UNIVERSITY
OF LONDON

Introduction to natural language processing

R. Kibble

CO3354

2013

Undergraduate study in
Computing and related programmes

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London by:

R. Kibble

This is one of a series of subject guides published by the University. We regret that due to pressure of work the author is unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

In this and other publications you may see references to the 'University of London International Programmes', which was the name of the University's flexible and distance learning arm until 2018. It is now known simply as the 'University of London', which better reflects the academic award our students are working towards. The change in name will be incorporated into our materials as they are revised.

University of London
Publications Office
32 Russell Square
London WC1B 5DN
United Kingdom

london.ac.uk

Published by: University of London

Contents

Preface

About this half unit

This half unit course combines a critical introduction to key topics in theoretical and computational linguistics with hands-on practical experience of using existing software tools and developing applications to process texts and access linguistic resources. The aims of the course and learning outcomes are listed in Chapter 1. This course has no specific prerequisites. There will be some programming involved and you will need to acquire some familiarity with the Python language, but you will not be expected to develop substantial original code or to encode specialised algorithms. The course involves some statistical techniques, but the only mathematical knowledge assumed is an understanding of elementary probability and familiarity with the concept of logarithms.

Before the advent of the world wide web, most machine-readable information was stored in structured databases and accessed via specialised query languages such as Structured Query Language (SQL). Nowadays the situation is reversed: most information is found in unstructured or semi-structured natural language documents and there is increasing demand for techniques to 'unlock' this data. Computing graduates with knowledge of natural language processing techniques are finding employment in areas such as text analytics, sentiment analysis, topic detection and information extraction.

Assessment

The course is assessed via an unseen written examination. A sample examination paper is provided in the Appendix at the end of this subject guide, with some guidelines on how to answer the questions. You will be required to attempt three questions out of a choice of five. The questions will cover 'book knowledge', problem solving and short essays on more theoretical topics. The examination is not a memory test but will be designed to assess your understanding of the course content. There will also be coursework which will include a similar mix of questions, but with a stronger focus on practical problem-solving.

You will be expected to provide electronic copies of your coursework for plagiarism checking purposes. It is very important that any material that is not original to you should be properly attributed and placed in quotation marks, with a full list of references at the end of your submission. You should follow the style used in this subject guide for citing references, for example:

> *Segaran (2007, pp.117–118) discusses some problems with rule-based spam filters.*

Answers which consist entirely or mostly of quoted material are unlikely to get many marks even if properly attributed, as simply reproducing an answer in someone else's words does not demonstrate that you have fully understood the material.

In order to give you some practice in problem-solving and writing short essays, there

are a number of Activities throughout this subject guide. The Appendix includes a section 'Answers to selected activities', although these will not always provide complete answers to the questions but are intended to indicate how particular types of questions should be approached. Sample examination questions are provided at the end of each chapter. Some, but not all, of these are included in the sample examination paper with suggested answers at the end of the guide.

The subject guide and other learning resources

This subject guide is not intended as a self-contained textbook but sets out specific topics for study in the CO3354 half unit. There is a recommended textbook and a number of other readings are listed at appropriate places. There are also links to websites providing useful resources such as software tools and access to online linguistic data. The learning outcomes listed in the next chapter assume that you are working through the recommended readings, activities and sample examination questions. It will not be possible to pass this half unit by reading only the subject guide. Please refer to the Computing VLE for other resources, which should be used as an aid to your learning.

Suggested study time

The Student Handbook states that 'To be able to gain the most benefit from the programme, it is likely that you will have to spend at least 300 hours studying for each full unit, though you are likely to benefit from spending up to twice this time'. Note that this subject is a half unit.

The course is designed to be delivered over a ten-week term as one of four concurrent modules, and this guide has six chapters. Chapter 1 goes into more detail about the structure of the guide and the course, while Chapters 2 to 6 are each dedicated to a particular topic. It is suggested that you spend about two weeks on Chapters 1 and 2 together and each of Chapters 3 to 6, including the associated reading and web-based material, and work through the activities and sample examination questions during this time.

Acknowledgement

This subject guide draws closely on:

Bird, S., E. Klein and E. Loper, *Natural Language Processing with Python*. (O'Reilly Media 2009) [ISBN 9780596516499; http://nltk.org/book].

Chapter 1

Introduction: how to use this subject guide

1.1 Introduction

The idea of computers being able to understand ordinary languages and hold conversations with human beings has been a staple of science fiction since the first half of the twentieth century and was envisaged in a classic paper by Alan Turing (1950) as a hallmark of computational intelligence. Since the start of the twenty-first century this vision has been starting to look more plausible: artificial intelligence techniques allied with the scientific study of language have emerged from universities and research laboratories to inform a variety of industrial and commercial applications. Many websites now offer automatic translation; mobile phones can appear to understand spoken questions and commands; search engines like Google use basic linguistic techniques for automatically completing or 'correcting' your queries and for finding relevant results that are closely matched to your search terms. We are still some way from full machine understanding of natural language, however. Automated translations still need to be reviewed and edited by skilled human translators while no computer system has yet come close to passing the 'Turing Test' of convincingly simulating human conversation. Indeed it has been argued that the Turing Test is a blind alley and that research should focus on producing effective applications for specific requirements without seeking to generate an illusion that users are interacting with a human rather than a machine (Hayes and Ford, 1995). Hopefully, by the time you finish this course you will have come to appreciate some of the challenges posed by full understanding of natural language as well as the very real achievements that have resulted from focusing on a range of specific, well-defined tasks.

1.2 Aims of the course

This course combines a critical introduction to key topics in theoretical linguistics with hands-on practical experience of developing applications to process texts and access linguistic resources. The main topics covered are:

- accessing text corpora and lexical resources
- processing raw text
- categorising and tagging
- extracting information from text
- analysing sentence structure.

1.3 Learning outcomes

On successful completion of this course, including recommended readings, exercises and activities, you should be able to:

1. utilise and explain the function of software tools such as corpus readers, stemmers, taggers and parsers
2. explain the difference between regular and context-free grammars and define formal grammars for fragments of a natural language
3. critically appraise existing Natural Language Processing (NLP) applications such as chatbots and translation systems
4. describe some applications of statistical techniques to natural language analysis, such as classification and probabilistic parsing.

Each main chapter contains a list of learning outcomes specific to that chapter at the beginning, as well as a summary at the end of the chapter.

1.4 Reading list and other learning resources

This is a list of textbooks and other resources which will be useful for all or most parts of the course. Additional readings will be given at the start of each chapter. See the bibliography for a full list of books and articles referred to, including all ISBNs. In some cases several different books will be listed: you are not expected to read all of them, rather the intention is to give you some alternatives in case particular texts are hard to obtain.

Essential reading

Bird, Klein, and Loper (2009): *Natural Language Processing with Python*. The full text including diagrams is freely available online at *http://nltk.org/book* (last visited 13th April 2013). The main textbook for this course, *Natural Language Processing with Python* is the outcome of a project extending over several years to develop the Natural Language Toolkit (NLTK), which is a set of tools and resources for teaching computational linguistics. The NLTK comprises a suite of software modules written in Python and a collection of corpora and other resources. See section 1.5 below for advice on installing the NLTK and other software packages.

In the course of working through this text you will gain some experience and familiarity with the Python language, though you will not be expected to produce substantial original code as part of the learning outcomes of the course.

Recommended reading

Pinker (2007). *The Language Instinct*. This book is aimed at non-specialists and deals with many psychological and cultural aspects of language. Chapter 4 is particularly relevant to this course as it provides a clear and accessible presentation of two standard techniques for modelling linguistic structure: finite-state machines and context-free grammars (though Pinker does not in fact use these terms, as we will see in Chapter 2 of the subject guide).

Jurafsky and Martin (2009): *Speech and Language Processing*, second edition. Currently the definitive introductory textbook in this field, covering the major topics in a way which combines theoretical issues with presentations of key technologies, formalisms and mathematical techniques. Much of this book goes beyond what you will need to pass this course, but it is always worth turning to if you're looking for a more in-depth discussion of any particular topics.

Perkins (2010): *Python Text Processing with NLTK 2.0 Cookbook*. This book will be suitable for students who want to get more practice in applying Python programming to natural language processing. Perkins explains several techniques and algorithms in more technical detail than Bird et al. (2009) and provides a variety of worked examples and code snippets.

Segaran (2007) *Programming Collective Intelligence*. This highly readable and informative text includes tutorial material on machine learning techniques using the Python language.

Additional reading

Russell and Norvig (2010) *Artificial Intelligence: a modern approach*, third edition. This book is currently regarded as the definitive textbook in Artificial Intelligence, and includes useful material on natural language processing as well as on machine learning, which has many applications in NLP.

Mitkov (2003) *The Oxford Handbook of Computational Linguistics*. Edited by Ruslan Mitkov. A collection of short articles on major topics in the field, contributed by acknowledged experts in their respective disciplines.

Partee et al. (1990) *Mathematical Methods in Linguistics*. A classic text, whose contents indicate how much the field has changed since its publication. A book with such a title nowadays would be expected to include substantial coverage of statistics, probability and information theory, but this text is devoted exclusively to discrete mathematics including set theory, formal logic, algebra and automata. These topics are particularly applicable to the content of Chapters 2 and 6.

Websites

Introductory/Reference *The Internet Grammar of English* is a clear and informative introductory guide to English grammar which also serves as a tutorial in grammatical terminology and concepts. The site is hosted by the Survey of English Usage at University College London (http://www.ucl.ac.uk/internet-grammar/home.htm, last visited 27th May 2013).

Hands-on corpus analysis

BNCWeb is a web-based interface to the British National Corpus hosted at Lancaster University which supports a variety of online queries for corpus analysis (http://bncweb.info/; last visited 27th May 2013).

The Bank of English forms part of the Collins Corpus, developed by Collins Dictionaries and the University of Birmingham. Used as a basis for Collins Advanced Learner's Dictionary, grammars and various tutorial materials for learners of English. Limited online access at http://www.collinslanguage.com/wordbanks; (last visited 27th May 2013).

Journals and conferences

Computational Linguistics is the leading journal in this field and is freely available at http://www.mitpressjournals.org/loi/coli (last visited 27th May 2013).

Conference Proceedings are often freely downloadable and many of these are hosted by the ACL Anthology at http://aclweb.org/anthology-new/ (last visited 27th May 2013).

1.5 Software requirements

This course assumes you have access to the Natural Language Toolkit (NLTK) either on your own computer or at your institution. The NLTK can be freely downloaded and it is strongly recommended that you install it on your own machine: Windows, Mac OSX and Linux distributions are available from `http://nltk.org` (last visited April 10th 2013) and some distributions of Linux have it in their package/software managers. Full instructions are available at the cited website along with details of associated packages which should also be installed, including Python itself which is also freely available. Once you have installed the software you should also download the required datasets as explained in the textbook (Bird et al., 2009, p. 3).

You should check the NLTK website to determine what versions of Python are supported. Current stable releases of NLTK are compatible with Python 2.6 and 2.7. A version supporting Python 3 is under development and may be available for testing by the time you read this guide (as of April 2013).

1.6 How to use the guide/structure of the course

This section gives a brief summary of each chapter. These learning outcomes are listed at the beginning of each main chapter and assume that you have worked through the recommended readings and activities for that chapter.

1.6.1 Chapter 2: Introducing NLP: patterns and structures in language

This chapter looks at different approaches to analysing texts, ranging from 'shallow' techniques that focus on individual words and phrases to 'deeper' methods that produce a full representation of the grammatical structure of a sentence as a hierarchical tree diagram. The chapter introduces two important formalisms: **regular expressions**, which will play an important part throughout the course, and **context-free grammars** which we return to in Chapter 6 of the subject guide.

1.6.2 Chapter 3: Getting to grips with natural language data

This chapter looks at the different kinds of data resources that can be used for developing tools to harvest information that has been published as machine-readable documents. In particular, we introduce the notion of a 'corpus' (plural *corpora*) – for the purposes of this course, a computer-readable collection of text or speech. The NLTK includes a selection of excerpts from several well-known corpora and we provide brief descriptions of the most important of these and of the different formats in which corpora are stored.

1.6.3 Chapter 4: Computational tools for text analysis

The previous chapter introduced some relatively superficial techniques for language analysis such as concordancing and collocations. This chapter covers some fundamental operations in text analysis:

- tokenisation: breaking up a character string into words, punctuation marks and other meaningful expressions;
- stemming: removing affixes from words, e.g. *mean+ing*, *distribut+ion*;
- tagging: associating each word in a text with a grammatical category or part of speech.

1.6.4 Chapter 5: Statistically-based techniques for text analysis

Statistical and probabilistic methods are pervasive in modern computational linguistics. These methods generally do not aim at complete understanding or analysis of a text, but at producing reliable answers to well-defined problems such as sentiment analysis, topic detection or recognising named entities and relations between them in a text.

1.6.5 Chapter 6: Analysing sentences: syntax and parsing

This chapter resumes the discussion of natural language syntax that was introduced in Chapter 2, concentrating on context-free grammar formalisms and various ways they need to be modified and extended beyond the model that was presented in that chapter. Formal grammars do not encode any kind of processing strategy but simply provide a declarative specification of the well-formed sentences in a language. Parsers are computer programs that use grammar rules to analyse sentences, and this chapter introduces some fundamental approaches to syntactic parsing.

1.6.6 Appendices

The Appendices include:

A. A bibliography listing all works referenced in the subject guide, including publication details and ISBNs.
B. A glossary of technical terms used in this subject guide.
C. Answers to selected activities.
D. A trace of a recursive descent parse as described in Chapter 6 of the subject guide.
E. A sample examination paper with guidelines on how to answer questions.

1.7 What the course does not cover

The field of natural language processing or computational linguistics is a large and diverse one, and includes many topics we will not be able to address in this course. Some of these are listed below:

- speech recognition and synthesis
- dialogue and question answering
- machine translation
- semantic analysis, including word meanings and logical structure
- generating text or speech from non-linguistic inputs.

However, the course should provide you with a basis for investigating some of these areas for your final year project. Some of these topics are dealt with in the later chapters of Bird et al. (2009) and most of them are touched on by Jurafsky and Martin (2009).

Chapter 2

Introducing NLP: patterns and structure in language

Essential reading

Steven Pinker (2007), *The Language Instinct*, Chapter 4.

Recommended reading

Jurafsky and Martin (2009), *Speech and Language Processing* second edition, Chapters/Sections 2 'Regular Expressions and Automata', 5.1 '(Mostly) English Word Classes', 12.1 'Constituency', 12.2 'Context-Free Grammars', 12.3 'Some Grammar Rules for English'.

Additional reading

- The Internet Grammar of English; *http://www.ucl.ac.uk/internet-grammar/home.htm* especially sections 'Word Classes' and 'Introducing Phrases'.
- Partee, ter Meulen and Wall, (1990), *Mathematical Methods in Linguistics*, Chapters/Sections 16.1–4, 17.1–3 (omitting 17.1.2–5, 17.2.1), 18.2, 18.6.

2.1 Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- explain the concept of **finite state machines** (FSMs) and their connections with regular expressions; work through simple FSMs
- write regular expressions for well-defined patterns of symbols
- analyse sentences in terms of parts of speech (POS) and constituent structure, including the use of tree diagrams
- write regular and context-free grammars for small fragments of natural language
- explain the concept of **stemming** and specify word-formation rules for given examples.

2.2 Introduction

People communicate in many different ways: through speaking and listening, making gestures, using specialised hand signals (such as when driving or directing traffic), using sign languages for the deaf, or through various forms of **text**.

By text we mean words that are written or printed on a flat surface (paper, card, street signs and so on) or displayed on a screen or electronic device in order to be read by their intended recipient (or by whoever happens to be passing by).

This course will focus only on the last of these: we will be concerned with various ways in which computer systems can analyse and interpret texts, and we will assume for convenience that these texts are presented in an electronic format. This is of course quite a reasonable assumption, given the huge amount of text we can access via the World Wide Web and the increasing availability of electronic versions of newspapers, novels, textbooks and indeed subject guides. This chapter introduces some essential concepts, techniques and terminology that will be applied in the rest of the course. Some material in this chapter is a little technical but no programming is involved at this stage.

We will begin in section 2.3 by considering texts as strings of characters which can be broken up into sub-strings, and introduce some techniques for informally describing patterns of various kinds that occur in texts. Subsequently in section 2.4 we will begin to motivate the analysis of texts in terms of hierarchical structures in which elements of various kinds can be embedded within each other, in a comparable way to the elements that make up an HTML web document. This section introduces some technical machinery such as: finite-state machines (FSMs), regular expressions, regular grammars and context-free grammars.

2.3 Basic concepts

2.3.1 Tokenised text and pattern matching

One of the more basic operations that can be applied to a text is **tokenising**: breaking up a stream of characters into words, punctuation marks, numbers and other discrete items. So for example the character string

“Dr. Watson, Mr. Sherlock Holmes”, said Stamford, introducing us.

can be tokenised as in the following example, where each token is enclosed in single quotation marks:

```
‘"’ ‘Dr.’ ‘Watson’ ‘,’ ‘Mr.’ ‘Sherlock’ ‘Holmes’ ‘"’ ‘,’
‘said’ ‘Stamford’ ‘,’ ‘introducing’ ‘us’ ‘.’
```

At this level, words have not been classified into grammatical categories and we have very little indication of syntactic structure. Still, a fair amount of information may be obtained from relatively shallow analysis of tokenised text. For example, suppose we want to develop a procedure for finding all personal names in a given text. We know that personal names always start with capital letters, but that is not enough to distinguish them from names of countries, cities, companies, racehorses

and so on, or from capitalisation at the start of a sentence. Some additional ways to identify personal names include:

- Use of a title *Dr.*, *Mr.*, *Mrs.*, *Miss*, *Professor* and so on.
- A capitalised word or words followed by a comma and a number, usually below 100: this is a common way of referring to people in news reports, where the number stands for their age – for example *Pierre Vinken, 61, . . .*
- A capitalised word followed by a verb that usually applies to humans: *said*, *reported*, *claimed*, *thought*, *argued* . . . This can over-generate in the case of country or organisation names as in *the Crown argues* or *Britain claimed*.

We can express these more concisely as follows, where | is the disjunction symbol, *Word* stands for a capitalised word and *Int* is an integer:

- (Dr. | Professor | Mr. | Mrs. | Miss | Ms) *Word*
- *Word Word*, *Int*
- *Word* (said | thought | believed | claimed | argued | ...)

Learning activity

1. Write down your own examples of names that match each of the above patterns.
2. Pick a newspaper article or webpage that provides a variety of examples of people's names. Do they match the patterns we have encoded above? If not, see if you can devise additional rules for recognising names and write them out in a similar format.

2.3.2 Parts of speech

A further stage in analysing text is to associate every token with a grammatical category or **part of speech** (POS). A number of different POS classifications have been developed within computational linguistics and we will see some examples in subsequent chapters. The following is a list of categories that are often encountered in general linguistics: you will be familiar with many of them already from learning the grammar of English or other languages, though some terms such as *Determiner* or *Conjunction* may be new to you.

Noun fish, book, house, pen, procrastination, language

Proper noun John, France, Barack, Goldsmiths, Python

Verb loves, hates, studies, sleeps, thinks, is, has

Adjective grumpy, sleepy, happy, bashful

Adverb slowly, quickly, now, here, there

Pronoun I, you, he, she, we, us, it, they

Preposition in, on, at, by, around, with, without

Conjunction and, but, or, unless

Determiner the, a, an, some, many, few, 100

Bird et al. (2009, pp. 184–5) make the standard distinction that nouns 'generally refer to people, places, things or concepts' while verbs 'describe events or actions'. This may be helpful when one is starting to learn grammatical terminology but is something of an over-simplification. One can easily find or construct examples where the same concept can be expressed by a noun or a verb, or by an adjective or an adverb. And on the other hand, there are many words that can take different parts of speech depending on what they do in a sentence:

1. Rome *fell swiftly*.
2. The *fall* of Rome was *swift*.
3. The enemy *completely destroyed* the city.
4. The enemy's *destruction* of the city was *complete*.
5. John likes to *fish* on the river bank.
6. John caught a *fish*.

Additionally, some types of verbs do not correspond to any particular action but serve a purely grammatical function: these include the auxiliary verbs such as *did*, *shall* and so on. So in summary, we can often only assign a part of speech to a word depending on its function in context rather than how it relates to real things or events in the world.

Learning activity

Identify parts of speech in these examples:

1. The cat sat on the mat.
2. John sat on the chair.
3. The dog saw the rabbit.
4. Jack and Jill went up the hill.
5. The owl and the pussycat went to sea.
6. The train travelled slowly.

2.3.3 Constituent structure

You will have noticed several recurring patterns in the above examples: *Det Noun*, *Prep Det Noun* and so on. You may also have noticed that some types of phrase can occur in similar contexts: *(John | the cat) sat*, a *Proper Noun* or a sequence *Det Noun* can come before a *Verb*. Some of these possibilities can be captured using the pattern-matching notation introduced above, for example:

(((the | a)(cat | dog))(John | Jack | Susan))(barked | slept)

This will match any sequence which ends in a verb *barked* or *slept* preceded by **either** a Determiner *a* or *the* followed by a Noun *cat* or *dog* **or** a proper name *John*, *Jack* or *Susan*.

Patterns that have similar distributions (meaning that they can occur in similar contexts) are standardly identified by phrasal categories such as *Noun Phrase* or *Verb*

Phrase. A common way to represent information about constituent structure is by means of production rules of the form $X \rightarrow A, B, C \ldots$. Using rules of this form, grammatical sentences can be broken down into constituent phrases consisting of various combinations of POS:

Sentence $\rightarrow$ Noun Phrase, Verb Phrase

Noun Phrase $\rightarrow$ Determiner, Noun (**Example:** the, dog)

Noun Phrase $\rightarrow$ Proper Noun (**Example:** Jack)

Noun Phrase $\rightarrow$ Noun Phrase, Conj, Noun Phrase (**Examples:** Jack and Jill, the owl and the pussycat)

Verb Phrase $\rightarrow$ Verb, Noun Phrase (**Example:** saw the rabbit)

Verb Phrase $\rightarrow$ Verb, Preposition, Noun Phrase (**Examples:** went up the hill, sat on the mat)

Learning activity

Read through the recommended sections of the UCL 'Internet Grammar of English'. Write production rules that cover some of the examples in these sections.

2.4 A closer look at syntax

This section aims to motivate the idea that texts can be analysed as hierarchical structures rather than 'flat' sequences whose elements are organised in various patterns. The Essential reading for this chapter by Steven Pinker gives a concise and accessible introduction to some fundamental distinctions we will make in this section, from the point of view of Chomskyan linguistics (compare Chomsky, 1957/2002). Chomsky and his followers argue that some components of our knowledge of language are innate, and Pinker (2007, chapter 4) sketches some arguments in support of this claim. This position is considered to be contentious by many linguists and we will not address it in this course. However, Pinker's chapter provides a useful introduction to syntactic analysis and clearly distinguishes between two formal techniques for modelling grammatical knowledge, which underlie **regular** and **context-free** grammars respectively (these terms will be explained as we go along).

Learning activity

If you have access to it, read through the recommended chapter by Pinker and make notes, and have it to hand while working through the remainder of this section.

Pinker notes that language makes 'infinite use of finite means', in Humboldt's phrase[1]. That is, there is no principled upper limit to the length of a grammatical sentence: we can always add another phrase, even if it's a banal one like 'one could say that', 'and that's a fact' or 'and you can tell that to the Marines'. A large

[1] 'Sie [die Sprache] muss daher von endlichen Mitteln einen unendlichen Gebrauch machen' (von Humboldt, 1836, p. 122).

proportion, perhaps most of the sentences we read, hear or speak every day may be entirely novel, at least to us. Consequently, knowledge of a language seems to consist in knowing rules that specify what sentences belong to the language, rather than memorising long lists of sentences to be produced on appropriate occasions.

Pinker considers two different formal systems for generating or recognising sentences in English:

- 'wordchain' devices, equivalent to finite state machines. These devices incorporate three distinct operations: **sequence**, **selection** and **iteration**.
- Phrase structure grammars, which include the additional operation of **recursion**.

Note that Pinker deliberately uses the more descriptive expression 'wordchain' as he is concerned to avoid the use of forbidding technical terminology. In what follows we will stick to the standard term *finite-state machine* which you are more likely to find in textbooks. You may also encounter the terms *finite-state automaton* or just *finite automaton*.

2.4.1 Operation of a finite-state machine

A wordchain device or finite-state machine (FSM) can be seen as a set of lists of symbols (such as words or fixed phrases) and rules for going from list to list. A simple example:

Word lists

1. The, a, one
2. Cat, dog, fish
3. Barked, slept, swam

Rules

It is important to keep in mind that FSMs are neutral between accepting and generating strings. That is to say, one way to operate a FSM is to read a string, one symbol at a time, and determine whether the symbol is found in the list at the current state of the machine. If it is, we advance to the next state and read the next symbol. Alternatively, this FSM could be used to generate strings by picking one word from each list in sequence. Some possible matching strings are:

- The dog swam
- A cat barked
- A fish slept
- . . .

A more complex example:

1. *John/Mary/Fred* **OR**

   1a. *the/a/one*

   1b. *cat/dog/fish*

2. (optional): *and/or* **GO TO 1**
3. *slept/barked/swam* **OR**

   3a. *sat/walked*

   3b. *on*

   3c. *a/the*

   3d. *mat/hill*
4. (optional) *and/or* **GO TO 3**
5. (optional) *and/or/but* **GO TO 1**.

This formulation involves the basic operations of sequence, selection and iteration as follows:

SEQUENCE

Moving from list to list in numerical order: 1, 2, 3 . . .

SELECTION

Choosing an item from a list, for example *cat*, *dog* or *fish*; choosing between lists.

ITERATION

Repeating particular sequences, for example:

- John and Mary or a fish *(repeats step 1.)*
- The cat barked but Fred walked on the hill. *(Repeats steps 1–5, omitting step 4.)*

Learning activity

1. Find the shortest sentence generated or accepted by the above FSM.
2. Write out four sentences between six and 20 words long which are accepted by the FSM.

2.4.2 Representing finite-state machines

There are various conventional ways of representing a non-deterministic FSM in terms of a number of states and the permissible transitions between states. In our informal exposition above, the numbered steps represent states and each symbol or word in a list counts as a possible transition to the next state. Pinker adopts a graphical convention where states are depicted as nodes in a graph and transitions are directed, labelled arcs between the nodes; see also Partee et al. (1990, p. 457 and following). Alternatively, the states and transitions can be shown in tabular form as in Table 2.1 where $q1$ is the initial state and $q4$ the final state:

q1	john	q2
q1	mary	q2
q1	the	q1a
q1	a	q1a
q1a	cat	q2
q1a	dog	q2
q2	slept	q3
q2	barked	q3
q2	swam	q3
q3	and	q1
q3	or	q1
q3	.	q4

Table 2.1: A finite-state machine represented as a state-transition table.

