Modern Natural Language Processing

Valentin Malykh, Qun Liu

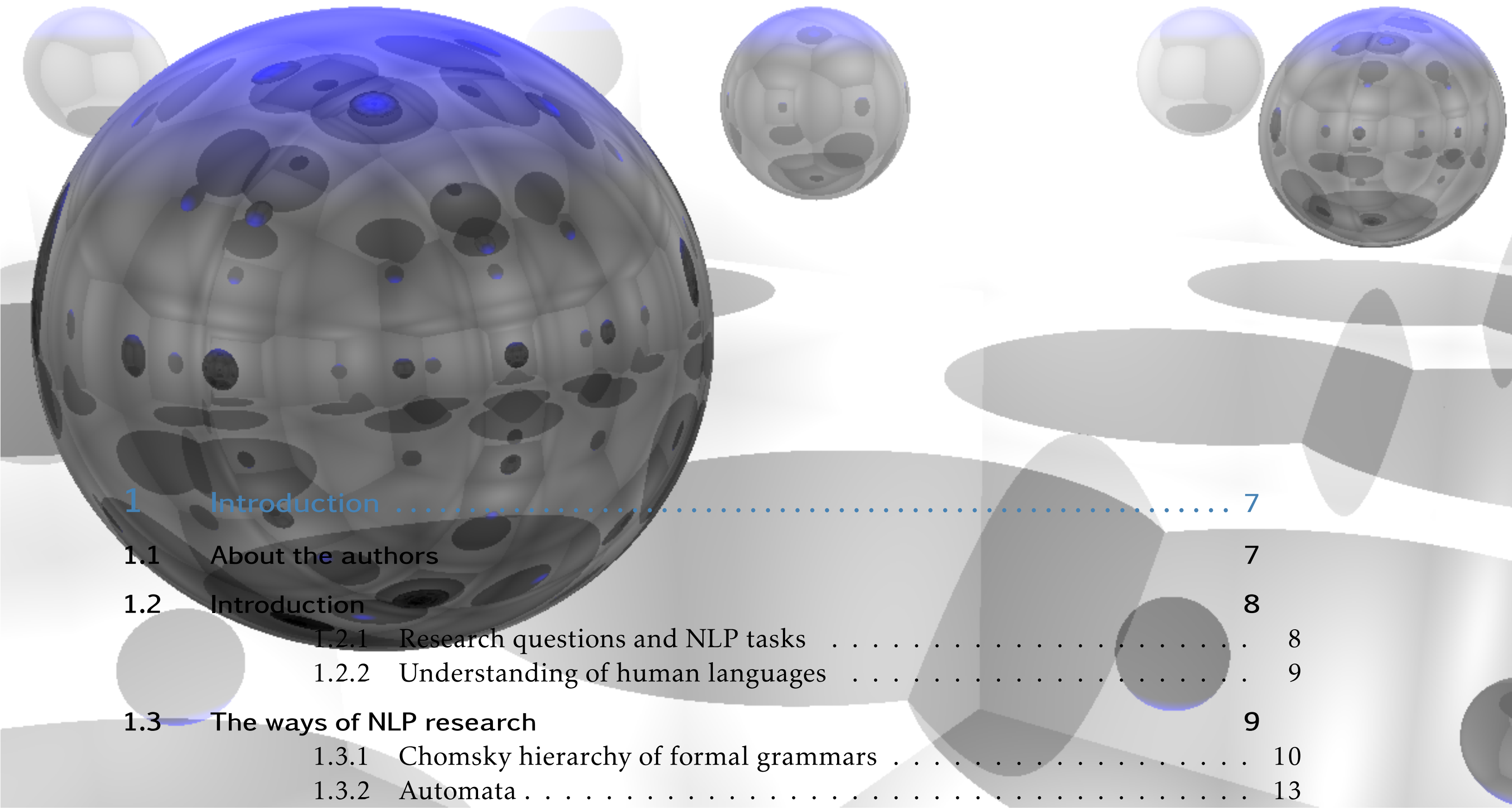
Publisher

Art. No xxxxx
ISBN xxx–xx–xxxx–xx–x
Edition 0.0

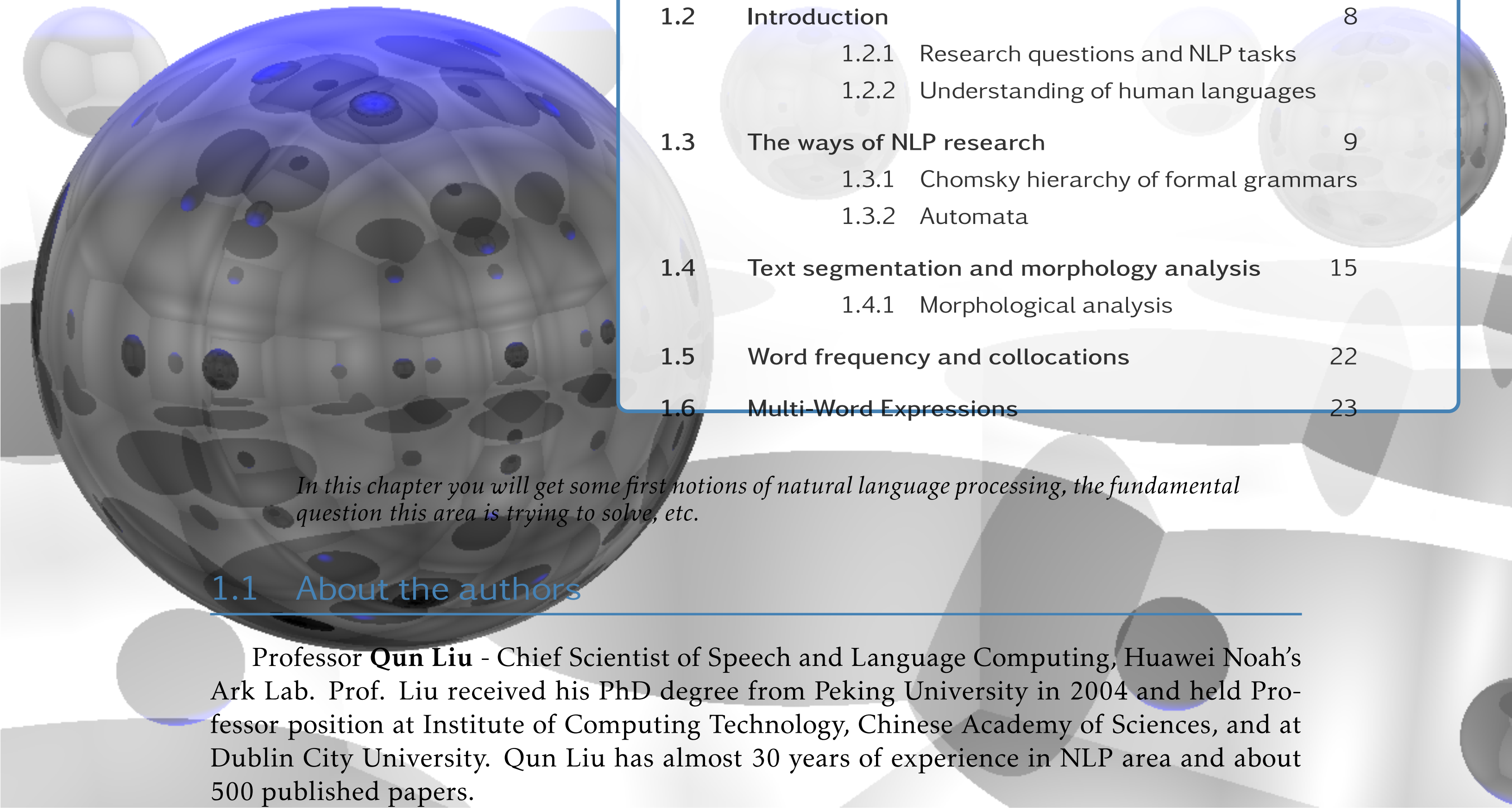
Cover design by Cover Designer

Published by Publisher
Printed in City

Contents



1. Introduction



In this chapter you will get some first notions of natural language processing, the fundamental question this area is trying to solve, etc.

1.1 About the authors

Professor **Qun Liu** - Chief Scientist of Speech and Language Computing, Huawei Noah's Ark Lab. Prof. Liu received his PhD degree from Peking University in 2004 and held Professor position at Institute of Computing Technology, Chinese Academy of Sciences, and at Dublin City University. Qun Liu has almost 30 years of experience in NLP area and about 500 published papers.

Doctor **Valentin Malykh** - Senior Research Scientist in Speech and Semantics Lab, Huawei Noah's Ark Lab. Valentin had written his thesis at Moscow Institute of Physics and Technology and received his PhD degree at Institute for Systems Programming, Russian Academy of Sciences in 2019. Valentin has 8 years of industry experience, including Yandex as a Machine

Figure 1.1: Course instructors: Prof. Qun Liu (left), Dr. Valentin Malykh (right).

Learning Engineer and VK.com as Applied Scientist. Dr. Malykh has published more than 30 papers.

1.2 Introduction

Natural Language Processing (NLP) is a domain of research whose objective is to analyze and understand human languages and develop technologies to enable human machine interactions with natural languages. NLP is an interdisciplinary field involving linguistics, computer sciences and artificial intelligence. The goal of this course is to provide students with comprehensive knowledge of NLP. Students will be equipped with the principles and theories of NLP, as well as various NLP technologies, including rule-based, statistical and neural network ones. After this course, students will be able to conduct NLP research and develop state-of-the-art NLP systems.

1.2.1 Research questions and NLP tasks

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

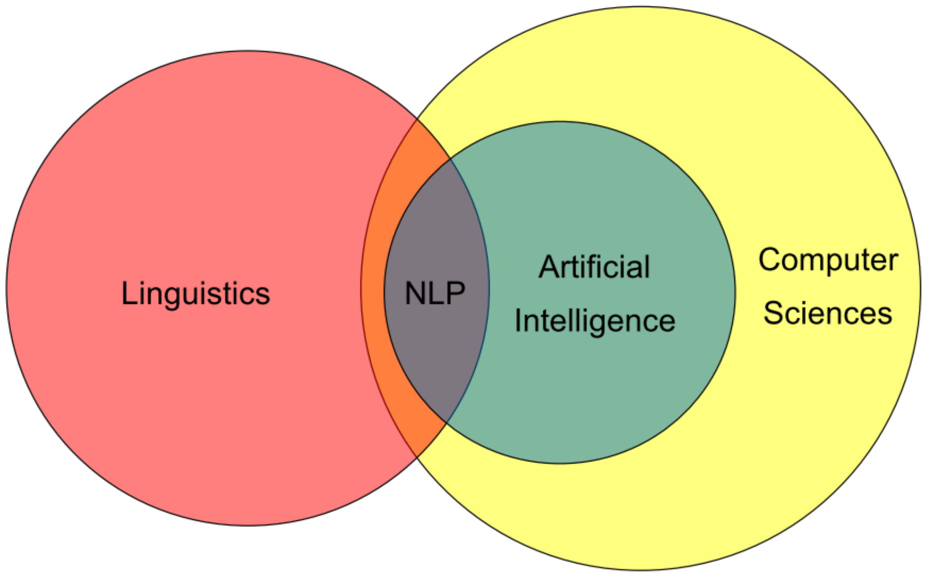


Figure 1.2: How NLP relates to the close entities.

The are several other names for NLP which are in use simultaneously (see Fig. 1.2 for their relationship). Each nom-de-plume has its own specific nuances of meaning. **Computational Linguistics** is more regarded as a branch of Linguistics, whose main purpose is to understand the mechanism of human languages by means of computing. While **Natural Language Processing** is a branch of computer sciences and artificial intelligent, whose main purpose is to develop technologies to enable human-computer interactions using human languages. **Natural Language Understanding** is also used as a synonym of NLP. It is one of the two main challenges in Natural Language Processing, while the other is **Natural Language**

Generation. At last **Human Language Technologies** mainly refer to NLP technologies, but may also include other language related technologies, include speech technologies, optical character recognition (OCR), computer typesetting, etc.

1.2.2 Understanding of human languages

We are getting used to the fact that human beings can understand each other using language communication. Although it is a natural result of evolution for human to obtain the language competence. It seems to be a miracle due to language overwhelming complexity. No other species in this planet can use languages at the degree as humans do. For example, our far relatives apes and monkeys could learn some simplified human language, but for example monkeys and apes seem to not understand each other languages while we can learn other languages with ease or at least with some effort. The mechanism behind human languages is not fully discovered. Understanding human languages by computer is difficult, since they seems to be more complex than typical algorithms a computer works with.

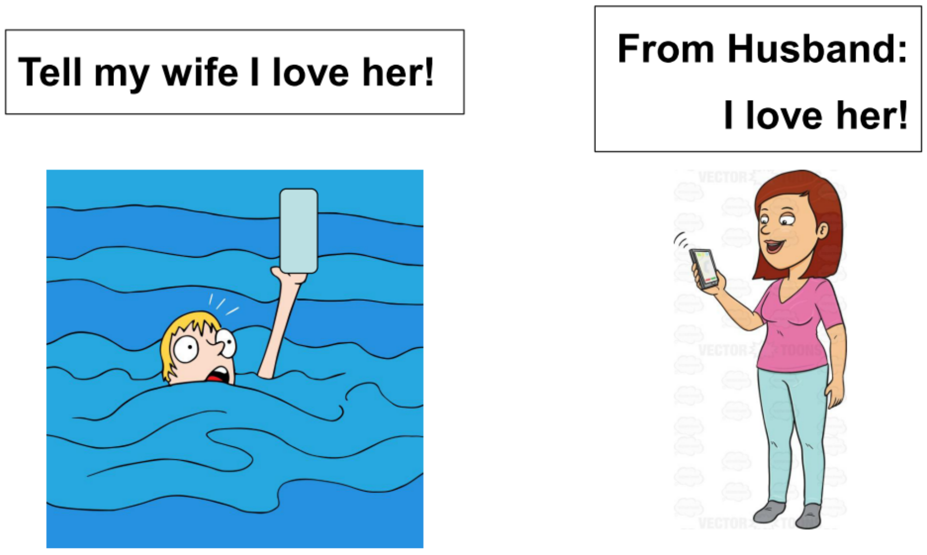


Figure 1.3: The context is crucial.

At Fig. 1.3 we could see an example of a misunderstanding. The indirect speech is misused and the meaning of a man's phrase is completely lost.

1.3 The ways of NLP research

Unlike linguists who develop numerous theories to explain the language mechanisms, NLP researchers try to simulate human language behaviors by computing, not necessary to understand the language mechanisms.

There were three main periods in NLP development. At the early stage of it, in **1960-1990s**, everything was based on the rules. The researches created more and more sophisticated rule sets for more and more complex problems. Relatively simple rule set stands behind famous in its day ELIZA chat-bot. It seems to be one of the first (if not the first) chat-bots, and it is pretty impressive even now. You could find some versions of it running at the Internet.

The next stage begun with more powerful computational engines **in 1990s** and it lasted **up to 2010s**. These were statistical models. The most famous models are Support Vector Machine and Decision Tree (with their variants). These models could make use of some data and generalize over it.

And the current period, which begun **in 2010s**. This is an era of Deep Learning. The deep learning models are also statistical, but since the computational power has grown significantly, especially with rise of graphical processing units, these models could handle the abundance of data. The data has become available conveniently with the triumph of the Internet on our planet.

It would be incorrect to say that the previously developed techniques are completely lost in time. They are still in use in some specific cases, for example in the case of low resource computational devices or in case where we need to be to understand why an answer is like that (for the rules and decision trees).

Let us try to define a language. It could be defined as the set of sentences which can be accepted by the speakers of that language. It is not possible to define a natural language by enumerate all the sentences, because the number of sentences in a natural languages is infinite. There are two feasible ways to define a language with infinite sentences: by a grammar or by an automaton.

Let us first define a grammar. A grammar G is defined as a finite set of rules, and, a mechanism to generate word sequences by applying the rules in G in a finite number of time steps.

A sentence of a language is defined as: a word sequence S is called a sentence of a language L if and only if S belongs to L.

Given a grammar G, a language L could be defined by G as: a word sequence S is a sentence of L if and only if S can be generated by G.

Next, we define an automaton. An automaton A is a abstract machine which: Takes a symbol sequence S as input, and determines if A will *accept* or *reject* S; has a finite number of states and a finite number of actions; at each time step, S is in a state, and points to a position in S; the current state and current symbol determines the action which A will execute, which determines the next state of A and the next position of S where A will point to; given a input S, A will run until it stops, and the final state of A determines if A will *accept* or *reject* S.

A language L can be defined by an automaton A as: a word sequence S is a sentence of L, if and only if, when we input S to A, A will stop in a finite number of time steps at an *accept* state.

