12th International Symposium on Intelligent Systems and Informatics, pp XX. 2014

[10] László Kovács, Zsolt Tóth *Inference of Probabilistic Grammars in Different Rules Systems of Natural Languages* In Procedia Technology, volume 12, pp. 3 - 10, 2014. (The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania)

[11] Zsolt Tóth, László Kovács *Classification based Learninig of Inflection Rules Enhanced with Associative–Memory* , Scientific Bulletin of "Petru Maior", 2014, vol. 11, no. 2, pp. 9-16.

[12] Zsolt Tóth, László Kovács *Lattices based Classification for Learning of Inflection rules in Hungarian* , to appear