2.4.3 Declarative alternatives to finite-state machines

The FSMs shown above combine a formal specification of a language with a processing strategy. It is often convenient to separate the two and define the language using expressions from a declarative formalism which can be manipulated using various different algorithms. This section considers two such formalisms: *regular expressions* and *regular grammars*.

Regular expressions (REs) provide a simple but powerful means of identifying patterns in text and are widely used in various applications of computer science. REs are based on three fundamental concepts which as we have seen are characteristic of finite-state machines:

sequence – to do with the order in which items occur: may include a wildcard character which is written as the period or full stop '.' and may be replaced by any character.

selection – specifying a choice between alternative items or sequences, indicated by the '|' operator

iteration – repetition of items or sequences, indicated by the '*' operator, meaning zero or more occurrences of whatever precedes the star.

Some simple examples:

a* matches sequences of zero or more **a**'s: **a**, **aaaa**, **aaaaaaaaaa** and so on. A sequence of zero elements is known as the 'empty string' and conventionally denoted by the Greek letter *epsilon* or ϵ.

aa* sequences of one or more **a**'s

ab* sequences of one **a** followed by zero or more **b**'s: **a**, **ab**, **abbbb**, ...

(ab)* sequences of zero or more pairs **ab**: ϵ, **ab**, **abab**, **ababab** ...

(ab)|(ba) **ab** or **ba**

((ab)|(ba))* possibly empty sequences of **ab** and **ba** pairs: ϵ, **ab**, **abab**, **baab**, **bababa**, **abba** ... Note that parentheses operate in the usual manner as in mathematical or logical expressions, to denote the **scope** of operators.

b.*a all strings that start with **b** and end with **a**: **ba**, **bbbaaaa**, **bccccccca** ...

Programming languages such as Java, Perl and Python implement extensions of REs with operators which are mostly redundant in that they can be reduced to combinations of the above operations, but can make programs much more compact and readable, including:

+ – one or more of the previous item

? – the previous item is optional

[A-Z], [0-9] – this expression matches one of a range of characters

^abc – matches pattern *abc* at the start of a string

abc$ – matches pattern *abc* at the end of a string.

See also Bird et al. (2009, Table 3.3) and the other recommended readings on this topic.

Here are some examples of our suggested ways of recognising personal names coded as regular expressions. These are intended to be applied to tokenised text and every sequence enclosed by angled brackets $< \ldots >$ stands for an individual token. In Examples 1, 3 and 4 below, the material within parentheses represents the target string and sequences outside parentheses provide the context.

1. <Mrs?>(<.+>) matches 'Mr' or 'Mrs' followed by any string. The first token consists of the sequence *Mr* followed optionally by the character *s*. The second consists of a sequence of one or more characters: any character may occur in place of the wildcard '.'.
2. <[A-Z][a-z]+>+ matches any sequence of one or more capitalised words.
3. (<[A-Z][a-z]+>+)<,><[0-9]+> matches capitalised word(s) followed by a comma and a number (age).
4. (<[A-Z][a-z]+>+)<said|reported|claimed>.

Learning activity

1. Write a regular expression for all strings consisting of an odd number of **a**'s followed by an even number of **b**'s.
2. Write a regular expression for all sequences of **a**'s and **b**'s of length 3.
3. Write a regular expression for all strings that contain **abba** somewhere within them.

As you have probably observed, the pattern-matching notation we used in section 2.3 employed a subset of the RE syntax, and we could in principle use regular expressions to encode simple grammars as presented in that section. For example:

((John|Mary|Fred) | ((the|a)(cat|dog|fish))

(barked | slept | swam)

*((and | or) (barked | slept | swam))**

matches sentences like:

1. John slept
2. The cat barked or swam
3. Mary swam and barked or slept

4. ...

It can be seen that even conceptually simple REs can rapidly become almost unreadable. A more manageable formalism is a **regular grammar**, made up of production rules or rewrite rules of the kind you have seen in the previous section:

S $\rightarrow$ John | Mary | Fred VP

S $\rightarrow$ the | a S1

S1 $\rightarrow$ cat | dog | fish VP

VP $\rightarrow$ barked | slept | swam VP1

VP1 $\rightarrow$ and | or VP

VP1 $\rightarrow \epsilon$

A sequence of words forms a grammatical sentence according to a grammar of this type if one can draw a tree diagram like Figure 2.1 such that:

1. The root node is *S* or *Sentence*.
2. For every node that matches the left hand side (LHS) of a grammar rule, one can draw a subtree with the items on the right hand side (RHS) as daughter nodes.
3. When no more grammar rules can be applied, every leaf node of the tree matches a word in the language or the empty string ϵ.

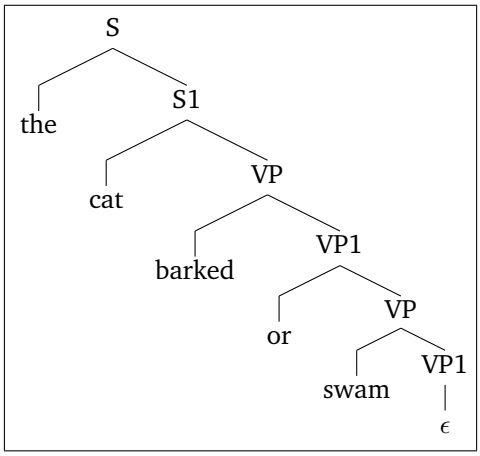

Figure 2.1: A right-branching tree.

Symbols which only occur on the right-hand side of rules, and so can only appear as leaf-nodes in a tree, are known as **terminal symbols**. Regular grammars have the restriction that when non-terminal symbols appear on the RHS they must either always be the rightmost symbol, or always the leftmost. These classes of grammars are known as **right-linear** and **left-linear** respectively. A right-linear grammar will always result in a right-branching tree as in the above example.

Learning activity

Draw tree diagrams according to the above grammar for the sentences:

1. The dog slept.
2. Mary swam and barked or slept.

2.4.4 Limitations of finite-state methods – introducing context-free grammars

Pinker (2007, p.86) gives an example of a 'wordchain device' or FSM which is intended to show the limitations of finite-state methods for handling natural language. The procedure is apparently designed to deal with complex sentences including constructions like *If. . . then. . .* and *Either. . . or . . .*. If we look at a few possible matching strings, we see clearly that some are grammatical sentences but others are nonsensical. (Following a standard convention in linguistics, the unacceptable cases are marked with an asterisk '*'.)

- Either a happy girl eats ice cream or the boy eats hot dogs.
- *Either a happy girl eats ice cream then one dog eats candy.
- If a girl eats ice cream then the boy eats candy.
- *If a girl eats ice cream or the boy eats candy.

Learning activity

1. Write a regular grammar that is equivalent to the FSM in Pinker (2007, p. 86).
2. Convince yourself that it allows you to draw well-formed trees for the ungrammatical examples above.
3. What characteristic of the grammar prevents it from ruling out the ill-formed examples?

See 'Answers to Activities' in Appendix C (p. 98) for further discussion.

In order to handle these kinds of sentences correctly we need to add new kinds of rewrite rules, going beyond the class of right- or left-linear grammars:

1. To match pairs of words like *if . . . then*, *either . . . or*, we need rules where a non-terminal symbol on the RHS can have additional material on both sides as in the first two rules below.
2. In order to allow for indefinite nesting – *if either John will come if Mary does, or . . .* we need rules where the same symbol can occur on both sides of the arrow. This is known as **self-embedding** or **centre-embedding**.

Note that centre-embedding is an instance of **recursion** in grammar; right-linear grammars may also include recursive rules but they can always be processed iteratively rather than recursively (Jurafsky and Martin, 2009, p. 447).

S →Either S or S

S →If S then S

S → NP VP

NP → Det N

Det → a | the | ϵ

N → girl | boy | dog | cat | burgers | candy | cream | cake

VP → V NP

VP → V PP

PP → P NP

V → eats | likes | sat

P → on

The above grammar handles these cases correctly as well as simple sentences like *The cat sat on the mat*:

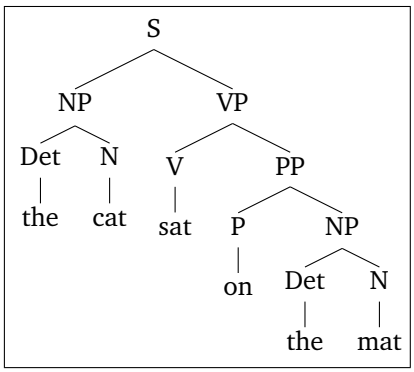

Figure 2.2: Tree diagram for *The cat sat on the mat.*

It is also acceptable to represent trees using *labelled bracketed strings* as in the example below:

```
(S
(NP (Det the ) (N cat ))
(VP (V sat )
 (PP
  (P on )
  (NP
  (Det the ) (N mat ) )
  )
   )
   )
```

Figure 2.3 is an example of self-embedding.

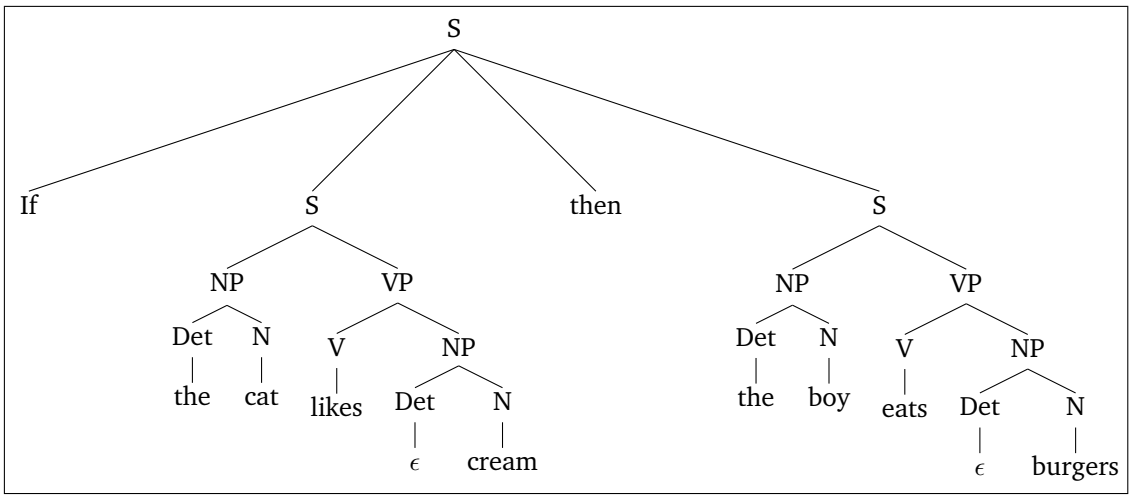

Figure 2.3: Tree diagram with self-embedding.

Learning activity

1. Trace through the grammar rules and satisfy yourself that Figure 2.3 represents the structure of the sentence *If the cat likes cream then the boy eats burgers* according to the grammar.
2. What is the shortest sentence generated by the above grammar?
3. Using the above grammar, draw complete tree diagrams for:
   (a) If the girl likes cake then either the boy eats burgers or the boy eats candy.
   (b) If either the boy likes cake or the girl likes burgers then the dog eats candy.
4. Think of ways to modify the grammar to generate more natural-sounding sentences.

2.4.5 Looking ahead: some further uses of regular expressions

In this chapter we have so far looked at finite-state formalisms as techniques for generating or recognising short phrases as well as whole sentences, and found them to be wanting. Many current applications in language technology do not, in fact, require complete analysis of sentences but proceed by looking for patterns of interest within a text and discarding what does not match these patterns. Finite-state methods are often quite adequate for these applications and you will see many uses for regular expressions in later chapters of this guide. Some examples we will look at in more detail in later chapters are:

- stemming: extracting the 'base form' of a word as informally presented in section 2.5 of this chapter
- tagging: automatically assigning POS or other forms of mark-up to elements in a text
- chunking: grouping together a sequence of words as a phrase
- information extraction: identifying chunks that denote meaningful entities, events or other items of interest.

2.4.6 Looking ahead: grammars and parsing

The pseudocode and graphical representations of wordchains (FSMs) combine a specification of the well-formed sentences in a language fragment with a processing strategy. It is important to keep in mind that formal grammars made up of a series of production rules do **not** encode a processing strategy. As stated above, a grammar is a declarative specification of the strings that make up a language while parsers use a variety of algorithms to apply the grammar rules. We will look at some of these parsing strategies in Chapter 6 of this subject guide.

2.5 Word structure

Words combine in different orders to form sentences and phrases; they also have internal structure. Nouns in English may have different endings according to whether they are singular (*a box*) or plural (*some boxes*) while in some languages this information may be expressed at the start of the word, for example Swahili *ziwa* ('lake') vs *maziwa* ('lakes'). In English, endings of verbs can indicate person, number, tense and mood[2], while other languages may make different dictinctions. Nouns and verbs are sometimes classified as regular or irregular according to whether their inflected forms can be derived by following simple rules. Table 2.2 shows examples of some common past tense forms in English.

Present	**Past**
become	became
come	came
mistake	mistook
misunderstand	misunderstood
ring	rang
sell	sold
shake	shook
sing	sang
sink	sank
stand	stood
take	took
tell	told
travel	travelled
understand	understood
withstand	withstood

Table 2.2: Past tense forms (1).

The subfield of linguistics known as **morphology** is concerned with the structure of words and is concerned, among other things, with formulating rules for deriving different forms of a word according to its grammatical role. Here are some rules which appear to cover the examples in the table:

[2] See the Internet Grammar of English at http://www.ucl.ac.uk/internet-grammar/verbs/verbs.htm (last visited 27th May 2013) for explanations of these terms.

Some rules for past-tense formation

-come → -came
-take → -took
-ing → -ang
-ink → -ank
-ell → -old
-and → -ood
-el → -elled

Some of these rules could be made more general: we could combine the *-ing* and *-ink* rules to a single rule, -in_ → -an_. On the other hand, some rules which work for these particular examples would fail if applied to a wider range of data: we have *come* → *came*, *become* → *became* but not *welcome* → **welcame*. This is an example of rules **overfitting** the data.

Learning activity

Modify the above rules so that they will account for the past tense forms in Table 2.3 as well as in Table 2.2.

Present	**Past**
bake	baked
command	commanded
bring	brought
sling	slung
smell	smelt
think	thought
wake	woke

Table 2.3: Past tense forms (2).

A natural language application such as a machine translation system will typically include a database of words or *lexicon* along with rules for deriving word endings: for example, a translation from English into Dutch might handle the word *brought* as follows:

1. Find the stem of *brought* and interpret the inflection: *bring+past*
2. Find the Dutch equivalent of *bring*: *brengen*
3. Find the past tense of *brengen*: *bracht*

The process of removing affixes from words to derive the basic form is called *stemming*. We will look at some tools for doing this in Chapter 4, and you will also have the opportunity to encode your own rules as regular expressions.

2.6 A brief history of natural language processing

The field of natural language processing or computational linguistics builds on techniques and insights from a number of different disciplines, principally

theoretical linguistics and computer science but with some input from mathematical logic and psychology.

The notions of a finite-state machine and context-free grammar (CFG) were first introduced to linguistics by Chomsky (1957; see Pullum (2011) for a somewhat critical reappraisal). Chomsky argued that both formalisms were inadequate for modelling natural language and proposed an additional operation of transformations, which could essentially permute the output string of a CFG in various ways. Chomsky's work introduced a methodology which was to dominate theoretical linguistics for the next couple of decades: linguists concentrated on postulating formal rules of grammar which were tested against their own intuitions or those of native speakers of other languages, rather than seeking to induce rules from large collections of data. Part of the rationale for this was that native speakers of a language are able to recognise whether a sequence of words makes up an acceptable sentence in their language, even if they have never encountered those words in that particular order before. Prior to what was to become known as the Chomskyan revolution, corpus-based approaches had been the norm in general linguistics. This tradition was overshadowed for a time by so-called 'generative' linguistics, but corpus-based research continued in some quarters until its resurgence in the 1980s, including the development of the first machine-readable corpus by the Jesuit priest Fr Robert Busa. Busa developed a 10 million-word corpus of medieval philosophy on punch-cards, with the support of Thomas Watson of IBM (McEnery and Wilson, 2001, pp. 20–21).

Work in formal grammar tended to assume a 'backbone' of context-free rules, augmented with various mechanisms to handle data that appeared to go beyond the context-free model; some important developments were Generalised Phrase Structure Grammar (Gazdar et al., 1985) and Head-driven Phrase Structure Grammar (Pollard and Sag, 1994). We will see examples of these extra mechanisms in Chapter 6.

Early work on automated language processing was essentially procedural in its methodology, working with a type of finite-state machine called **transition networks** which were extended as augmented transition networks to cope with various linguistic constructions (Woods, 1970). Later work based on declarative grammar formalisms employed techniques including **deductive parsing** (Pereira and Warren, 1983) and **unification** (Kay, 1984). The former adopts techniques from the AI field of automated reasoning: the core idea is that parsing a sentence can be seen as constructing a logical proof that a particular sequence of words forms a proper sentence according to a given set of grammar rules. Unification grammars treat linguistic objects as sets of attributes or features with a finite range of values, and grammar rules specify that particular items in a sentence must have the same or compatible values for certain features. For example, the subject and main verb of a sentence should have the same value for the *number* feature. We will consider detailed examples in Chapter 6.

Meanwhile, substantial progress was made in lower-level tasks such as speech recognition and morphological analysis using probabilistic techniques and finite-state models. During the late 1990s these techniques were extended to cover tasks such as parsing, part-of-speech tagging and reference resolution (recognising whether or not different expressions in a document referred to the same person or entity). These developments were driven by a number of factors: the continuing increase in the processing speed and memory capacity of computers; the availability of massive amounts of spoken and written material, both in unstructured form on the world wide web and with various types of annotation in corpora such as the

Penn Treebank[3] or the British National Corpus[4], and events such as the Message Understanding Conferences[5] which were initially sponsored by the US Department of Defense to measure and foster progress in extracting information from unstructured text.

Much work since around the year 2000 has involved the use of machine learning techniques such as Bayesian models and maximum entropy (see Chapter 5). This has involved using annotated corpora to train systems to segment and annotate texts according to various morphological, syntactic or semantic criteria. These techniques have been systematically applied to particular tasks such as parsing, word sense disambiguation, question answering and summarisation.

2.7 Summary

1. This chapter has characterised the subject matter of the course as being concerned with various ways of using computer programs to analyse **text**, by which we mean words, numbers, punctuation and other meaningful symbols that are printed on paper or some other flat surface, or displayed on a screen.
2. Some fundamental operations in text analysis include **tokenisation**, which involves extracting these meaningful elements from a stream of electronic characters and discarding white space, line feed characters and other material which has no explicit meaning for a human reader, and using regular expressions to identify patterns in a text.
3. Regular expressions are composed of the three basic operations of sequence, selection and iteration, and have many applications in computational linguistics and computer science at large. A finite-state machine is a process whose operations can be specified by means of regular expressions. A regular grammar is a set of production rules or rewrite rules that defines the sentences that make up a language, and any language defined by a regular grammar can be processed by a finite state machine or described using a regular expression.
4. A complete syntactic analysis of natural language sentences is generally held to require the additional operation of centre-embedded recursion, which is beyond the power of finite-state methods. Recursion is formally encoded in context-free grammars.
5. Not only do words combine in various patterns and structures to form sentences; they also have internal structure which can be described to an extent using rules for regular and irregular forms.
6. The current state of NLP or computational linguistics builds on research results and concepts from many different fields, and we have sketched some of the highlights in a very short history of the discipline.

2.8 Sample examination questions

You can expect a list of RE operators to be included as an appendix in the examination paper.

[3] http://www.cis.upenn.edu/∼treebank/; last visited 27th May 2013
[4] http://www.natcorp.ox.ac.uk/; last visited 27th May 2013
[5] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/; last visited 27th May 2013

1. S → NP VP

   NP → Det N

   NP → PN

   VP → V

   VP → V NP

   VP → V NP PP

   VP → VP Adv

   PP → P NP

   Det → the | a

   N → waiter | chairs | tables | hotel | manager

   PN → Oscar | Paris

   V → died | put | saw | called

   Adv → suddenly | quickly | slowly

   P → in | on

   (a) Using the grammar rules above, draw syntax trees for:
      i. Oscar died suddenly.
      ii. The waiter put the chairs on the tables.
      iii. Oscar called the waiter.

   (b) Modify the grammar so that it generates the unstarred sentences below as well as (i–iii) above but not the starred ones. Explain the reasons for your modifications.
      i. Oscar died in Paris.
      ii. Oscar died in a hotel in Paris.
      iii. The waiter came to the table when Oscar called him.
      iv. When Oscar called him the waiter came to the table.
      v. * Oscar put
      vi. * The waiter saw on the tables
      vii. * The waiter put in the chairs
      viii. * The waiter put the chairs
      ix. * Oscar died the table
      x. * When Oscar called him when the waiter came to the table.

2. Write a regular expression that will identify male and female names in context, in an English-language text. Discuss ways in which this might over- or under-generate.

3. Explain the difference between regular and context-free grammars and discuss the claim that natural language grammars need at least context-free power.

4. (a) Write a regular grammar which generates the following sentences:
      i. This is the kid that my father bought.
      ii. This is the cat that killed the kid that my father bought.
      iii. This is the dog that worried the cat that killed the kid that my father bought.
      iv. This is the stick that beat the dog that worried the cat that killed the kid that my father bought.

      (*Brewer's Dictionary of Phrase and Fable*, 1896)

   (b) Write out three more sentences generated by your grammar.

Chapter 3

Getting to grips with natural language data

Essential reading

Bird et al. (2009) *Natural Language Processing with Python* Chapters 1 and 2 particularly: 1.1–1.3, 2.1–2.2, 2.5.

Recommended reading

McEnery (2003) 'Corpus Linguistics' in Mitkov (2003) is a succinct overview of the topic from one of the leading scholars in the field.

Additional reading

McEnery and Wilson (2001) *Corpus Linguistics* is an established undergraduate textbook; Chapters 2 and 3 are especially relevant for this topic.

McEnery and Hardie (2011) *Corpus Linguistics: Method, Theory and Practice*. Chapter 3 addresses the important topic of research ethics for corpus linguistics, which is often neglected in technical or academic presentations of the subject.

3.1 Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- explain what is meant by a corpus in the context of natural language processing, and describe some different types, structures and uses of corpora
- describe the characteristics of some well-known corpora and other language resources such as the Brown corpus, Penn treebank, Project Gutenberg and WordNet
- Use online interfaces and other software tools to do some basic corpus analysis, including concordancing and finding collocations
- locate and format raw text documents and analyse them using corpus tools.

3.2 Using the Natural Language Toolkit

As stated in Chapter 1, this subject guide is not intended as a stand-alone tutorial in using the NLTK or the Python language. You are advised to read through the recommended sections of Bird et al. (2009) and work through the exercises marked

Your turn. You may also find it useful to attempt some of the exercises provided at the end of each chapter.

From this chapter onwards you will be running Python sessions and using the NLTK. You should get into the habit of starting sessions with the following commands:

```
>>> from __future__ import division
>>> import nltk, re, pprint
```

One of the features that makes the Python language suitable for natural language applications is the very flexible treatment of data structures such as lists, strings and sequences. You should be familiar with these structures from previous programming courses, but should ensure you understand the way they are handled in Python. For this chapter, only lists are relevant and you should study Bird et al. (2009, section 1.2) before trying any of the learning activities in this chapter.

3.3 Corpora and other data resources

As explained in the previous chapter, much natural language processing relies on large collections of linguistic data known as *corpora* (plural of *corpus*). A corpus can be simply defined as no more than a collection of language data, composed of written texts, transcriptions of speech or a combination of recorded speech and transcriptions.

Corpora fall into three broad categories (McEnery, 2003, p.450):

- **Monolingual** corpora consist, as the name suggests, of data from a single language.
- **Comparable** corpora include a range of monolingual corpora in different languages, preferably with a similar level of balance and representativeness, and can be used for contrastive studies of those languages.
- **Parallel** corpora include original texts in one language with translations of those texts in one or more different languages. Parallel corpora can be used to train statistical translation systems.

A corpus is generally expected to have additional characteristics: corpora are usually constructed so as to be *balanced* and *representative* of a particular domain (McEnery and Wilson, 2001, pp. 29–30). (Sometimes the term is used more loosely to cover any large collection of language data which need not have been compiled systematically, as in the phrase 'the web as corpus'.) **Sampling theory** is a branch of statistics that deals with questions such as: how many respondents are needed in an opinion poll for the results to be considered to represent public opinion at large? Similar considerations arise in corpus linguistics. This is particularly important if a corpus is to be used for quantitative analysis of the kind described in Chapter 5: if the corpus data is skewed or unrepresentative then results of the analysis may not be reliable. These considerations may be less important if the corpus is collected for the literary or historical interest of the documents that make it up, as is the case with Project Gutenberg for example.

For example, Bird et al. (2009, pp. 407–412) refer to the TIMIT corpus, an annotated speech corpus developed by Texas Instruments and MIT. To ensure representativeness, it was designed to include a wide coverage of dialect variations. Corpus builders need to exercise expert judgment in deciding on the *sampling frame*,

or 'the entire population of texts from which we will take our samples' (McEnery and Wilson, 2001, p. 78) and calculating the size of the corpus that is necessary for it to be maximally representative. The sampling frame may, for example, be *bibliographic*, based on some comprehensive index or the holdings of a particular library, or *demographic*, selecting informants on the basis of various social categories as is often done in public opinion research.

Corpora have tended to be of a finite size and to remain fixed once they have been compiled. There are also what is known as *monitor corpora* which are continually updated with new material. This is particularly useful for compilers of dictionaries who need to be able to track new words entering the language and the changing or declining use of old ones. An example of a monitor corpus is the COBUILD Bank of English™, which had around 300 million words when it was referred to by McEnery (2003) and has since more than doubled in size to 650 million words.

A further distinction is between corpora consisting solely of the original or 'raw' text and those that have been marked up with various annotations. One common technique is *standoff annotation* where the mark-up is stored in a different file from the original text (McEnery and Wilson, 2001, p.38); (Bird et al., 2009, p.415).

Finally, corpora can be further classifed according to their structure:

Isolated – an unorganised collection of individual texts such as the Gutenberg online collection of literary works.

Categorised – texts are organised by categories such as genre; an example is the Brown corpus described below.

Overlapping – some categories overlap. A news corpus such as Reuters may contain stories which cover both politics and sport, for example.

Temporal – texts indicate language use over time. Examples are the Inaugural corpus of all inaugural speeches by US Presidents, and the Helsinki Diachronic corpus of about 1.6 million words of English dating from the early 9th century CE to 1710.