1.3.1 Chomsky hierarchy of formal grammars

The Chomsky hierarchy (occasionally referred to as the Chomsky–Schützenberger hierarchy) is a nested hierarchy of classes of formal grammars. This hierarchy of grammars was described by Noam Chomsky in 1956. It is also named after Marcel-Paul Schützenberger, who played a crucial role in the development of the theory of formal languages.

Formal grammars

- A formal grammar G is a quadruple $\{N, T, S, R\}$:

Figure 1.4: Noam Chomsky

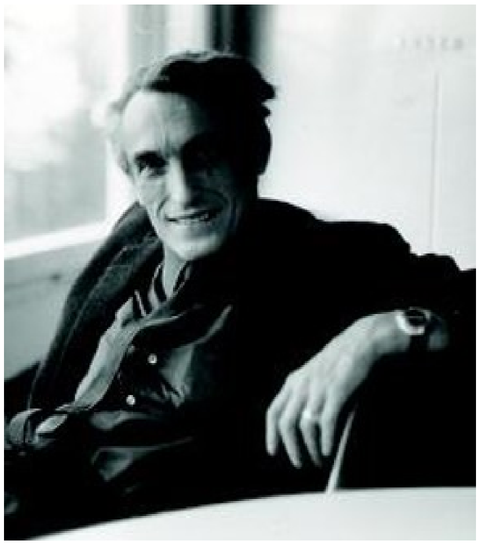


Figure 1.5: Marcel-Paul Schützenberger

- R: a finite set of production rules (left-hand side $\rightarrow$ right-hand side, or LHS $\rightarrow$ RHS), where each side consists of a finite sequence of the symbols from N, T or $\{S\}$
- N: a finite set of non-terminal symbols (indicating that some production rule can yet be applied),
- T: a finite set of terminal symbols (indicating that no production rule can be applied),
- S: a start symbol (a distinguished non-terminal symbol).

A *formal grammar G* provides an axiom schema for a *formal language L*, which is a set of finite-length sequences of symbols that may be constructed by applying *production rules* to another sequence of symbols, which initially contains just the *start symbol*. A *production rule* may be applied by replacing an occurrence of the symbols on its left-hand side (LHS) with those that appear on its right-hand side (RHS).

A sequence of rule applications is called a *derivation*. A formal grammar G defines a formal language L: all sequences of symbols consisting solely of terminal symbols which can be reached by a derivation from the start symbol.

Let us look at the example. The language $\{a^n b^n\}$ (i.e. n copies of a followed by n copies of b) can be defined by the following grammars:

Terminals: $\{a, b\}$	Terminals: $\{a, b\}$
Nonterminals:$\{S, A, B\}$	Nonterminals:$\{S\}$
Rules:	Rules:
$S \rightarrow AB$	$S \rightarrow aSb$
$S \rightarrow \epsilon$	$S \rightarrow \epsilon$
$S \rightarrow aS$	
$S \rightarrow b$	

Grammar	**Languages**	**Production Rules**	**Examples**
Type 0	Recursively Enumerable	$\alpha A\beta \rightarrow \delta$	
Type 1	Context Sensitive	$\alpha A\beta \rightarrow \alpha\gamma\beta$	$L = \{ a^n b^n c^n \| n > 0\}$
Type 2	Context Free	$A \rightarrow \alpha$	$L = \{ a^n b^n \| n > 0\}$
Type 3	Regular	$A \rightarrow a$ or $A \rightarrow aB$	$L = \{ a^n \| n > 0\}$

Table 1.1: Chomsky's hierarchy of grammars.

where ϵ is an empty string. These grammars are equivalent although they differ in the number of rules, the rules themselves and even the set of non-terminals.

Noam Chomsky has defined a hierarchy of grammars. It is showed in Tab. 1.1. Let us describe each type in more details.

Type 0 grammars

These grammars are also called Unrestricted Phrase Structure Grammars. A Type 0 Grammar generates a Recursively Enumerable Language. A Recursively Enumerable Language L said to be semi-decidable by a Turing Machine T:

- For any sentence S in L, T will accept it in a finite number of time steps.
- For a sentence S not in L, it is not guaranteed that T can reject it in a finite number of time steps.

Type 1 grammars

A grammar of this type is called Context Sensitive Grammar. A Type 1 Grammar generates a Context Sensitive Language. A Context Sensitive Language is decidable by a Linear Bounded Automaton and the complexity of this decision problem is NP-complete. It is not practical to use Type 1 Grammars in NLP because of its time complexity.

Type 2 grammars

These are Context Free Grammars. A Type 2 Grammar generates a Context Free Language. A Context Free Language is decidable by a Pushdown Automaton and the complexity of this decision problem is polynomial. Type 2 Grammars are the theoretical basis of all programming languages. Type 2 Grammars are commonly used in NLP, however, natural languages are not context free languages actually.

Type 3 grammars

And finally there are Regular Grammars. A Type 3 Grammar generates a Regular Language. A Context Free Language is decidable by a Finite State Automaton / Machine and the complexity of this decision problem is linear. Type 3 Grammars are the theoretical basis of the lexical analyzers of all programming languages. Type 3 Grammars are very broadly used in NLP for many different purposes.

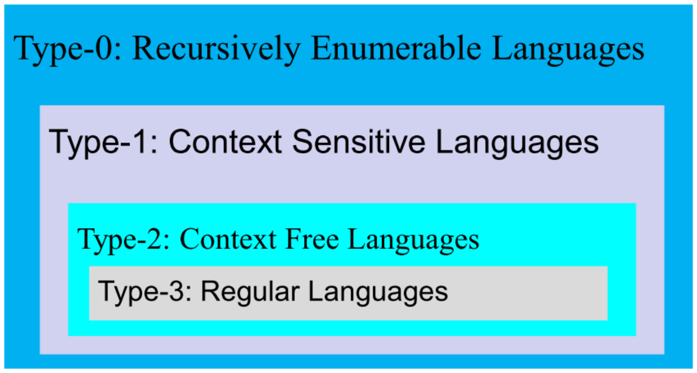

Type-0: Recursively Enumerable Languages
Type-1: Context Sensitive Languages
Type-2: Context Free Languages
Type-3: Regular Languages

Figure 1.6: Set inclusions described by the Chomsky hierarchy

Grammar	**Languages**	**Automaton**	**Decidability and Complexity**
Type 0	Recursively Enumerable	Turing Machine	Semi-Decidable
	Recursive	Decider, or Total Turing Machine	Decidable
Type 1	Context Sensitive	Linear Bounded Automaton (LBA)	NP Complete
Type 2	Context Free	Pushdown Automaton (PDA)	Polynomial
	Deterministic Context Free	Deterministic Pushdown Automaton (PDA)	Linear
Type 3	Regular	Deterministic / Nondeterministic Finite State Machine (FSM)	Linear

Table 1.2: Grammars and automata

Regular expressions

Regular Expressions are very useful tools which are supported by most of the modern programming languages and text editors. Regular Expression is equivalent to a Regular Grammar, and vice versa.

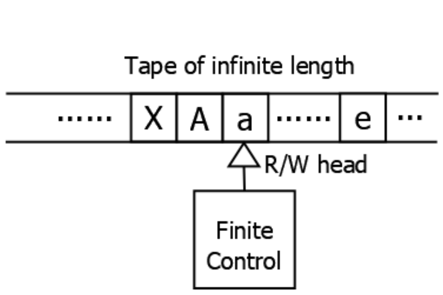
Other grammar classes

There are so called Mild Context Sensitive Grammars. These are grammar classes which define subsets of Context Sensitive Languages, but beyond Context Free Languages. Examples: Index Grammars (IGs), Tree Adjoin Grammars (TAGs).

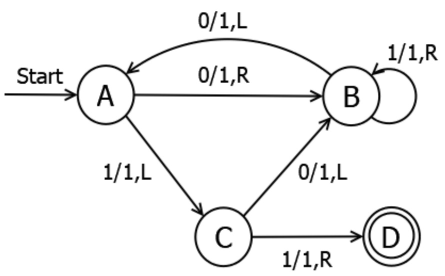
1.3.2 Automata

There is a special automaton, which has a significant impression on the whole field of computation, and the natural language processing as its part. It is Turing machine. A Turing machine is presented on Fig. 1.7ab. Alongside it a portrait of Alan Turing in young age is presented at Fig. 1.7c. A Turing machine consists of:

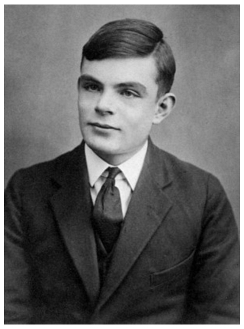
- A *tape* divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol and some other symbols. The *tape* is assumed to be arbitrarily extendable to the left and to the right.



a



b



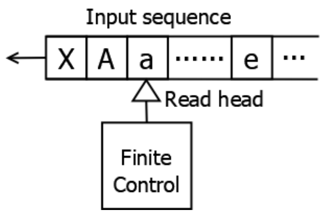
c

Figure 1.7: a) Turing machine tape; b) Turing machine state represented as automaton; c) Alan Turing, the machine author.

- A *read/write head* that can read and write symbols on the *tape* and move the *tape* left and right one (and only one) cell at a time.
- A *state register* that stores the state of the Turing machine, one of finitely many. Among these is the special *start state* with which the state register is initialized.
- A finite *table* of instructions that, given the *state* the machine is currently in and the symbol it is reading on the *tape*, tells the machine to do the following in sequence:
  - Either erase or write a symbol.
  - Move the *head* to the left or right cell.
  - Assume the same or a *new state* as prescribed.

After regarding probably most famous automaton, we are now will have a look on several other types of automata. The next thing which we will look into is a linear bounded automaton. It is a Turing Machine that satisfies the following three conditions:

- Its input alphabet includes two special symbols, serving as *left and right endmarkers*.
- Its *transitions* may not print other symbols over the *endmarkers*.
- Its *transitions* may neither move to the left of the *left endmarker* nor to the right of the *right endmarker*.



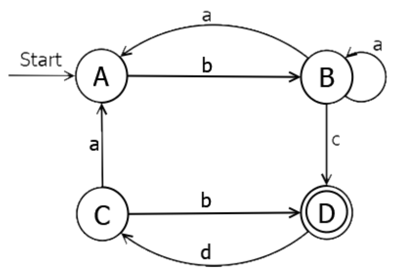


Figure 1.8: Finite state automaton / machine (FSA/FSM).

A Finite State Automaton (FSA), or Finite State Machine (FSM) presented at Fig. 1.8, consists of:

- A finite number of *states*, while the FSM can be in one *states* at each given time;
- A *head* which read a symbol from a sequence of symbols as the *input*. The *head* always goes to the next symbol at the next time step;
- A *transition* matrix which determines the next *states* of the FSM according to the current *states* and the current symbol.

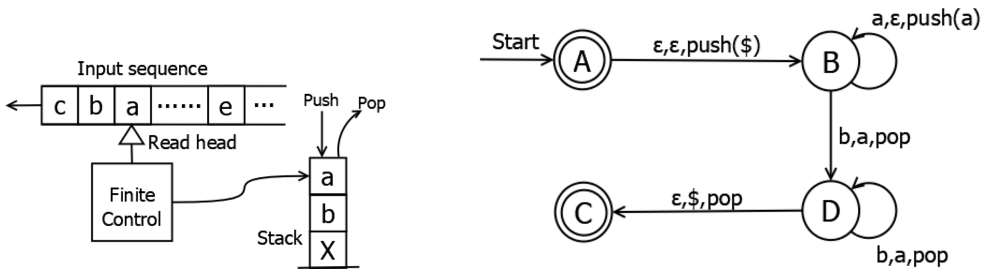


Figure 1.9: Pushdown automaton (PDA).

The last interesting automaton type is a pushdown Automaton (PDA) presented at Fig. 1.9. This automaton is similar to FSM except that it maintains a *stack*:

- It can use the top of the *stack* to decide which *transition* to take. In each step, it chooses a *transition* by indexing a table by *input symbol*, *current state*, and the *symbol* at the top of the *stack*.
- It can manipulate the *stack* as part of performing a *transition*. The manipulation can be to *push* a particular symbol to the top of the *stack*, or to *pop* off the top of the *stack*.

1.4 Text segmentation and morphology analysis

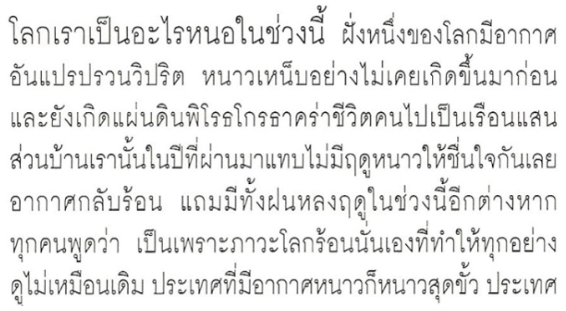
NLP is working with a text, but it is common that a NLP technique is working with smaller units that the texts. In NLP, text is segmented into units of various granularities, which include, but not limited to: Chapters and sections; Paragraphs; Sentences; Clauses; Phrases; Words; Subword units (stems, suffixes, prefixes).

Text segmentation is not straightforward in many cases:

- For languages like Chinese, Japanese, Tibetan, Thai, there are no spaces between words;
- For languages like Thai and Tibetan, the delimiters between sentences, clauses or phrases are ambiguous, which makes it hard to segment sentences;
- Even for English, sentence segmentation is not a trivial task, because the full stop mark (.) is also used for abbreviations, decimals, etc., which may or may not terminate a sentence.

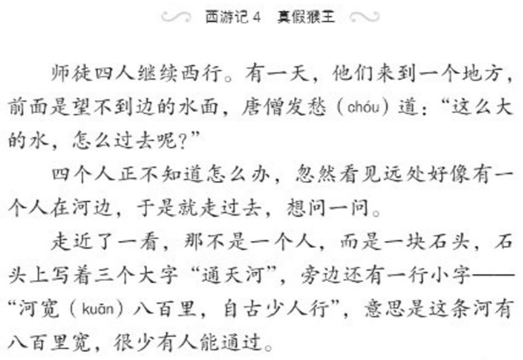
Some examples from English sentence segmentation. Why the dot marks (.) are ambiguous? The dots could be used as:

- Full stop: *This is an apple.*
- Decimal delimiter: *235.6*



โลกเราเป็นอะไรหนอในช่วงนี้ ฝั่งหนึ่งของโลกมีอากาศ
อันแปรปรวนวิปริต หนาวเหน็บอย่างไม่เคยเกิดขึ้นมาก่อน
และยังเกิดแผ่นดินพิโรธโกรธาคร่าชีวิตคนไปเป็นเรือนแสน
ส่วนบ้านเรานั้นในปีที่ผ่านมาแทบไม่มีฤดูหนาวให้ชื่นใจกันเลย
อากาศกลับร้อน แถมมีทั้งฝนหลงฤดูในช่วงนี้อีกต่างหาก
ทุกคนพูดว่า เป็นเพราะภาวะโลกร้อนนั่นเองที่ทำให้ทุกอย่าง
ดูไม่เหมือนเดิม ประเทศที่มีอากาศหนาวก็หนาวสุดขั้ว ประเทศ

Figure 1.10: Thai text sample. One cannot rely on the spaces as boundaries between the sentences.



西游记 4　真假猴王

师徒四人继续西行。有一天，他们来到一个地方，
前面是望不到边的水面，唐僧发愁（chóu）道："这么大
的水，怎么过去呢？"

四个人正不知道怎么办，忽然看见远处好像有一
个人在河边，于是就走过去，想问一问。

走近了一看，那不是一个人，而是一块石头，石
头上写着三个大字"通天河"，旁边还有一行小字——
"河宽（kuān）八百里，自古少人行"，意思是这条河有
八百里宽，很少有人能通过。

Figure 1.11: Chinese text sample. There are no spaces between the words.

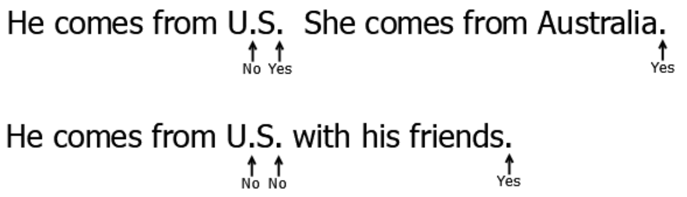
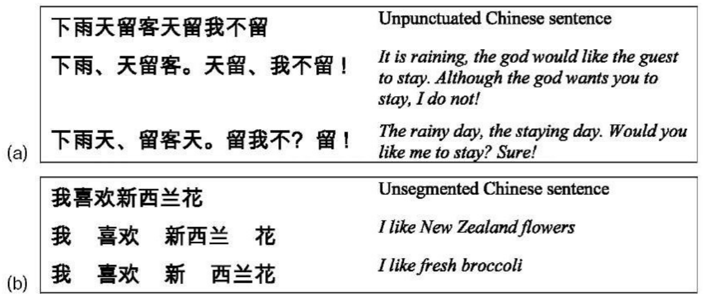

He comes from U.S. She comes from Australia.
No Yes
Yes
He comes from U.S. with his friends.
No No
Yes

Figure 1.12: English sentence segmentation.



