Mastering spaCy

An end-to-end practical guide to implementing NLP applications using the Python ecosystem

Duygu Altınok

Packt
BIRMINGHAM—Mumbai
To my mother, Ülker, for her life-long support and endless love. To my sister, for her support and inspiration. To my besties, Umutcan, Simge, and Aydan, for their friendship and support.
About the author

Duygu Altınok is a senior Natural Language Processing (NLP) engineer with 12 years of experience in almost all areas of NLP, including search engine technology, speech recognition, text analytics, and conversational AI. She has published several publications in the NLP area at conferences such as LREC and CLNLP. She also enjoys working on open source projects and is a contributor to the spaCy library. Duygu earned her undergraduate degree in computer engineering from METU, Ankara, in 2010 and later earned her master’s degree in mathematics from Bilkent University, Ankara, in 2012. She is currently a senior engineer at German Autolabs with a focus on conversational AI for voice assistants. Originally from Istanbul, Duygu currently resides in Berlin, Germany, with her cute dog Adele.
About the reviewers

Kevin Lu is currently a student studying software engineering at the University of Waterloo, with experience in full-stack web development, machine learning, computer vision, and natural language processing, and is the founder of the Python package PyATE (Python Automated Term Extraction). His interests include discrete mathematics, data science, algorithmic optimization, and deep learning. In the future, he is interested in pursuing research in NLP with deep learning and applications of it in accelerating academic research.

Usama Yaseen is currently a PhD candidate at Siemens AG (Munich) and the University of Munich. His research interests lie in data-efficient information extraction. Before starting his PhD, he was the lead data scientist at SAP SE, where he led a machine learning team focused on information extraction from semi-structured documents. He holds a master's from the Technical University of Munich in informatics; his master's thesis explored recurrent neural networks with external memory for question-answering systems. Overall, he has worked at Siemens (AG) (on corporate technology research), SAP SE (on machine learning), and Intel Corporation (on software development).

Souvik Roy is an NLP researcher. He primarily works on recurrent neural networks and transformer model compression methodologies such as pruning, quantization, tensor decomposition, and knowledge distillation to reduce the challenges faced by larger models, including longer training and inference times. He is passionate about working with textual data to solve underlying problems. Souvik has a master's in engineering from the University of Waterloo, specializing in text processing. Additionally, he has worked with Scribendi on document summarization and grammatical error correction. Since then, he has been working in diverse industrial research labs.

Carlos Fernando Schiaffin is passionate about analyzing and describing the underlying phenomena of human language. He is an NLP developer currently focused on conversational AI. He has a degree in linguistics and is a self-taught Python programmer. For more than five years, he has been working on NLP systems to try to understand and explain some of the speakers' linguistic behaviors. He started his career as a data tagger and soon went on to design annotation processes for linguistic data in Spanish, English, and Portuguese. Currently, he works with Rasa, spaCy and others, on the development of a conversational AI in Spanish. I thank Duygu Altinok for giving me the chance to participate in this book and my colleagues who always accompany my learning process.
## Table of Contents

### Preface

### Section 1: Getting Started with spaCy

#### 1 Getting Started with spaCy

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical requirements</td>
<td>4</td>
</tr>
<tr>
<td>Overview of spaCy</td>
<td>4</td>
</tr>
<tr>
<td>Rise of NLP</td>
<td>4</td>
</tr>
<tr>
<td>NLP with Python</td>
<td>7</td>
</tr>
<tr>
<td>Reviewing some useful string operations</td>
<td>8</td>
</tr>
<tr>
<td>Getting a high-level overview of the spaCy library</td>
<td>12</td>
</tr>
<tr>
<td>Tips for the reader</td>
<td>15</td>
</tr>
<tr>
<td>Installing spaCy</td>
<td>15</td>
</tr>
<tr>
<td>Installing spaCy with pip</td>
<td>15</td>
</tr>
<tr>
<td>Installing spaCy with conda</td>
<td>16</td>
</tr>
<tr>
<td>Installing spaCy on macOS/OS X</td>
<td>16</td>
</tr>
<tr>
<td>Installing spaCy on Windows</td>
<td>17</td>
</tr>
<tr>
<td>Troubleshooting while installing spaCy</td>
<td>17</td>
</tr>
<tr>
<td>Installing spaCy's statistical models</td>
<td>18</td>
</tr>
<tr>
<td>Installing language models</td>
<td>21</td>
</tr>
<tr>
<td>Visualization with displaCy</td>
<td>23</td>
</tr>
<tr>
<td>Getting started with displaCy</td>
<td>24</td>
</tr>
<tr>
<td>Entity visualizer</td>
<td>26</td>
</tr>
<tr>
<td>Visualizing within Python</td>
<td>26</td>
</tr>
<tr>
<td>Using displaCy in Jupyter notebooks</td>
<td>29</td>
</tr>
<tr>
<td>Exporting displaCy graphics as an image file</td>
<td>31</td>
</tr>
<tr>
<td>Summary</td>
<td>31</td>
</tr>
</tbody>
</table>