Some examples of corpora, which will be described in more detail later in the chapter, are:

Brown Developed at Brown University in the early 1960s.

BNC British National Corpus, created and managed by BNC consortium which includes Oxford and Lancaster universities, dictionary publishers OUP, Longmans and Chambers, and the British Library.

COBUILD (Bank of English) The Bank of English™forms part of the Collins Corpus, developed by Collins Dictionaries and the University of Birmingham, and contains 650 million words.

Gutenberg An archive of free electronic books in various formats hosted at `http://www.gutenberg.org/`

Penn Treebank A corpus of reports from the *Wall Street Journal* and other sources in various different formats.

3.4 Some uses of corpora

McEnery and Wilson (2001, Chapter 4) discuss a variety of uses for corpora, some of which are briefly discussed below.

3.4.1 Lexicography

Modern dictionaries such as Chambers, Collins and Longmans now rely heavily on corpus data in order to classify and inventorise the various ways words can be used in contemporary English as well as any ways these uses may have changed. For example, a lexicographer who wishes to determine whether the words *scapegoat*, *thermostat* or *leverage* can be used as verbs can enter the appropriate search string and be presented with examples such as the following (from the BNC):

- ***Scapegoating*** *an individual prevents the debate and delays community understanding*.
- *The measuring cell is immersed in a vat of liquid, usually benzene or xylene which can be* ***thermostatted*** *at temperatures between 273 and 400 K*.
- *These one-time costs once met could be* ***leveraged*** *over much more business activity around the globe than we then enjoyed*.

McEnery and Wilson (2001, p. 107) discuss a case where they claim that two well-known dictionaries had 'got it wrong' by listing *quake* as a solely intransitive verb, while examples in a transitive construction can in fact be found through a corpus search:

- These sudden movements quake the Earth. (BNC)

It is perhaps debatable whether the dictionaries in question were 'wrong' to disregard examples like this, or the compilers may have judged this to be an idiosyncratic usage which did not merit being included in a work of reference with the status of standard usage.

3.4.2 Grammar and syntax

Large-scale grammars for pedagogic and reference use such as the *Comprehensive Grammar of the English Language* (Quirk et al., 1985) or the *Cambridge Grammar of the English Language* (Huddleston and Pullum, 2002) use corpora among their sources of information along with results of linguistic research and the compilers' own subjective intuitions as competent speakers of the language, although this has tended to involve qualitative rather than quantitative analysis. Recent advances in computational power and developments in parsed corpora and tools to analyse them have made it possible for researchers to carry out quantitative studies of various kinds of grammatical frequency, such as the relative frequency of different clause types in English. Other studies use corpora to test predictions made by formal grammars that have been developed within the generative school of linguistics. The COBUILD project which provided the resources for Collins English dictionaries has also resulted in a series of small handbooks covering various kinds of grammatical construction, which are useful both for advanced language learners and for linguists in search of examples.

3.4.3 Stylistics: variation across authors, periods, genres and channels of communication

The notion of *style* in communication is that people generally have a choice between different ways of expressing themselves and not only do individuals tend to make

similar choices each time they communicate, but their particular choices may be more characteristic of particular genres (romantic fiction, financial news, court reports and so on), time periods and channels of communication. By **channels** we mean distinctions such as written text compared with spoken discourse, both of which can be further subdivided: people will make different choices when composing emails, sending text messages or (rarely) writing a letter by hand. We probably also adopt different styles when talking face-to-face and on the telephone. Literary scholars as well as law enforcement and intelligence agencies may have an interest in identifying the author of a document from internal evidence. There have been various famous and less well-known instances of controversial attribution of authorship: for example, various figures have been put forward as the authors of Shakespeare's plays.

3.4.4 Training and evaluation

In addition to the applications listed above, corpora are routinely used in linguistic research for training and testing machine-learning systems for specific tasks in *text analytics* such as:

- detecting the topic of a document
- analysing the sentiments expressed for or against some product or policy
- identifying individuals described in a text, relations between them and events they participate in
- statistical parsing
- statistical machine translation.

For example, the Brown corpus and the WSJ corpus are typically used for evaluating text segmentation among other text processing tasks (Mikheev, 2003, p. 203).

These topics will be covered in more detail in Chapter 5 of this subject guide, where you will be introduced to various machine-learning techniques. These will all be types of **supervised learning**, where a system is trained on 'corpora containing the correct label for each input' (Bird et al., 2009, p. 222), as opposed to **unsupervised learning** where the system is designed to detect patterns in the input without any feedback. This normally means that the corpus has been marked up by human annotators. Standard practice is to divide a corpus into training and test sets; the test set is considered a **gold standard** for comparing the accuracy of trained learning systems with that of the annotators. Of course humans are fallible, and it is good practice to use multiple annotators for at least a sample of the corpus and report the level of **inter-annotator agreement** that was achieved. This will set an upper bound for the performance that can be expected from the system, as it seems meaningless to say that a computer program can achieve 100 per cent accuracy on tasks where human annotators disagree (see Jurafsky and Martin, 2009, p. 189).

3.5 Corpora

This section provides brief descriptions of various corpora, some of which are distributed in full or in part with the NLTK and others can be accessed online.

3.5.1 Brown corpus

This was one of the first 'large-scale' machine readable corpora, though it may seem rather small by today's standards, consisting of one million words. It was developed at Brown University from the early 1960s and took around two decades to complete. It was intended as a 'standard corpus of present-day edited American English' and is caterorised by genre under headings such as:

News *Chicago Tribune*: Society Reportage

Editorial *Christian Science Monitor*: Editorials

Reviews *Time Magazine*: Reviews

Government US Office of Civil Defense: *The Family Fallout Shelter*

Science Fiction Heinlein: *Stranger in a Strange Land*

Humour Thurber: *The future, if any, of comedy*.

The Brown corpus is provided with the NLTK in tagged and untagged versions and can be accessed using the various methods listed by Bird et al. (2009, Table 2.3, p. 50).

3.5.2 British National Corpus

The BNC is created and managed by the BNC consortium, which includes Oxford and Lancaster universities, dictionary publishers OUP, Longmans and Chambers, and the British Library. It was developed between 1991 and 1994 and consists of 100 million words: 90 per cent written and 10 per cent transcriptions of speech. This was one of the first corpora to include spontaneous spoken English. The corpus was marked up using an automated part-of-speech tagger which resulted in a significant saving of time and expense compared with manual annotation by competent speakers of the language, but means that there is inevitably a degree of error – as you may discover in the course of the exercise given later in this chapter.

You can access this corpus online and perform various kinds of analysis using the Simple Query language. Registration is required via the following link but there is currently no charge:

`http://bncweb.lancs.ac.uk/bncwebSignup/user/login.php` (last visited 27th May 2013)

3.5.3 COBUILD Bank of English

The COBUILD project involved Collins Dictionaries and the University of Birmingham. The Collins corpus is a 2.5-billion word analytical database of English. It contains written material from websites, newspapers, magazines and books published around the world, and spoken material from radio, TV and everyday conversations. The Bank of English™forms part of the Collins Corpus and contains 650 million words. It was used as a basis for the Collins Advanced Learner's Dictionary, grammars and various tutorial materials for learners of English. It is not included in the NLTK but there is limited online access via `http://www.collinslanguage.com/wordbanks`.

3.5.4 Penn Treebank

The Penn Treebank with its various offshoots is one of the widely used linguistic resources among empirical researchers.

It includes a collection of texts in four different formats:

- Raw text (original).
- Tagged with POS using a tagset which was developed as part of the project.
- 'Parsed'; that is, marked up with constituent structure.
- Combined, including both POS tags and constituent structure.

The Penn Treebank project . . . has produced treebanks from the Brown, Switchboard, ATIS and *Wall Street Journal* corpora of English, as well as treebanks in Arabic and Chinese.

Jurafsky and Martin (2009, p. 438)

The project began at the University of Pennsylvania in the 1990s and the results have been used as a basis for further annotation efforts involving semantics and rhetorical structure. The NLTK includes a selection from the *Wall Street Journal (WSJ)* component of the Treebank, which can be accessed in each of the above formats and additionally with a simplified POS tagset (Bird et al., 2009, Table 5-1, p. 183). Here is an excerpt showing the four different formats:

Raw text

```
Pierre Vinken, 61 years old, will join the board as a
nonexecutive director Nov. 29.
```

Tagged

```
[ Pierre/NNP Vinken/NNP ]
,/,
[ 61/CD years/NNS ]
old/JJ ,/, will/MD join/VB
[ the/DT board/NN ]
as/IN
[ a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ]
./.
```

Parsed

```
( (S (NP-SBJ (NP Pierre Vinken)
             ,
             (ADJP (NP 61 years) old)
```

```
        ,)
    (VP will
        (VP join
            (NP the board)
            (PP-CLR as
    (NP a nonexecutive director))
    (NP-TMP Nov. 29)))
    .))
```

Combined

```
( (S
     (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

3.5.5 Gutenberg archive

The NLTK includes a small selection of out-of-copyright literary texts from Project Gutenberg, an archive of free electronic books hosted at *http://www.gutenberg.org/* Some of the texts included are:

Jane Austen: *Emma*, *Persuasion*

GK Chesterton: *Father Brown stories*, *The Man Who Was Thursday*

William Blake: *Poems*

Milton: *Paradise Lost*

Shakespeare: *Julius Caesar*, *Macbeth*, *Hamlet*

3.5.6 Other corpora

Some further corpora included with the NLTK are:

- The Reuters corpus distributed with the NLTK contains 10,788 news documents totalling 1.3m words, partitioned into 'training' and 'test' sets. This split is for training and testing machine learning algorithms: we return to this topic in Chapter 5 of this subject guide.

- US Presidents' inaugural and State of the Union addresses, organised as separate files.
- UN Declaration of Human Rights in 300+ languages. Here are a few excerpts:
  - All human beings are born free and equal in dignity and rights.
  - *Abantu bonke bazalwa bekhululekile njalo belingana kumalungelo abo*.
  - *Todos os seres humanos nascem livres e iguais em dignidade e em direitos*.

Other corpora with online query interfaces include:

1. The Corpus of Contemporary American English, hosted at Brigham Young University, is claimed to be 'the only large and balanced corpus of American English'. *http://corpus.byu.edu/coca/* (last visited 27th May 2013)
2. The Intellitext project at the University of Leeds 'aims to facilitate corpus use for academics working in various areas of the humanities' and currently provides access to monolingual and parallel corpora in several European and Asian languages. *http://corpus.leeds.ac.uk/it/* (last visited 27th May 2013)

Learning activity

1. Pick two or three of the corpora mentioned above and research them online, focusing on questions such as:
   - how large is the corpus?
   - what language(s) and genre(s) does it represent?
   - when was it constructed and what is its intended use?
   - what is the sampling frame?
   - what level of interannotator agreement was achieved, if reported?
2. Logon to the BNC Web (free registration needed) or another online corpus. Study the documentation provided and search for data to answer the following questions:
   - What syntactic categories can the following words have? *total*, *pancake*, *requisition*, *acquisition*.
   - The guide to Simple Query Syntax provided with the BNC warns that 'part-of-speech tags have been assigned by an automatic software tool and are not always correct'. Have your answers to the previous question shown up any examples of incorrect classification, in your view?
   - What prepositions can follow the verb *talk*? Give an example in each case.

3.5.7 WordNet

The NLTK also includes English WordNet, with 155,287 words and 117,659 synonym sets or *synsets*. A synset consists of a set of synonymous words paired with a definition and linked to words with more general or more specific meanings. For example, *table* has various meanings:

```
table.n.01 ['table', 'tabular\_array'], "a set of data arranged in
rows and columns"

table.n.02 ['table'], "a piece of furniture having a smooth flat top
```

```
that is usually supported by one or more vertical legs"

table.n.02 hyponyms: drop-leaf\_table, coffee\_table, pool\_table, altar,

table.n.02 hypernyms: furniture
```

3.6 Some basic corpus analysis

This chapter describes some relatively simple techniques, extracting various kinds of data in suitable formats for human interpretation of the results. Chapters 4 and 5 of the subject guide will look at ways the analysis and interpretation itself can be automated to varying degrees.

Concordancing involves locating every instance of a word or phrase within a text or corpus and presenting it in context, usually a fixed number of words before and after each occurrence.

Collocations are pairs of sequences of words that occur together in a text more frequently than would be expected by chance, and so provide a rough indication of the content or style of a document.

Conditional frequency distributions support an elementary form of statistical analysis. A **frequency distribution** counts observable events and a **conditional frequency distribution** pairs each event with a condition. Some sample applications are:

- Comparing the use of particular words in different genres.
- Comparing word lengths in different languages.

3.6.1 Frequency distributions

The following worked example displays some rudimentary stylistic analysis by ranking the POS tags in a corpus according to frequency.

Calculating tag frequency

1. Import the Brown corpus.
2. List the different categories within the corpus.
3. Count the number of sentences in the science fiction category.
4. Extract all the word tokens from the science fiction category, paired with their tags, and store them in the variable `bsf`. Note that the simplified tagset is selected.
5. Calculate a frequency distribution of the tags: this gives an ordered list of the tags paired with their frequency in the variable `sf_tag_fd`. (Only the 12 most frequent are shown.)

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction',
'government', 'hobbies', 'humor', 'learned', 'lore',
```

```
'mystery', 'news', 'religion', 'reviews', 'romance', 'science_fiction']
>>> sf_sents = brown.sents(categories = 'science_fiction')
>>> len(sf_sents)
948
>>> bsf = brown.tagged_words(categories = 'science_fiction',
simplify_tags=True)
>>> sf_tag_fd = nltk.FreqDist(tag for (word,tag) in bsf)
>>> sf_tag_fd.keys()
['N', 'V', 'DET', 'PRO', 'P', '.', 'ADJ', ',', 'CNJ', 'ADV', 'VD', 'NP',  ]
>>> sf_tag_fd.tabulate()
N    V   DET  PRO  P    .     ADJ  ,    CNJ  ADV   VD   NP
2232 1473 1345 1268 1182 1078  793  791  685  644   531  467
```

Learning activity

1. Repeat the above process for other categories such as romance, news and religion. How do the frequency distributions and sentence counts enable you to compare the literary styles of these genres? Explain any assumptions you make.
2. Having read through Bird et al. (2009, section 2.1), with particular attention to Table 2-3 on page 50, answer the following questions:
   (a) Summarise the README file from the Reuters corpus.
   (b) Create a variable `soysents` containing all sentences from reports concerning soy products.
   (c) Display the first ten sentences in `soysents`.
   (d) Create a variable `metalwords` containing all words from reports concerning metals.
   (e) What are the most frequently mentioned metals in this collection? Caution: why might this result be less than 100 per cent reliable?

3.6.2 DIY corpus: some worked examples

NLTK's plain text corpus reader can be used to build a 'corpus' from a collection of text files. The resulting corpus will be formatted for access as raw text, lists of words or lists of sentences and can be re-formatted for other functions such as concordancing and finding collocations.

The first example involves a one-text 'corpus' of the recent report from the UK Equality and Human Rights Commission: *How fair is Britain*?

Step 1 Download the report as a PDF from `http://www.equalityhumanrights.com`

Step 2 Manually extract text using Adobe Acrobat or another PDF reader and save as a `.txt` file

Step 3 Point the corpus reader to the directory where you have saved the text file.

```
>>> import nltk
>>> from nltk.corpus import PlaintextCorpusReader
>>> corpus_root = 'C:\NLP-stuff\Corpora'
>>> mycorpus = PlaintextCorpusReader(corpus_root,'.*')
```

We can now use the `raw`, `words` and `sents` methods to display the content in different formats:

```
>>> mycorpus.fileids()
['howfair.txt']
>>> mycorpus.words('howfair.txt')
['Equality', 'and', 'Human', 'Rights', 'Commission', ]
>>> hf_raw = mycorpus.raw('howfair.txt')
>>> hf_raw[:100]
'Equality and Human Rights Commission\r\nTriennial
Review 2010\r\nExecutive Summary\r\nHow fair\r\nis Britain'
>>> mycorpus.sents('howfair.txt')
[['Equality', 'and', 'Human', 'Rights', 'Commission', 'Triennial',
'Review', '2010', 'Executive', 'Summary', 'How', 'fair', 'is',
'Britain', '?'], ...
```

Concordancing and collocations

The `Text` method formats the content in a way which can be accessed by the `concordance` and `collocation` methods. Note that concordancing will always return fixed-length strings which include your target text as a substring, and so may be cut off in the middle of a word.

```
>>> fair=nltk.Text(mycorpus.words('howfair.txt'))
>>> fair.concordance('equal')
Building index...
Displaying 3 of 3 matches:
has narrowed considerably since the equal Pay Act 1970 came into force in 1975
sonal circumstances , should have an equal opportunity to have a say in decisio
that every individual should have an equal chance to make the most of their tal
>>> fair.collocations()
Building collocations list
Human Rights; Rights Commission; Significant findings; Headline data;
Executive Summary; less likely; ethnic minority; life expectancy;
0845 604; domestic violence; hate crime; labour market; disabled people;
mental health; Black Caribbean; different groups; religiously motivated;
sexual assault; minority groups; formal childcare
```

Conditional frequency distribution

Recall that a **frequency distribution** is a set of ordered pairs $< event,\ count >$ where *count* is the number of times *event* occurs. In our context *event* is a word-type and *count* is the number of tokens of that type in a text. A **conditional frequency distribution** is a collection of frequency distributions, each one for a different condition.

For this example we add a second document to the corpus, extracted from a PDF 'Guide to data protection'.

Step 1 Create a single variable `text_word` consisting of pairs of each word-token with the fileid of the document it occurs in.

Step 2 Create a conditional frequency distribution, which will tell you the frequency of each word in both texts.

Step 3 Pick a sample of words which are likely to occur in both documents, and tabulate their comparative frequency.

```
>>> text_word = [(text,word) for text in ['howfair.txt','guide.txt']
     for word in mycorpus.words(text)]
>>> text_word[:10]
[('howfair.txt', 'Equality'), ('howfair.txt', 'and'),
('howfair.txt', 'Human'),('howfair.txt', 'Rights'),
('howfair.txt', 'Commission'), ('howfair.txt', 'Triennial'),
('howfair.txt', 'Review'), ('howfair.txt', '2010'),
('howfair.txt', 'Executive'),
('howfair.txt', 'Summary')]
>>> cfd = nltk.ConditionalFreqDist(text_word)
>>> cfd
<ConditionalFreqDist with 2 conditions>
>>> cfd.conditions()
['guide.txt', 'howfair.txt']
>>> cfd['howfair.txt']
<FreqDist with 16391 outcomes>
>>> cfd['guide.txt']
<FreqDist with 47064 outcomes>
Testing the CFD
>>>  cfd['guide.txt']['fair']
31
>>> cfd['howfair.txt']['fair']
30
>>> keywords = ['fair','police','crime','office','equal','privacy']
>>>>cfd.tabulate(conditions=['howfair.txt','guide.txt'],
samples=keywords)
              fair police crime office equal privacy
howfair.txt   30   15     29    4      2     0
guide.txt     31   16     17    2      0     26
```

Learning activity

Find some suitable electronic documents and follow the above techniques to construct a 'mini-corpus'. The documents in these examples were sourced from UK government websites: you may find similar documents on the website of your own country's government, or of transnational organisations like the European Union or the United Nations. Think of some terms which are likely to occur in several of these documents and compare them using a conditional frequency distribution. If you can find a lengthy report which is issued along with a shorter summary, it is an interesting exercise to pick some key terms and see if their comparative frequency is the same or similar in the original document and the summary.

3.7 Summary

1. A corpus is a collection of linguistic data which was not originally produced for the purposes of linguistic analysis. Properly speaking it should be constructed so as to be balanced and representative. If a corpus includes any kind of

annotation, it is good practice to use multiple annotators for at least a sample of the corpus and report the level of **inter-annotator agreement** that was achieved.

2. Some uses of corpora include:
   - Lexicography (compiling dictionaries).
   - Compiling grammars for education and reference purposes.
   - Stylistics: developing techniques to identify the author or genre of a document; investigating the effect on language use of different channels such as email, chat, face-to-face conversation, telephone calls and hand-written letters.
   - Training and evaluation in linguistic research, using machine learning techniques.
3. This chapter includes brief descriptions of several well-known and widely-used corpora such as the Brown corpus, the BNC and the Penn Treebank.
4. Students on this course can access a variety of corpora through online interfaces or by using corpus tools provided with the NLTK.
5. Some simple techniques for analysing corpora include concordancing, collocations and (conditional) frequency distributions. None of these techniques involve automated linguistic analysis: the interpretation of the outputs is down to the analyst.

3.8 Sample examination question

a) Explain what is meant by the following types of corpus, and describe an example in each category that you have encountered during this course:

- isolated
- categorised
- overlapping
- temporal.

b) What applications would a **tagged** and **parsed** corpus be suitable for? What are some advantages and disadvantages of using an automated tagger to build such a corpus?

c) Suppose the following lists show the number of sentences and the most commonly occurring part-of-speech tags in three different categories of text in a corpus, with their frequency of occurrence in brackets. What can you say about the styles of these documents from studying these results? Discuss any assumptions you make.

Category A (4623 sentences) N (22236) P (10845) DET (10648) NP (8336) V (7346) ADJ (5435) ‘,’ (5188) ‘.’ (4472) CNJ (4227) PRO (3408) ADV (2770) VD (2531) . . .

Category B (948 sentences) N (2232) V (1473) DET (1345) PRO (1268) P (1182) ‘.’ (1078) ADJ (793) ‘,’ (791) CNJ (685) ADV (644) VD (531) NP (467) . . .

Category C (1716 sentences) N (7304) P (4364) DET (4239) V (3863) ADJ (2620) CNJ (2324) PRO (2243) ‘,’ (1913) ‘.’ (1892) ADV (1485) NP (1224) VN (952) . . .

Chapter 4

Computational tools for text analysis

Essential reading

Bird et al. (2009) *Natural Language Processing with Python*, Chapter sections 3.1–2, 3.4–6, 4.2, 5.1–2, 5.4–5.

Recommended reading

Mitkov (2003) *The Oxford Handbook of Computational Linguistics*, Chapter 10, 'Text segmentation' by Andrei Mikheev, pp. 201–218 and Chapter 11, 'Part-of-Speech tagging' by Aitro Voutilainen, pp. 219–232.

McEnery and Wilson (2001) *Corpus Linguistics*, Chapter 5.

Additional reading

Jurafsky and Martin (2009) *Speech and Language Processing* Chapter 5 (**advanced**).

4.1 Introduction and learning outcomes

The previous chapter introduced some fairly superficial techniques for language analysis such as concordancing and collocations. This chapter covers some fundamental operations in text analysis:

- tokenisation: breaking up a character string into words, punctuation marks and other meaningful expressions
- stemming: removing affixes from words, for example: *mean+ing*, *distribut+ion*
- tagging: associating each word in a text with a grammatical category or part of speech.

There will be some worked examples and exercises showing how these tasks can be carried out to a limited extent using Python's implementation of regular expressions. We will also demonstrate some tools that are built in to the NLTK and consider some of their strengths and limitations. In this chapter these tools are treated as 'black boxes' in that we do not consider their internal workings. The following chapter will introduce some of the underlying algorithms.

4.1.1 Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- Explain what is meant by terms such as *token*, *tag* and *stem* in the context of natural language processing.
- Explain the uses of programming constructs such as **strings** and **lists** in natural language processing.
- Critically evaluate different approaches to stemming.
- Design and implement some simple models for stemming and tagging using regular expressions.
- Understand how to train and test some automated taggers and evaluate the results.

4.2 Data structures

In addition to lists, language processing involves data structures such as **strings** and **sequences** (also referred to as tuples). As with lists, you should have encountered these constructs in previous programming courses but it is important to be familiar with the way they are handled in Python.

Learning activity

Read through sections 3.2 and 4.2 of the NLTK book. Start a Python session and execute the following commands:

```
seq = 'bang', 'on', 'a', 'can'
list = ['bang', 'on', 'a', 'can']
str =  'bang on a can'
```

Which of the following queries evaluates to `True`?

i `'an' in seq`
ii `'an' in list`
iii `'an' in str`
iv `'an' in seq[1]`
v `'an' in list[0]`
vi `'on' in seq`

What should result from the following operation? Try to work it out before executing it.

```
list2 = [list[0], 'hard', str[5:7], seq[2], list[-1]]
```

See how many other ways you can get the same result with different combinations of **indexing** and **slicing**.

4.3 Tokenisation

A machine-readable text consists essentially of a stream of electronic characters, which we perceive when they are printed or rendered on a screen as letters, white

space, punctuation marks, emoticons and so on. Some characters are only indirectly perceived as they control operations such as line breaks and tabbing. Tokenisation is the task of breaking up this stream into the discrete units that are recognised by a competent, literate speaker of the language.

4.3.1 Some issues with tokenisation

The simplest way to tokenise a text would be to treat everything between successive white spaces as a token, but this does not go far enough. Consider again the example from Chapter 2:

> *"Dr. Watson, Mr. Sherlock Holmes", said Stamford, introducing us.*

If we simply split this up on white space, we get the following list of 'tokens' (separated by slashes '/'):

```
``Dr. / Watson, / Mr. / Sherlock / Holmes", / said / Stamford,
/ introducing / us.
```

Clearly we need to treat punctuation marks as tokens, separate from the words they are adjacent to. However, there are a few difficulties in interpreting punctuation (Mikheev, 2003):

- Perhaps the most obvious source of difficulty is the full stop or 'period'. As you may have noted in the previous chapter, the same symbol can occur to mark the end of a sentence, to indicate an abbreviation, as a decimal point, or as part of an internet domain name. Additionally, if an abbreviated form such as *Y.M.C.A.* comes at the end of a sentence, the final 'dot' performs two functions. Looking for sequences 'full stop-white space-capital letter' will usually find the end of a sentence but would also match title and proper name sequences like *Mr. Jones*, *Dr. Kildare* or sequences of initials and names: *H. Hatterr*, *G.V. Desani*, *T.S. Eliot*. These kinds of examples can be caught to some extent by listing standard abbreviations in the *lexicon*.
- Hyphenated expressions may form a single token, conjoin two tokens or simply signal a line break. Mikheev (2003, p. 207) gives *self-assessment* and *F-16* as examples of single tokens, which we can contrast with *self-centred* or *10–16* (expressing a range of numbers). Mikheev also notes what we might call 'dynamic' hyphenation, where a hyphen is inserted within a phrase to prevent incorrect parsing: *ditch-delivered*, *Nietzsche-influenced*.
- Inverted commas may act as quote marks, they may indicate elision as in *he's*, *can't* or they may form an integral part of a word: *O'Brien*, *o'clock*. Where the comma does indicate elision, it is not always the same position within a word which has been removed. In verb contractions in English, the first vowel or diphthong of the verb is removed: *he's* is short for *he is* or *he has*, *they'll* stands for *they will*.[1] In negated forms the comma appears to stand for the elided *o* in *not*: *has not* becomes *hasn't*, *does not* becomes *doesn't*.