(a)	下雨天留客天留我不留	Unpunctuated Chinese sentence
	下雨、天留客。天留、我不留！	*It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!*
	下雨天、留客天。留我不？留！	*The rainy day, the staying day. Would you like me to stay? Sure!*
(b)	我喜欢新西兰花	Unsegmented Chinese sentence
	我　喜欢　新西兰　花	*I like New Zealand flowers*
	我　喜欢　新　西兰花	*I like fresh broccoli*

Figure 1.13: Chinese word segmentation may results in different meanings.

- Integral part of an abbreviation: *U.S. Ph.D. etc.*
- Sometimes a dot could has multiple roles: *He comes from U.S.*

To segment English text into sentences, we need to determine whether a dot mark is an end of sentence or not. It can be solved as a classification problem. There is an illustrative example of ambiguous case on Fig. 1.12.

Input: Another **ex-Golden** *Stater,* Paul Stankowski from *Oxnard,* is contending for a berth on the **U.S.** Ryder Cup team after winning his first PGA Tour event last year and staying within three strokes of the lead through three rounds of last *month's* **U.S.** *Open.* **H.J.** Heinz Company said it completed the sale of its **Ore-Ida** frozen-food business catering to the service industry to McCain Foods Ltd. for about *$500* million. *It's* the first group action of its kind in Britain and one of only a handful of lawsuits against tobacco companies outside the U.S.

Output: Another **ex-Golden** *Stater ,* Paul Stankowski from *Oxnard ,* is contending for a berth on the **U.S.** Ryder Cup team after winning his first PGA Tour event last year and staying within three strokes of the lead through three rounds of last *month 's* **U.S.** *Open .* **H.J.** Heinz Company said it completed the sale of its **Ore-Ida** frozen-food business catering to the service industry to McCain Foods Ltd. for about *$ 500* million . *It 's* the first group action of its kind in Britain and one of only a handful of lawsuits against tobacco companies outside the *U.S. .*

Note: *Text in italic*: change, **text in bold**: Keep

Table 1.3: English word segmentation - Tokenization. An example of Stanford Tokenizer work.

flies	→	fly +N +PL
flies	→	fly +V +3rd +Sg

Table 1.4: Example of ambiguous morphology.

1.4.1 Morphological analysis

To break a word down into component morphemes and build a structured representation. A morpheme is the minimal meaning-bearing unit in a language. We should notice that a morpheme could not bear a meaning by itself but in some combination with some other morphemes only. The types os the morphemes are:

- **Stem** - the morpheme that forms the central meaning unit in a word. Another name for it is *root*.
- **Affix**. There are several types of affixes. It is simpler to present them by their usage:
  - **Prefix**: e.g., possible → **im**possible
  - **Suffix**: e.g., walk → walk**ing**
  - **Infix**: e.g., hingi → h**um**ingi (Tagalog)
  - **Circumfix**: e.g., sag*en* → **ge**sag**t** (German)

Practically there are two different tasks which are considered to be close. Those are Stemming and Lemmatization. Stemming itself is a process of stem extraction from a word. While the lemmatization is a process of finding a word normal form, which could or could not be the same as its stem. Let us have a look on some examples:

- Stemming:
  - Ex: writing → writ (+ ing)
- Lemmatization:
  - Ex1: writing → write (+V +Prog)
  - Ex2: books → book (+N +Pl)
  - Ex3: writes → write (+V +3Per +Sg)

The stemming example shows the stem (*writ*) and a departed suffix (*ing*). The lemmatization examples show the normal form of the verb "to write", which is *write* (an infinitive form), alongside with some special tags referring to the original word. For the word "writes" these are: *V* - reflecting the fact that this is a verb; *3Per* - the verb in the 3rd person form of a *Sg* singular verb form.

The morphological analysis could be ambiguous. Let us consider an example on Tab. 1.4.1. There are two possible interpretation of a word "flies" - either it is a plural form of a noun "a fly", or it is a 3rd person singular form of a verb "to fly". Thus ambiguity could be resolved based only on a context.

Basing on their morphology features languages could be roughly divided into three groups:

- Analytic languages: e.g., Chinese
- Synthetic flexive languages: e.g., Russian

- Synthetic agglutinate languages: e.g., Turkish

These three groups are drastically differ from each other. The analytical languages have (almost) no morphology and use function words to represent a grammatical meaning. The agglutinate languages have in contrast one morpheme for one grammatical feature at once (e.g. grammar gender and grammar number in Spanish word "amigas", where suffix *a* before suffix "s" represents that these are female gender, while *s* itself represents plural form[1]). And the flexive languages are mixing the grammatical meaning as we have seen in "writes" example.

There should be said a few words about English language this notes is written in. This language once was flexive in its history (see Middle English for reference), but the modern English language use morphemes less with time, reducing for example the verb forms to 4 most common ones: infinitive, 3rd person singular form of present tense, past tense, and continuous tense. While in the past there were at least 8 forms for a verb in only one present tense.

There are some ways to combine morphemes to form a word:

- Inflection: stem + gram. morpheme → same class
  - Ex: help + ed → helped
- Derivation: Derivation: stem + gram. morpheme → different class
  - Ex: civil + -zation → civilization
- Compounding: multiple stems
  - Ex: cabdriver, doghouse
- Cliticization: stem + clitic
  - Ex: they**'ll**, she**'s** (*I don't know who she is)

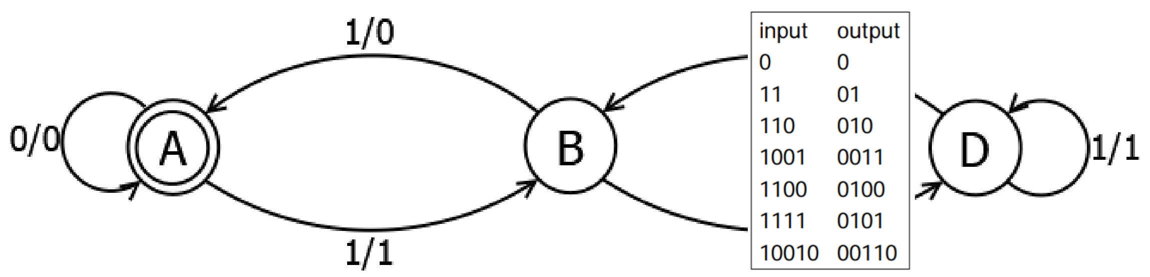
The compounding usually considered to be a word creation mechanism, rather the morphology one, but if we are using a vocabulary of stems in a morphology parser, we need to consider this case nevertheless. Also clitication is a process of creation a word from the parts of the other words and should be considered due to the same reason.

Once we had considered the morphology, let us return to the automata. There is a class of ones which is widely used to create a morphology parsers. This is the Finite state transducers (FSTs).

- Finite State Transducers are an extension to Finite State Machines, where an output symbol will be given for each input symbol.
- FSTs are commonly used tools for morphological analysis.
- A FST can be used in a inverse direction with the input and the output swapped.

Let us consider more closely English morphology. The existing word creation methods in the languages are:

[1]Spanish language itself is not an agglutinate, but a flexive one. This word is chosen as convenient example since it is pretty commonly used.



input	output
0	0
11	01
110	010
1001	0011
1100	0100
1111	0101
10010	00110

Figure 1.14: Finite state transducers (FSTs)

- Affixes: prefixes, suffixes; no infixes, no circumfixes.
- Inflectional:
  - Noun: -s
  - Verbs: -s, -ing, -ed, -ed
  - Adjectives: -er, -est
- Derivational:
  - Ex: V + suf → N

    computerize + -ation → computerization

    kill + er → killer
- Compound: pickup, database, heartbroken, etc.
- Cliticization: 'm, 've, 're, etc.

To make a morphology analyzer for English we need three components:

- Lexicon: the list of stems and affixes, with associated features.
  - Ex1: book: N
  - Ex2: -s: +PL
- Morphotactics:
  - Ex: +PL follows a noun
- Orthographic rules (spelling rules): to handle exceptions that can be dealt with by rules.
  - Ex1: y → ie: fly + -s → flies
  - Ex2: ϵ → e: fox + -s → foxes
  - Ex3: $\epsilon \rightarrow e$ / x ^ _ s#

The rewrite rule from the Fig. 1.16 could be presented as a finite-state machine, see Fig. 1.17. The examples with words "cats, foxes and geese" are presented in Fig. **??**.

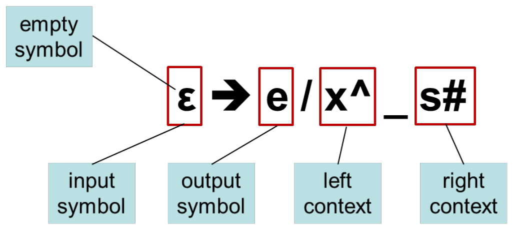


Figure 1.15: Rewrite rules for the orthography.

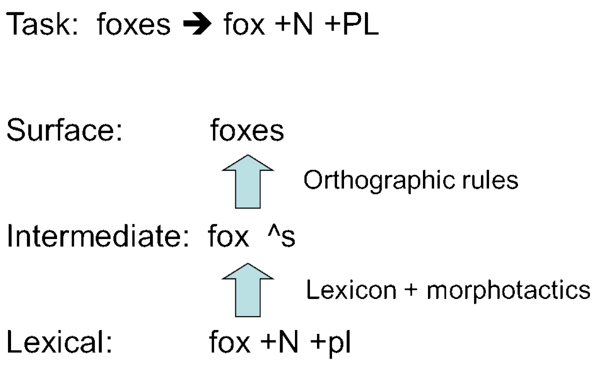


Figure 1.16: An example of the rewrites rules application.

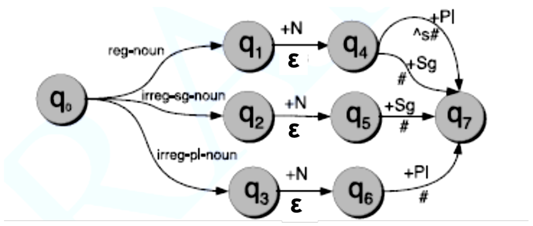


Figure 1.17: Finite-State Machine to rewrite a word to its plural form.

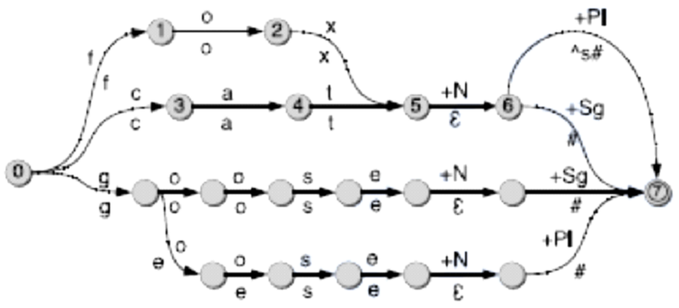
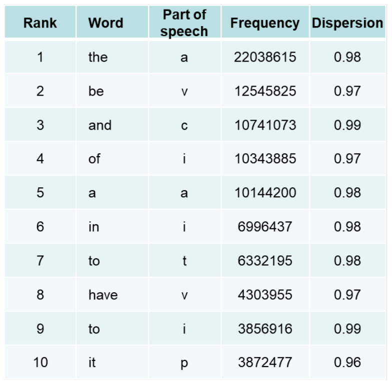


Figure 1.18: Application of a FST to create plural forms for words “cat, fox, and goose”.



Rank	Word	Part of speech	Frequency	Dispersion
1	the	a	22038615	0.98
2	be	v	12545825	0.97
3	and	c	10741073	0.99
4	of	i	10343885	0.97
5	a	a	10144200	0.98
6	in	i	6996437	0.98
7	to	t	6332195	0.98
8	have	v	4303955	0.97
9	to	i	3856916	0.99
10	it	p	3872477	0.96

Figure 1.19: Top 5000 words in American English

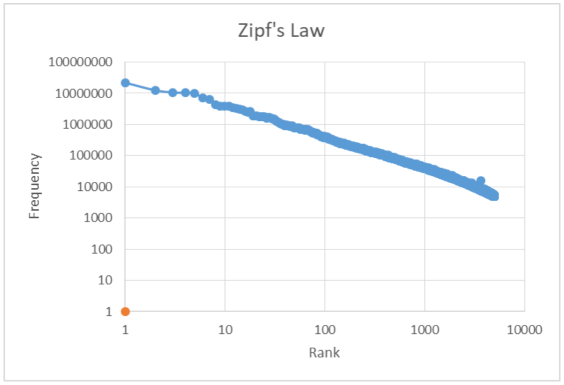


Figure 1.20: Zipf law for American English

1.5 Word frequency and collocations

Statics from Corpus of the Contemporary American English[2] is presented on the Fig. 1.19. This figure shows first dozens of 5000 most common words in English. Obviously these words are the most frequent ones, but also they have no specific meaning, like the articles "the" and "a". These not meaningful words usually called as *stopwords* in practice. The stopwords usually are being removed in the preprocessing of a particular NLP task, although there are some exceptions from this rule. But more importantly, we could use the statistics in another way. We could search for a dependency between the number of occurrences of a particular word in a corpus and the number of the words with the same number of occurrences. And there is such dependency, which is described by so called Zipf law after George Kingsley Zipf. George Zipf described this dependency for the English language (see Fig. 1.20.

The frequency of any word is inversely proportional to its rank in the frequency table. Formally, let:

N be the number of elements; k be their rank; s be the value of the exponent characterizing the distribution. Zipf's law then predicts that out of a population of N elements, the

[2]http://www.wordfrequency.info/

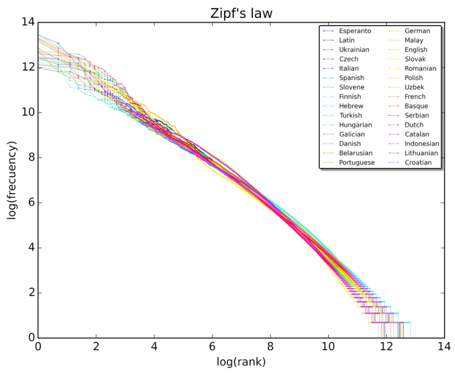


Figure 1.21: A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias (dumps from October 2015) in a log-log scale. Adopted from Wikipedia.

normalized frequency of elements of rank k, $f(k;s,N)$, is:

$$f(k;s,N) = \frac{1/k^s}{\sum\limits_{n=1}^{N} (1/n^s)}$$

Interestingly, that this law is applicable to the most natural languages, a comparison for 30 languages is presented at Fig. 1.21.

1.6 Multi-Word Expressions

A *collocation* is an expression consisting of two or more words that correspond to some conventional way of saying things. The words together can mean more than their sum of parts. These are some examples: The Times of India, disk drive; hot dog, mother in law. The samples are adopted from Manning & Schütze, Fundamentals of Statistical Natural Language Processing, 1999.

These are noun phrases like *strong tea* and *weapons of mass destruction* which could be considered as collocations. The phrasal verbs like to *make up*, and other phrases like the rich and powerful, also could be listed as ones. The collocations usually cannot be translated into other languages word by word. It is worth mentioning, that a phrase can be a collocation even if it is not consecutive (as in the example *knock ... door*).

But not all the multi-word expressions could be considered as collocations. The typical criteria for collocations:

- non-compositionality,
- non-substitutability,
- non-modifiability.

A phrase is *compositional* if the meaning can be predicted from the meaning of the parts, e.g. “new companies”. A phrase is *non-compositional* if its meaning cannot be predicted from the meaning of its parts, e.g. “hot dog”. Collocations are not necessarily fully compositional in that there is usually an element of meaning added to the combination, e.g. *strong tea*. The idioms are the most extreme examples of non-compositionality, e.g. *to hear it through the grapevine* (to know something from gossip).