### Summary

#### 2 Core Operations with spaCy

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical requirements</td>
<td>34</td>
</tr>
<tr>
<td>Overview of spaCy conventions</td>
<td>34</td>
</tr>
<tr>
<td>Introducing tokenization</td>
<td>39</td>
</tr>
<tr>
<td>Customizing the tokenizer</td>
<td>42</td>
</tr>
<tr>
<td>Debugging the tokenizer</td>
<td>43</td>
</tr>
<tr>
<td>Sentence segmentation</td>
<td>44</td>
</tr>
<tr>
<td>Understanding lemmatization</td>
<td>45</td>
</tr>
<tr>
<td>Lemmatization in NLU</td>
<td>46</td>
</tr>
<tr>
<td>Understanding the difference between lemmatization and stemming</td>
<td>47</td>
</tr>
</tbody>
</table>
Section 2: spaCy Features

3

Linguistic Features

Technical requirements 68
What is POS tagging? 68
WSD 76
Verb tense and aspect in NLU applications 76
Understanding number, symbol, and punctuation tags 78
Introduction to dependency parsing 79

4

Rule-Based Matching

Token-based matching 108
Extended syntax support 115
Regex-like operators 116
Regex support 121
Matcher online demo 122
PhraseMatcher 123
EntityRuler 126
Combining spaCy models and

5

Working with Word Vectors and Semantic Similarity

Technical requirements 140
Understanding word vectors 140
## 6 Putting Everything Together: Semantic Parsing with spaCy

### Technical requirements
- Extracting named entities 168
- Getting to know the ATIS dataset 168
- Extracting named entities with Matcher 172
- Using dependency trees for extracting entities 179

### Using dependency relations for intent recognition
- Linguistic primer 183
- Extracting transitive verbs and their direct objects 185

### Extracting multiple intents with conjunction relation 186
- Recognizing the intent using wordlists 188

### Semantic similarity methods for semantic parsing 190
- Using synonyms lists for semantic similarity 190
- Using word vectors to recognize semantic similarity 192

### Putting it all together 194
- Summary 196

## Section 3: Machine Learning with spaCy

### 7 Customizing spaCy Models

<table>
<thead>
<tr>
<th>Technical requirements</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting started with data preparation</td>
<td>200</td>
</tr>
<tr>
<td>Do spaCy models perform well enough on your data?</td>
<td>201</td>
</tr>
<tr>
<td>Does your domain include many labels that are absent in spaCy models?</td>
<td>202</td>
</tr>
</tbody>
</table>

### Annotating and preparing data
- Annotating data with Prodigy 204
- Annotating data with Brat 205
- spaCy training data format 206
### 8

**Text Classification with spaCy**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical requirements</td>
<td>226</td>
</tr>
<tr>
<td>Understanding the basics of text classification</td>
<td>226</td>
</tr>
<tr>
<td>Training the spaCy text classifier</td>
<td>229</td>
</tr>
<tr>
<td>Getting to know TextCategorizer class</td>
<td>230</td>
</tr>
<tr>
<td>Formatting training data for the TextCategorizer</td>
<td>231</td>
</tr>
<tr>
<td>Defining the training loop</td>
<td>233</td>
</tr>
<tr>
<td>Testing the new component</td>
<td>234</td>
</tr>
<tr>
<td>Training TextCategorizer for multilabel classification</td>
<td>234</td>
</tr>
<tr>
<td>Sentiment analysis with spaCy</td>
<td>237</td>
</tr>
</tbody>
</table>