[1] Mikheev (2003, p. 210) simplifies this a little when he refers to the 'first character' of the second word.

4.3.2 Tokenisation in the NLTK

The NLTK includes a built-in tokeniser, `nltk.word_tokenize()` (Bird et al., 2009, p. 80) which can be invoked as in this example:

```
>>> holmes = "‘Dr. Watson, Mr. Sherlock Holmes’, said Stamford
introducing us."
>>> tokens = nltk.word_tokenize(holmes)
>>> tokens
[’‘’, ’Dr.’, ’Watson’, ’,’, ’Mr.’, ’Sherlock’, ’Holmes’, "’", ’,’,
’said’, ’Stamford’, ’,’, ’introducing’, ’us’, ’.’]
```

There is also a regular expression tokeniser which uses an RE defined by the user to break up text. Here is an example using the same string as above; see Bird et al. (2009, p. 111) for the full RE.

```
>>> text = "‘Dr. Watson, Mr. Sherlock Holmes’, said Stamford,
introducing us."
>>> nltk.regexp_tokenize(text, pattern)
[’‘’, ’Dr’, ’.’, ’Watson’, ’,’, ’Mr’, ’.’, ’Sherlock’, ’Holmes’,
"’", ’,’, ’said’, ’Stamford’, ’,’, ’introducing’, ’us’, ’.’]
```

Interestingly, this version has stumbled on the abbreviations Dr. and Mr., separating the stops as distinct tokens. We also get different results when processing a URL such as *http://www.facebook.com/*:

```
>>> nltk.word_tokenize("http://www.facebook.com/")
[’http’, ’:’, ’//www.facebook.com/’]
>>> nltk.regexp_tokenize("http://www.facebook.com/",pattern)
[’http’, ’:’, ’www’, ’.’, ’facebook’, ’.’, ’com’]
```

Neither tokeniser distinguishes the different uses of hyphens identified by Mikheev: all hyphenated forms are treated as a single token.

Learning activity

Use some constructed or found documents to compare the performance of `nltk.word_tokenize()` and `nltk.regexp_tokenize()` on texts which include currency symbols, the @ sign, or complex numerical expressions.

4.4 Stemming

Stemming is the process of removing word affixes to obtain the *stem* or base form. Stemming may be useful in information retrieval applications such as web search, in order to return documents containing terms with the same stem as those in the query. For example, a search on *cinematographer* will also return *cinematography*. As Bird et al. (2009, p. 107) note, stemming is not a well-defined process and different stemmers will be based on differing ideas of what the ‘stem’ of a word should be. This can be seen by comparing the outputs of two widely-used stemmers distributed

with the NLTK, the Porter and Lancaster stemmers. The following paragraph is taken from a Goldsmiths press release dated 31 May 2012:

Three artists from Northern Ireland are working with school children from South East London at Goldsmiths, University of London to create a large mural painting reflecting on issues from the past and present which dominate the local area. The artists – Danny Devenny, Mark Ervine and Marty Lyons – have worked with pupils from Addey & Stanhope School throughout the week in the Department of Educational Studies at Goldsmiths.

This is the result of running the above text through the Lancaster Stemmer:

```
['three', 'art', 'from', 'northern', 'ireland', 'ar', 'work',
'with', 'school', 'childr', 'from', 'sou', 'east', 'london', 'at', 'goldsmith',
',', 'univers', 'of', 'london', 'to', 'cre', 'a', 'larg', 'mur', 'paint',
'reflect', 'on', 'issu', 'from', 'the', 'past', 'and', 'pres', 'which',
'domin', 'the', 'loc', 'are', '.', 'the', 'art', '-', 'danny', 'devenny',
',', 'mark', 'ervin', 'and', 'marty', 'lyon', '-', 'hav', 'work', 'with',
'pupil', 'from','addey', '&', 'stanhop', 'school', 'throughout', 'the',
'week', 'in', 'the', 'depart', 'of', 'educ', 'study', 'at', 'goldsmith', '.']
```

And this is the result of running it through the Porter Stemmer:

```
 ['Three', 'artist', 'from', 'Northern', 'Ireland', 'are', 'work',
'with', 'school', 'children', 'from', 'South', 'East', 'London','at',
'Goldsmith', ',', 'Univers', 'of', 'London', 'to', 'creat', 'a',
'larg', 'mural', 'paint', 'reflect', 'on', 'issu', 'from', 'the',
'past', 'and', 'present', 'which', 'domin', 'the', 'local', 'area',
'.', 'The', 'artist', '-', 'Danni', 'Devenni', ',', 'Mark', 'Ervin',
'and', 'Marti', 'Lyon', '-', 'have', 'work', 'with', 'pupil', 'from',
'Addey', '&', 'Stanhop', 'School', 'throughout', 'the', 'week', 'in',
'the', 'Depart', 'of', 'Educ', 'Studi', 'at', 'Goldsmith', '.']
```

There are some obvious similarities and differences between the two: Lancaster converts everything to lower case while Porter leaves cases as they are; both remove *-s* from the end of apparently plural forms. There are also quite a few subtle differences.

Learning activity

1. Make lists of rules which the stemmers appear to have applied in each instance. Note that this question does not require you to read up on the history and design of these stemmers: you should hypothesise rules that would result in the particular decisions taken in these examples.
2. The following Python code implements a very simple stemmer which simply removes the ending *-s* if present, except from words ending in *-ss*. Extend the regular expression to incorporate up to six of the rules you have given in your previous answer.

   ```
   >>> def stemmer(word):
   [(stem,end)] = re.findall('^(.*ss|.*?)(s)?$',word)
   return stem
   ```

   The input string *word* is split into two substrings which match the REs *(.*ss|.*?)* and *(s)?*. The first matches any string that ends *-ss* or any sequence of zero or more characters. Since we use the 'non-greedy' star operator **?* this will not consume the letter *s* if matched by the second RE. Examples: *stemmer('less') = 'less'*, *stemmer ('s') = ''*, *stemmer ('lesss') = 'lesss'*, *stemmer ('lets') = 'let'*, *stemmer ('lest') = 'lest'*.
3. Are there any rules which cannot be simply encoded as regular expressions? Explain your answer.

4.5 Tagging

Tagging is the process of assigning annotations to items in a text, such as parts of speech. This chapter is concerned with part-of-speech or POS tagging. One problem for taggers is dealing with homonyms: words which look the same but can take different grammatical categories according to context, as in the following example:

```
>>> refuse = nltk.word_tokenize('They refuse to permit a refuse permit')
>>> nltk.pos_tag(refuse)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'),
('a', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

The words *refuse* and *permit* occur both as verbs and nouns. The program calculates the relative likelihood of these categories in context:

- A verb is more likely than a noun to follow a pronoun: *they refuse*
- A noun is more likely than a verb to follow a determiner: *a refuse* ...

Taggers also need to 'guess' at unfamiliar words according to context:

```
>>> junk = nltk.word_tokenize('Moraya zarked the dweeble with a tolchock')
>>> nltk.pos_tag(junk)
[('Moraya', 'NNP'), ('zarked', 'VBD'), ('the', 'DT'), ('dweeble', 'JJ'),
('with', 'IN'), ('a', 'DT'), ('tolchock', 'NN')]
```

The words *Moraya*, *zarked*, *dweeble*, *tolchock* are made up for this example. The tagger has guessed the categories:

Moraya: *NNP* (proper name)

zarked: *VBD* (past tense verb)

dweeble: *JJ* (adjective)

tolchock: *NN* (noun)

What is the basis for these decisions? Has the tagger made any mistakes?

4.5.1 RE tagging

The NLTK allows tagged corpora to be accessed using either the Penn Treebank tagset or a 'simplified' set (Bird et al., 2009, p. 183). This section works through a case study which rewrites the regular expression tagger of Bird et al. (2009, p. 199) using the simplified tagset and tests it on the science fiction section of the Brown corpus. As the simplified set (unlike the Penn set) does not have separate tags for singular present verbs, possessives or plural nouns, we have to use the simple tags V and N for these patterns.

Simplified RE tagger:

```
        (r'.*ing$', 'VG'),                # gerunds
...     (r'.*ed$', 'VD'),                 # simple past
...     (r'.*es$', 'V'),                  # 3rd singular present
...     (r'.*ould$', 'MOD'),              # modals
...     (r'.*\'s$', 'N'),                 # possessive nouns
...     (r'.*s$', 'N'),                   # plural nouns
...     (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
...     (r'.*', 'N')                      # nouns (default)
```

For this example we pick three random sentences from the SF corpus and run them through the tagger. Here are some results; the output has been annotated by coding F (false) or T (true) after each tagged element.

```
>>> regexp_tagger = nltk.RegexpTagger(patterns)
>>> regexp_tagger.tag(sents[10])
[('He', 'N')F, ('was', 'N')F, ('closer', 'N')F, ('to', 'N')F,
('understanding', 'VG')v, ('it', 'N')F, ('in', 'N')F,
('English', 'N')T, ('now', 'N')F, (',', 'N')F, ('although', 'N')F,
('it', 'N')F, ('could', 'MOD')T, ('never', 'N')F, ('have', 'N')F,
('the', 'N')F, ('inevitability', 'N')T, ('of', 'N')F, ('the', 'N')F,
('Martian', 'N')F, ('concept', 'N')T, ('it', 'N')F, ('stood', 'N')F,
('for', 'N')F, ('.', 'N')F]
>>> regexp_tagger.tag(sents[500])
[('This', 'N')F, ('old', 'N')F, ('world', 'N')T, ('lost', 'N')F,
('all', 'N')F, ('of', 'N')F, ('its', 'N')F, ('helium', 'N')T,
('and', 'N')F, ('trace', 'N')F, ('gases', 'V')F, ('long', 'N')F,
('ago', 'N')F, ('.', 'N')F]
>>> regexp_tagger.tag(sents[900])
[('In', 'N')F, ('between', 'N')F, ('teratologies', 'V')F,
(',', 'N')F, ('she', 'N')F, ('was', 'N')F, ('a', 'N')F,
('pretty', 'N')F, ('and', 'N')F, ('shapely', 'N')F, ('woman', 'N')T,
(';', 'N')F, (';', 'N')F]
```

54 tokens, 8 correctly tagged; accuracy = 14.8 per cent.

The results are clearly rather poor. Most correct taggings come from the default 'N', while non-default tags have introduced errors: 'gases', 'teratologies'. The simplified tagset doesn't have a tag for punctuation marks so these all get the default tag. Reading through the texts, many of the items which have been wrongly tagged are *closed-class* words such as pronouns and conjunctions. The patterns below have been extended to include some of these closed-class items as well as tagging any capitalised words as proper nouns:

```
(r'^he|him|she|her|it|they|them$', 'PRO'),    # pronouns
(r'^who|which|when|what|where|how$','WH'),    # Wh- words
(r'^and|but|or|if|while|although$','CNJ'),    # conjunctions
(r'^a|the|some|many|most$','DET'),            # determiner
(r'^[A-Z].*','NP'),                           # proper noun
(r'.*ing$', 'VG'),                            # gerunds
(r'.*ed$', 'VD'),                             # simple past
(r'.*es$', 'V'),                              # 3rd singular present
(r'.*ould$', 'MOD'),                          # modals
(r'.*\'s$', 'N'),                             # possessive nouns
(r'.*s$', 'N'),                               # plural nouns
(r'^-?[0-9]+(.[0-9]+)?$', 'NUM'),             # cardinal numbers
(r'.*', 'N')                                  # nouns (default)
```

Tested on the same sample of sentences from the science-fiction collection, the tagger now performs rather better:

```
>>> regexp_tagger.tag(sents[10])
[('He', 'NP')F, ('was', 'N')F, ('closer', 'N')F, ('to', 'N')F,
('understanding', 'VG')T, ('it', 'PRO')v, ('in', 'N')F,
('English', 'NP')T?, ('now', 'N')F, (',', 'N')F, ('although', 'CNJ')v,
('it', 'PRO')T, ('could', 'MOD')T, ('never', 'N')F, ('have', 'N')F,
('the', 'DET')T, ('inevitability', 'N')T, ('of', 'N')F, ('the', 'DET')T,
('Martian', 'NP')F, ('concept', 'N')v, ('it', 'PRO')T, ('stood', 'N')F,
('for', 'N')F, ('.', 'N')F]
>>> regexp_tagger.tag(sents[500])
[('This', 'NP')F, ('old', 'N')F, ('world', 'N')T, ('lost', 'N')F,
('all', 'N')F, ('of', 'N')F, ('its', 'N')F, ('helium', 'N')T, ('and', 'CNJ')T,
('trace', 'N')F, ('gases', 'V')F, ('long', 'N')F, ('ago', 'N')F,
('.', 'N')F]
>>> regexp_tagger.tag(sents[900])
[('In', 'NP')F, ('between', 'N')F, ('teratologies', 'V')F, (',', 'N')F,
('she', 'PRO')T, ('was', 'N')F, ('a', 'DET')T, ('pretty', 'N')F,
('and', 'CNJ')T, ('shapely', 'N')F, ('woman', 'N')T, (';', 'N')F,
(';', 'N')F]
```

18 correct = 33.3 per cent accuracy.

Note that the list of tags as given by Bird et al. (2009) does not include punctuation, although punctuation marks do have defined tags as in this excerpt:

```
[('Now', 'ADV'), ('that', 'CNJ'), ('he', 'PRO'), ('knew', 'VD'),
('himself', 'PRO'), ('to', 'TO'), ('be', 'V'), ('self', 'N'),
('he', 'PRO'), ('was', 'V'), ('free', 'ADJ'), ('to', 'TO'),
('grok', 'V'), ('ever', 'DET'), ('closer', 'ADV'), ('to', 'P'),
('his', 'PRO'), ('brothers', 'N'), (',', ','), ('merge', 'V'),
```

```
('without', 'P'), ('let', 'N'), ('.', '.')]
```

If we discount punctuation marks we now have 18/50 = 36 per cent correct.

Learning activity

Read through the examples above and see if any closed-class words have still been wrongly tagged. If so, extend the tagger patterns to include these words as well as punctuation symbols, and test the result for accuracy using random sentences from any corpus.

4.5.2 Trained taggers and backoff

You should have learned from the preceding example that while an RE tagger can be moderately successful in choosing the right word class, the results of trying to do this based solely on the surface form of a word in isolation will always be limited. Some reasons for this are:

- Individual words can often belong to different word classes: just in this sentence, *individual* can be a noun or an adjective; *can*, *word(s)*, *classes*, *sentence* and *forms* can all be noun or verb forms; *just* can be an adjective or an adverb.
- When this is the case, identifying the correct form requires looking at the **context** and not just the word in isolation. The context here may be the preceding word class or a short sequence.
- Few word endings uniquely identify a particular class: for example the RE tagger uses *-ing*, *-es* and *-ed* as indicators of verb forms, but all of these can also be found at the end of nouns.

In the taggers we have looked at so far, it is up to the developer to specify some sort of model which guides tagging decisions. A more convenient technique is called **training**: essentially, the tagger builds its own model by analysing pre-tagged sentence data, and then applies the model to other sentences in the same genre. It is essential to separate the training and test data to get a clear idea of the accuracy of a tagger. Note that at this point we will not consider how the training process works but treat it as a 'black box'. The next chapter describes some **machine learning** algorithms which can be applied to various training tasks. The following examples use the `evaluate()` method which reports how accurately a tagger marks up the test data. 'Accuracy' is used here as a technical term with a precise meaning, namely the percentage of inputs in the test set which are correctly classified (Bird et al., 2009, p. 239).

Training a unigram tagger

In this example the tagger is trained on 90 per cent of the sentences in the Brown news corpus and tested on the remaining 10 per cent. The tagger achieves an accuracy of 0.81.

```
>>> from nltk.corpus import brown
>>> bts=brown.tagged_sents(categories='news')
```

```
>>> bs=brown.sents(categories='news')
>>> size=int(len(bts)*0.9)
>>> size
4160
>>> train=bts[:size]
>>> test=bts[size:]
>>> unigram_tagger=nltk.UnigramTagger(train)
>>> unigram_tagger.evaluate(test)
0.81371474135353339
```

Bringing in context

A trained unigram tagger may achieve accuracy of above 80 per cent but it has the inherent limitation that any given word will always be assigned the same tag regardless of context. N-gram taggers (where $n > 1$) try to calculate the most likely tag in context, where the context is the word-class of the previous n-1 words. N-gram taggers on their own do not score very highly owing to the 'sparse data' problem: texts we wish to tag may contain contexts that were not present in the training data. However, better results can be achieved by combining different types of tagger through **backoff**.

Bigram tagging

Just one unfamiliar sequence is enough to throw the tagger off. In this example, the tagger has not encountered the sequence *RB, 'existed'* and so cannot tag the latter word. This means that there is no context for any remaining words in the text and the process breaks down.

```
>>> bg_tag=nltk.BigramTagger(train)
>>> bg_tag.evaluate(test)
0.10305990232233629
>>> bg_tag.tag(bs[1024])
[('Then', 'RB'), ('Dick', 'NP'), ('Hyde', 'NP'), (',', ','),
('submarine-ball', 'NN'), ('hurler', 'NN'), (',', ','),
('entered', 'VBD'), ('the', 'AT'), ('contest', 'NN'),
('and', 'CC'), ('only', 'AP'), ('five', 'CD'), ('batters', 'NNS'),
('needed', 'VBD'), ('to', 'TO'), ('face', 'VB'), ('him', 'PPO'),
('before', 'IN'), ('there', 'RB'), ('existed', None), ('a', None),
('3-to-3', None), ('deadlock', None), ('.', None)]
```

Combining taggers

While reading the above example, you may have thought that even if the tagger has not seen the word *existed* in this particular context before, it would make sense to guess that it is a verb-form based on the ending. This in fact is the essential intuition behind **backoff** tagging, as illustrated in the example below:

1. Try tagging the token with a trained bigram tagger.
2. If the bigram tagger can't find a tag, try a trained unigram tagger.
3. If this doesn't work, use a default or RE tagger.

```
>>> t0=nltk.DefaultTagger('NN')
>>> t1=nltk.UnigramTagger(train,backoff=t0)
>>> t2=nltk.BigramTagger(train,backoff=t1)
>>> t2.evaluate(test)
0.84720422605402168
```

4.5.3 Transformation-based tagging

This technique is known as Brill tagging after its inventor. The general idea is to guess the tag of each word on the first pass, then construct a set of **transformation rules** to fix mistakes. Transformation rules can take account of actual words in context or their word class (part of speech). The resulting models are a fraction of the size of n-gram taggers and are linguistically informative. See Bird et al. (2009, section 5.6), Jurafsky and Martin (2009, section 5.6).

4.5.4 Evaluation and performance

Evaluation involves the idea of a 'Gold standard': taggers are trained and evaluated against corpora which have been manually annotated and accepted as a standard. To ensure the validity of a standard annotated corpus, texts are marked up independently by multiple annotators and tested for agreement. 100 per cent agreement among annotators is very rare. Therefore, there is an inherent limitation on the accuracy of taggers themselves. See section 3.4.4 of this subject guide and Jurafsky and Martin (2009, section 5.7) for further discussion.

Learning activity

Read through Bird et al. (2009, section 5.5) and answer the following:

1. Divide the science fiction corpus into training and test sets, and evaluate the following methods of tagging:
   i. Trained unigram tagger on its own
   ii. Trained bigram tagger on its own
   iii. Bigram tagger with backoff to the unigram tagger, which backs off to your modified RE tagger
2. Given that your tagger has been developed using the science fiction corpus, which of the following other categories in the Brown corpus do you think it would score highest on, and why? Test your hypothesis.

   news; mystery (detective stories); romance

4.6 Summary

1. Some fundamental operations in text analysis are tokenisation, stemming and tagging.
2. These tasks all present particular difficulties arising from the idiosyncrasies of natural language.

3. A certain level of success can be achieved in stemming and tagging based solely on the surface form of words, using regular expressions.
4. More successful techniques involve training the analysis tools on corpora which have been marked up by human experts. Producing an annotated corpus is a highly labour-intensive task which is subject to a probably irreducible degree of error.
5. Different types of tagger are subject to particular sources of error; however, low error rates can be achieved by combining different taggers in sequence. This is known as *backoff*.

4.7 Sample examination question

The following paragraph is taken from a government report, How Fair is Britain?

> *Today, we live in a society where overt displays of prejudice are usually unlawful, and almost always socially unacceptable. Surveys suggest that we are more tolerant of difference, and less tolerant of discrimination. This is mirrored in the evolution of new laws which prohibit discrimination and require public bodies to promote equality. It is borne out by our expectations of public figures: a career in the public eye can be cut short by a bigoted comment.*

This is the result of running the above text through the Lancaster Stemmer:

```
['today', ',', 'we', 'liv', 'in', 'a', 'socy', 'wher', 'overt',
'display', 'of', 'prejud', 'ar', 'us', 'unlaw', ',', 'and',
'almost', 'alway', 'soc', 'unacceiv', '.', 'survey', 'suggest',
'that', 'we', 'ar', 'mor', 'tol', 'of', 'diff', ',', 'and',
'less', 'tol', 'of', 'discrimin', '.', 'thi', 'is', 'mir', 'in',
'the', 'evolv', 'of', 'new', 'law', 'which', 'prohibit',
'discrimin', 'and', 'requir', 'publ', 'body', 'to', 'promot',
'eq', '.', 'it', 'is', 'born', 'out', 'by', 'our', 'expect',
'of', 'publ', 'fig', ':', 'a', 'car', 'in', 'the', 'publ', 'ey',
'can', 'be', 'cut', 'short', 'by', 'a', 'bigot', 'com', '.',]
```

a) Explain what is meant by word stems, with references to examples from the above text. How can stemming be useful in applications such as machine translation?

b) Make a list of rules which the stemmer has applied in this example and discuss the motivations for the rules, including any cases where you believe rules have been applied inappropriately.

c) The following is part of the output of a bigram tagger, tested on an excerpt from the Brown corpus news genre:

```
[('Then', 'RB'), ('Dick', 'NP'), ('Hyde', 'NP'), (',', ','),
('submarine-ball', 'NN'), ('hurler', 'NN'), (',', ','),
('entered', 'VBD'), ('the', 'AT'), ('contest', 'NN'), ('and', 'CC'),
('only', 'AP'), ('five', 'CD'), ('batters', 'NNS'), ('needed', 'VBD'),
('to', 'TO'), ('face', 'VB'), ('him', 'PPO'), ('before', 'IN'),
('there', 'RB'), ('existed', None), ('a', None), ('3-to-3', None),
('deadlock', None), ('.', None)]
```

   i. What is a probable reason that the last five tokens have been assigned the tag None?

ii. Explain how more accurate tagging can be achieved by using a procedure that includes multiple taggers.

Chapter 5

Statistically-based techniques for text analysis

Essential reading

Bird et al. (2009) *Natural Language Processing with Python*, Chapters 6 and 7 especially: 6.1, 6.3–6, 7.1–2, 7.5.

Recommended reading

Cunningham (2006) 'Information extraction, automatic'.

Samuelsson (2003) 'Statistical methods', in Mitkov (2003).

Additional reading

Bird et al. (2009) Section 11.2.

Segaran (2007) *Programming Collective Intelligence*, Chapters 6, 7 and 12.

Perkins (2010) *Python Text Processing With NLTK 2.0 Cookbook*, Chapters 5 and 7.

Jurafsky and Martin (2009) *Speech and Language Processing*, Chapters 5, 6 and 22.

5.1 Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- explain the difference between supervised and unsupervised learning and list some typical applications
- describe in general terms some commonly used machine learning techniques
- describe the tasks of document classification and information extraction as applications of statistical techniques
- show understanding of information extraction by coding regular expressions for named entity recognition
- use software tools from the NLTK to train classifiers with provided datasets, and test them on novel data.

5.2 Introduction

This chapter will introduce the basics of supervised learning, training and evaluation and give an overview of applications such as types of **classification** and **information extraction**. This is to be contrasted with unsupervised learning, where an agent learns patterns in its input without being provided with any feedback (Russell and Norvig, 2010, pp. 694, 817–20). By 'supervised learning' we mean that a computer system is provided with data that has been marked up in some way by human experts, and its task is to learn how to annotate new data on the same principles. This is done by dividing the data into a *training set* and a *test set*: the system will attempt to learn a classification from the training data and this is checked by applying it to the test set. The applications we will consider in this chapter come under two main headings:

Classification is the task of determining which class input items belong to. If the inputs are words, the classes might be parts of speech. If the inputs are documents or texts, the task could be to classify them along one of the following dimensions:

- Spam or not spam: this is a standard feature of email servers nowadays, and is also applicable to other forms of electronic communication such as social media and blogs.
- Authorship: many different kinds of investigators are interested in determining from internal evidence who the author of a document is or was, from literary scholars and historians to law enforcement and intelligence agencies.
- Sentiment: businesses can apply these techniques to find out whether users of social media websites or authors of online reviews are generally expressing favourable or unfavourable views on their products.
- Relevance: information retrieval systems such as search engines need to identify documents which are likely to be relevant to a user's query.

Information extraction (IE) is concerned not just to find documents that may help to answer a user's query, but to generate answers to the query in some fixed format. In fact IE can be regarded as an application of joint or sequential classification, as it involves identifying a sequence of terms and classifying them in terms of their informational role.

Section 5.3 provides a high-level overview of some commonly used machine learning techniques. This section does not assume any mathematical knowledge beyond elementary probability and familiarity with the concept of logarithms. Section 5.4 sets out a worked example using some NLTK tools and provides some activities for you to work through yourself.

5.3 Some fundamentals of machine learning

5.3.1 Naive Bayes classifiers

Thomas Bayes (1701–1761) was a mathematician and churchman who is known to posterity for the rule which bears his name and 'underlies most modern AI systems for probabilistic inference' (Russell and Norvig, 2010, p. 495).

$$P(b \mid a) = \frac{P(a \mid b)P(b)}{P(a)}$$

Using this rule, if we have good estimates for the first three of the following terms we can calculate the fourth:

- $P(a)$ – prior probability of a
- $P(b)$ – prior probability of b
- $P(a \mid b)$ – probability of event a given the observation b
- $P(b \mid a)$ – probability of event b given the observation a

The following worked example shows how we can apply this formula to predict the probability that a given word-token belongs to a particular grammatical category. The Brown News corpus contains 100,554 word-tokens of which 35 are *run* and 7346 are tagged as *V* (using the simplified tagset). 11 instances of *run* are tagged as *V*. So the probabilities corresponding to the components of Bayes' Rule are:

- $P(a)$ – prior probability of *run* = 35/100554 = 0.000348
- $P(b)$ – prior probability of *V* = 7346/100554 = 0.073055
- $P(a \mid b)$ – probability of *run* given *V* = 11/7346 = 0.001497
- $P(b \mid a)$ – probability of *V* given *run* = (P(*run*|*V*)*P(*V*))/P(*V*) = 0.314286

Learning activity

1. Verify these calculations and confirm that the final result is equal to 11/35, the proportion of instances of *run* which are tagged with *V*. You may get slightly different results as the above figures have been rounded.
2. Reread Bird et al. (2009, section 2.2) on conditional frequency distributions and see if you can reproduce these figures. Once you have done that and compared your results with the suggested answer in Appendix C, repeat the analysis on your own choice of corpora, word tokens and grammatical categories.

Suppose we apply Bayesian reasoning to a classification task such as determining whether film reviews are 'good' (favourable) or 'bad' (unfavourable). In this case event b is either *good* or *bad* while a is a vector of features, such as a list of certain keywords which are present or absent in the review. For example Bird et al. (2009, p. 228) list some words which are strongly correlated with positive reviews in a particular corpus (*outstanding*, *wonderfully*, *damon*) and others which are more likely to be found in negative reviews (*seagal*, *wasted*). So if we assume a feature set comprising names of actors, directors, scriptwriters and so on, and various adjectives and adverbs such as [Astaire, Bardot, classic, Depp, excellent, flop, . . .], we might apply Bayes' rule as

$$P(\text{good} \mid [0,0,1,1,1,0,\ldots]) = \frac{P([0,0,1,1,1,0,\ldots] \mid \text{good})P(\text{good})}{P([0,0,1,1,1,0,\ldots])}$$

In this example, 1 indicates the presence of a particular term and 0 means it is absent. The difficulty here is that 'it is unlikely that any corpus to which we have

access will provide coverage to adequately train this kind of feature vector' (Jurafsky and Martin, 2009, p. 675). This problem is addressed by relaxing the analysis with an assumption that the features are conditionally independent, which allows us to make use of the equivalence $P(a \wedge b \mid c) = P(a \mid c)P(b \mid c)$. While a corpus will provide very few observations of the precise combination of features in a particular review, it will generally include many instances of the features individually. The above example can be reformulated using conditional independence as (highly simplified for purposes of exposition):

$$P(\text{good}) \mid P(\text{classic})P(\text{Depp})P(\text{excellent})) =$$
$$\frac{P(\text{classic} \mid \text{good})P(\text{Depp} \mid \text{good})P(\text{excellent} \mid \text{good})P(\text{good})}{P(\text{classic})P(\text{Depp})P(\text{excellent})}$$

In practice, it is rare for feature values to be completely independent – for example, reviews which contain *Herzog* are more likely than the average to include *Kinski*, as the actor Klaus Kinski featured in several films directed by Werner Herzog. On the other hand, *Herzog* is probably less likely than average to co-occur with *heartwarming* or *hilarious*. This limitation is reflected in the term 'naive Bayes' for classifiers which make a simplifying independence assumption. As a consequence they may not make fully accurate predictions of the probability of a particular classification, but have proved generally robust in predicting which alternative is more likely (Segaran, 2007, p. 124). Bayesian classifiers have the advantage that they can be trained *incrementally*: if new items are added, it is not necessary to rerun the training using the entire dataset (Segaran, 2007, p. 280).

5.3.2 Hidden Markov models

Andrei Markov (1856–1922) was a Russian statistician who amused himself by devising a procedure to predict the occurrence of letters in Pushkin's *Eugene Onegin* based on the previous one or two letters. The idea that the state of a system can be predicted on the basis of only the immediately preceding state (or a small finite number of preceding states) is generally called the **Markov property** or **Markov assumption**. Markov models are an extension of the finite-state automata we looked at in Chapter 2. These models have been applied to speech processing for some time, and more recently a variant called the **hidden Markov model** or **HMM** has found wide application; for example in part-of-speech tagging. Choosing tags for a series of words rather than for individual words in isolation is an application of *sequence classification*, which is itself a special case of *joint classification*. The term 'hidden' means that we are not only dealing with directly observable events such as words or phonemes but also with POS tags which cannot be observed but can only be hypothesised with a certain probability. HMMs generate a probability distribution over tags, which are then combined to calculate the tag sequence with the highest probability. Jurafsky and Martin (2009, pp. 173–183) describe a simplified HMM tagger which calculates for each word w_i in an input sequence:

- the probability distribution of the possible tags for that word, given the immediately previous tag
- the probability of observing w_i given any hypothesised tag t_i.

These calculations will be computed on the basis of observed frequencies within a particular corpus. They give an example of a sentence including the sequence *to race*

tomorrow, for which the possible tag sequences include TO,VB,NR and TO,NN,NR (TO in the Brown corpus is reserved for infinitival *to*). In the Brown corpus, the relevant probabilities are:

- p(NN|TO) = 0.00047
- p(VB|TO) = 0.83
- p(NR|NN) = 0.0012
- p(NR|VB) = 0.0027
- p(race|NN) = 0.00057
- p(race|VB) = 0.00012

Multiplying out the probabilities gives us:

- p(to/TO race/NN tomorrow/NR) = 0.00000000032
- p(to/TO race/VB tomorrow/NR) = 0.00000027

So the HMM tagger correctly tags *race* as a VB in this instance.

5.3.3 Information and entropy

Machine learning techniques use some concepts from information theory such as **entropy** and **information** itself, which has a precise technical meaning in this context.

Recall that the *logarithm* of a number is the power or exponent by which a fixed number (the base) has to be raised to produce that number. For example, the logarithm of 8 to the base 2 is 3, since 8 is 2 to the power of 3. This is written as $\log_2(8) = 3$. If we are in a situation where one of eight events is equally likely, such as the roll of an octahedral (eight-sided) die, we would need three binary digits or *bits* to report which one has occurred, assuming binary '000' stands for event type 1, '001' for type 2 and so on. In information theory we would say that we have received three bits of information from this event. The formula for calculating this value is

$$I(E) = \log_2 \frac{1}{P(E)}$$

So in our example of eight equally likely probabilities, the probability of the observed event is 0.125; 1/0.125 is 8 and $\log_2(8) = 3$.

Clearly the term *information* here is used in a rather specialised manner. Intuitively, it can be considered as a measure of the *reduction of uncertainty*. If you're about to witness the spin of a roulette wheel, there is rather high uncertainty as the ball could land in any one of 37 slots. In the case of a coin being tossed there are only two possible outcomes, so a correspondingly lesser degree of uncertainty. So we can say that the outcome of the roulette case gives a higher reduction of uncertainty and thus more information than the toss of a coin.

In practice, of course, we will rarely be dealing with events that all have equal probabilities. Let's consider a slightly more complex case, where we are rolling two standard (cubical) dice and are interested in the sum of the two numbers that come up. Although there are 36 different outcomes, the probability distribution of the

sums is uneven: for example, the probability of 12 (6+6) or of 2 (1+1) is 1/36 (0.028) in each case, while the probability of 7 is 6/36 or 1/6 (0.17; 1+6, 2+5 . . .). By our intuitive definition above we would expect a score of 12 to generate more information than a score of 7 and so it turns out:

$$\log_2 \frac{1}{0.17} = \log_2(5.88) = 2.56$$

$$\log_2 \frac{1}{0.028} = \log_2(35.7) = 5.16$$

Information theorists would describe the first scenario (one die) as having a greater degree of *entropy* than the two dice case (where we are only interested in the sum of the two faces). The term was adopted from thermodynamics by Claude Shannon, the founder of information theory, and for our purposes can be seen as a measure of how disorganised a particular set of values is. The formula for calculating entropy can be stated in various ways, one of which is (Bird et al., 2009, p. 243):

$$-\sum_x p(x).\log_2 p(x)$$

We can illustrate this with two simple examples: a four-sided (tetrahedral) die which has four equally likely outcomes (p = 0.25), and a toss of two coins which has three: two heads (p = 0.25), two tails (p = 0.25) and one of each (p = 0.5).

1. Four-sided die: p(1) = p(2) = p(3) = p(4) = 0.25
   Entropy = –(4 x (0.25 x log2(0.25))
   = –(4 x (0.25 x –2)
   = 2.
2. Two-coin toss: p(HH) = p(TT) = 0.25; p(TH|HT) = 0.5
   Entropy = - ((2 x (0.25 x log2(0.25))) + (0.5 x log2(0.5)))
   = – ((2 x (0.25 x –2)) + (0.5 x –1))
   = – (–1 + –0.5)
   = 1.5

You will not be expected to calculate these values manually or to write a program to do so, but it will be helpful to understand the basis of the calculations.

5.3.4 Decision trees and maximum entropy classifiers

One application of entropy is in algorithms for constructing decision trees (see Bird et al. (2009, p. 243); (Segaran, 2007, pp. 142–166)). The problem is which of a set of features to use for the topmost node of the tree, and which features to use recursively for each sub-tree. This is done by calculating the information gain or reduction in entropy for each candidate. The main advantage of decision trees is that they are very simple to understand: see the following example that was trained on the NLTK movie-review corpus:

```
if contains(bad) == False:
  if contains(worst) == False:
    if contains(waste) == False: return 'neg'
    if contains(waste) == True: return 'pos'
  if contains(worst) == True:
    if contains(present) == False: return 'pos'
    if contains(present) == True: return 'pos'
if contains(bad) == True:
  if contains(wonderfully) == False:
    if contains(realistic) == False: return 'neg'
    if contains(realistic) == True: return 'pos'
  if contains(wonderfully) == True:
    if contains(heard) == False: return 'pos'
    if contains(heard) == True: return 'neg'
```

On the other hand, decision trees have some disadvantages which are spelled out by Bird et al. (2009, pp. 245): there is a danger of overfitting the data, and they can produce less accurate results when dealing with features that are relatively independent. For example, the decision tree classifier achieved only 0.59 accuracy on one run in the movie-review task, compared with between 0.77 and 0.86 for the Bayes classifier. (Because the documents are randomly shuffled before training, we get different training and test sets each time and consequently the accuracy of prediction will vary.)

Maximum entropy classifiers work by building some probability distributions that fit the training data, and choosing the one that has the highest entropy. See Bird et al. (2009, pp. 252–3) for a worked example of a maximum entropy model. Maximum entropy models can be quite computing-intensive. It is recommended that you install the `scipy` package[1] in addition to `numpy`, as this provides faster algorithms that use less memory.

These classifiers are not documented in detail by Bird et al. (2009). One way to learn more about them is by studying the code in the `classify` folder in your NLTK distribution, which is quite thoroughly commented. Perkins (2010, pp. 170–183) gives some worked examples and advises on optimal parameter settings for Decision Tree and Maximum Entropy classifiers.

Learning activity

Read up and make notes on how *decision trees* are constructed: see for example Bird et al. (2009, pp. 242–5), Russell and Norvig (2010, pp. 697–707) or Segaran (2007, Chapter 7).

5.3.5 Evaluation

Various different methods exist for evaluating the performance of tools for the analysis of natural language. One of the simplest, which has already been used for examples in the subject guide, is *accuracy*: what proportion of the application's judgments are correct. This, however, is not always particularly useful or informative in itself. Consider the problem of *information retrieval*, where the application is

[1] `http://www.scipy.org`

required to identify documents that are relevant to a user's query. Given that most documents in any collection are not likely to be relevant, an application which simply returned *false* for each document would have an accuracy close to 100 per cent, but would not be of much help to the user. A similar situation would arise if we were searching documents for some rare syntactic construction, such as sentences which show self-embeddings to a depth of three or more. Information retrieval uses the concepts of *precision* and *recall* to provide more fine-grained and useful evaluation and these have proved to be of use in computational linguistics as well.

These values are calculated on the basis of:

precision: proportion of positive answers which are correct

recall: proportion of available positive answers which have been found.

These values may be reported separately, or combined in the so-called *F-Measure* which is defined as the *harmonic mean* of precision and recall: *(2*Precision*Recall)/(Precision+Recall)*.

5.4 Machine learning in action: document classification

In this section we will work through an example of document classification from Bird et al. (2009, pp. 227–8). This example uses the Movie Reviews Corpus and a naive Bayes classifier provided with the NLTK.

```
>>> from __future__ import division
>>> import nltk, random
>>> from nltk.corpus import movie_reviews
>>> documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
>>> random.shuffle(documents)
```

This gives us a list of pairs in which the first item is the text of a review as a list of tokens, and the second is a category from the set {`pos`, `neg`} indicating a positive or negative review. The list is randomly shuffled before dividing it into training and test sets. This randomisation means that we should not expect to get identical results every time we run this procedure, as we will be working with different training and test data each time.

The point of the procedure is to learn what characteristics of a review are associated with positive or negative judgments. This is done by training on specified **features** of the documents rather than on the raw text. In this case, a feature set is defined using the most frequently used tokens in the corpus:

```
>>> all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
>>> word_features = all_words.keys()[:2000]
```

Recall from Chapter 3 of the subject guide (p. 38) that `Freqdist` returns a frequency distribution with the most frequently occurring items coming first. We use `keys()` to extract the 2,000 most frequent tokens and assign the result to `word_features`. The function `document_features` (Bird et al., 2009, p. 228) creates a Python **dictionary** for each review, indicating whether or not it contains each word in `word_features`. This function is then called to create a list of pairs of the feature set for each review with the value *neg* or *pos*.

```
>>> featuresets = [(document_features(d), c) for (d,c) in documents]
```

This list is then divided into training and test sets; the classifier learns associations between the contents of the feature set and the values *neg* or *pos* in the training set, and is then tested on the test set. Note that we do not train the classifier directly on the contents of the reviews, but on abstracted data which is assumed to contain the most relevant and useful information. The outcome can be tested with the `accuracy` method which simply returns the proportion of times the classifier has made the correct decision: as said before this may vary somewhat as the training and test sets are randomly determined but the result 0.81 shown in the book example is fairly typical. See Bird et al. (2009, pp. 227–8; pp. 189–98) for details of this procedure, and for an explanation of Python dictionaries respectively.

Having worked through the procedure as shown in the recommended textbook, it is also interesting to test it on novel examples which are not part of the provided corpus. For the purposes of this subject guide, the classifer was tested on a small number of reviews of the film *Ginger and Rosa* (dir. Sally Potter, 2012) which varied between one and four stars out of five. The classifier correctly assessed the one-star review as 'neg' and the three- and four-star reviews as 'pos'. This test required pasting raw text from online reviews and formatting it in a suitable way for the above procedures[2]:

```
>>> review_4star = "For a long time, Sally Potter's new film -
about teen friendship overshadowed by the Cuban
missile crisis - was called Bomb. [ ... ]"
>>> review_4star_tok =nltk.word_tokenize(review_4star)
>>> df_4star = document_features(review_4star_tok)
>>> classifier.classify(df_4star)
'pos'
>>>
```

5.4.1 Summary: document classification

This summarises a procedure for supervised learning of document classification using the NLTK:

1. Documents which are to provide the training and test data need to be annotated with the appropriate class and tokenised. If you are using a corpus that comes as part of the NLTK, a tokenised version will generally be available via the `words()` method. If you wish to classify raw text data from another source, you can use the `nltk.word_tokenise()` function as in the above example or you may prefer to develop your own regular expression-based tokeniser (Bird et al., 2009, pp. 109–112).
2. The analyst must decide on the appropriate structure and content of a feature set, which will determine the data the classifier is actually trained on. This is the part of the process where linguistic intuition and expertise come into play.
3. The dataset is divided into training and test sets; or, for rigorous error analysis, into training, 'dev-test' and test sets (see Bird et al. (2009, p. 225)). Training and test datasets consist of pairs of feature sets and classes.
4. The classifier is trained using one of the implemented algorithms, such as:

[2]The example text is from Catherine Shoard's review in the *Guardian*, 7th September 2012; `http://www.guardian.co.uk/film/2012/sep/07/ginger-and-rosa-review` (last visited 27th May 2013).

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
classifier = nltk.DecisionTreeClassifier.train(train_set)
classifier = nltk.MaxentClassifier.train(train_set)
```

Perkins (2010, pp. 170–183) gives some guidance on optimal parameter settings for the Decision Tree and Maximum Entropy classifiers.

5. Once trained, the classifier can be tested on a test or dev-test set, which may lead to revision and refinement of the feature set to improve its accuracy:
`print nltk.classify.accuracy(classifier, test_set)`
6. The trained and tested classifier can then be used to classify feature sets extracted from new documents:
`classifier.classify(features)`

Learning activity

1. The worked example above taken from (Bird et al., 2009, pp. 227–8) builds a feature set using the 2,000 most frequent tokens in the corpus. This includes punctuation marks and 'stopwords' which might seem likely to occur in all or most reviews whether positive or negative. Try removing some of these tokens and making your own revisions to the feature set to see if you can improve the accuracy of the classifiers.
2. Construct a small corpus of classified movie reviews using raw text from online sources. Suitable websites include the Internet Movie Database *http://www.imdb.com* or the BBC's archive of 10 years' worth of film reviews at *http://www.bbc.co.uk/movies/*. You will have to tokenise the reviews and tag each one as 'pos' or 'neg'. Reviews are often flagged with a number of stars out of five or ten and you will have to decide a cut-off point between positive and negative reviews.
3. Train three classifiers on the NLTK Movie Reviews Corpus and test them on your mini-corpus: a decision tree, a naive Bayes classifier and a maximum entropy model. Compare their performance.
4. Test the classifiers on reviews from other domains such as books or music.

5.5 Machine learning in action: information extraction

Along with classification, information extraction (IE) is probably the largest area of interest in applied natural language processing. This is a process that takes unstructured text as its input and generates 'fixed-format, unambiguous data' as its output (Cunningham, 2006, p. 665). IE needs to be distinguished from information retrieval (IR):

- IR identifies particular documents which may be relevant to a user's query; it is up to the user to study the documents and find the information they need.
- IE analyses the documents and stores the required information in a structured format; for instance by populating a relational database or spreadsheet.

IE is of interest to businesses, scientific researchers and so on as it can significantly reduce the time and costs involved in retrieving and correlating information from heterogenous sources. (Jurafsky and Martin, 2009, p. 759) present a hypothetical example of an analyst who wants to investigate the impact of airline price increases on the value of their shares on the stock market. The analyst will most likely have access to reliable, structured historical data on share prices, but finding

announcements of price increases will mean locating and sifting through news reports and press releases. Another particularly important application area is information extraction from biomedical journal articles, as the number of publications reporting on advances in genomics is growing beyond the abilities of scientists to keep up with their results. IE also facilitates the construction of 'large databases of genomic and related information' which would otherwise take decades to complete (Jurafsky and Martin, 2009, p. 791).

IE is an example of language computation without full understanding Cunningham (2006), which does not rely on complete syntactic or semantic analysis. Research in IE techniques has been largely driven by formal evaluations with shared benchmark datasets, especially:

- The Message Understanding Conferences (MUC) sponsored by the US Department of Defense (DoD) from 1987–98
- Automated Content Extraction conferences, sponsored by the US National Institute of Standards and Technology (NIST) and the Linguistic Data Consortium (LDC) from 2000–08.

5.5.1 Types of information extraction

IE can be divided into a number of discrete tasks. As a preliminary, the documents to be analysed must be segmented into sentences, tokenised into words and other symbols, and tagged with POS labels. The main tasks that emerged from the MUC series of evaluations were (Cunningham, 2006, p. 667):

NE Named entity recognition via **chunking** of the text into noun phrases

CO Identifying repeated mentions of the same entity (coreference)

TE Template element construction: adding descriptive information to the named entities

TR Template relation construction: identifying relations between entities

ST Scenario template production: identifying events involving the TE and TR results.

Chunking here refers to a process of labelling sequences within a text such as noun phrases. It falls short of full parsing of a string as it does not aim for exhaustive syntactic analysis; the structure of sequences in between chunks may be ignored, and it typically does not apply recursively: NP-chunks do not contain other NP-chunks. This means that chunking is a suitable application of the finite-state methods that were discussed in Chapter 2 of the subject guide and can be implemented using regular expressions. Jurafsky and Martin (2009, p. 760) list two further tasks: **temporal expression recognition** and **temporal analysis**, which has to do with situating events in time. (Recognising temporal expressions is sometimes considered to be part of NE recognition.)

5.5.2 Regular expressions for personal names

As a preliminary to briefly considering statistical information extraction, we illustrate some techniques that can be executed manually using regular expression. These examples illustrate a few 'rules of thumb' (ROTs) for recognising personal names in text:

1. Words starting with capital letters and preceded by *Mr* or *Mrs*.
2. Any sequence of one or more capitalised words.
3. One or more capitalised word(s) followed by a comma and a number (indicating the person's age) such as *Pierre Vinken, 62*.
4. One or more capitalised word(s) followed by a verb that usually applies to humans: *Simon says . . .* , *Cameron congratulates . . .*

You can use NLTK's `findall()` method ((Bird et al., 2009, pp. 105–6)) to search across multiple words in a tokenised text. In this example we take a range of sentences from the Penn Treebank corpus, originally from the *Wall Street Journal*, and concatenate them as a single list:

```
>>> from nltk.corpus import treebank
>>> def concat(lists):
  biglist = []
  while len(lists)>0:
    biglist = biglist+lists[0]
    lists=lists[1:]
  return biglist

>>> tbsents = concat(treebank.sents()[200:250])
```

The list is then formatted using the `nltk.Text()` method before running it through `findall()`:

```
>>> wsj = nltk.Text(tbsents)
>>> wsj.findall(r"<Mrs?\.?><[A-Z][a-z]+>+" )
Mr. Stevens; Mr. Hatch; Mrs. Hills; Mrs. Hills; Mrs. Hills; Mrs.
Hills; Mrs. Hills; Mrs. Hills; Mr. Rapanelli; Mr. Rapanelli; Mr.
Rapanelli
```

This strategy has high precision as all these appear to be correct examples of proper names. Recall is probably low as many names occur without the titles *Mr* or *Mrs*. Note that at some point since this corpus was constructed, it has become normal to drop the full stop in titles like *Mr*, *Mrs*, *Dr* and so on, so it is safer to code this as optional in the RE.

```
>>> wsj.findall(r"<[A-Z][a-z]+>+")
Besides; Cray; Barnum; Neil Davenport; Joseph; Blanchard; Malcolm;
Hammerton; Douglas; Wheeland; All; Cray Research; Cray Computer; John;
Stevens; He; Donald Pardus; Stevens; Arthur; Hatch; He; Eastern
Edison; John; Carney; Hatch; Eastern Edison; Previously; Eastern
Edison; Robert; Tassinari; Eastern Utilities; He; The; South Korea;
Taiwan; Saudi Arabia; However; China; Thailand; India; Brazil; Mexico;
[ ... ]
```

This strategy seems good at picking out named entities but not so good at focusing on names of persons: several names of countries and companies are included in the results. Calculating precision involves some uncertainties: it is not always obvious whether a term actually refers to a person or an organisation. For example *Cray* could mean the company 'Cray Research' or 'Cray Computer'; an actual computer made by Cray; or the name of the founder Seymour Cray. Inspecting the text shows that it is used as part of the company name in this instance. Likewise, without further knowledge we can't be sure whether *Douglas* is a name of a person or a

place. And of course, searching on capitalised words picks up many sentence-initial words which are not names.

Learning activity

1. Repeat the above procedures using the same input or another corpus of your choice.
2. Continue by coding REs to implement strategies (3) and (4) and calculate the precision of your results.
3. See if you can think of any other ROTs for identifying personal names in English and implement them as REs.

5.5.3 Information extraction as sequential classification: chunking and NE recognition

As noted above, named entity recognition involves identifying a sequence of words as a proper name and classifying the entity that the name refers to. It can thus be handled as an application of **joint** or **sequential** classification as discussed in section 5.3.2 above (p. 60), where the task is to assign labels to a sequence of items; statistical classifiers do this by calculating a probability distribution over possible tag sequences and choosing the one with the highest probability.

This section will demonstrate approaches to chunking and NE recognition using this topical text (at the time of writing):

President Barack Obama has been re-elected to a second term, defeating Republican challenger Mitt Romney. America's first black president secured more than the 270 votes in the electoral college needed to win. In his victory speech before supporters in Chicago, Mr Obama said he would talk to Mr Romney about 'where we can work together to move this country forward'.

BBC News website, 7th November 2012.

The first step as usual is to tokenise and tag the text: we use NLTK's built-in tokeniser and tagger. Only the results from the first sentence will be shown here – the remainder is left as an exercise for the reader.

```
>>> s1="President Barack Obama has been re-elected to a second term,
defeating Republican challenger Mitt Romney."
>>> t1 = nltk.word_tokenize(s1)
>>> tagged1=nltk.pos_tag(t1)
>>> tagged1
[('President', 'NNP'), ('Barack', 'NNP'), ('Obama', 'NNP'),
('has', 'VBZ'), ('been', 'VBN'), ('re-elected', 'JJ'),
('to', 'TO'), ('a', 'DT'), ('second', 'JJ'), ('term', 'NN'),
(',', ','), ('defeating', 'VBG'), ('Republican', 'JJ'),
('challenger', 'NN'), ('Mitt', 'NNP'), ('Romney', 'NNP'), ('.', '.')]
```

Running the text through the regular expression chunker presented by Bird et al. (2009, Example 7-2, p. 267) gives the following output:

```
(S
  (NP President/NNP Barack/NNP Obama/NNP)
  has/VBZ
  been/VBN
  re-elected/JJ
  to/TO
  (NP a/DT second/JJ term/NN)
  ,/,
  defeating/VBG
  (NP Republican/JJ challenger/NN)
  (NP Mitt/NNP Romney/NNP)
  ./.)
```

The NLTK includes a named entity chunker that has been pre-trained on data from the ACE programme, using a maximum entropy sequential classifier. The training data itself is not provided with the NLTK (Bird et al., 2009, pp. 283–4); (Perkins, 2010, pp. 133–4). This chunker takes tagged text as input and marks it up with named entity labels: either simply `NE` if the option `Binary=True` is selected, or `PERSON`, `ORGANISATION`, `GPE`[3] etc with `Binary = False`. Note that this chunker has made different decisions from the RE chunker shown previously, as it only marks up chunks which correspond to specific types of individuals such as, in this example, persons and organisations.

```
(S
 President/NNP
 (PERSON Barack/NNP Obama/NNP)
 has/VBZ
 been/VBN
 re-elected/JJ
 to/TO
 a/DT
 second/JJ
 term/NN
 ,/,
 defeating/VBG
 (ORGANIZATION Republican/JJ)
 challenger/NN
 (PERSON Mitt/NNP Romney/NNP)
 ./.)
```

The expected output of a coreference module would consist of the following chains:

1. {Barack Obama, America's first black president, his, Mr Obama, he}
2. {Mitt Romney, Mr Romney}
3. {Barack Obama and Mitt Romney, we}

Some of these are easier to compute than others:

- *Mr Obama* has a high probability of referring to the same person as *President Barack Obama*, likewise with *Mr Romney*.
- Identifying *America's first black president* with *President Barack Obama* is more of a challenge; this is easy for us as it is part of our general knowledge, but there is

[3] Geo-Political Entity: country, region, city, etc.

a limit to how much 'world knowledge' can be built in to reference resolution algorithms.