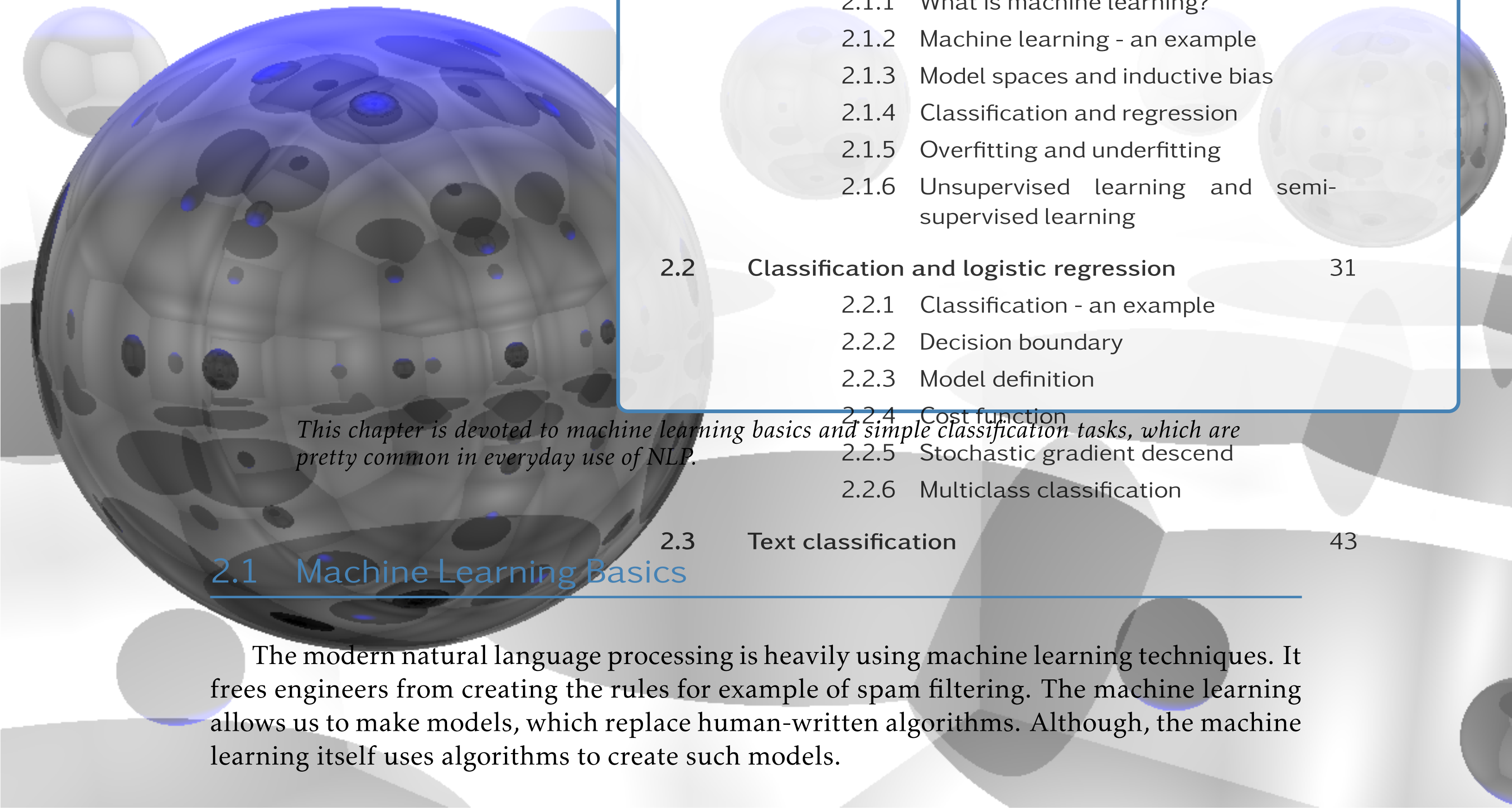
We cannot substitute near-synonyms for the components of a collocation. For example, we can't say *yellow wine* instead of *white wine* even though *yellow* is as good a description of the color of *white* wine as white is (it is kind of a yellowish white). Many collocations cannot be freely modified with additional lexical material or through grammatical transformations (*non-modifiability*). Some examples: *white wine*, but not *whiter wine*; *mother in law*, but not *mother in laws*.

There are additional criteria, which could used to determine the collocation: Frequency, Mean and Variance of Distances between Words, Hypothesis Testing, Mutual Information, Left and Right Context Entropy, C-Value.

Additional information could be found in these books and articles:

- Manning & Schütze, Fundamentals of Statistical Natural Language Processing, 1999, Chapter 3 (A general introduction to collocation)
- Katerina T. Frantzi, Sophia Ananiadou, Junichi Tsujii, The C-value / NC-value Method of Automatic Recognition for Multi-word Terms, ECDL 1998: Research and Advanced Technology for Digital Libraries pp 585-604 (proposed the C-value metric)
- Zhiyong Luo, Rou Song, An integrated method for Chinese unknown word extraction, SIGHAN 2004. Barcelona, Spain. (proposed the context entropy method).

2. Classification



This chapter is devoted to machine learning basics and simple classification tasks, which are pretty common in everyday use of NLP.

2.1 Machine Learning Basics

The modern natural language processing is heavily using machine learning techniques. It frees engineers from creating the rules for example of spam filtering. The machine learning allows us to make models, which replace human-written algorithms. Although, the machine learning itself uses algorithms to create such models.

2.1.1 What is machine learning?

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence.

Machine learning algorithms build a mathematical model based on sample data, known as ”training data”, in order to make predictions or decisions without being explicitly programmed to perform the task.

Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

Machine learning is closely related to *computational statistics*, which focuses on making predictions using computers. The study of *mathematical optimization* delivers methods, theory and application domains to the field of machine learning.

Data mining is a field of study within machine learning, it focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as *predictive analytics*.

(Supervised) Machine Learning techniques automatically learn a model of the relationship between a set of descriptive features and a target feature (or label) from a set of historical examples (example shown on Figure 2.1).

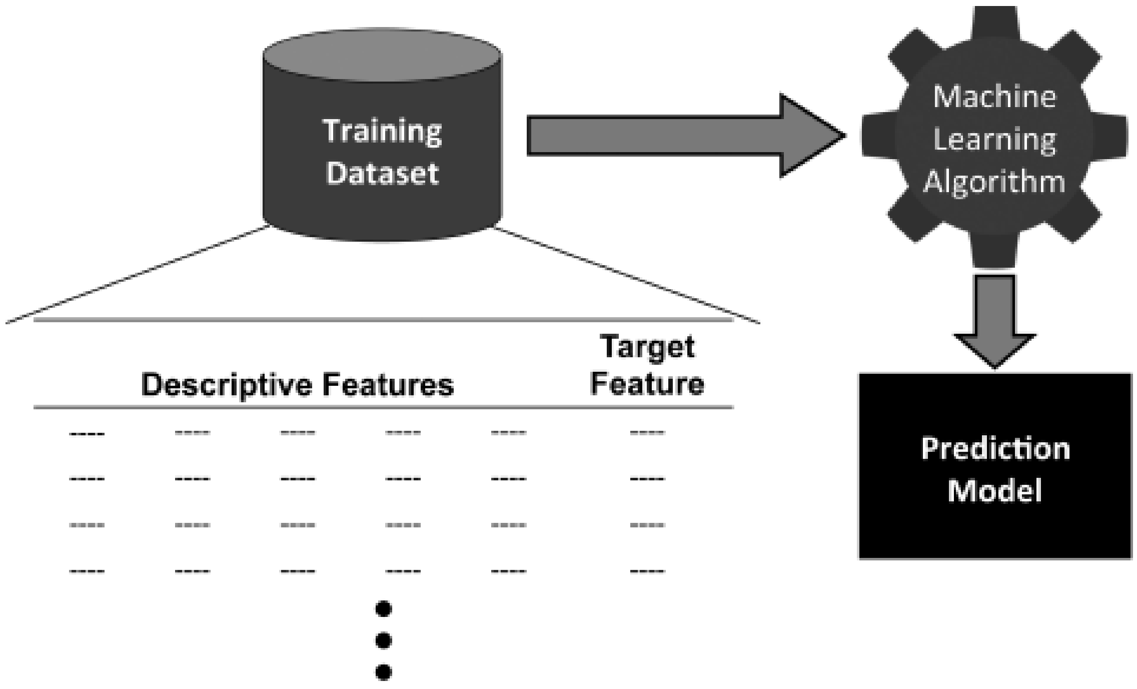


Figure 2.1: Using machine learning to introduce a prediction model from a training dataset.

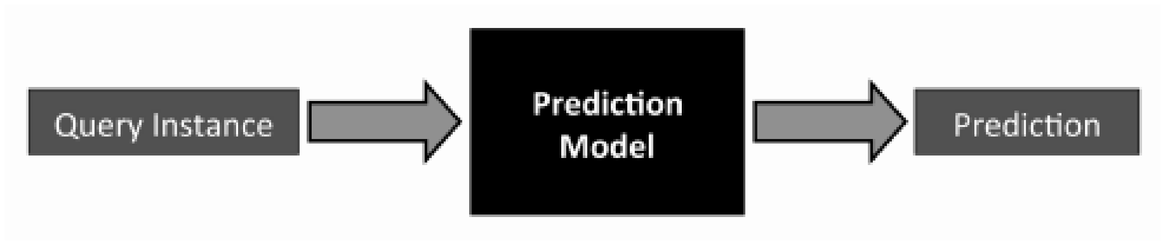


Figure 2.2: Using the model to make predictions for new query instances.

2.1.2 Machine learning - an example

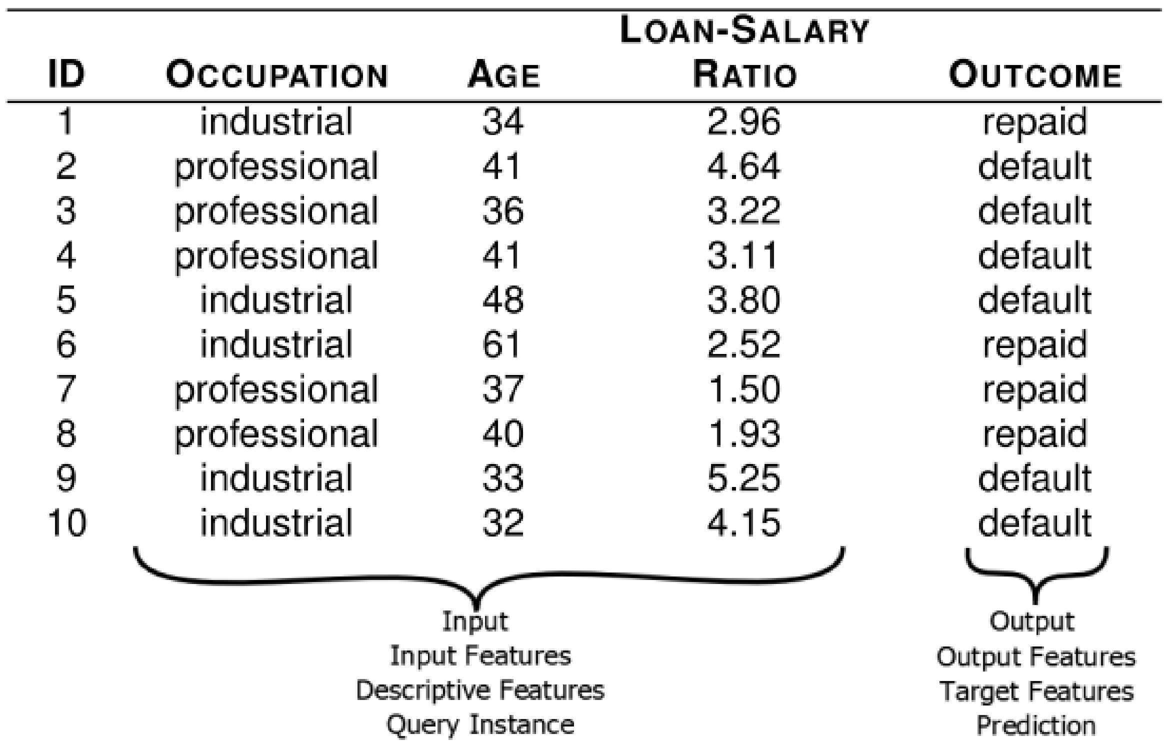
What is the relationship between the descriptive features (Occupation, Age, Loan-salary ratio) and the target, that shown at example on Figure 2.3? We could suggest an algorithm for that:

```
if LOAN-SALARY RATIO > 3 then
    OUTCOME='default'
```

```
else
    OUTCOME='repay'
end if
```

This is an example of a *prediction model*. This is also an example of a *consistent* prediction model. That means this model is consistent with the presented data. Notice that this model does not use all the features and the feature that it uses is a derived feature (in this case a ratio): *feature design* and *feature selection* are two.

Let us propose a new algorithm, which all the features presented in the table (Figure 2.3).



ID	OCCUPATION	AGE	LOAN-SALARY RATIO	OUTCOME
1	industrial	34	2.96	repaid
2	professional	41	4.64	default
3	professional	36	3.22	default
4	professional	41	3.11	default
5	industrial	48	3.80	default
6	industrial	61	2.52	repaid
7	professional	37	1.50	repaid
8	professional	40	1.93	repaid
9	industrial	33	5.25	default
10	industrial	32	4.15	default

Input
Input Features
Descriptive Features
Query Instance
Output
Output Features
Target Features
Prediction

Figure 2.3: A typical example of model input and output data.

```
if LOAN-SALARY RATIO < 1.5 then
    OUTCOME='repay'
else if LOAN-SALARY RATIO > 4 then
    OUTCOME='default'
else if AGE < 40 and OCCUPATION ='industrial' then
    OUTCOME='default'
else
    OUTCOME='repay'
end if
```

In this simplistic example one could easily come with an algorithm to predict an out-

ID	**Bby**	**Alc**	**Org**	**Grp**
1	no	no	no	couple
2	yes	no	yes	family
3	yes	yes	no	family
4	no	no	yes	couple
5	no	yes	yes	single

Table 2.1: A simple retail dataset

Bby	**Alc**	**Org**	**Grp**	M_1	M_2	M_3	M_4	M_5	**...**	M_{6561}
no	no	no	?	couple	couple	single	couple	couple	...	couple
no	no	yes	?	single	couple	single	couple	couple	...	single
no	yes	no	?	family	family	single	single	single	...	family
no	yes	yes	?	single	single	single	single	single	...	couple
yes	no	no	?	couple	couple	family	family	family	...	family
yes	no	yes	?	couple	family	family	family	family	...	couple
yes	yes	no	?	single	family	family	family	family	...	single
yes	yes	yes	?	single	single	family	family	couple	...	family

Table 2.2: A full set of potential prediction models before any training data becomes available

come. But in the real world data there are several thousand or more recently several million of such rows, which are called *data sample*s or data points. The real value of machine learning becomes apparent in situations like this when we want to build prediction models.

2.1.3 Model spaces and inductive bias

Machine learning algorithms work by searching through a set of possible prediction models for the model that best captures the relationship between the descriptive features and the target feature (see Fig. 2.1, Fig. 2.2).

An obvious search criteria to drive this search is to look for models that are *consistent* with the data (see Fig. 2.3). However, because a training set is only a sample of the whole data (which is called general set), machine learning is an *ill-posed* problem. In the real world problems there are virtually unlimited number of possible models, the choice from this multidimensional space is an essential problem of a data science in general and natural language processing in particular.

Notice that there is more than one candidate model left! It is due to a single consistent model cannot be found. Consistency $\approx$ **memorizing** the dataset. Consistency with **noise** in the data is not desirable. Our goal will be a model that **generalises** beyond the given data

Table 2.3: A sample of the models that are consistent with the training data

ID	Age	Income
1	21	24000
2	32	48000
3	62	83000
4	72	61000
5	84	52000

Table 2.4: An age-income dataset

and that is not influenced by the noise in the dataset.

So what criteria should we use for choosing between models? **Inductive bias** is a set of assumptions that define the model selection criteria of an ML algorithm. There are two types of bias that we can use: *restriction bias* and *preference bias*. Inductive bias is necessary for learning, and learning means going beyond the dataset.

2.1.4 Classification and regression

Let us consider a sample of classification task. We need to predict a target feature with categorical values (see Table 2.1). Here we could see that there are several descriptive features both numerical and categorical and the target feature is categorical. That means that we need to predict a category (a class) for a data sample.

This example also gives a good opportunity to see the difference between the categorical and numerical data. The numerical data has the fundamental property - it is comparable, we always could say that 2 is bigger than 1. While in the categorical case there could be no such ordering.

Another type of a machine learning task is **regression**. The regression task is to predict a target feature with numerical values (see Table 2.4). This particular dataset comes with only one numerical descriptive feature. But there could be both numerical and categorical features, that is only the numerical outcome which matters in this case.

2.1.5 Overfitting and underfitting

Let us visualize our age-income dataset as it shown at Figure 2.4. There are only the dots which are dropped somewhat randomly on the plane. But we could imagine that this dots are not random and could be described with mode function. On the Figure 2.5 one could see four subplots. One could see that there are three possible models which describe feature dependency and the origina data as plot (a). The perfectly describing the data model (c) seems to be going too far, i.e. it predicts the more complex dependency which is obvious to a human eye. The (b) model in contrary goes not far enough: it is too simple to catch the dependency. And only the (d) model is catching the essential issue. This model is not perfectly describing data, but it is instead adopted to the noise which is always there in the real data. So the machine learning algorithm should be able to choose such a model among other possible ones.