### 9

**spaCy and Transformers**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical requirements</td>
<td>260</td>
</tr>
<tr>
<td>Transformers and transfer learning</td>
<td>260</td>
</tr>
<tr>
<td>Understanding BERT</td>
<td>265</td>
</tr>
<tr>
<td>BERT architecture</td>
<td>266</td>
</tr>
<tr>
<td>BERT input format</td>
<td>268</td>
</tr>
<tr>
<td>How is BERT trained?</td>
<td>270</td>
</tr>
<tr>
<td>Transformers and TensorFlow</td>
<td>271</td>
</tr>
<tr>
<td>HuggingFace Transformers</td>
<td>272</td>
</tr>
<tr>
<td>Using the BERT tokenizer</td>
<td>273</td>
</tr>
<tr>
<td>Obtaining BERT word vectors</td>
<td>276</td>
</tr>
<tr>
<td>Using BERT for text classification</td>
<td>278</td>
</tr>
<tr>
<td>Using Transformer pipelines</td>
<td>280</td>
</tr>
<tr>
<td>Transformers and spaCy</td>
<td>282</td>
</tr>
<tr>
<td>Summary</td>
<td>286</td>
</tr>
</tbody>
</table>
# 10

## Putting Everything Together: Designing Your Chatbot with spaCy

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical requirements</td>
<td>290</td>
</tr>
<tr>
<td>Introduction to conversational AI</td>
<td>290</td>
</tr>
<tr>
<td>NLP components of conversational AI products</td>
<td>292</td>
</tr>
<tr>
<td>Getting to know the dataset</td>
<td>294</td>
</tr>
<tr>
<td><strong>Entity extraction</strong></td>
<td>298</td>
</tr>
<tr>
<td>Extracting city entities</td>
<td>298</td>
</tr>
<tr>
<td>Extracting date and time entities</td>
<td>300</td>
</tr>
<tr>
<td>Extracting phone numbers</td>
<td>303</td>
</tr>
<tr>
<td>Extracting cuisine types</td>
<td>304</td>
</tr>
<tr>
<td>Intent recognition</td>
<td>306</td>
</tr>
<tr>
<td>Pattern-based text classification</td>
<td>306</td>
</tr>
<tr>
<td>Classifying text with a character-level LSTM</td>
<td>309</td>
</tr>
<tr>
<td>Differentiating subjects from objects</td>
<td>313</td>
</tr>
<tr>
<td>Parsing the sentence type</td>
<td>316</td>
</tr>
<tr>
<td>Anaphora resolution</td>
<td>320</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>323</td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>324</td>
</tr>
<tr>
<td><strong>Why subscribe?</strong></td>
<td>325</td>
</tr>
</tbody>
</table>

### Other Books You May Enjoy

### Index
spaCy is an industrial-grade, efficient NLP Python library. It offers various pre-trained models and ready-to-use features. Mastering spaCy provides you with end-to-end coverage of spaCy features and real-world applications.

You’ll begin by installing spaCy and downloading models, before progressing to spaCy’s features and prototyping real-world NLP apps. Next, you’ll get accustomed to visualizing with spaCy’s popular visualizer displaCy. The book also equips you with practical illustrations for pattern matching and helps you advance into the world of semantics with word vectors. Statistical information extraction methods are also explained in detail. Later, you’ll cover an interactive business case study that shows you how to combine spaCy features to create a real-world NLP pipeline. You’ll implement ML models such as sentiment analysis, intent recognition, and context resolution. The book further focuses on classification with popular frameworks such as TensorFlow’s Keras API together with spaCy. You’ll cover popular topics, including intent classification and sentiment analysis as well as using them on popular datasets and interpreting the classification results.

By the end of this book, you’ll be able to confidently use spaCy, including its linguistic features, word vectors, and classifiers, to create your own NLP apps.

Who this book is for

This book is for data scientists and machine learners who want to excel in NLP as well as NLP developers who want to master spaCy and build applications with it. Language and speech professionals who want to get hands-on with Python and spaCy and software developers who want to quickly prototype applications with spaCy will also find this book helpful. Beginner-level knowledge of the Python programming language is required to get the most out of this book. A beginner-level understanding of linguistic terminology, such as parsing, POS tags, and semantic similarity, will also be useful.
What this book covers

Chapter 1, *Getting Started with spaCy*, begins your spaCy journey. This chapter gives you an overview of NLP with Python. In this chapter, you'll install the spaCy library and spaCy language models and explore displaCy, spaCy's visualization tool. Overall, this chapter will get you started with installing and understanding the spaCy library.

Chapter 2, *Core Operations with spaCy*, teaches you the core operations of spaCy, such as creating a language pipeline, tokenizing the text, and breaking the text into its sentences as well as the Container classes. The Container classes token, Doc, and Span are covered in this chapter in detail.

Chapter 3, *Linguistic Features*, takes a deep dive into spaCy's full power. This chapter explores the linguistic features, including spaCy's most used features, such as POS-tagger, dependency parser, named entity recognizer, and merging/splitting.