- The pronoun *he* is easy to identify with *Mr Obama* as it follows very shortly after it; recognising that *his* also belongs to Obama is a little more tricky and relies on some syntactical analysis.
- The last example is the hardest, as the pronoun *we* is not being matched with a single referring expression in the preceding text; rather its referent has to be constructed from the combination of two noun phrases which occur at different places.

Learning activity

1. Repeat the above procedures for the second and third sentences in the example, and on your own selections of news items or other suitable texts.
2. Note any disagreements you may have with the results.

5.6 Limitations of statistical methods

Quantitative and statistical techniques have become the standard way of doing things in natural language processing and it is important to keep in mind that they are not the only viable approach, and not without their own shortcomings:

1. Computational complexity: Bird et al. (2009, p. 233) observe that for example a 10-word sentence could have 30^{10} or 600 trillion possible mark-ups using a set of 30 tags, making the problem of calculating a probability distribution over possible labellings computationally intractable. This issue has driven the development of models such as HMMs which only look at very short sequences.
2. Sparse data problem: some patterns may simply not occur in training data which may prevent them being recognised in test data or 'real text'. One way to tackle this is **smoothing**, which essentially consists of assigning a very small non-zero probability to non-occurring patterns and subtracting the same amount from higher frequency counts to make everything balance out (Jurafsky and Martin, 2009, p. 132).
3. While machine learning methods may make accurate enough predictions and classifications it is not always easy to understand how they have done this; linguists from more traditional backgrounds sometimes feel uncomfortable that they are not getting genuine insights into the way language works, when confronted with research results consisting solely of graphs and columns of numbers. One reason for the popularity of **decision trees** is that they do generate what is effectively a flow chart that is relatively easy to interpret.
4. Supervised learning requires a significant investment of time and effort in marking up text with a clearly documented, unambiguous annotation scheme. It is important that texts are marked up independently by multiple annotators with a good level of inter-annotator agreement. Sometimes several iterations are needed in order to arrive at a robust mark-up. The reason this is so important is that as with any scientific investigation, results must be *repeatable*. That is, other researchers should be able to apply the same annotation scheme to their own choice of texts, run the same algorithms and report whether or not they have

obtained the predicted results. See Bird et al. (2009, pp. 413–4) for discussion of this issue. There is increasing interest in unsupervised learning but it remains a minority pursuit for the time being.

5. While computational applications of statistical techniques, applied to large volumes of data, may seem to minimise the 'human factor' in research, they ultimately depend on human judgments and expertise in areas such as designing the appropriate features for training and deciding on which factors to apply in constructing a balanced and representative corpus (as discussed in Chapter 3 of this subject guide).
6. Finally: the majority of data-driven research has focused on the English language, and the techniques that have been developed are not always appropriate for other languages. In particular, sequential classifiers can be trained to perform well on English with its relatively fixed word order and poor morphology, but taggers based on hand-crafted grammars have achieved lower error rates on languages with 'richer morphology and less fixed word order' (Voutilainen, 2003, p. 228); see also McEnery and Wilson (2001, p. 136).

5.7 Summary

- Statistical and probabilistic methods are pervasive in modern computational linguistics. These methods generally do not aim at complete understanding or analysis of a text, but at producing reliable answers to well-defined problems such as sentiment analysis, topic detection or recognising named entities and relations between them in a text.
- This chapter has looked at the application of these methods to supervised learning, where classifiers are developed with the aid of training data which has been marked up with the correct label for each input. The classifiers are not trained on the raw data itself but on *feature sets* which are abstracted from the data according to the analyst's specifications.
- Some widely-used techniques include decision trees, naive Bayes classifiers, hidden Markov models and maximum entropy. The learning outcomes of this course do not include a deep understanding of these techniques or the ability to encode them in computer programs, but you are expected to have a general understanding of the principles which underlie them.
- This chapter has considered two important application areas for statistical methods: sentiment detection and information extraction. The former is aimed at determining whether positive or negative attitudes are being expressed towards some entity and can be directed towards the analysis of online user reviews, Twitter posts and so on. Information extraction covers a variety of activities but we have been mainly concerned with named entity recognition.

5.8 Sample examination question

a) Explain the difference between supervised and unsupervised learning, giving some general examples in the field of Natural Language Processing.

b) Briefly describe two of the following techniques in machine learning:

   i. Hidden Markov models

   ii. Decision trees

iii. Maximum entropy.

c) i. Explain why naive Bayes classifiers are called ‘naive’. Your answer should not assume prior knowledge of Bayes’ Rule.

ii. Why are machine learning methods useful for spam filtering, and why is the naive Bayes approach particularly appropriate?

d) The NLTK `findall()` method allows you to search for sequences of tokens in a text and choose what part of the result to display, enabling you to find instances of strings occurring in particular contexts. For example, assuming `news` is a tokenised text:

```
>>>  news.findall(r"<President><K.*>")
President Kennedy; President Kasavubu; President Kennedy’s;
President Krushchev
>>>  news.findall(r"<President>(<K.*>)")
Kennedy; Kasavubu; Kennedy’s; Krushchev
```

Construct search patterns to find the following types of phrase. You should think about what kinds of context these expressions can occur in. You may give more than one pattern for each answer. Explain your answers.

i. Personal names of the form firstname, lastname: Daniel Radcliffe, Nitin Sawhney, Barack Obama, etc.

ii. First names which are more commonly male: John, Abdul, Ivan.

iii. First names which are more commonly female: Laura, Fatima, Mary.

iv. Names of streets: Downing Street, St James, Pennsylvania Avenue.

v. Place names like Paris, Gaza, Germany, Tibet, New York.

Chapter 6

Analysing sentences: syntax and parsing

Essential reading

Bird et al. (2009): *Natural Language Processing with Python* Chapters 8.1–8.4, 8.6, 9.1, 9.3.

Recommended reading

Jurafsky and Martin (2009) *Speech and Language Processing*, Chapters 12.1–12.3, 12.6, 13.1–13.4, 14.1

Additional reading

Russell and Norvig (2010) *Artificial Intelligence: a Modern Approach*, Chapters 23.1–23.2.
Carroll (2003) 'Parsing' in Mitkov (2003).

6.1 Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- use a variety of syntactic parsers, including some provided with the NLTK and some from other sources
- explain some fundamental parsing techniques at a general level, focusing on top-down, bottom-up and well-formed substring approaches
- define formal context-free grammars which can handle phenomena such as agreement, unbounded dependencies, verb categories and ambiguity, and explain how traditional context-free grammars have been extended to deal with these issues
- construct tree diagrams to represent the syntactic structure of grammatical sentences
- explain the rationale for probabilistic parsing methods, and define and work with a simple probabilistic CFG.

6.2 Grammars and parsing

It is important to keep in mind the differences between a **grammar** and a **parser**. A grammar, in the sense that the term is used by computer scientists and linguists, is a

set of rules that defines a **language**, where a language is understood as a set of sequences of items from a **vocabulary**. That is, by applying the rules of a grammar we can (in theory) determine whether or not a particular sequence is a **sentence** in the language defined by the grammar, and we may also use the rules to analyse the internal structure of sentences (or structures, in the case of ambiguity). A **parser** is a computer program that applies the rules of a grammar in a systematic manner to carry out these tasks. A **recogniser** is a program that executes only the first task. Parsing has been a very active field of research both in computational linguists and in mainstream computer science, as many of the techniques used to parse computer programs can be carried across to the analysis of natural language.

6.3 Complicating CFGs

This section picks up on some topics that were introduced in Chapter 2, which introduced a distinction between regular and context-free grammars. In this chapter we will focus on context-free grammars since most (if not all) researchers consider that at least context-free power is needed to cope with the syntax of natural languages. Recall that a context-free grammar was defined as a series of production rules of the form $LHS \rightarrow RHS$ where:

- LHS and RHS stand simply for 'left hand side' and 'right hand side' respectively
- LHS is a single atomic non-terminal symbol
- RHS is a sequence of atomic symbols which may be:
  - terminal symbols in the vocabulary of the language
  - non-terminal symbols or **categories** in the grammar of the language
  - the empty string ϵ which has no audible pronunciation.

It turns out that rules of this kind are not optimal for representing grammatical structures in natural language as will shortly be demonstrated. Different ways of extending the grammar have been proposed, including Chomsky's notion of extending the grammar with a new category of rules called **transformations** which could re-arrange the output of the CF rules in various ways, such as deleting items, introducing new content or re-ordering material ((Chomsky, 1957/2002)). A more common approach in computationally-oriented formalisms is to relax the requirement that non-terminal symbols be atomic, so that they can carry different kinds of information. We will consider three different phenomena which are problematic for the kind of grammar that was presented in Chapter 2 of the subject guide: verb categories, agreement and unbounded dependencies.

6.3.1 Verb categories

For the sample examination question in Chapter 2 of this guide, you were asked to revise a grammar including the rules shown below to prevent it generating sequences like **Oscar put*, **Oscar died the table*:

VP $\rightarrow$ *V*

VP $\rightarrow$ *V NP*

VP $\rightarrow$ *V NP PP*

You probably concluded that the problem stemmed from having a single category *V* for verbs like *die*, *put*, *saw* and so on: these are all verbs but enter into different patterns with other words in the sentence. So we can say *Oscar died*, but not usually *Oscar put*; *Oscar saw the table* but not *Oscar put the table*. Traditionally, these verbs are classed as belonging to different categories:

Intransitive verbs do not take any object (do not have an NP following them) as in *Oscar died*, *Oscar slept*.

Transitive verbs have a direct object: *Oscar saw the table*, *Oscar called his wife*.

Ditransitive verbs take 'double objects': *give a dog a bone*, *tell the children a story*.

'Dative' verbs take an object NP followed by an indirect object as a PP: *put the rubbish in the bin*, *throw the plates on the floor*.

Sentential complement verbs are followed by a clause: *he thinks it will rain*, *she said she was hungry*.

This can be formulated in different ways: by defining distinct atomic categories like IV, TV etc, or by specifying the subcategory as a feature value in a complex node. In the first case, we could reformulate the grammar to include rules like:

$VP \rightarrow IV$

$VP \rightarrow TV\ NP$

$VP \rightarrow DatV\ NP\ PP$

$IV \rightarrow sleep \mid die \mid \ldots$

$TV \rightarrow see \mid eat \mid \ldots$

$DatV \rightarrow give \mid put \mid \ldots$

While this technique succeeds in capturing the correct distributions, we have lost a generalisation in that nothing indicates that words like *sleep*, *die*, *eat*, *prevaricate* and so on belong to a single category of *verb*. Using complex nodes consisting of clusters of features, we can still label items as verbs while expressing their subcategorisation as feature values. The example rules below use this technique:

S	→	NP VP[SUBCAT=y]
VP[SUBCAT=intrans]	→	V[SUBCAT=intrans]
VP[SUBCAT=trans]	→	V[SUBCAT=trans] NP
[...]		
V[SUBCAT=intrans]	→	slept
V[SUBCAT=trans]	→	saw
[...]		

In syntactic sentence structures licensed by grammars of this kind, the value of a feature such as SUBCAT on a verb node must **unify** with the value of the same feature on the dominating VP node. Unification is a computational technique for 'merging the information content of two structures and rejecting the merge of structures that are incompatible' (Jurafsky and Martin, 2009, p. 256). Grammars which incorporate this kind of formalism are termed **unification grammars**. This makes it impossible for the grammar to generate a sentence which contains, for example, an intransitive verb in a context where transitive verbs are required. See Figures 6.1 and 6.2 where the SUBCAT values on the VP nodes match those on the V node. So, for example, we cannot generate a sentence like *Oscar died the table* as the SUBCAT features on the *V* and *VP* nodes would have different values and would fail to unify.

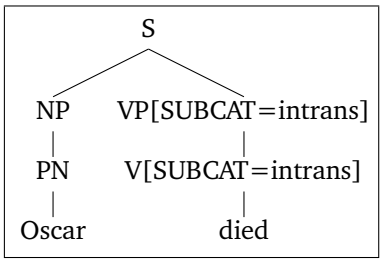

Figure 6.1: Intransitive verb.

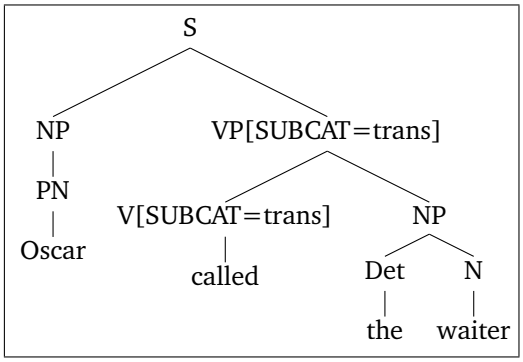

Figure 6.2: Transitive verb.

Learning activity

Pick some of the example sentences in Question (1b) in Section 2.8 (p. 28) and draw syntax trees labelled with feature structures to demonstrate why they are grammatical or ungrammatical respectively, according to reasonable grammar rules which you should also provide.

6.3.2 Agreement

Agreement is the phenomenon whereby the morphological form of some words in a sentence is determined by the form of other words they agree with; this includes number, person:

Number agreement: we can say *The boy sees some girls*, *Many girls see a hill*, but not **The boy see many girl*, **Some boys sees a girls*. (Recall that the '*' symbol is conventionally used in linguistics to mark ungrammatical or otherwise incorrect sequences.) Many languages other than English have number agreement between adjectives and nouns, as in the following examples from Swahili:[1].

- *mtoto* – 'child' singular
- *watoto* – 'child' plural
- *mdogo* – 'small' singular

[1] Joan Russell, 2010, *Complete Swahili: Teach Yourself*.

- *wadogo* – 'small' plural
- *mtoto mdogo* – 'a small child'
- *watoto wadogo*– 'small children'

Tense agreement: tenses of verbs must be consistent between clauses in a sentence as in:

- When I was a child, I spoke as a child.
- *When I was a child, I speak as a child.

Person agreement: verb forms must agree with their subject (or in some languages, with other arguments in a sentence).

- I am the judge.
- *You am the judge.

In this chapter we will concentrate on number agreement. You may like to investigate other forms of agreement as an exercise.

The grammar rules need to capture number agreement in some way; so that, for example, a plural *Det* should only go with a plural noun to form a plural *NP*, and plural *NPs* should only go with plural *VPs*. This can be done by coding different versions of the rules for each combination, for example:

$\text{NP}_{pl} \rightarrow \text{Det}_{pl}\ \text{N}_{pl}$
$\text{NP}_{sing} \rightarrow \text{Det}_{sing}\ \text{N}_{sing}$

However, this rapidly leads to a combinatorial explosion, and a more usual practice is to use **features** which can take the value **singular** or **plural**, and code the grammar rules so that features have to have the same value in appropriate cases. See Table 6.1 and Figure 6.3. Note that the use of parentheses in the VP rule means that the element in question is optional.

S	→	NP[NUM=x] VP[NUM=x]
NP[NUM=x]	→	Det[NUM=x] N[NUM=x] PP
NP[NUM=x]	→	Det[NUM=x] N[NUM=x]
VP[NUM=x]	→	V[NUM=x] NP (PP)
PP	→	P NP
Det[NUM=sg]	→	a | the
Det[NUM=pl]	→	the|several
V[NUM=sg]	→	saw | was
V[NUM=pl]	→	saw | were
N[NUM=sg]	→	boy | girl | man | dog | hill | park | telescope
N[NUM=pl]	→	boys | girls | men | dogs | hills | parks | telescopes
P	→	in | on | by | near | with

Table 6.1: Grammar rules for number agreement.

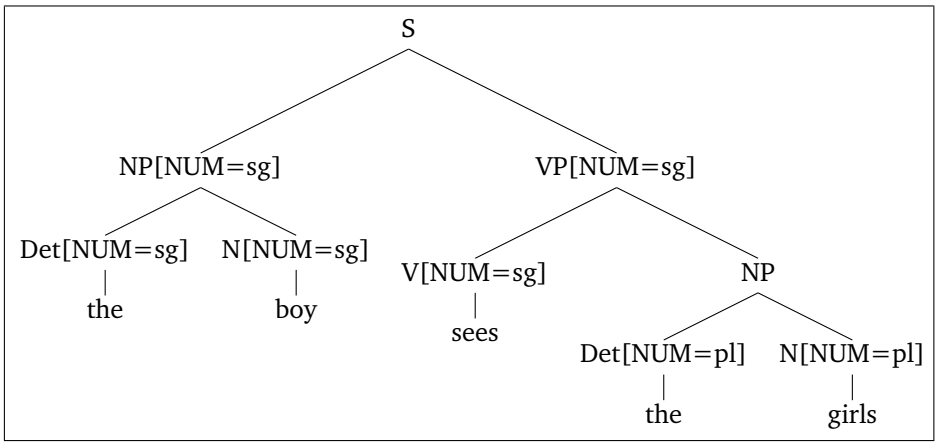

Figure 6.3: Number agreement

Learning activity

1. Extend the above grammar to combine verb categorisation and number agreement, so that it will correctly predict the following judgments:
   - The waiter pours the wine.
   - *The waiter pour the wine.
   - Oscar dies.
   - *Oscar dies the table.
   - *Oscar die.
   - The waiter put several bottles on the table.
   - *The waiter puts the bottle.
   - *The waiters puts the bottle on the table.
2. Extend the grammar further so that it can handle person as well as number agreement as in the following examples. This will involve a new feature PERS for person, as in the tree diagram shown in figure 6.4.
   (a) The dog sees the rabbit.
   (b) *You sees the rabbit.
   (c) I have a headache.
   (d) *John have a headache.
   (e) John sneezes.
   (f) *We sneezes.

6.3.3 Unbounded dependencies

Note that the starred strings are not grammatical sentences:

– You like linguistics.

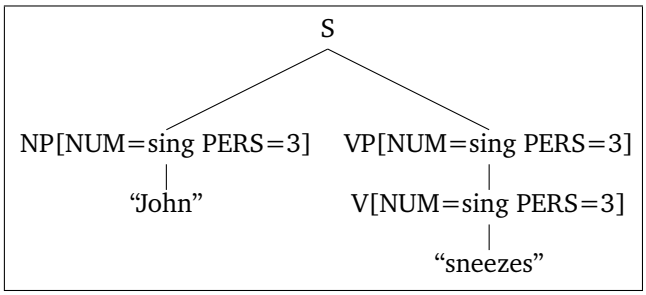

Figure 6.4: Person agreement.

– *You like.

– You put the card in the slot.

– *You put in the slot.

– *You put the card.

So, a formal grammar should not include rules like:

VP → TV
VP → DatV PP
VP → DatV NP

However, we seem to need such rules for sentences like:

– What subject do **you like**?

– Which card did **you put in the slot**?

– Where did **you put the card**?

In each of these examples, something seems to be missing if you take the bold phrase on its own. The problem here is that the grammar needs to generate 'incomplete' clauses such as *put the rubbish*, *the girls saw* but only in the context of a question or relative clause where there is a noun phrase or interrogative phrase corresponding to the gap (indicated by the underscore _):

– *Which girls* did John think he saw _ at the park?

There can be an unlimited number of words and phrasal boundaries between the wh-item and the gap, hence *unbounded*:

– *Who* did you say Mary told John she thinks Bill saw _ at the racecourse?

Standard context-free grammars can't model unbounded dependencies as there is no way a particular rule can be triggered by content in another part of the tree – that is what 'context-free' means. As explained in Bird et al. (2009, Chapter 9), one standard solution is the use of 'slash categories' to license a 'gap' in the syntax tree which can be arbitrarily far from the 'licensing' item. The basic idea is that if a clause starts with an interrogative or relative pronoun, this is a signal that the current clause or one further down the tree is missing an NP. Questions involving this construction are known in the literature as **Wh-questions**. This technique was introduced by Gerald Gazdar in the 1980s.

The general format of the rules needed for these examples is shown in Table 6.2. Note that this grammar includes 'metarules' which actually stand for a class of rules that can be obtained by replacing XP with categories such as NP or PP. Note also that having introduced feature structures for verb categories and agreement, we revert to atomic symbols for this section to simplify the exposition, as replacing them all with feature structures would take up space and make the resulting tree structures rather hard to read. We have also suppressed the number and verb category features in the interests of readability. There is no one 'correct' way to formulate these rules; for example, the grammar in Bird et al. (2009, example 9.3) does not have a separate Aux category but treats this as a feature value on the V node.

S	$\rightarrow$	XP[WH=yes] Aux S/XP
S/XP	$\rightarrow$	NP/XP VP
S/XP	$\rightarrow$	NP VP/XP
VP/XP	$\rightarrow$	V NP/XP
VP/XP	$\rightarrow$	V S/XP
Aux	$\rightarrow$	can | may | must | did | will ...
NP/NP	$\rightarrow$	ϵ (the "empty symbol")
NP	$\rightarrow$	Det N (WH-Pro) S/NP
NP[WH=x]	$\rightarrow$	Det[WH=x] N
NP[WH=x]	$\rightarrow$	Pro[WH=x]
Det[WH=yes]	$\rightarrow$	which | what | whose | ...
Det[WH=no]	$\rightarrow$	the | a| that| those | ...

Table 6.2: A grammar for unbounded dependencies.

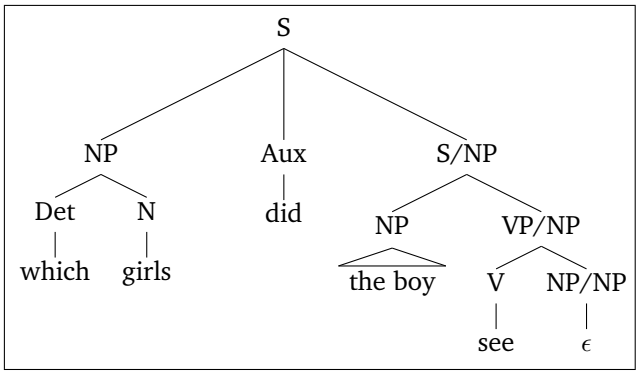

Figure 6.5: Wh-question.

6.3.4 Ambiguity and probabilistic grammars

Ambiguity is pervasive in natural language but we rarely notice this because one reading is usually most plausible in context:

1. *The boy saw a girl with a telescope* – who had the telescope?
2. *The builders dumped the rubble from the house in the truck* – where was the house? Where were the builders?
3. *Add three cloves chopped garlic and one fresh chilli or three dried chillis* – if you use dried chillis, should you add garlic?

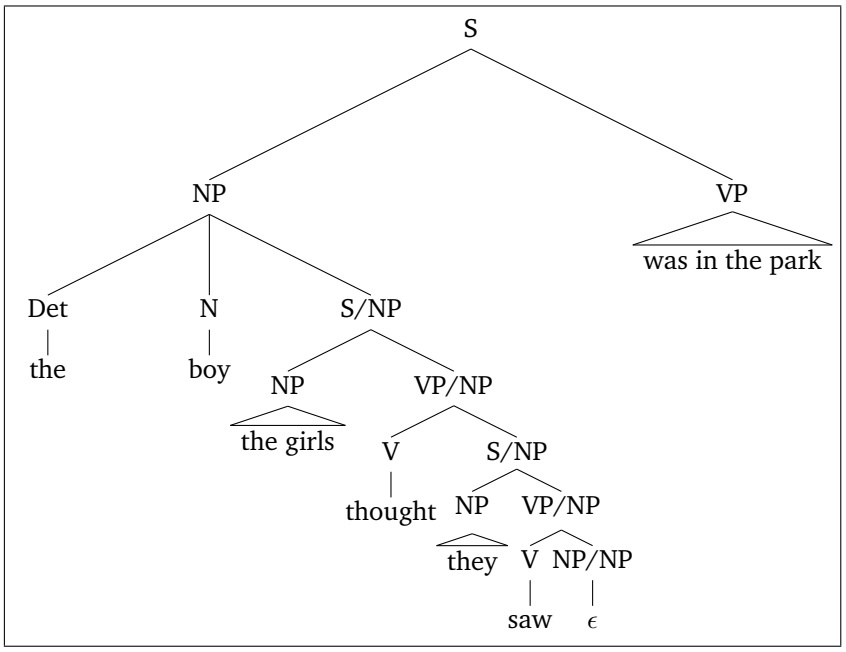

Figure 6.6: Relative clause.

These are examples of very common sources of syntactic ambiguity: **attachment** and **coordination**. In examples (2) and (3), the question is which phrasal nodes the PP should be attached to. In example (3) the issue is the relative scope of the conjunctions *and* and *or*. Because of our experience of looking through telescopes, seeing rubble being dumped and cooking, we tend to see one reading as most likely to be the intended one in each case.

However, automated parsers lack our common-sense judgments of the most likely interpretation but can instead choose the syntactic analysis that has the highest **probability**. This is not based on evaluating the possible meanings of a sentence but on the relative frequency of the grammatical structures that may be assigned to particular sequences of words. A **probabilistic context-free grammar** (PCFG) as shown in Table 6.3 calculates the most likely parse tree for a sentence on the basis of the probabilities of the possible sub-trees for each constituent phrase. Every rule of the form LHS $\rightarrow$ RHS is assigned a probability, and the probability of a parse is the product of the probabilities of all the rules used to expand each non-terminal node in the parse tree.

S	$\rightarrow$	NP VP	[1.0]
NP	$\rightarrow$	NP PP	[0.3]
NP	$\rightarrow$	Det N	[0.7]
VP	$\rightarrow$	V NP	[0.9]
VP	$\rightarrow$	V NP PP	[0.1]
PP	$\rightarrow$	P NP	[1.0]

Table 6.3: A probabilistic grammar

Notes

- Because these are probability distributions, the numbers for all rules with the same LHS must sum to 1.0.
- These numbers are entirely made up for pedagogical purposes.