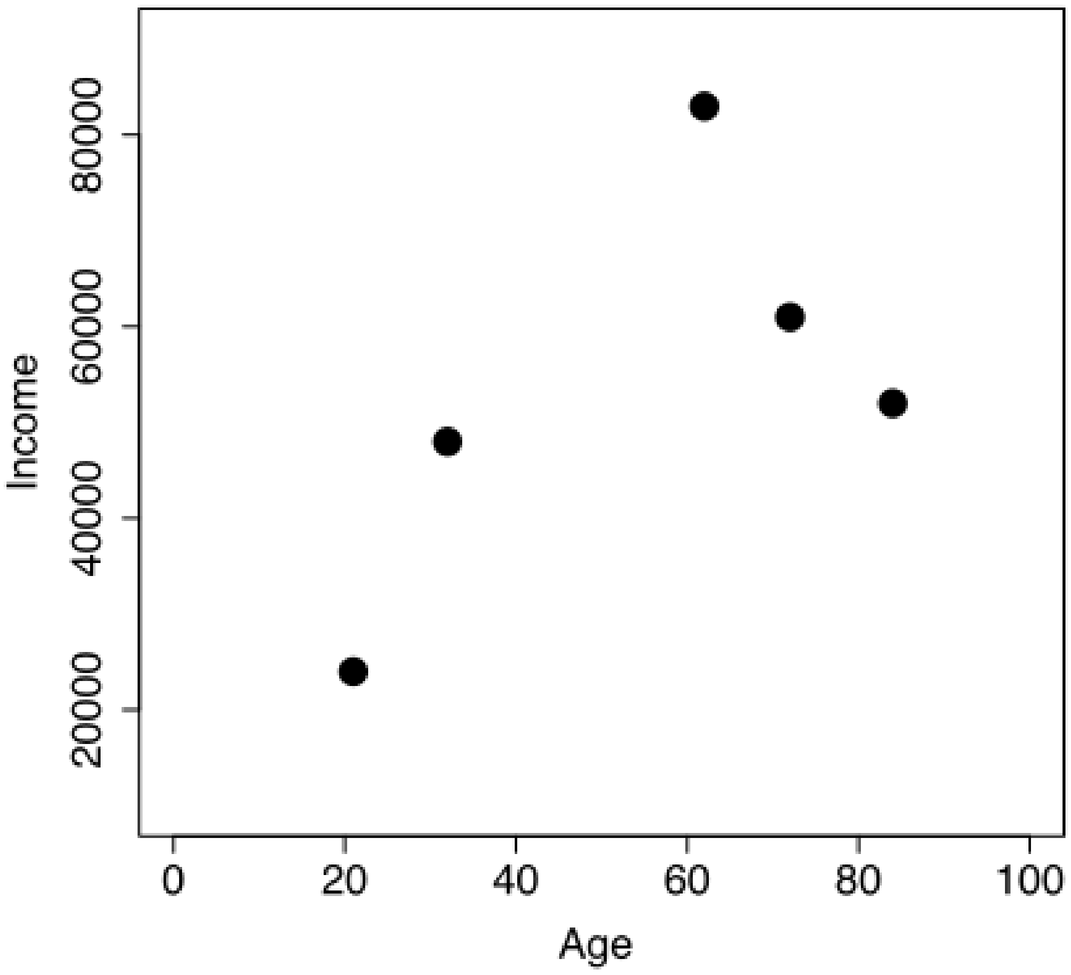


Figure 2.4: Age-income data.

How **overfitting** and **underfitting** plots would be looks like (Figure 2.5)?

2.1.6 Unsupervised learning and semi-supervised learning

Unsupervised learning is the *machine learning task* of inferring a function to describe hidden structure from unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning (Figure 2.6).

The typical unsupervised learning task is clustering.

Cluster analysis or *clustering* is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).

For a some data we always have a lot of ways how to clusterize it (as it shown on Fig-

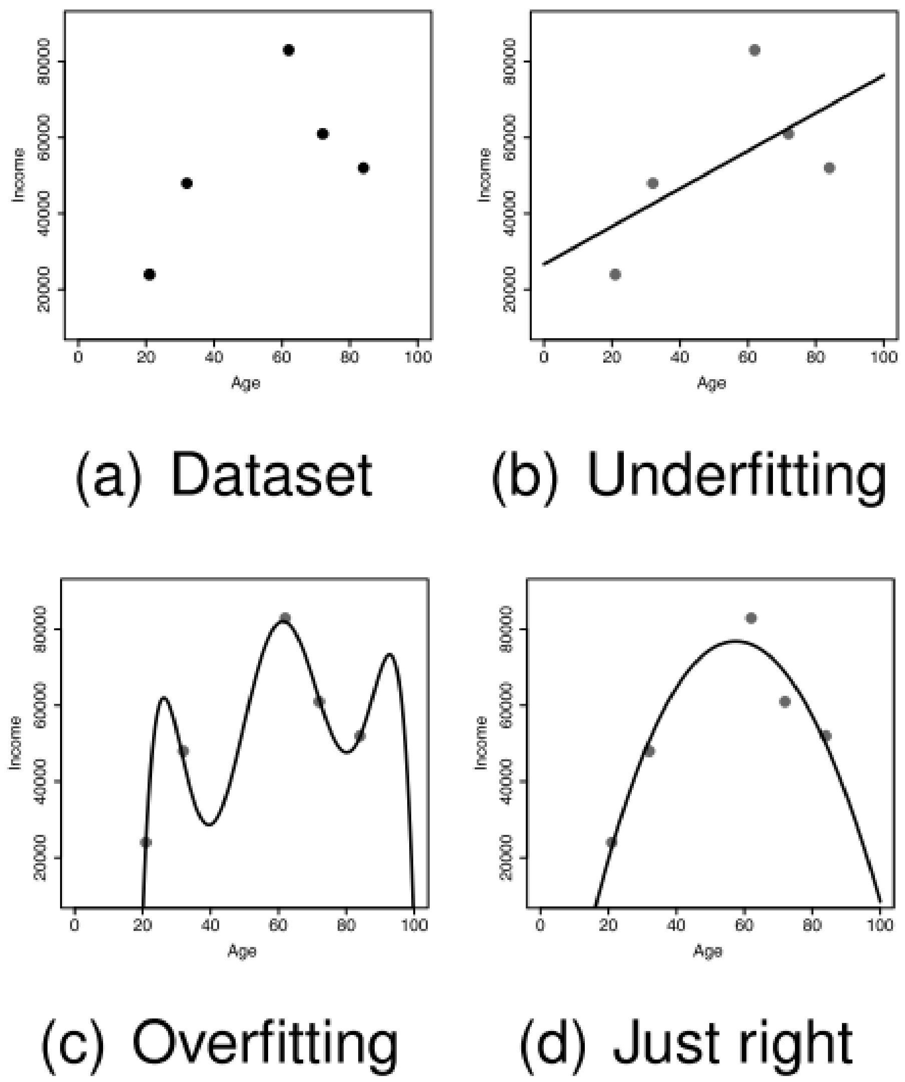


Figure 2.5: Striking a balance between overfitting and underfitting when trying to predict age from income.

ure 2.7).

2.2 Classification and logistic regression

2.2.1 Classification - an example

Consider a set of some tabular data (Table 2.5).

If we plot the dependence of RPM on Vibration, we get the following picture as it shown on Figure 2.8.

ID	RPM	Vibration	Status		ID	RPM	Vibration	Status
1	568	585	good		29	562	309	faulty
2	586	565	good		30	578	346	faulty
3	609	536	good		31	593	357	faulty
4	616	492	good		32	626	341	faulty
5	632	465	good		33	635	252	faulty
6	652	528	good		34	658	235	faulty
7	655	496	good		35	663	299	faulty
8	660	471	good		36	677	223	faulty
9	688	408	good		37	685	303	faulty
10	696	399	good		38	698	197	faulty
11	708	387	good		39	699	311	faulty
12	701	434	good		40	712	257	faulty
13	715	506	good		41	722	193	faulty
14	732	485	good		42	735	259	faulty
15	731	395	good		43	738	314	faulty
16	749	398	good		44	753	113	faulty
17	759	512	good		45	767	286	faulty
18	773	431	good		46	771	264	faulty
19	782	456	good		47	780	137	faulty
20	797	476	good		48	784	131	faulty
21	794	421	good		49	798	132	faulty
22	824	452	good		50	820	152	faulty
23	835	441	good		51	834	157	faulty
24	862	372	good		52	858	163	faulty
25	879	340	good		53	888	91	faulty
26	892	370	good		54	891	156	faulty
27	913	373	good		55	911	79	faulty
28	933	330	good		56	939	99	faulty

Table 2.5: A dataset listing features for a number of power-generators

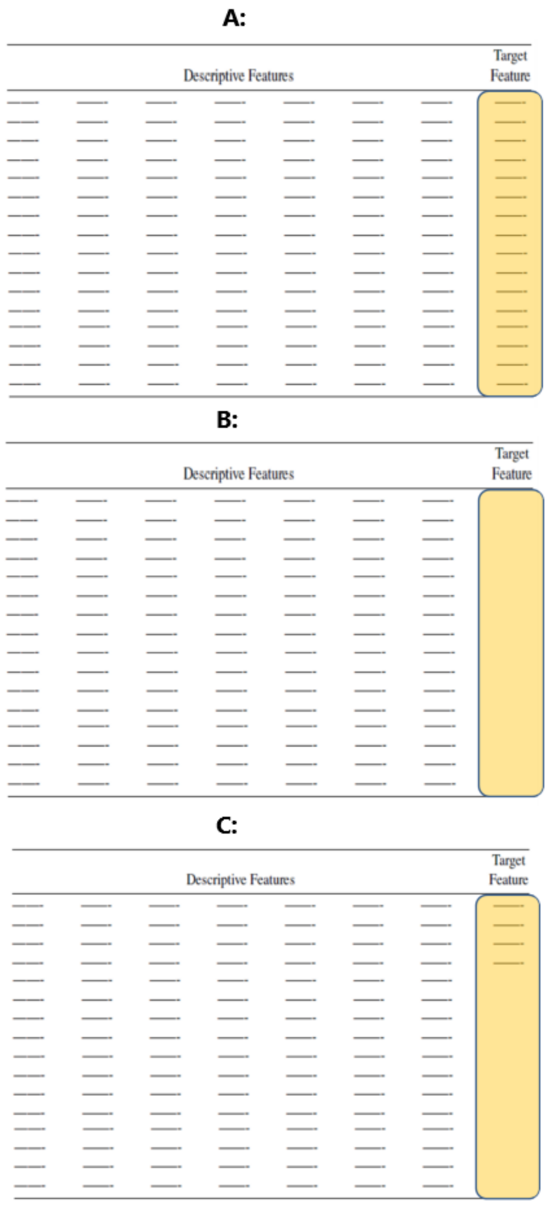


Figure 2.6: Data of three types of learning: A) Supervised learning; B) Unsupervised learning; C) Semi-supervised learning.

2.2.2 Decision boundary

A decision boundary is the region of a problem space in which the output label of a classifier is ambiguous. If the decision surface is a hyperplane, then the classification problem is linear, and the classes are linearly separable (Figure 2.9). Decision boundaries are not always clear cut.

$$830 - 0.667 \text{ x RPM - Vibration} = 0$$
$$\theta_0 + \theta_1 x_1 + x_2 = 0$$

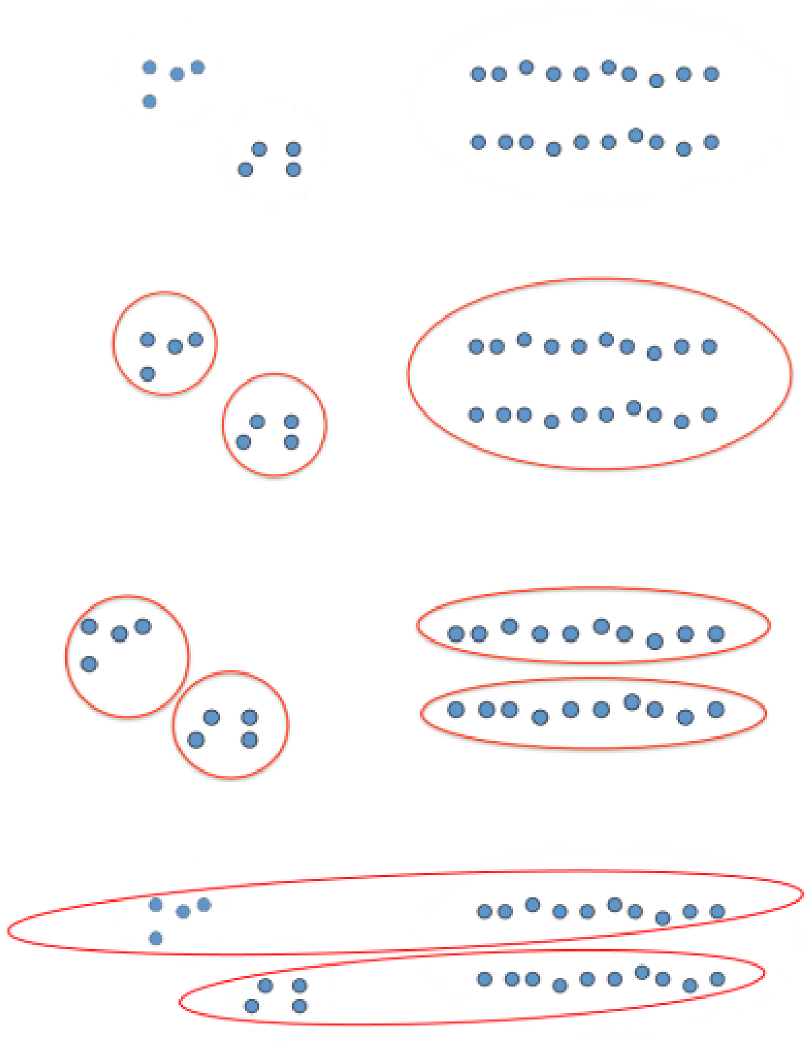


Figure 2.7: Three examples of clustering for the same data.

For all “good” generators, we have: $\theta_0 + \theta_1 x_1 + x_2 >= 0$
For all “faulty” generators, we have: $\theta_0 + \theta_1 x_1 + x_2 < 0$

$$\text{Let: } \theta = \begin{bmatrix} 1 \\ \theta_1 \\ \theta_0 \end{bmatrix}, x = \begin{bmatrix} x_2 \\ x_1 \\ 1 \end{bmatrix}$$

Then we have the decision boundary:

$$d_\theta(x) = \theta^T x = 0$$

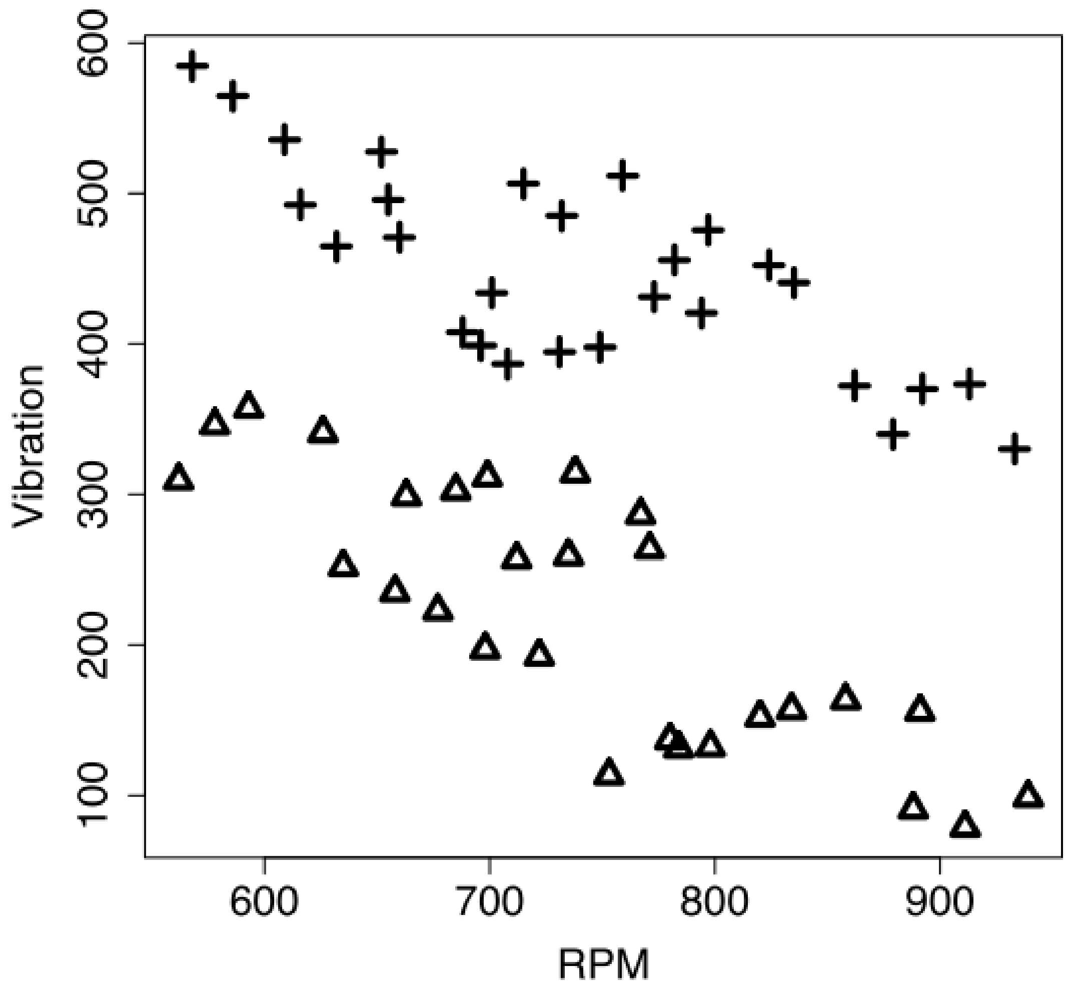


Figure 2.8: An example of visualization of data

2.2.3 Model definition

The Heaviside step function, or the unit step function, usually denoted by H or θ, is a discontinuous function, named after Oliver Heaviside (1850–1925), whose value is zero for negative arguments and one for positive arguments (Figure 2.10). It is an example of the general class of step functions, all of which can be represented as linear combinations of translations of this one.