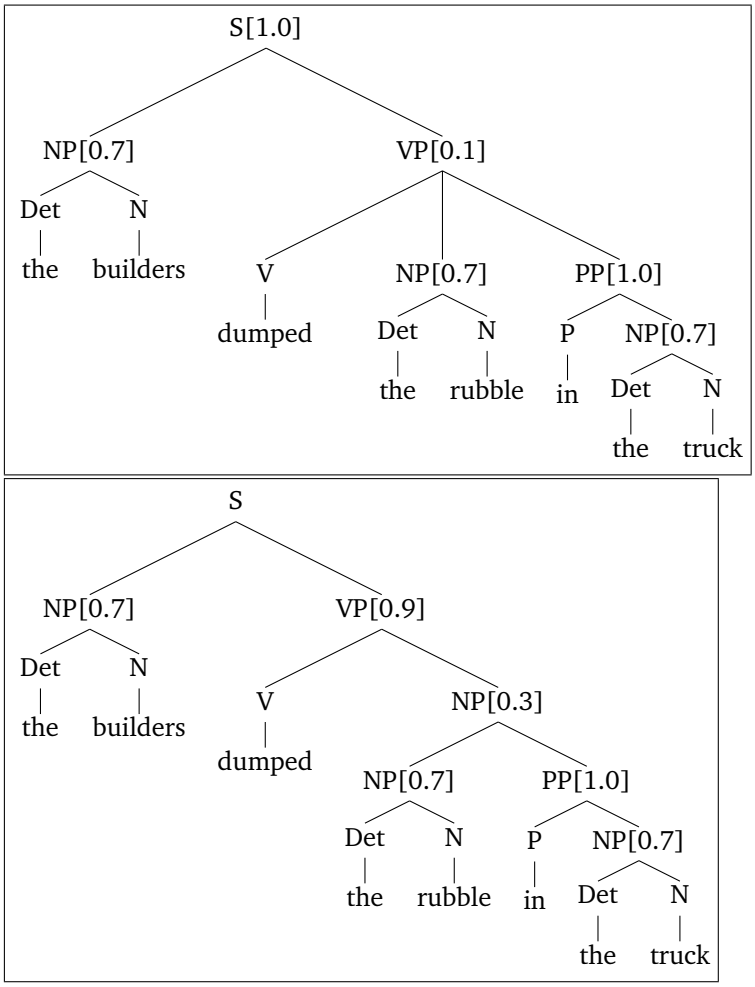

Figure 6.7: Two analyses of *The builders dumped the rubble in the truck*.

Learning activity

1. Calculate which of the two analyses for *The builders dumped the rubble in the skip* has the higher probability according to the weightings of the various rule applications.
2. The trees could be seen to correspond with two readings of the sentence:

   a) The builders dumped the rubble from somewhere else into the truck.

   b) The builders took the rubble that was in the truck and dumped it somewhere else.

   Which of these seems more natural to you? Does this accord with your answer to Question 1 above?
3. The sentence could also be read as saying that the builders were in the truck when they dumped the rubble. This would require an additional grammar rule *VP* → *VP PP*. Add this rule to the grammar, adjust the probabilities arbitrarily (ensuring that the probabilities for each rule expanding a non-terminal symbol sum to one) and recalculate the relative likelihood of the three different readings that are now allowed.

6.4 Parsing

Parsers apply grammar rules in order to determine whether a particular string is a sentence in the language defined by the grammar, and if so, to construct a representation of the grammatical structure(s) of the sentence. Two main classes of strategy are:

- Top-down: essentially, using the grammar rules to hypothesise structures and checking the input matches the proposed structure. A fundamental top-down strategy is **recursive descent**.
- Bottom-up: taking the input words in sequence, determining which grammatical class(es) they belong to and computing whether they can be combined to form a sentence according to the rules. A fundamental bottom-up algorithm is **shift-reduce**.
- Pure top-down and bottom-up strategies can be inefficient and may sometimes fail to find a parse. The technique of **chart parsing** uses dynamic programming methods to store intermediate results for later reuse.

As noted earlier, a parser is a program which processes a sequence of words according to a set of grammar rules, and outputs a representation of the syntactic structure if the sequence forms a grammatical sentence.

Example: given the grammar:

S → NP VP

NP → Det N

VP → V

Det → 'a'

N → 'dog'

V → 'barks'

and the sequences:

1. A dog barks
2. The dog barked

a parser should output the following:

1. ```
   (S
   (NP
   (Det 'a')
   (N 'dog'))
         (VP
   (V 'barks') ))
   ```
2. None (failure)

Parsers implement various techniques for systematically searching the space of trees licensed by a grammar to find one or more that matches an input string. The simplest of these parsing algorithms are probably:

1. Recursive-descent, a form of top-down parsing.
2. Shift-reduce, a form of bottom-up parsing.

The following sections will briefly describe these two techniques but will abstract away from issues of **search**. Readers can find a fuller exposition in Jurafsky and Martin (2009, section 13.1, pp. 462–466) and many other textbooks. You will not be expected to encode parsing algorithms from scratch as part of the learning outcomes of this course but you should try out the various parsers provided with the NLTK on your own examples of English sentences. Additionally, we will not go into the complications of parsing with feature-structure grammars but will only consider grammars that use atomic non-terminal symbols.

6.4.1 Recursive descent parsing

Top-down means that the parser starts from the top-level goal of finding an *S* and uses the grammar rules to generate sub-goals.

Given the example above, the goal is to recognise *A dog barks* as a sentence (*S*). The procedure (simplified for exposition) is:

- Does 'A' match *S*?
- No; expand $S \rightarrow NP\ VP$
- Does 'A' match *NP*?
- No; expand $NP \rightarrow Det\ N$
- Does 'A' match *Det*?
- Yes, by rule $Det \rightarrow$ 'a'
- Does 'dog' match *N*?
- Yes, by rule $N \rightarrow$ 'dog'
- Does 'barks' match *VP*?
- No; expand $VP \rightarrow V$
   ```

- Does 'barks' match *V*?
- Yes; *success*.

The input string is now exhausted and in the process the parser has built the structure `(S (NP Det N) (VP V))`.

Important points

A recursive-descent parser frequently has to backtrack and try different rules. For instance, in the trace shown below, the parser first tries the rule $VP \rightarrow V\ NP$. However as it cannot match *a man in the park* to *NP*, it has to discard this attempt and retry with $VP \rightarrow V\ NP\ PP$. Note that this means re-parsing the word 'saw' as a *V* and 'a man' as an *NP*, even though these items were correctly parsed on the first attempt.

Grammars to be used with top-down parsers have to be carefully designed to avoid left-recursion; namely rules of the form $X \rightarrow X \ldots$. Suppose, for example, the first *NP* rule were $NP \rightarrow NP\ PP$. Given that there is no lexical item of the category *NP*, the parser will go into an infinite loop recursively executing this rule and trying to match the current input item with *NP*.

Figure 6.8 shows an example using NLTK's built-in parser. In the trace, the asterisk '*' can be seen as the 'read head'; anything to the left of it is input that has been consumed, anything to the right represents outstanding goals.

(See Appendix D for a full trace of the parse.)

6.4.2 Shift-reduce parsing

Bottom-up means that the parser starts by trying to match input items against the RHS of grammar rules, working upwards as larger units of structure are recognised.

> *The shift-reduce parser repeatedly pushes the next input word onto a stack . . . this is the shift operation. If the top n items on the stack match the n items on the righthand side of some production then they are all popped off the stack, and the item on the lefthand side of the production is pushed onto the stack. This . . . is the reduce operation". (Bird et al., 2009, p. 305)*

Unlike the recursive-descent parser, the shift-reduce parser as implemented in the NLTK cannot backtrack to undo earlier decisions and so may reach a dead end even when parsing a sentence that has a grammatical structure according to the grammar. Figure 6.9 shows an example of a failed parse. See Bird et al. (2009, Figure 8-5, p. 305) for a successful parse.

6.4.3 Parsing with a well-formed substring table

We have seen that both types of parsers may fail on particular formulations of grammatical rules and that a recursive-descent parser may introduce inefficiencies by repeatedly re-parsing the same strings on backtracking. Chart parsing is a technique which stores intermediate results in a **well-formed substring table (WFST)** so that

```
>>>s = "John saw a man in the park".split()
>>> grammar1 = nltk.parse_cfg("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw"
  NP -> "John" |  Det N
  Det -> "a" | "the"
  N -> "man" | "park"
  P -> "in"
  """)
>>> rdp=nltk.RecursiveDescentParser(grammar1,trace=2)
>>> for tree in rdp.nbest_parse(s):
print tree

Parsing 'John saw a man in the park'
    [ * S ]
  E [ * NP VP ]
  E [ * 'John' VP ]
  M [ 'John' * VP ]
  E [ 'John' * V NP ]
  E [ 'John' * 'saw' NP ]
  M [ 'John' 'saw' * NP ]
    [ ... ]
  E [ * Det N VP ]
  E [ * 'a' N VP ]
  E [ * 'the' N VP ]
(S
  (NP John)
  (VP
    (V saw)
    (NP (Det a) (N man))
    (PP (P in) (NP (Det the) (N park)))))
```

Figure 6.8: Trace of recursive descent.

they can be looked up when needed rather than cycling through all the rules over and over again. For example, when parsing *John saw a man in the park*:

- The beginnings and ends of all substrings are indexed as follows: *(0) John (1) saw (2) a (3) man (4) in (5) the (6) park (7)*.
- At a certain point in the parse the table will include the following information:
  - (0–1) is a NP
  - (1–2) is a V
  - (2–3) is a Det
  - (3–4) is a N
  - (2–4) is a NP
- This is the point where the recursive-descent parser had to backtrack and the shift-reduce parser failed because the input matched the rule $VP \rightarrow V\ NP$ but the sentence was not completely parsed. The parser can retry with the rule $VP \rightarrow V\ NP\ PP$ but does not need to re-parse the words *saw a man*, as it can just retrieve

```
>>> srp=nltk.ShiftReduceParser(grammar1)
>>> srp=nltk.ShiftReduceParser(grammar1,trace=2)
>>> print srp.parse(s)
Parsing 'John saw a man in the park'
    [ * John saw a man in the park]
  S [ 'John' * saw a man in the park]
  R [ NP * saw a man in the park]
  S [ NP 'saw' * a man in the park]
  R [ NP V * a man in the park]
  S [ NP V 'a' * man in the park]
  R [ NP V Det * man in the park]
  S [ NP V Det 'man' * in the park]
  R [ NP V Det N * in the park]
  R [ NP V NP * in the park]
  R [ NP VP * in the park]
  R [ S * in the park]
  S [ S 'in' * the park]
  R [ S P * the park]
  S [ S P 'the' * park]
  R [ S P Det * park]
  S [ S P Det 'park' * ]
  R [ S P Det N * ]
  R [ S P NP * ]
  R [ S PP * ]
None
```

Figure 6.9: Tracing a shift-reduce parser.

these entries from the table. See Jurafsky and Martin (2009, pp. 482ff.) for details of the **chart parsing** approach including the notion of an **agenda** that determines the order in which WFST entries are processed.

6.4.4 Finite-state machines and context-free parsing

Finite-state machines of the kind shown in Chapter 2 specify processes for recognising regular languages but are not adequate for parsing context-free languages. A context-free parser cannot be restricted to a finite number of states but will contain data structures such as the stack in the shift-reduce algorithm which may need to hold an indefinite number of items. This corresponds to the structural characteristic of context-free grammars that allows for indefinite levels of nesting where bracketed items such as *if* ... *then*, *either* ... *or* must be matched in the correct order. The technical term for this is that context-free grammars may include recursive **self-embedding** or **centre-embedding** rules of the form $A \rightarrow \alpha A \beta$ where α and β are non-empty strings of terminal or non-terminal symbols. See Jurafsky and Martin (2009, Section 12.6) for further discussion.

Learning activity

1. Run the graphical recursive descent demonstration program `nltk.app.rdparser()` (Bird et al., 2009, pp. 303–4). Note the amount of backtracking that takes place when parsing the verb phrase.
2. Run the graphical shift-reduce demonstration program `nltk.app.srparser()` (Bird et al., 2009, p. 305). Experiment with selecting `Shift` and `Reduce` in different orders until you have successfully parsed the input sentence.
3. Type some of the example sentences from this chapter and Chapter 2 of the subject guide, as well as your own examples, into the online Stanford Parser at `http://nlp.stanford.edu/software/lex-parser.shtml` and study the resulting output.
4. Revisit the sample examination question 1(b) from Chapter 2 of the subject guide. Code your new grammar rules in the format shown in Figure 6.8 and run the NLTK recursive descent and shift-reduce parsers on all the examples from this question. Check whether they correctly parse the well-formed sentences and fail on the ill-formed examples.

6.5 Summary

This chapter resumes the discussion of natural language syntax that was introduced in Chapter 2 of the subject guide, concentrating on context-free grammar formalisms and various ways they need to be modified and extended beyond the model that was presented in that chapter:

1. The grammar needs to distinguish different categories of verbs, while retaining the generalisation that they all belong to the same grammatical class at an appropriate level of abstraction. This is achieved by replacing atomic terminal nodes with more complex feature structures, which can encode several different kinds of information.
2. Feature structures can also be used for handling agreement between different elements such as verbs and their subjects, or adjectives and nouns, as well as for unbounded dependencies.
3. Context-free grammars frequently allow a sentence to be analysed in many different ways, although language users are not always aware that sentences are multiply ambiguous as one particular reading is likely to be the most salient in context. This can be simulated in a formal grammar by assigning probabilities to individual grammar rules, which provides a probability distribution over the different possible analyses.
4. Formal grammars do not encode any kind of processing strategy but simply provide a declarative specification of the well-formed sentences in a language. Parsers are computer programs that use grammar rules to analyse sentences; parsing of both artificial and natural languages is a long-standing, highly active area of research and we have briefly looked at some of the most basic techniques.

6.6 Sample examination question

S $\rightarrow$ NP VP

NP $\rightarrow$Det N

NP $\rightarrow$ Det N PP

VP $\rightarrow$ V NP

VP $\rightarrow$ Cop Adj

PP $\rightarrow$ P NP

Det $\rightarrow$ a | the | my | twelve | some | most| ϵ

N $\rightarrow$ friend | coat | sales | buttons | shoe | shoes | car | cars | student | students

V $\rightarrow$ bought | likes | like | has | have

Cop $\rightarrow$ is | are | was | were

P $\rightarrow$ in | with | of

Adj $\rightarrow$ happy | angry | sad

a) Using the above grammar rules, draw syntax trees for:
   i. My friend is angry.
   ii. My friend bought a coat with twelve buttons.
   iii. Some students have cars.

b) Show some key steps in the execution of a recursive descent parser for the above grammar including any backtracking, with the input 'My friend bought a coat in the sales'.

c) Modify the grammar using feature structures so that it generates the unstarred sentences below as well as (a)(i–iii) above but not the starred ones. Explain the reasons for your modifications.
   i. My friends are students.
   ii. My friend is a student.
   iii. Most coats have buttons.
   iv. * Most coats have button.
   v. * The student are my friend.
   vi. * Twelve student bought a car.
   vii. * The students bought some shoe.

Appendix A

Bibliography

Bird, S., E. Klein, and E. Loper *Natural Language Processing with Python.* (Sebastopol, CA: O'Reilly Media, 2009) [ISBN 9780596516499].

Carroll, J. 'Parsing' , 2003. In Mitkov, R., (ed.) *The Oxford Handbook of Computational Linguistics*, pp. 233–248.

Chomsky, N. *Syntactic Structures.* (The Hague: Mouton, 1957/2002) second edition [ISBN 9780596516499].

Cunningham, H. 'Information extraction, automatic' *Encyclopedia of Language and Linguistics, Second Edition* , 2006. Electronic preprint available at `http://gate.ac.uk/sale/ell2/ie/`; last visited 27th May 2013.

Gazdar, G., E. Klein, G. Pullum, and I. Sag *Generalised Phrase Structure Grammar.* (Oxford: Blackwell, 1985) [ISBN 9780674344556].

Hayes, P. and K. Ford 'Turing test considered harmful' in 'Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence', (1995) pp. 972–997.

Huddleston, R. and G. Pullum *The Cambridge Grammar of the English Language.* (Cambridge: Cambridge University Press, 2002) [ISBN 9780521431460].

Jurafsky, D. and J. H. Martin *Speech and Language Processing.* Prentice Hall Series in Artificial Intelligence (New Jersey: Pearson Education, 2009) second edition [ISBN 9780131873216].

Kay, M. 'Functional unification grammar: a formalism for machine translation' in 'Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting of the Association for Computational Linguistics', (1984) pp. 75–78.

McEnery, T. 'Corpus linguistics' , 2003. In Mitkov, R., (ed.) *The Oxford Handbook of Computational Linguistics*, pp. 448–463.

McEnery, T. and A. Hardie *Corpus Linguistics: Method, Theory and Practice.* Cambridge Textbooks in Linguistics (Cambridge: Cambridge University Press, 2011) [ISBN 9780521547369].

McEnery, T. and A. Wilson *Corpus Linguistics: An Introduction.* Edinburgh Textbooks in Empirical Linguistics (Edinburgh: Edinburgh University Press, 2001) second revised edition [ISBN 9780748611652].

Mikheev, A. 'Text segmentation' , 2003. In Mitkov, R., (ed.) *The Oxford Handbook of Computational Linguistics*, pp. 201–218.

Mitkov, R. (ed.) *The Oxford Handbook of Computational Linguistics.* (Oxford: Oxford University Press, 2003) [ISBN 9780199276349].

Partee, B., A. ter Meulen, and R. Wall *Mathematical Methods in Linguistics.* (Dordrecht: Kluwer, 1990) [ISBN 9789027722454].

Pereira, F. and D. Warren 'Parsing as deduction' in 'Proceedings of the 21st Annual Meeting of the ACL', (1983) pp. 137–144.

Perkins, J. *Python Text Processing With NLTK 2.0 Cookbook*. (Birmingham: Packt Publishing, 2010) [ISBN 9781849513609].

Pinker, S. *The Language Instinct*. (New York: Harper Perennial Modern Classics, 2007) [ISBN 9780061336461].

Pollard, C. and I. Sag *Head-Driven Phrase Structure Grammar*. (Chicago: University of Chicago Press, 1994) [ISBN 9780226674476].

Pullum, G. 'On the mathematics of *Syntactic Structures*' *Journal of Logic, Language and Information* , 2011, pp. 277–296.

Quirk, R., S. Greenbaum, G. Leech, and J. Svartvik *A Comprehensive Grammar of the English Language*. (London: Longman, 1985) second edition [ISBN 9780582517349].

Russell, S. J. and P. Norvig *Artificial Intelligence – A Modern Approach*. (New Jersey: Pearson Education, 2010) third international edition [ISBN 9780132071482].

Samuelsson, C. 'Statistical techniques' , 2003. In Mitkov, R., (ed.), *The Oxford Handbook of Computational Linguistics*, pp. 358–375.

Segaran, T. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. (Sebastopol, CA: O'Reilly Media, 2007) [ISBN 9780596529321].

Turing, A. 'Computing machinery and intelligence' *Mind 49* , 1950, pp. 433–460.

von Humboldt, W. *Über die Verschiedenheit des menschlichen Sprachbaues und ihren Einfluss auf die geistige Entwicklung des Menschengeschlects*. (Berlin: Königlich Preussische Akademie der Wissenschaft, 1836).

Voutilainen, A. 'Part-of-speech tagging' , 2003. In Mitkov, R., (ed.) *The Oxford Handbook of Computational Linguistics*, pp. 219–232.

Woods, W. 'Transition network grammars for natural language analysis' *Communications of the ACM* , 1970, pp. 591–606.

Appendix B

Glossary

accuracy in the context of classification: the percentage of inputs which are correctly labelled by the classifier.

backoff Technique for using multiple **taggers** in sequence.

Bayes' rule Rule for calculating conditional probabilities.

chunk Sequence of POS-tagged items grouped as a syntactic unit.

classifier, classification Automated tool or technique for assigning linguistic items to classes or categories; includes POS tagging and document classification for applications such as topic detection or sentiment analysis.

collocation Pair or small sequence of words which co-occur in a text more frequently than normal.

concordance List of every occurrence of a word or phrase in a text including a certain amount of context.

context-free grammar Formal grammar with rules of the form $X \rightarrow Y$, where X is a non-terminal symbol and Y a sequence of one more terminal or non-terminal symbols, which may include the empty string ϵ.

coreference Phenomenon whereby two or more linguistic items refer to the same entity.

corpus Collection of linguistic data designed to be balanced and representative.

decision tree Machine learning technique for constructing a branching decision procedure.

entropy Measure of disorganisation.

feature set Abstract characterisation of a document used for training and evaluating machine learning procedures.

finite-state machine/automaton A process with a finite number of states and no memory.

frequency distribution Count of the frequencies that items occur in some collection.

grammar List of rules which define a language.

information extraction Analysing text to provide answers to queries in some fixed form.

information retrieval Identifying documents which may be relevant to a user's query.

Markov model; Hidden Markov model (HMM) Model which predicts the next state solely based on the immediately prior state or a small number of prior states; HMM considers not the observed states but hypothesised 'hidden' values such as POS.

morphology The area of linguistics that deals with the internal structure of words and rules of word-formation.

n-grams (bigrams, trigrams) An n-gram is a sequence of n items.

parser Program which determines whether a sequence of words makes up a sentence according to a **grammar**, and if so, analyses its internal structure.

POS Part of speech; syntactic category.

precision and recall Terms for measuring accuracy of classification tasks, originating from the field of **information retrieval**.

probabilistic grammars Grammars which predict preferred readings of ambiguous phrases by assigning probabilities to the application of individual rules.

regular expression Formalism for defining patterns or sequences in terms of the basic operations of sequence, selection and iteration.

regular grammar Context-free grammar which has equivalent generative power to a regular expression or finite-state machine.

ROT Rule of thumb, an imprecise, fallible but useful working rule. From the practice among engineers of using the first thumb-joint to measure an inch.

stem, stemming The stem is the base form of a word when stripped of affixes.

supervised learning Training a machine learning program on text which has been marked up by human experts.

tag, tagger Labelling linguistic items with particular categories.

token, tokenise Breaking up an electronic text into discrete items such as words, punctuation marks and other meaningful symbols.

unbounded dependencies Phenomenon whereby a relative or interrogative construction licenses a 'gap' at some point in a sentence which can be arbitrarily remote from the phrase that 'fills' the gap.

unification Merging information from two feature structures, which must have no conflicting values.

Appendix C

Answers to selected activities

This section includes model solutions to some of the exercises and activities where appropriate. In other cases there is no 'correct' answer and the point of the activity is to stimulate you to engage in independent self-directed learning.

Chapter 2: Introducing NLP: patterns and structure in natural language

Identify parts of speech, page 14

- Jack *(Proper Noun)* and *(conjunction)* Jill *(Proper Noun)* went *(verb)* up *(preposition)* the *(determiner)* hill *(noun)*.
- The *(determiner)* owl *(noun)* and *(conjunction)* the *(determiner)* pussycat *(noun)* went *(verb)* to *(preposition)* sea *(noun)*.

Operation of a finite-state machine, page 17

1. John swam.
2. (a) John and Mary walked on the hill.
   (b) The cat sat on the mat and slept.
   (c) John or a fish walked on a hill and barked.
   (d) ...

Coding regular expressions, page 19

1. a(aa)*(bb)*
2. (aaa)|(aab)|(abb)|(bbb)|(bba)|(baa)|(aba)|(bab)

Regular grammars, page 21

S → either | if S1

S → the | a | one S2

S2 → happy S2

S2 → (boy | girl | dog) eats ('ice cream' | 'hot dogs' | candy) S3

S3 → or | then S2

S3 → ϵ

This is a slightly idealised rendering of Pinker's state diagram which appears to have no halting state.

The problem with this grammar can be clearly seen: the rule *S3 → or | then S2* has no connection with the rule that introduces *either* or *if* and so there is no way to ensure that the appropriate conjunction is used.

Past tense forms, page 25

The general idea is that rules need to be **conditional** in order to handle non-standard cases before applying general regularities: so a reasonable rule based on this data set might be:

welcome → welcomed **else**
-come → -came

Chapter 3: Getting to grips with natural language data

Online corpus queries, page 37

Examples of incorrect tagging: search on `to total/V` gives examples like:

- . . . a ticket **to total** oblivion.
- . . . to describe it **to total** strangers.

as well as 'correct' examples like

- . . . thought **to total** about 1,500 families . . .
- . . . a great opportunity **to total** and celebrate all the small wins made over the year

Using NLTK tools, page 39

1. 'Stylistic' analysis

Science Fiction (948 sentences) N (2232) V (1473) DET (1345) PRO (1268) P (1182) '.' (1078) ADJ (793) ',' (791) CNJ (685) ADV (644) VD (531) NP (467) ...

News (4623 sentences) N (22236) P (10845) DET (10648) NP (8336) V (7346) ADJ (5435) ',' (5188) '.' (4472) CNJ (4227) PRO (3408) ADV (2770) VD (2531) ...

Religion (1716 sentences) N (7304) P (4364) DET (4239) V (3863) ADJ (2620) CNJ (2324) PRO (2243) ',' (1913) '.' (1892) ADV (1485) NP (1224) VN (952) ...

Some counts:

SF: N 2232, PRO 1268, NP 467 ADJ 793

NE: N 22236, NP 8336, PRO 3408 ADJ 5435

RE: N 7304, PRO 2243 NP 1224 ADJ 2620

SF: S 948, COMMA 791, CNJ 685

NE: COM 5188 S 4623 CNJ 4227

RE: CNJ 2324 COM 1913 S 1716

Some tentative conclusions:

1. Reference and description: both SF and RE use pronouns more than proper names; News has more proper names. Hard to interpret without further analysis: if an SF work includes a lot of dialogue for example, it might be more natural for characters to refer to each other as I, we, you and so on rather than by name. And news stories tend to be about named individuals.
2. Syntactic complexity: we cannot be very precise with this data but it looks as if the SF genre has the least syntactic complexity and the RE genre the highest, judging from the numbers of commas and conjunctions per sentence. Of course we cannot tell whether these tokens are connecting clauses or other types of phrases.

In an examination or coursework question, you would get credit for discussing these and other characteristics in the light of your impressionistic understanding of the typical styles for these genres.

2. Reuters

- Display the README file from the Reuters corpus.

```
from nltk.corpus import reuters
desc = reuters.readme()
print desc
```

- Create a variable `soysents` containing all sentences from reports concerning soy products.

  ```
  reuters.categories()
  ```

  Pick out categories relating to soy:

  ```
  soysents = reuters.sents(categories=['soy-meal', ...])
  ```

- Display the first ten sentences in `soysents`.

  ```
  print soysents[:10]
  ```

- Create a variable `metalwords` containing all words from reports concerning metals.

  ```
  metalwords = reuters.words(categories = ['alum','copper','gold', ...])
  ```

  (Note that inspection of texts in the *alum* category confirms that they are about aluminium.)