The function was originally developed in operational calculus for the solution of differential equations, where it represents a signal that switches on at a specified time and stays switched on indefinitely. Oliver Heaviside, who developed the operational calculus as a tool in the analysis of telegraphic communications, represented the function as 1.

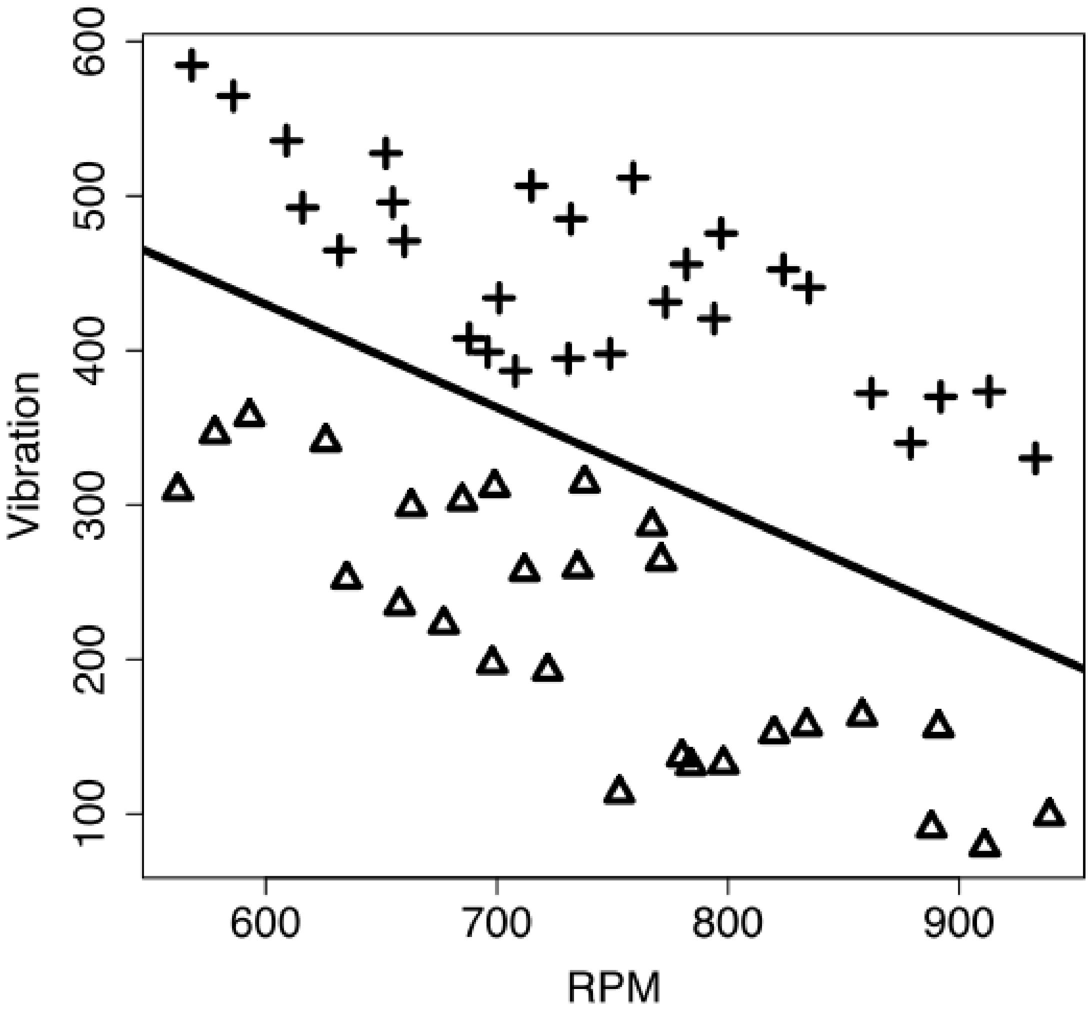


Figure 2.9: Decision boundary

$$Heaviside(x) = \begin{cases} 1 & \text{if: } x >= 0 \\ 0 & \text{if: } x < 0 \end{cases}$$

$$\begin{aligned} h_\theta(x) &= Heaviside(d_\theta(x)) \\ &= \begin{cases} 1 & \text{if: } d_\theta(x) = \theta^T x >= 0; \text{ for } \textit{good} \text{ generators} \\ 0 & \text{if: } d_\theta(x) = \theta^T x < 0; \text{ for } \textit{faulty} \text{ generators} \end{cases} \end{aligned}$$

You can imagine the Heaviside function as a logistic function. The function can be constructed both in two-dimensional and in three-dimensional space (Figure 2.11).

Heaviside step function is not derivable and hard to optimize. An alternative is using a Logistic function (sigmoid function) (Figure 2.12):

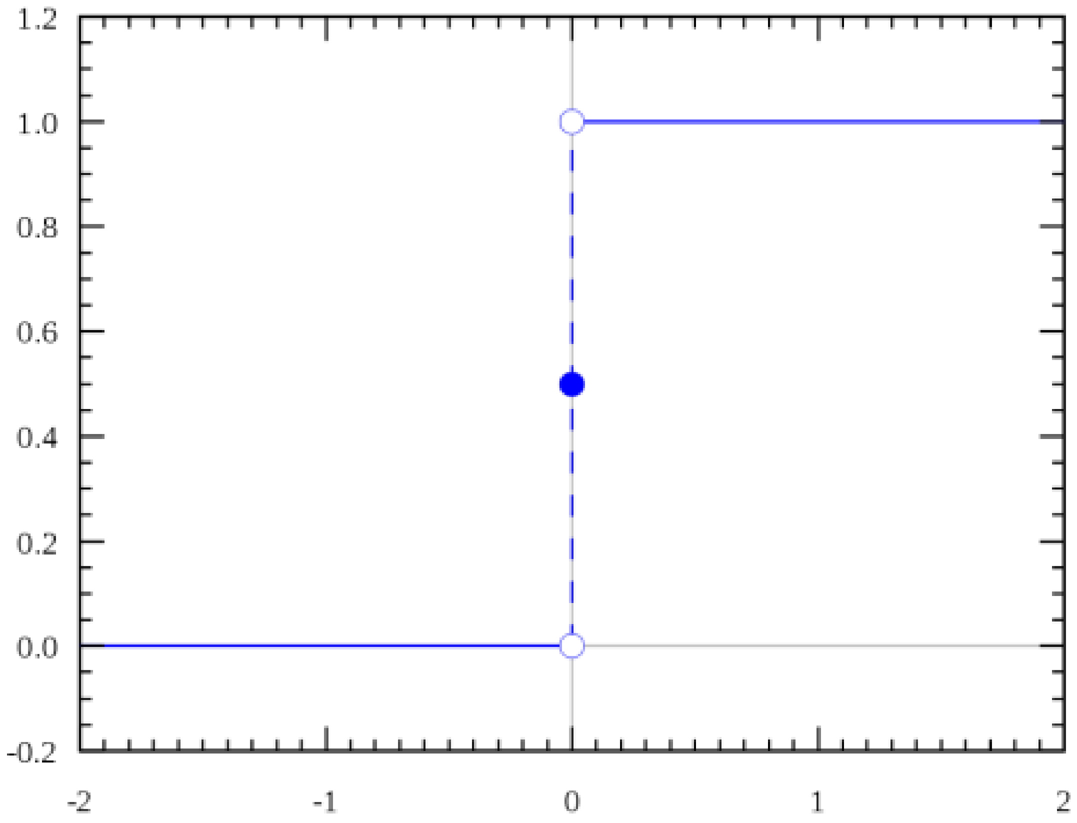


Figure 2.10: The Heaviside step function, using the half-maximum convention

$$Logistic(x) = \frac{1}{1+e^{-x}}$$

$$h_\theta(x) = Logistic(d_\theta(x)) = \frac{1}{1+e^{-d_\theta(x)}} = \frac{1}{1+e^{-\theta^T x}}$$

2.2.4 Cost function

The objective of a learning algorithm is to search a model to best predict the target feature on the training data given the inductive bias. One way to achieve this goal is to minimize the errors of the prediction over the training data. A cost function (loss function) is defined as the sum of the errors over the training samples.

A possible cost function:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(h_\theta(x^{\{i\}}) - y^{\{i\}})^2$$

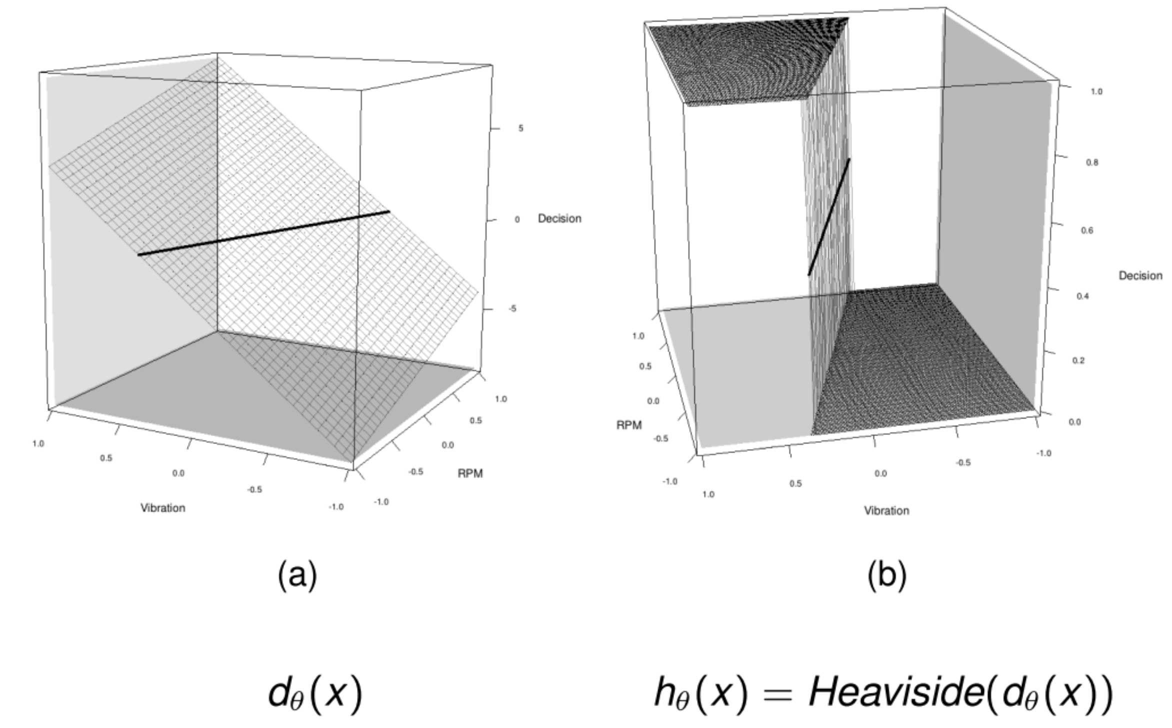


Figure 2.11: Model as a Heaviside step function

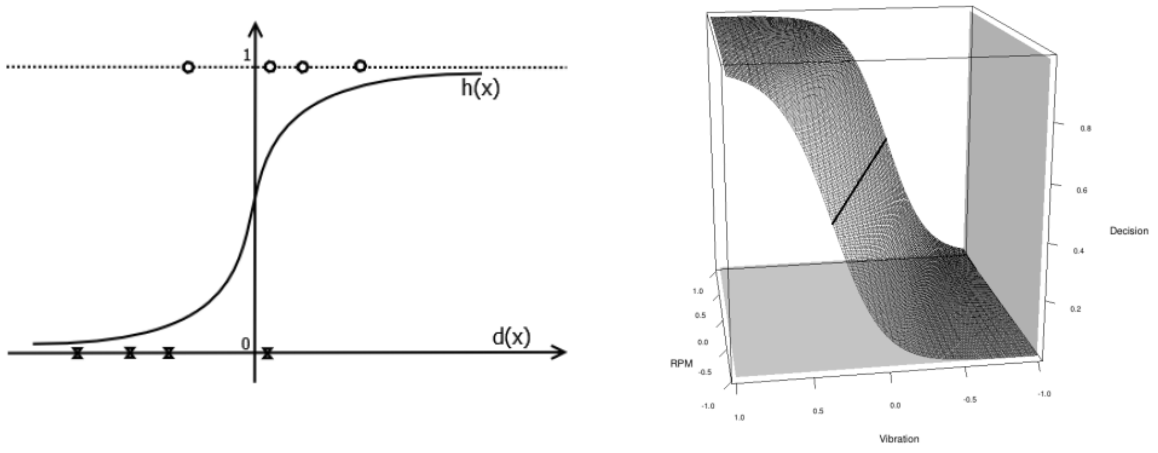


Figure 2.12: Model as a logistic function

It is not a good cost function for Logistic regression because it is non-convex for a Logistic function (Figure 2.14).

The logistic regression loss function is calculated using this formula

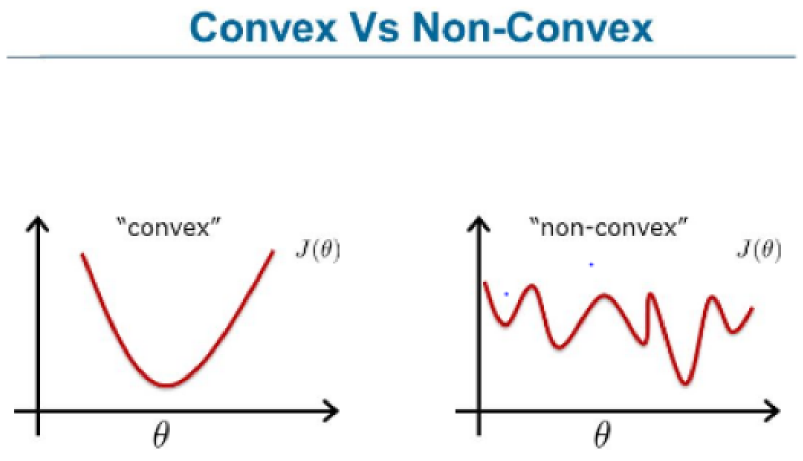


Figure 2.13: Convex and non-convex functions

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} Cost(h_\theta(x^{\{i\}}), y^{\{i\}})$$

where:

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$
$$= -[y\log(h_\theta(x)) + (1-y)\log(1-h_\theta(x))]$$
$$y^{\{*\}} \in \{0, 1\}$$

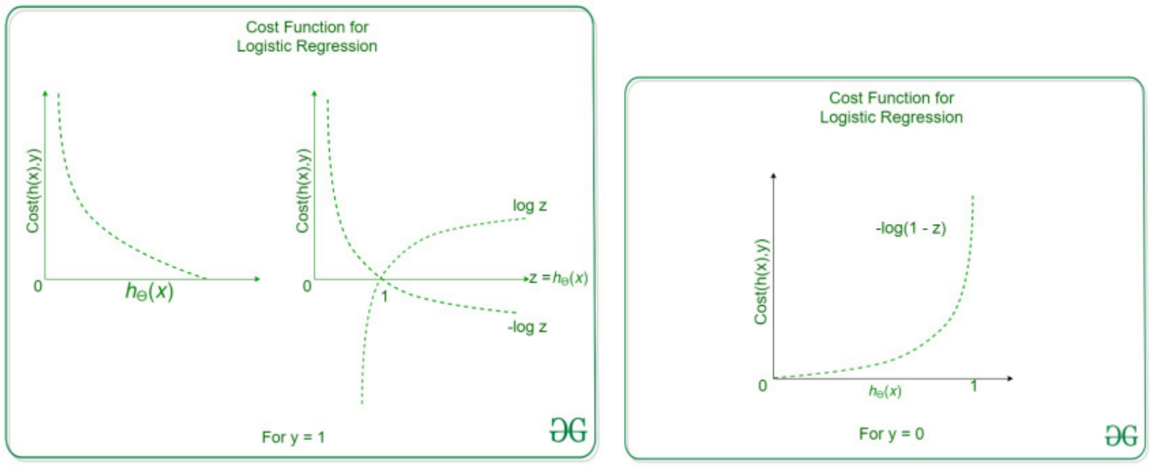


Figure 2.14: Cost function for Logistic regression

Cost function for Logistic regression formula:

$$Cost(h_\theta(x), y) = -[y\log(h_\theta(x)) + (1-y)\log(1-h_\theta(x))]$$

2.2.5 Stochastic gradient descend

Now we have a cost function which defines the errors of a model over the training data. Next we need to search all the possible models to find a model with the minimal cost. We use a gradient descend algorithm for this purpose.