- What are the most frequently mentioned metals in this collection? Caution: why might this result be less than 100 per cent reliable?

  ```
  from nltk.book import *
  freqmetal = FreqDist(metalwords)
  freqmetalkeys = freqmetal.keys()
  freqmetal[:100]
  ```

  Remember that the contents of a frequency distribution are listed in the order of their frequency of occurrence. By scanning the output you should see that the first three metals listed are gold, copper and steel. However caution is in order as Reuters is an overlapping corpus, so we may be double-counting some occurrences. These metals may also be mentioned under the category `strategic-metal`, or some reports may mention more than one kind of metal and so come under multiple categories.

Chapter 4: Computational tools for text analysis

Comparing stemmers, page 48

Lancaster rules
- Remove *ist/s/e/ing/en/th/ity/ate/al/a/ed/ment/ation*.
- Replace *-ies* with *-y*.
- Some motivations: reduce verbs to stem form, remove plural affix and return stem in irregular cases (*study*), remove ordinal affix *-th*, remove affixes that form nominalisations or adjectives: *-ment*, *-al*.

Porter rules
- Remove *s/ing/ity/e/ate/ed/ment/ational*.
- Replace *-y* with *-i*.
- Motivations: similar to Lancaster rules but applied more sparingly.

Errors Lancaster removes *-th* from *south* as if it were an ordinal and *-e* from *are* though *ar* is not a stem here; not clear why *-a* removed from *area*. Lancaster and Porter both treat some proper names as common nouns; for example, by removing the last letter from *Lyons* and *Stanhope* (both) or by postulating an *-i* stem for names ending in *-y* (Porter).

Tagging with REs, page 51

Patterns for RE tagger including punctuation. Note use of the backslash escape character for special characters ', ., ? .

```
[('^he|him|she|her|it|they|them$', 'PRO'),  # pronouns
('^who|which|what|where|how$', 'WH'),# WH-items
 ('^and|but|or|if|while|although|when$', 'CNJ'),# conjunctions
 ('^a|the|some|many|most$', 'DET'),  # determiners
('^[A-Z].*', 'NP'),# proper nouns
 ('.*ing$', 'VG'),
('.*ed$', 'VD'),
('.*es$', 'V'),
('.*ould$', 'MOD'),
(".*\'s$", 'N'),
('.*s$', 'N'),
('^-?[0-9]+(.[0-9]+)?$', 'NUM'),
 ('^"$', '"'),# double quotes
 ('^``$', '``'),  # more double quotes
('^,$', ','),  # comma
('^;$', ';'),  # semi-colon
('^:$', ':'),  # colon
('^!$', '.'),  # exclamation mark
('^\?$', '.'),  # question mark
('^\.$', '.'),  # full stop
('.*', 'N')]
```

Chapter 5: Statistically-based techniques for text analysis

Activity: Bayes' Rule, page 59

There are various ways of tackling this problem. Here is one of them.

1. Normalise the untagged and tagged word tokens to lower case:

   ```
   >>> news_words = [w.lower() for w in brown.words(categories = 'news')]
   >>> len(news_words)
   100554
   >>> tagged_news_words = [(word.lower(),tag) for (word, tag)
   in brown.tagged_words(categories = 'news',simplify_tags = True)]
   >>> len(tagged_news_words)
   100554
   ```

2. Create a frequency distribution over the words and a conditional frequency distribution for the word-tag pairs:

   ```
   >>> fd_words = nltk.FreqDist([w for w in news_words])
   >>> cfd_tag_word = nltk.ConditionalFreqDist((tag,word) for (word,tag)
   in tagged_news_words)
   ```

   Note the order of arguments `(tag,word)`: we are interested in calculating the frequency of word tokens for each tag type and not vice versa.

3. Find the frequency of `run`

```
>>> fd_words['run']
35
```

4. Find the frequency of *V* tags and the number of those with the word token *run*:

```
>>> cfd_tag_word['V']
<FreqDist with 7346 outcomes>
>>> cfd_tag_word['V']['run']
11
```

5. At this point we have enough values to apply Bayes' Rule to determine $P(b \mid a)$, the probability that a token of *run* will be tagged *V*. We can check this by computing $P(b \mid a)$ directly, using a conditional distribution the other way round:

```
>>> cfd_word_tag = nltk.ConditionalFreqDist((word,tag) for (word,tag)
in tagged_news_words)
>>> cfd_word_tag['run']
<FreqDist with 35 outcomes>
>>> cfd_word_tag['run']['V']
11
>>> 11/35
0.31428571428571428
>>>
```

Chapter 6: Analysing sentences: syntax and parsing

Activity: Verb categories, page 78

Examples:

i Oscar died in Paris.

ix *Oscar died the table.

Grammar rules

S	→	NP VP[SUBCAT=y]
VP[SUBCAT=x]	→	VP[SUBCAT=x] PP
VP[SUBCAT=intrans]	→	V[SUBCAT=intrans]
VP[SUBCAT=trans]	→	V[SUBCAT=trans] NP
PP	→	P NP
V[SUBCAT=intrans]	→	died
[...]		

In tree C.1 the SUBCAT values match on the VP and V nodes so everything is fine. In tree C.2 however there is a problem: the lexical rule for died gives the value `intrans` but the phrase structure rule for the VP mandates `trans`.

Activity: Feature-based grammar, page 80

Combining number and verb categories

S → NP[Num=n?] VP[Num=n?]

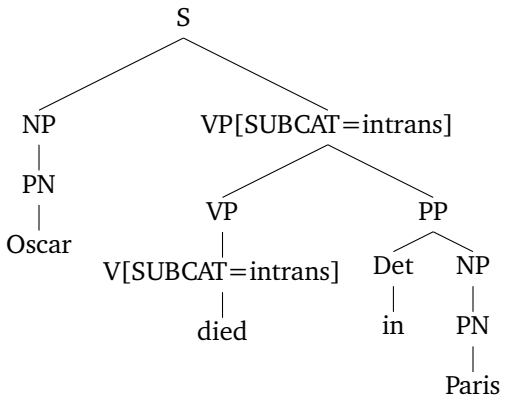

Figure C.1: Tree for Q1b(i).

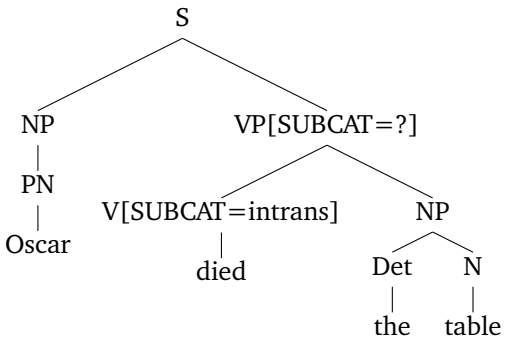

Figure C.2: Tree for Q1b(ix).

NP[Num=n?] → Det[Num=n?] N[Num=n?]

VP[Num=n?,SUBCAT=intrans] → V[Num=n?,SUBCAT=intrans]

VP[Num=n?,SUBCAT=trans] → V[Num=n?,SUBCAT=trans] NP

VP[Num=n?,SUBCAT=dative] → V[Num=n?,SUBCAT=dative] NP PP

PP → P NP

Det[Num=sg] → a | the

Det[Num=pl] → the | several

V[Num=?n,SUBCAT=trans] → pours

V[Num=?n,SUBCAT=dative] → puts

N[Num=sg] → waiter | wine | bottle | table

N[Num=pl] → waiters | bottles

P → on

Introducing person agreement

Some sample rules. Note that we are only concerned with subject NPs: no grammar rules in English are concerned with the person number of non-subject arguments.

S → NP[Num=x PERS=y] VP[Num=x PERS=y]

NP[Num=x PERS=y] → Pro[Num=x PERS=y]

Pro[Num=sg PERS=1] → I

Pro[Num=x PERS=2] → you

NP[Num=sg PERS=3] → PN[Num=sg PERS=3]

PN[Num=sg PERS=3] → John

Appendix D

Trace of recursive descent parse

```
>>> from __future__ import division
>>> import nltk
>>> s = "John saw a man in the park".split()
>>> s
['John', 'saw', 'a', 'man', 'in', 'the', 'park']
>>> grammar1 = nltk.parse_cfg("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw"
  NP -> "John" |  Det N
  Det -> "a" | "the"
  N -> "man" | "park"
  P -> "in"
  """)

>>> grammar1
<Grammar with 12 productions>
>>> rdp=nltk.RecursiveDescentParser(grammar1,trace=2)
>>> for tree in rdp.nbest_parse(s):
print tree

Parsing 'John saw a man in the park'
    [ * S ]
  E [ * NP VP ]
  E [ * 'John' VP ]
  M [ 'John' * VP ]
  E [ 'John' * V NP ]
  E [ 'John' * 'saw' NP ]
  M [ 'John' 'saw' * NP ]
  E [ 'John' 'saw' * 'John' ]
  E [ 'John' 'saw' * Det N ]
  E [ 'John' 'saw' * 'a' N ]
  M [ 'John' 'saw' 'a' * N ]
  E [ 'John' 'saw' 'a' * 'man' ]
  M [ 'John' 'saw' 'a' 'man' ]
  E [ 'John' 'saw' 'a' * 'park' ]
  E [ 'John' 'saw' * 'the' N ]
  E [ 'John' * V NP PP ]
  E [ 'John' * 'saw' NP PP ]
  M [ 'John' 'saw' * NP PP ]
  E [ 'John' 'saw' * 'John' PP ]
  E [ 'John' 'saw' * Det N PP ]
  E [ 'John' 'saw' * 'a' N PP ]
  M [ 'John' 'saw' 'a' * N PP ]
```

```
  E [ 'John' 'saw' 'a' * 'man' PP ]
  M [ 'John' 'saw' 'a' 'man' * PP ]
  E [ 'John' 'saw' 'a' 'man' * P NP ]
  E [ 'John' 'saw' 'a' 'man' * 'in' NP ]
  M [ 'John' 'saw' 'a' 'man' 'in' * NP ]
  E [ 'John' 'saw' 'a' 'man' 'in' * 'John' ]
  E [ 'John' 'saw' 'a' 'man' 'in' * Det N ]
  E [ 'John' 'saw' 'a' 'man' 'in' * 'a' N ]
  E [ 'John' 'saw' 'a' 'man' 'in' * 'the' N ]
  M [ 'John' 'saw' 'a' 'man' 'in' 'the' * N ]
  E [ 'John' 'saw' 'a' 'man' 'in' 'the' * 'man' ]
  E [ 'John' 'saw' 'a' 'man' 'in' 'the' * 'park' ]
  M [ 'John' 'saw' 'a' 'man' 'in' 'the' 'park' ]
  + [ 'John' 'saw' 'a' 'man' 'in' 'the' 'park' ]
  E [ 'John' 'saw' 'a' * 'park' PP ]
  E [ 'John' 'saw' * 'the' N PP ]
  E [ * Det N VP ]
  E [ * 'a' N VP ]
  E [ * 'the' N VP ]
(S
  (NP John)
  (VP
    (V saw)
    (NP (Det a) (N man))
    (PP (P in) (NP (Det the) (N park)))))
```

Appendix E

Sample examination paper with answering guidelines

This section will give you an idea of the types of questions you may encounter in the examination, in terms of subject matter and level of difficulty. Some guidelines are given on how to tackle the questions.

The examination duration will be 2 hours 15 minutes.

There will be FIVE questions in the paper. You should answer THREE of them. Each question is marked out of 25. The marks for each part of a question are indicated at the end of the part in [.] brackets.

A hand held calculator may be used when answering questions on this paper but it must not be pre-programmed or able to display graphics text or algebraic equations. The make and type of machine must be stated clearly on the front cover of the answer book.

E.1 Sample examination questions

1. Tables 1 and 2 below show a selection of past tense forms in English.

Table 1		**Table 2**	
Present	**Past**	**Present**	**Past**
bear	bore	bind	bound
bend	bent	clear	cleared
creep	crept	fear	feared
keep	kept	hear	heard
lay	laid	mend	mended
lend	lent	play	played
pay	paid	seep	seeped
pity	pitied	tend	tended
rely	relied	wind	wound
reply	replied		
say	said		
send	sent		
sleep	slept		
study	studied		
swear	swore		
tear	tore		
wear	wore		

   (a) Give a list of rules for deriving past tense forms based on the cases in Table 1 only. You should justify your answer with reference to specific examples. [6]

   (b) Revise your list of rules so that it covers all cases in both Table 1 and Table 2. [9]

   (c) i. Explain why naive Bayes classifiers are called ‘naive’. Your answer should not assume prior knowledge of Bayes’ Rule.

   ii. Why might a naive Bayes classifier be particularly useful for spam filtering?

   [10]

2. Consider the following grammar rules and answer questions (a–c) following.

S → NP VP

NP → (Det) N

NP → Det N PP

VP → V NP

VP → Cop Adj

PP → P NP

Det → a | the | my | twelve | some | most

N → friend | coat | sales | buttons | shoe | shoes | car | cars | student | students

V → bought | likes | like | has | have

Cop → is | are | was | were

P → in | with | of

Adj → happy | angry | sad

(a) Explain what is meant by the following, with some simple examples:

i. Attachment ambiguity
ii. Left-recursion
iii. Centre-embedding
iv. Recursive descent parsing

[10]

(b) Using the above grammar rules, draw syntax trees for:

i. My friend is angry.
ii. My friend bought a coat with twelve buttons.
iii. Some students have cars.

[5]

(c) Modify the grammar using feature structures so that it generates the unstarred sentences below as well as (2b)(i–iii) above but not the starred ones. Explain the reasons for your modifications. [10]

i. My friend is a student.
ii. My friends are students.
iii. Most coats have buttons.
iv. * Most coats have button.
v. * The student are my friend.
vi. * Twelve student bought a car.
vii. * The students bought some shoe.

3. The following paragraph is taken from a government report, 'How Fair is Britain?'

   Today, we live in a society where overt displays of prejudice are usually unlawful, and almost always socially unacceptable. Surveys suggest that we are more tolerant of difference, and less tolerant of discrimination. This is mirrored in the evolution of new laws which prohibit discrimination and require public bodies to promote equality. It is borne out by our expectations of public figures: a career in the public eye can be cut short by a bigoted comment.

   This is the result of running the above text through the Lancaster Stemmer:

```
['today', ',', 'we', 'liv', 'in', 'a', 'socy', 'wher', 'overt', 'display', 'of',
'prejud', 'ar', 'us', 'unlaw', ',', 'and', 'almost', 'alway', 'soc', 'unacceiv',
'.', 'survey', 'suggest', 'that', 'we', 'ar', 'mor', 'tol', 'of', 'diff', ',',
'and', 'less', 'tol', 'of', 'discrimin', '.', 'thi', 'is', 'mir', 'in', 'the',
'evolv', 'of', 'new', 'law', 'which', 'prohibit', 'discrimin', 'and', 'requir',
'publ', 'body', 'to', 'promot', 'eq', '.', 'it', 'is', 'born', 'out', 'by', 'our',
'expect', 'of', 'publ', 'fig', ':', 'a', 'car', 'in', 'the', 'publ', 'ey', 'can',
'be', 'cut', 'short', 'by', 'a', 'bigot', 'com', '.',]
```

   (a) Explain what is meant by word stems, with references to examples from the above text. How can stemming be useful in applications such as machine translation? [4]

   (b) Make a list of rules which the stemmer has applied in this example and discuss the motivations for the rules, including any cases where you believe rules have been applied inappropriately. [12]

   (c) i. Explain the difference between **supervised** and **unsupervised** learning.

   ii. Outline some steps you might go through in the process of training a classifier to tell the difference between male and female names. [9]

4. (a) Explain the difference between information retrieval and information extraction. [4]

   (b) Describe and give examples of the classes of strings matched by the following regular expressions: [6]

      i. [A-Z0-9]+
      ii. [A-Z][0-9]*
      iii. ([^aeiou][aeiou])*

   (c) The `findall()` method in the Natural Language Toolkit (NLTK) allows you to search for sequences of tokens in a text and choose what part of the result to display, enabling you to find instances of strings occurring in particular contexts. For example, assuming `news` is a tokenised text:

```
>>> news.findall(r"<President><K.*>")
President Kennedy; President Kasavubu; President Kennedy's; President Krushchev
>>> news.findall(r"<President>(<K.*>)")
Kennedy; Kasavubu; Kennedy's; Krushchev
```

   Construct search patterns to find the following types of phrase. You should think about what kinds of context these expressions can occur in. You may give more than one pattern for each answer. Explain your answers.

      i. Personal names of the form firstname, lastname: Daniel Radcliffe, Nitin Sawhney, Barack Obama, and so on.
      ii. Names of streets: Downing Street, St James, Pennsylvania Avenue
      iii. Place names like Paris, Gaza, Germany, Tibet, New York.

   [10]

   (d) Explain what is meant by **precision** and **recall** in the context of information retrieval and extraction. What kinds of **false negatives** and **false positives** might result from the queries you specified in your answers to (4c) above? [5]

5. (a) The following sentences are all ambiguous in some way. Express their different meanings using paraphrases and explain the source of the ambiguity in each case. [5]

   i. The builders dumped the rubble in the truck.

   ii. Every man loves a woman.

   iii. Flying planes can be dangerous.

   iv. Either Kim will stay or Dana will leave and everyone will be happy.

   v. Whenever John meets Bill he buys him a drink.

   (b) Explain how probabilistic grammars can deal with ambiguity, with reference to one of the examples above. [6]

   (c) The corpora that are distributed with the NLTK can be accessed in various ways, including:

   i. Raw text

   ii. Tokenised

   iii. POS-tagged

   iv. Parsed.

   Explain the distinctions between these terms and give an example of a corpus that includes parsed text. [5]

   (d) The following is part of the output of a bigram tagger, tested on an excerpt from the Brown corpus news genre:

```
[('Then', 'RB'), ('Dick', 'NP'), ('Hyde', 'NP'), (',', ','),
('submarine-ball', 'NN'), ('hurler', 'NN'), (',', ','),
('entered', 'VBD'), ('the', 'AT'), ('contest', 'NN'),
('and', 'CC'), ('only', 'AP'), ('five', 'CD'), ('batters', 'NNS'),
('needed', 'VBD'), ('to', 'TO'), ('face', 'VB'), ('him', 'PPO'),
('before', 'IN'), ('there', 'RB'), ('existed', None), ('a', None),
('3-to-3', None), ('deadlock', None), ('.', None)]
```

   i. What is a probable reason that the last five tokens have been assigned the tag None?

   ii. Explain how more accurate tagging can be achieved by using a procedure that includes multiple taggers. [9]

E.2 Answering guidelines for sample examination questions

ANSWERING QUESTION 1

a) Table 1 rules – need examples for full marks

-ear → -ore

-d → -t

-eep → -ept

-ay → -aid

-y → -ied

b) Tables 1 and 2 rules:

Regular past form: word+ed if word ends in 'e' else word+ed

Regular verbs: clear, mend, play, seep

Irregular forms:

-ear → -ore except hear ? heard

-end → -ent

-ind → -ound

-eep → -ept

-ay → -aid

-y → -ied

(last three as Table 1)

c)

i. Explain why naive Bayes classifiers are called 'naive'. Your answer should not assume prior knowledge of Bayes' Rule.

   Answer should show understanding of Bayes rule, explain that naive Bayes classifiers assume independence of features which is unrealistic in practice, but can still be reliable in predicting which alternative is more likely. Credit will be given for well-chosen examples.

ii. Why might a naive Bayes classifier be particularly useful for spam filtering?

   Machine learning in general is useful for spam filtering as spammers will keep changing the characteristics of their messages which the designers of filters may have identified. Naive Bayes is particularly useful as it can be trained incrementally; for instance, whenever a user reports an email as spam which has got through the filter.

[10]

ANSWERING QUESTION 2

a) i. Structural ambiguity where a node could be attached to more than one higher node; for example, the PP in *John saw a girl with a telescope*

ii. Rules of the form X $\rightarrow$ X ... such as NP $\rightarrow$ NP PP

iii. Constructions where a phrase is embedded within a phrase of the same type with material either side of it, such as *The horse raced past the barn fell*, *the cat the dog bit died*.

iv. Top-down parsing technique which recursively breaks down a high-level goal "find S" into lower level subgoals as specified by the grammar rules, terminating when all words in the input string can be matched to the RHS of a lexical rule.

b) [2/6] per example, such as:

```
(S
(NP (Det my) (N friend))
(VP (Cop is) (Adj angry)
)
```

c) Modified grammar using unification of feature values to percolate number agreement: some explanation required for full marks.

S $\rightarrow$ NP[Num=x] VP[Num=x]

NP $\rightarrow$ Det[Num=x] N[Num=x]

NP[Num=x] $\rightarrow$ Det[Num=x] N[Num=x] PP

VP[Num=x] $\rightarrow$ V[Num=x] NP

VP[Num=x] $\rightarrow$ Cop[Num=x] Adj

VP[Num=x] $\rightarrow$ Cop[Num=x] N[Num=x]

PP $\rightarrow$ P NP

Det[Num=sg] $\rightarrow$ a | the | my

Det[Num=pl] $\rightarrow$ ϵ | the | my | twelve | most | some

N[Num=sg] $\rightarrow$ friend | coat | sale | button | shoe | car | student

N[Num=pl] $\rightarrow$ friends | coats | sales | buttons | shoes | cars | students

V[Num=_] $\rightarrow$ bought

V[Num=sg] $\rightarrow$ likes | has

V[Num=pl] $\rightarrow$ like | have

Cop[Num=sg] $\rightarrow$ is | was

Cop[Num=pl] $\rightarrow$ are | were

P $\rightarrow$ in | with | of

Adj $\rightarrow$ happy | angry | sad

ANSWERING QUESTION 3

a) Stem: 'basic form' of a word with suffixes like verb endings, plural markers, etc. removed. Depending on application, a stem may or may not be a 'dictionary word'. In Machine Translation (MT), having a single lexical entry for the stem form can avoid duplication of information. [2/4]

b) Rules:

i. Remove endings:
+e except the, we, be
+s except is, less
+ice
+ually
+ful
+ially
+erant
+erence
+ation
+rored
+ic
+uality
+ures
+eer
+ed
+ment
Replace endings:
-iety → -y
-ceptable → -ceive
-ution → -v

ii. Motivations may include:
+e, +ed treated as verb endings
+s as a verb ending or plural marker
+(u|i)ally adverbial forms derived from adjectives
-ceptable → -ceive prob designed for forms like receive/reception, not really applicable here

c) i. Explain the difference between **supervised** and **unsupervised** learning.
Supervised learning involves data which is marked up with the correct label for each input. Unsupervised learning has no information of this kind.

ii. Outline some steps you might go through in the process of training a classifier to tell the difference between male and female names.
Main steps:

1. Decide what kind of classifier to use (naive Bayes, decision tree . . .)
2. Find a suitable corpus with names identified as male or female.
3. Decide what features of names are relevant and create a feature extractor function.
4. Randomly shuffle the input.
5. Use the feature extractor to extract a feature set, which is divided into training and test sets.
6. Train the classifier on the training set and test it on the test set.
7. Depending on the results, decide whether to refine the features and repeat the process.

Extra credit for explaining the rationale for division into training and test sets, and for including dev-test sets and error analysis.

ANSWERING QUESTION 4

a) Information Retrieval (IR) – finding documents that may contain information relevant to a query (typically expressed as a string of keywords), such as search engines.

Information Extraction (IE) – identifying information in unstructured text and presenting it in a structured form, such as names of people/corporations/countries, specific types of event and so on.

b) For full marks there must be both descriptions and examples.

i. One or more upper case letters and/or digits: 'A', '1B', 'T1000'

ii. Empty string, or one or more pairs <UC letter, digit>: 'A1', 'B4U2'

iii. Empty string, or sequence of consonant/vowel pairs: 'la', 'panama', 'topolino'.

c) For full marks, must specify well-motivated contexts but exclude it from answer with proper use of parentheses.

i. Full names are often preceded by titles or followed by age, so possible solutions are:

```
<Mr|Mrs|Miss|Dr>(<[A-Z][a-z]+><[A-Z] [a-z]+>)

(<[A-Z]  [a-z]+><[A-Z]  [a-z]+>)<,><[0-9]+>
```

ii. `<[0-9]+>(<[A-Z]  [a-z]+><[A-Z]  [a-z]+>`

iii. Place names occur in phrases like native of X, come from X, in X (X capitalised in all cases) so one example could be:

`<in>(<[A-Z][a-z]*>)`

In all cases, credit will be given for explaining the answer as required and using well-chosen examples.

d) Precision: percentage of answers returned that were correct. Recall: percentage of potential correct answers that were actually identified by the system.

False positives/negatives will obviously depend on students' answers to (c): some possibilities for the above answers include:

– False positives: identifying for example *Justice Holmes* as a personal name; *Christmas Day 1923* would match the second full name pattern.

– False negatives: names occurring without titles or age, place names occurring in different contexts.

ANSWERING QUESTION 5

a) i. *The builders dumped the rubble in the truck* – PP-attachment ambiguity: was the rubble in the truck to start with or not?

ii. *Every man loves a woman* – logical ambiguity: same woman or different woman for each man?

iii. *Flying planes can be dangerous* - syntactic ambiguity: the act of flying planes, or the planes themselves can be dangerous.

iv. *Either Kim will stay or Dana will leave and everyone will be happy* – scope of coordination: will everyone be happy anyway, or only if Dana leaves?

v. *Whenever John meets Bill he buys him a drink* – pronoun resolution: who buys the drink?

b) Probabilistic grammars encode relative probabilities of different possible expansions of non-terminal symbols, enabling most probable analysis of a sentence to be calculated – for example in (ai) the reading (V NP PP) may be more likely than (V (Det N PP))

c)
   i. Raw text – sequence of characters with no differentiation between letters, punctuation, spaces, and so on.
   ii. Tokenised – list of 'tokens' comprising separate words, numbers, punctuation and other symbols, white space normally removed.
   iii. POS-tagged – tokens annotated with grammatical parts-of-speech.
   iv. Parsed – sentences annotated with labels and brackets showing syntactic structure: an example is the Penn Treebank.

d) Tagging:
   i. Answer needs to explain concepts of bigram tagger and training. Probable answer is sparse data: sequence <RB 'existed'> not in training data so can't resolve context in test data or for subsequent tokens.
   ii. A solution is to combine taggers: if the bigram tagger breaks, then backoff to a unigram tagger and then a default tagger. Should briefly explain these for full marks.

Comment form

We welcome any comments you may have on the materials which are sent to you as part of your study pack. Such feedback from students helps us in our effort to improve the materials produced for the University of London.
If you have any comments about this guide, either general or specific (including corrections, non-availability of Essential readings, etc.), please take the time to complete and return this form.

Title of this subject guide: .....

.....

Name .....

Address .....

.....

Email .....

Student number .....

For which qualification are you studying? .....

Comments

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Please continue on additional sheets if necessary.

Date: .....

Please send your completed form (or a photocopy of it) to:
Publishing Manager, Publications Office, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.