The essence of the gradient descent is to find the minimum. Step by step, the algorithm searches for the minimum value of the function (Figure 2.15).

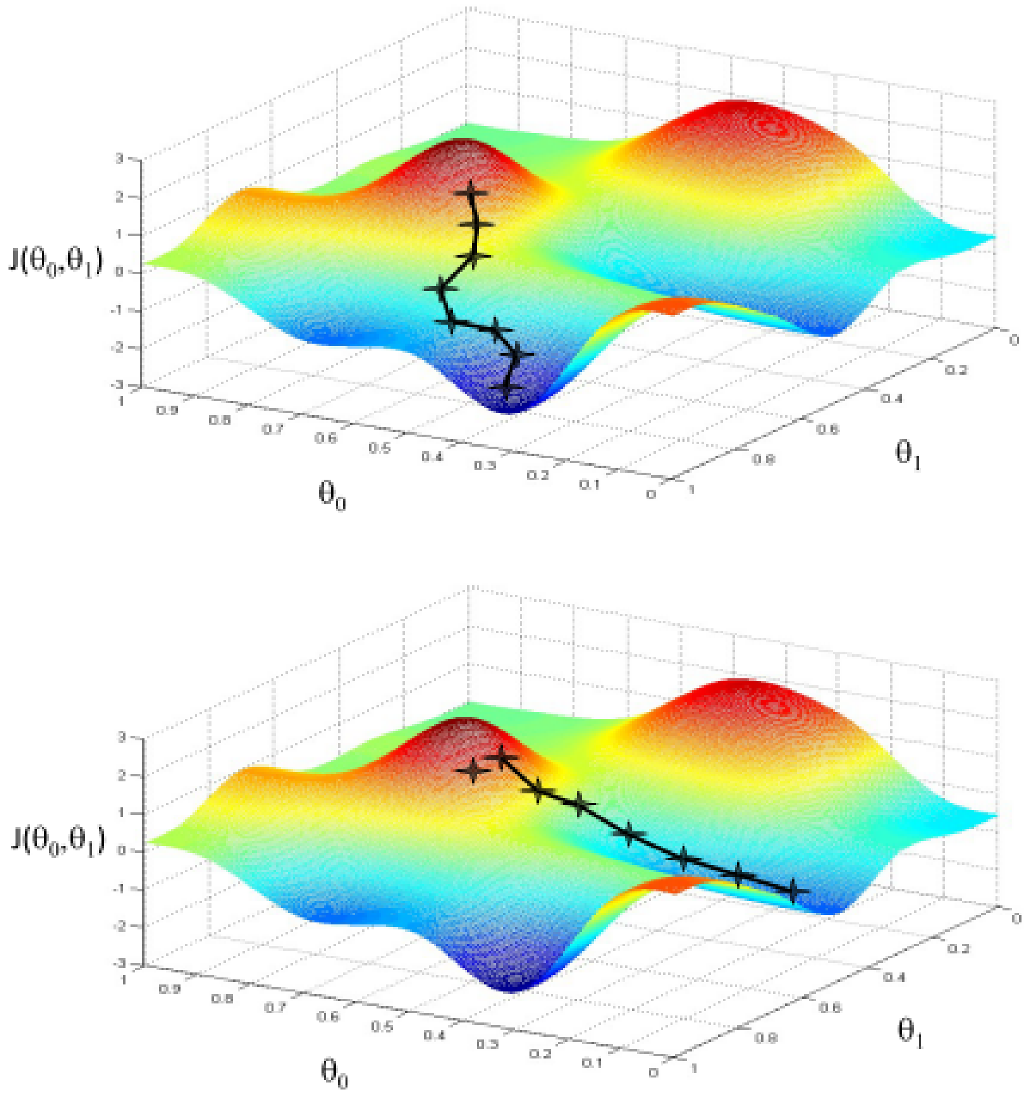


Figure 2.15: Different start points of SGD brings you different local minimums

$$\text{Have some function } J(\theta_0, \theta_1)$$
$$\text{Want } \min J(\theta_0, \theta_1)$$

Outline:

Start with some θ_0, θ_1
Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
until we hopefully end up at a minimum.

Gradient descent algorithm (Figure 2.16):

repeat until convergence
$\{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
(for $j = 0$ and $j = 1$)$\}$

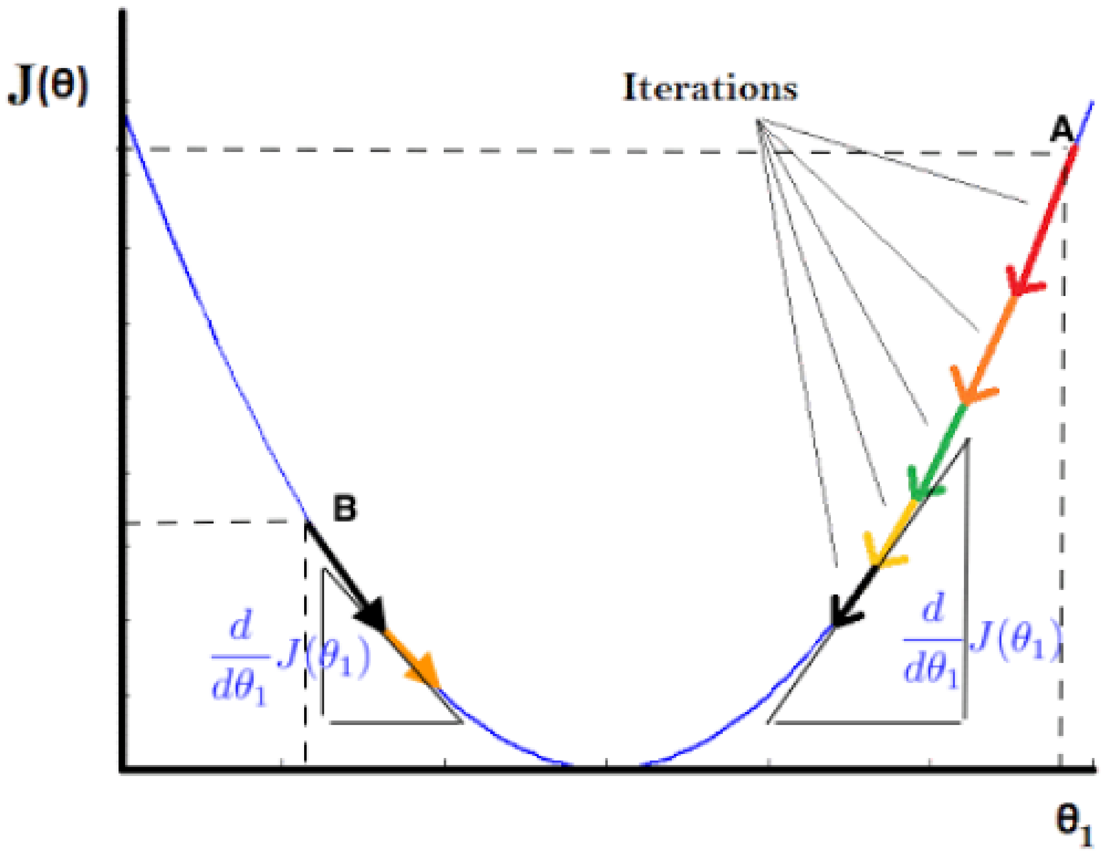

J(θ)
Iterations
A
B
$\frac{d}{d\theta_1}J(\theta_1)$
$\frac{d}{d\theta_1}J(\theta_1)$
θ1

Figure 2.16: SGD iterations

- The derivation $\frac{\partial J(\theta)}{\partial \theta}$ gives the direction of the movement.
- The learning rate α is used to adjust the size for each step.

Gradient descend for Logistic regression:

Model

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

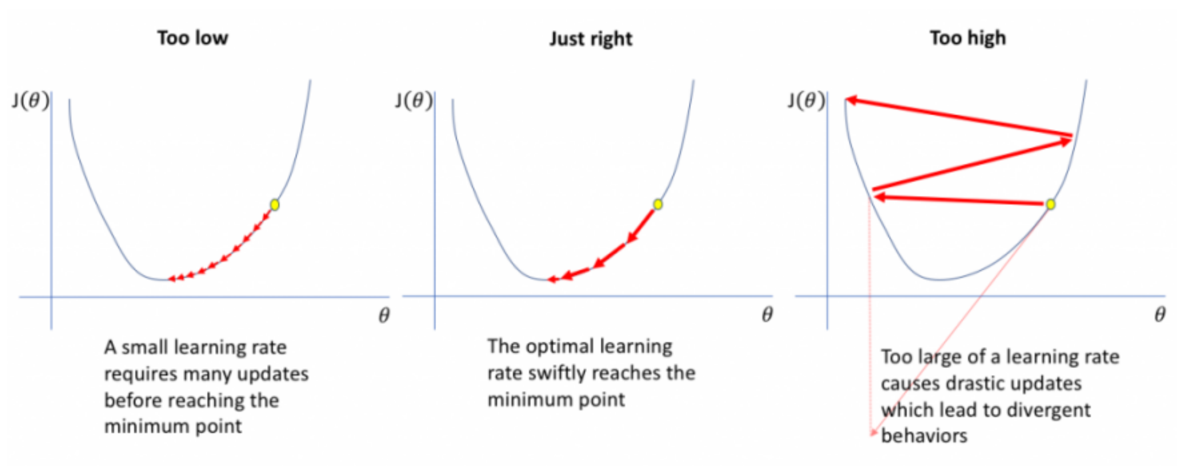

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 2.17: The influence of the learning rate

Parameters

$$\theta_0, \theta_1$$

Cost Function

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{\{i\}}\log(h_\theta(x^{\{i\}})) + (1-y^{\{i\}})\log(1-h_\theta(x^{\{i\}}))\right]$$

Goal

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta)$$

Gradient descent:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{\{i\}}\log(h_\theta(x^{\{i\}})) + (1-y^{\{i\}})\log(1-h_\theta(x^{\{i\}}))\right]$$

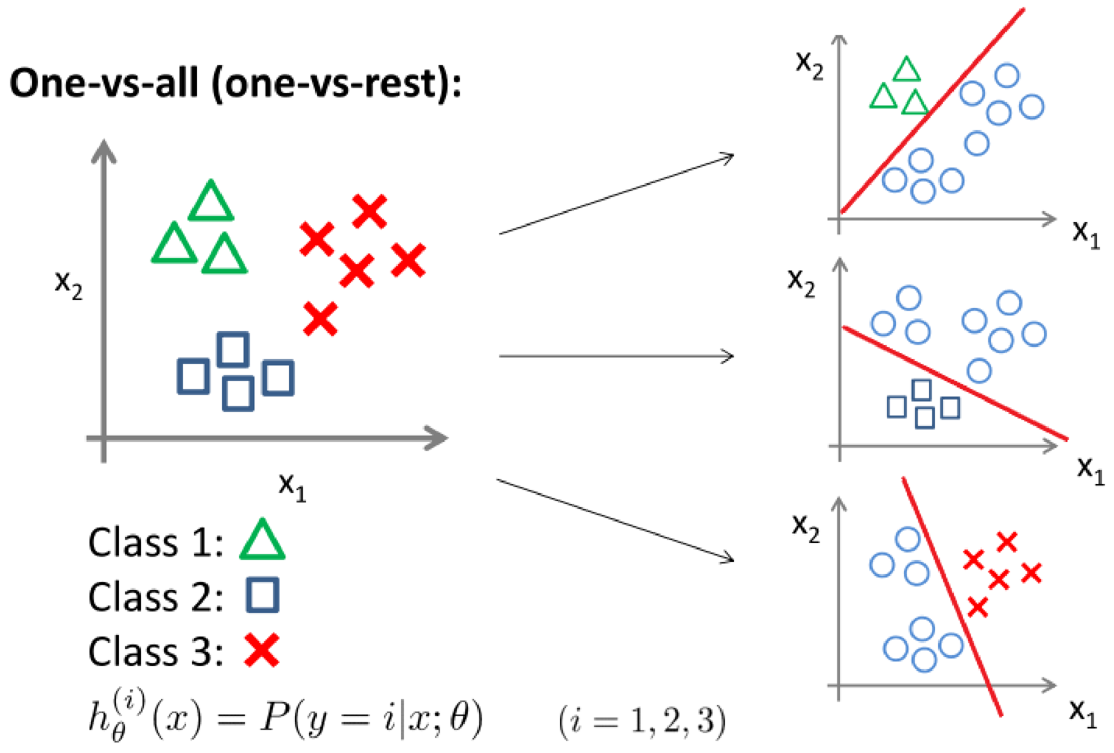
Want $\min_\theta J(\theta)$: Repeat { $\theta_j = \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}(h_\theta(x^{\{i\}}) - y^{\{i\}})x_j^{\{i\}}$ (simultaneously update all θ_j) }

2.2.6 Multiclass classification

- Email foldering/tagging: Work, Friends, Family, Hobby

$$y = \begin{cases} 1, & Work \\ 2, & Friends \\ 3, & Family \\ 4, & Hobby \end{cases}$$

- Medical diagrams: Not ill, Cold, Flu
- Weather: Sunny, Cloudy, Rain, Snow



One-vs-all (one-vs-rest):
x2
x1
Class 1: △
Class 2: □
Class 3: ✕
$h_\theta^{(i)}(x) = P(y = i|x;\theta) \quad (i = 1, 2, 3)$

Figure 2.18: One-vs-all classification

One-vs-all (Figure 2.18)

Train a logistic regression classifier $h_\theta{}^{(i)}(x)$ for each class i to predict the probability that $y = i$. On a new input x, to make a prediction, pick the class i that maximizes $\max h_\theta{}^{(i)}(x)$

2.3 Text classification

Text classification is the task of assigning a sentence or document an appropriate category (Figure 2.19). The categories depend on the chosen dataset and can range from topics.

A few examples of applications for text classification in business include for example:

- Junk email filtering
- News topic classification
- Authorship attribution
- Sentiment analysis
- Genre classification
- Offensive language identification
- Language identification

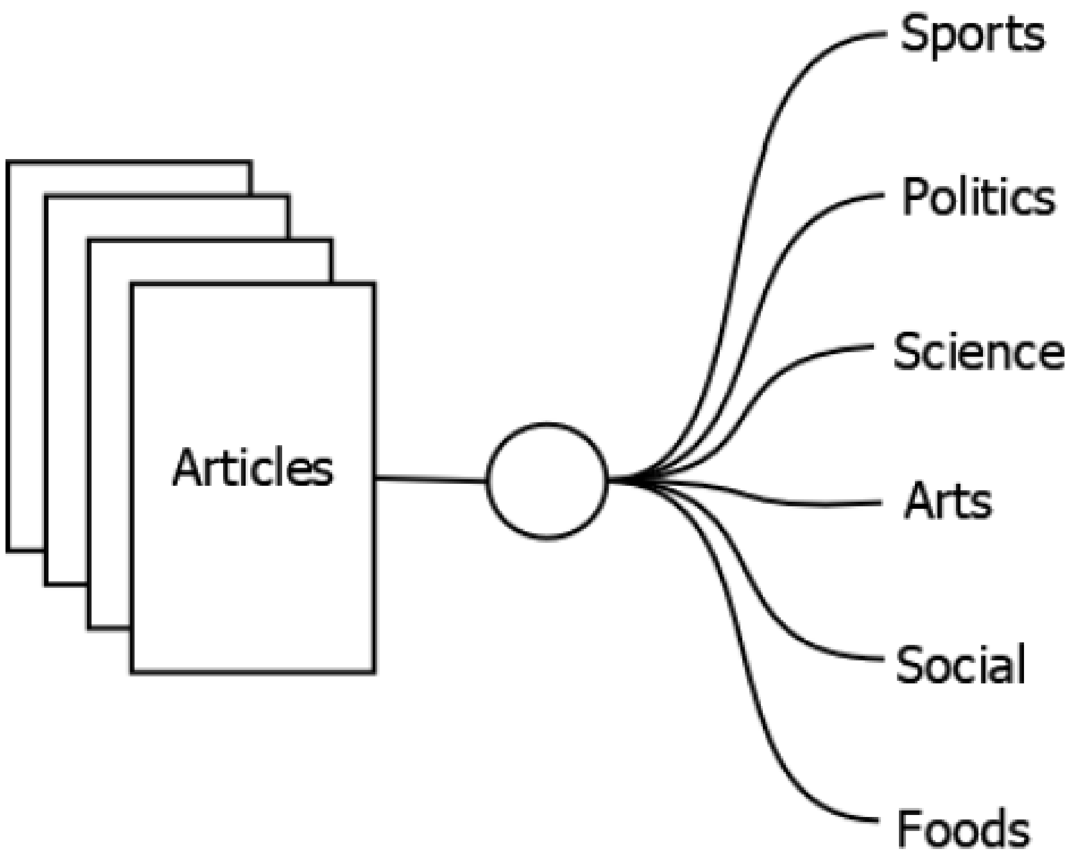


Figure 2.19: Typical example of text classification.

Typical pipeline of **text classification procedure** is:

- Text preprocessing
- Feature extraction
- Model training
- Model Application
- Evaluation

Text preprocessing usually include text cleaning (removing HTML/XML tags, figures, formula, etc.), removing stop words, tokenization and stemming. Stop words are a set of commonly used words in any language. For example, in English, "the", "is" and "and", would easily qualify as stop words. In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead. These words are not helpful for text classification because they occur in almost all documents, and should be removed before applying a text classification algorithm.

Feature extraction step means to extract and produce feature representations that are appropriate for the type of NLP task you are trying to accomplish and the type of model

doc-id	book	read	music	go
doc-1	3	1	0	5
doc-2	2	5	3	0
doc-3	0	0	7	2

Table 2.6: Words as features

you are planning to use. In case of text classification task each document is represented as a vector in order to applying a classification algorithm, each dimension of the input vector is called a feature. In text classification, the most straightforward idea is to use words as features (Table 2.6).

There is a different ways for **weighting of words** in document vectors:

Boolean weighting: $$w_{ik} = \begin{cases} 1 & \text{if } f_{ik} > 0 \\ 0 & \text{Otherwise} \end{cases}$$

Word frequency weighting: $$w_{ik} = f_{ik}$$

TF.IDF weighting: $$w_{ik} = f_{ik} \times \log \frac{N}{n_i}$$

i: word index
k: document index
f_{ik}: word frequency in a document
N: number of documents in the corpus
n_i: number of documents containing the word

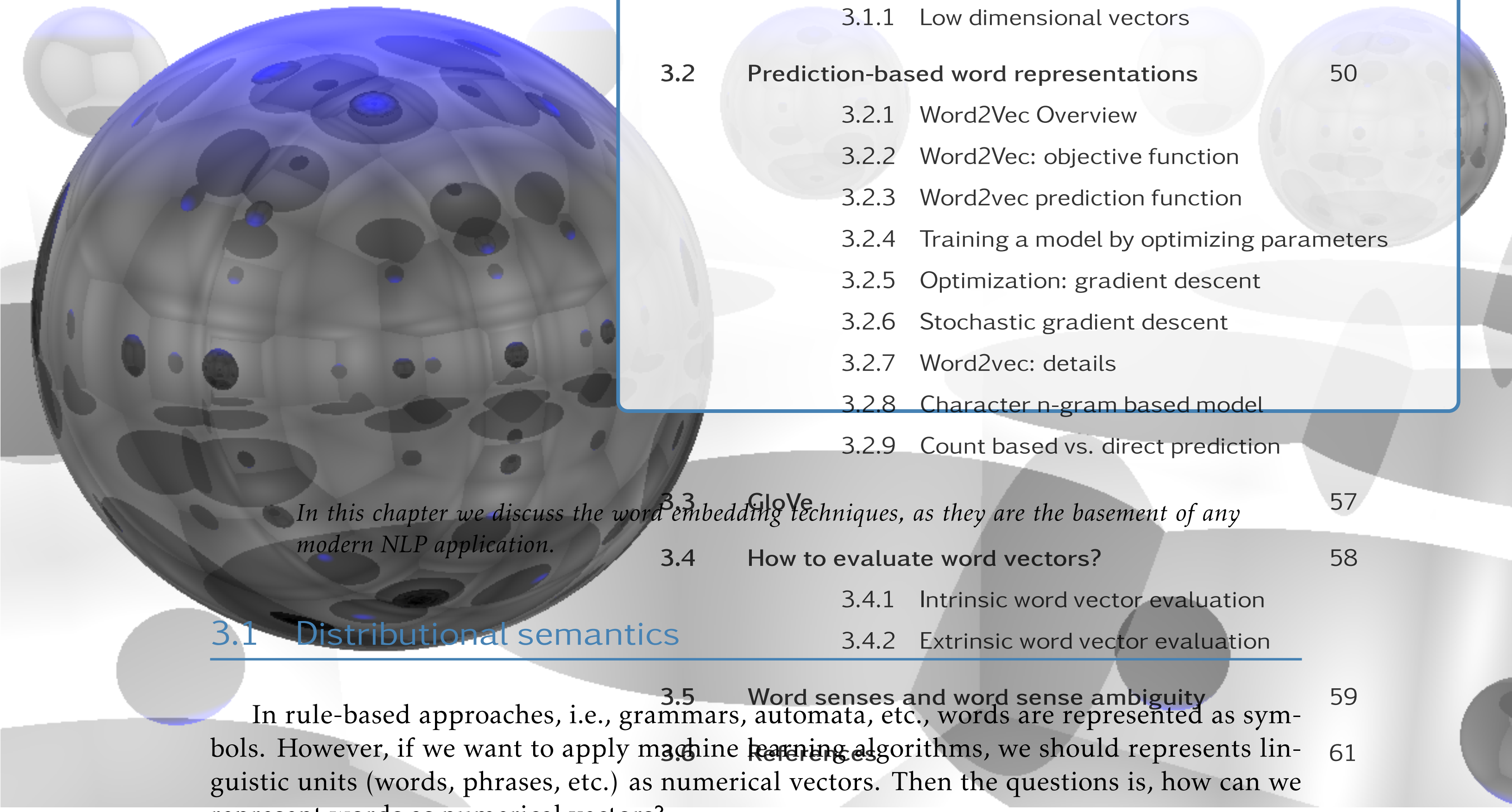
TF.IDF is short for **t**erm **f**requency–**i**nverse **d**ocument **f**requency. It's designed to reflect how important a word is to a document in a collection or corpus. The **TF.IDF** value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. The assumption behind the use of the inverse document frequency is: the more documents where a word (term) occurs, the less important the word is to that document. **TF.IDF** is proposed in information retrieval but also used in other areas including NLP. Which word weighting method is the best for text classification: no universal answer. It empirically depends on the data and the classification algorithm you use.

Algorithms what usually uses in text classification tasks:

- Logistic regression
- Nearest neighbor

- Decision trees
- Support vector machines
- Neural networks

3. Word Embeddings



In this chapter we discuss the word embedding techniques, as they are the basement of any modern NLP application.

3.1 Distributional semantics

In rule-based approaches, i.e., grammars, automata, etc., words are represented as symbols. However, if we want to apply machine learning algorithms, we should represents linguistic units (words, phrases, etc.) as numerical vectors. Then the questions is, how can we represent words as numerical vectors?

"You shall know a word by the company it keeps" (J. R. Firth, 1957)

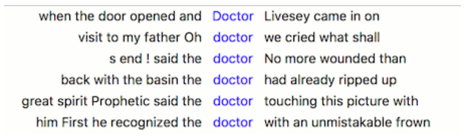
Distributional hypothesis: *Linguistic items with similar distributions have similar meanings.*

This hypothesis is a corner stone of the modern natural language processing approaches, starting from the ones of word meaning representation. Wikipedia says that *Distributional semantics* is a research area that develops and studies theories and methods for quantifying and categorizing semantic similarities between linguistic items based on their distributional properties in large samples of language data.

The most common approach nowadays is to collect distributional information in high-dimensional vectors, and define distributional/semantic similarity in terms of vector similarity. An example of so called concordance is presented at Fig. 5.31. The idea of this figure is to show that the contexts are semantically related for all the usages of specific word. Interestingly, the word Doctor in the very first sample is used as a part of person name, nevertheless it is still counts for semantic similarity.

There are a lot of possible distributional semantic models (or just word embeddings for short). They differ primarily with respect to the following parameters:

- Context type (text regions vs. linguistic items)



when the door opened and	Doctor	Livesey came in on
visit to my father Oh	doctor	we cried what shall
s end ! said the	doctor	No more wounded than
back with the basin the	doctor	had already ripped up
great spirit Prophetic said the	doctor	touching this picture with
him First he recognized the	doctor	with an unmistakable frown

Figure 3.1: Distributional semantics. A sample of concordance

- Context window (size, extension, etc.)
- Frequency weighting (e.g. entropy, pointwise mutual information, etc.)
- Dimension reduction (e.g. random indexing, singular value decomposition, etc.)
- Similarity measure (e.g. cosine similarity, Minkowski distance, etc.)

We will make more effort to discuss these differences below.

Let us consider an example. Let us make a window-based co-occurrence matrix. We take window length to be 1 (in practice 5-10 are more common sizes) and window itself to be symmetric, i.e. it is irrelevant for us whether a word stands to the left or to the right of target word. Our example corpus will be: *I like deep learning. I like NLP. I enjoy flying.* The table 3.1 is presenting the co-occurence matrix for this corpus and our limitations. We could already use this matrix as a set of vectors for the words in vocabulary. Although, this could be not a best decision, since there are issues, which cannot be solved with this approach:

- Matrix size increases with vocabulary growth (actually size of this matrix is $O(|V|)$);
- The vectors we have from it are very high dimensional, thus they require a lot of storage (although there are ways to solve this issue);
- Because the words are to present in each others context, the subsequent models will have sparsity issues;
- The models are of low robustness to new of rare words.

counts	**I**	**like**	**enjoy**	**deep**	**learning**	**NLP**	**flying**	**.**
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

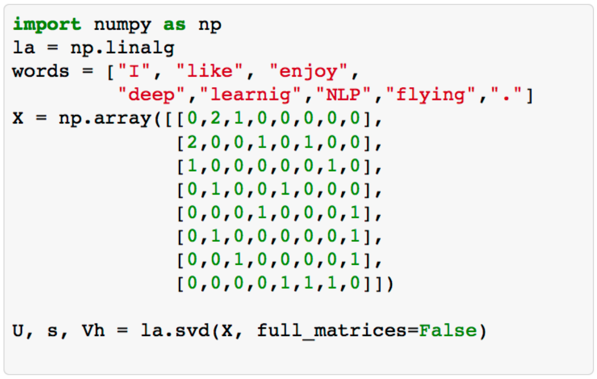
3.1.1 Low dimensional vectors

The issues with co-occurrence matrix lead us to the idea of overcoming the sparsity and high dimensionality. To achieve this the most obvious way is to reduce dimensionality of the matrix. It is usual to reduce matrix dimensionality to 25-1000 (word2vec model we will be discussing later typically use this number). The main idea of dimensionality reduction is to store “most” of the important information in a fixed, small number dimensions. The output of it we call a dense vector.

Let us have a look on the most popular method of dimensionality reduction - Singular Value Decomposition (SVD). Singular Value Decomposition of co-occurrence matrix X factorizes X into $U\Sigma V^T$, where U and V are “tall” and “wide” respectively. Σ matrix contains the eigenvalues of the original matrix X. This matrix is typically transformed to order the eigenvalues in descent.

$$\left[\hat{X} = X^k\right] = \left[U\right]\left[\Sigma\right]\left[V^T\right]$$

Retain only k singular values, in order to generalize. $\hat{X}$ is the best rank k approximation to X, in terms of least squares. Although, be aware that this operation is expensive to compute for large matrices.



```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

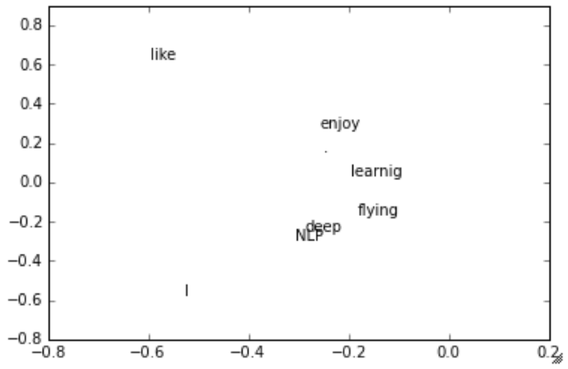
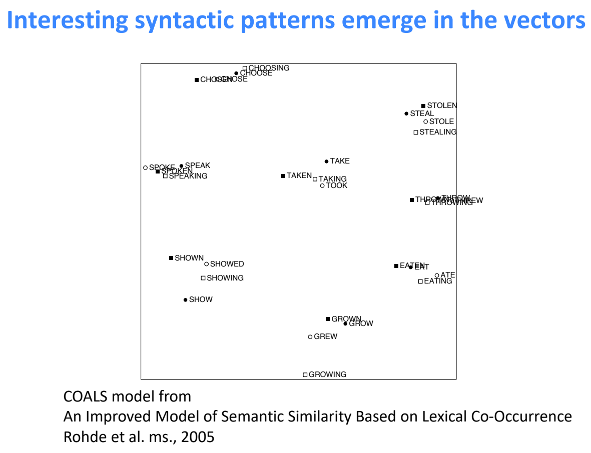


Figure 3.2: Simple SVD word vectors in Python.

There are several hacks which could be recommended to handle X (some of them are used in Rodhe et al. 2005):

- Scaling the counts in the cells; this sometimes can help a lot.
- So called function words (e.g. the, he, has) are too frequent, so syntax has too much impact on their meaning. There are some fixes:
  - You could subsample this words, thus using the capping limit of, say, 100;
  - You can ignore them at once.
- Inctead of standard uniform window, you can use so called ramped window where the closer to the target words weight more.
- Another option is to use Pearson correlation coefficient instead of word counts; if you do so, don’t forget to set negative values to 0.

Actually, there are more, you could easily found literature on this question in numerous sources.



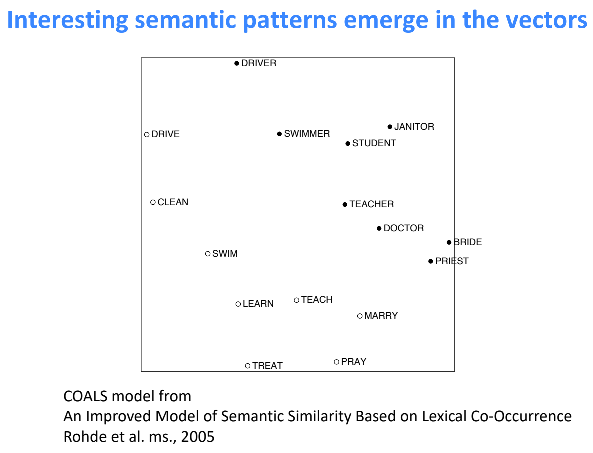


Figure 3.3: Interesting patterns emerge in the vectors

The vectors used in distributional semantics are based on frequencies, which are also called count-based representations. Count-based representations were successful in many similarity-related tasks, however, their usage was not able to extended to other NLP tasks. In order to take full advantages of machine learning (including deep learning) approaches, it is a necessity to represent words solely as vectors. Thus, we must use vectors to replace words completely, and get rid of symbols in computing, except for the output layer.

If a word can be predicted basing on the vectors of its content words, then we can expect to use the vectors to replace the words in any NLP task.

3.2 Prediction-based word representations

Some typical (although not exact) algorithm to obtain a model with word vector representations could be as follows:

- Randomly assign a vector to each word in the vocabulary;
- Prepare a corpus;
- For each of the words (referred to as the current word) in the corpus, repeat:
  - Calculate the probability of all content words given the current word (or vice versa);
  - Adjust the word vectors to maximize the above probability.

Various predict-based *word representations*, or *word embeddings*, are developed, including but not by any chance limited to: Word2Vec, GloVe, FastText, etc. We will discuss the mentioned models later in this chapter.

In some cases, the pretrained word embeddings (like Word2Vec) can be directly used to solve NLP problems. However, this is not always the case. Instead, word embeddings are defined as components of the whole NLP system, whose parameters are tuned together with