Chapter 4, *Rule-Based Matching*, teaches you how to extract information from the text by matching patterns and phrases. You will use morphological features, POS-tags, regex, and other spaCy features to form pattern objects to feed to the spaCy Matcher objects.

Chapter 5, *Working with Word Vectors and Semantic Similarity*, teaches you about word vectors and associated semantic similarity methods. This chapter includes word vector computations such as distance calculations, analogy calculations, and visualization.

Chapter 6, *Putting Everything Together: Semantic Parsing with spaCy*, is a fully hands-on chapter. This chapter teaches you how to design a ticket reservation system NLU for Airline Travel Information System (ATIS), a well-known airplane ticket reservation system dataset, with spaCy.

Chapter 7, *Customizing spaCy Models*, teaches you how to train, store, and use custom statistical pipeline components. You will learn how to update an existing statistical pipeline component with your own data as well as how to create a statistical pipeline component from scratch with your own data and labels.

Chapter 8, *Text Classification with spaCy*, teaches you how to do a very basic and popular task of NLP: text classification. This chapter explores text classification with spaCy's Textcategorizer component as well as text classification with TensorFlow and Keras.

Chapter 9, *spaCy and Transformers*, explores the latest hot topic in NLP – transformers – and how to use them with TensorFlow and spaCy. You'll learn how to work with BERT and TensorFlow as well as transformer-based pretrained pipelines of spaCy v3.0.
Chapter 10, Putting Everything Together: Designing Your Chatbot with spaCy, takes you into the world of Conversational AI. You will do entity extraction, intent recognition, and context handling on a real-world restaurant reservation dataset.

To get the most out of this book

First of all, you'll need Python 3 installed and working on your system. Code examples are tested with spaCy v3.0, however, most of the code is compatible with spaCy v2.3 due to backwards compatibility. For the helper libraries such as scikit-learn, pandas, NumPy, and matplotlib, the latest versions available on pip will work. We use TensorFlow, transformers, and helper libraries starting with Chapter 7, Customizing spaCy Models, so you can install these libraries by the time you reach Chapter 7.

<table>
<thead>
<tr>
<th>Software/Hardware covered in the book</th>
<th>OS Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python &gt;= 3.6</td>
<td>Windows, macOS X, and Linux (any)</td>
</tr>
<tr>
<td>spaCy v3.0</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>Tensorflow 2.0</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>Transformers</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>scikit-learn</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>pandas</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>NumPy</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>matplotlib</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
<tr>
<td>Jupyter</td>
<td>Windows, macOS X, Linux (any)</td>
</tr>
</tbody>
</table>

We used Jupyter notebooks from time to time. You can view the notebooks on the book's GitHub page. If you want to work with Jupyter notebooks, that's great; you can install Jupyter via pip. If you don't want to, you can still copy and paste the code into the Python shell and make the code work.

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.
Download the example code files

You can download the example code files for this book from GitHub at https://github.com/PacktPublishing/Mastering-spaCy. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: https://static.packt-cdn.com/downloads/9781800563353_ColorImages.pdf.

Conventions used

There are a number of text conventions used throughout this book.

**Code in text:** Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Finally, the validation_split parameter is used to evaluate the experiment."

A block of code is set as follows:

```python
import spacy
nlp = spacy.load("en_subwords_wiki_lg")
```

Any command-line input or output is written as follows:

```bash
wget https://github.com/PacktPublishing/Mastering-spaCy/blob/main/Chapter08/data/Reviews.zip
```

**Bold:** Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "The following diagram illustrates the distance between dog and cat and the distance between dog, canine terrier, and cat."

**Tips or important notes**

Appear like this.
Get in touch

Feedback from our readers is always welcome.

**General feedback**: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customer care@packtpub.com.

**Errata**: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

**Piracy**: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

**If you are interested in becoming an author**: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

**Reviews**

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packt.com.
Section 1: Getting Started with spaCy

This section will begin with an overview of natural language processing (NLP) with Python and spaCy. You will learn how the book is organized and how to make the best use of the book. You will then start by installing spaCy and its statistical models and take a quick dive into the spaCy world. Basic operations, general conventions, and visualization are the core attractions of this section.

This section comprises the following chapters:

- Chapter 1, Getting Started with spaCy
- Chapter 2, Core Operations with spaCy
Getting Started with spaCy

In this chapter, we will have a comprehensive introduction to natural language processing (NLP) application development with Python and spaCy. First, we will see how NLP development goes hand in hand with Python, along with an overview of what spaCy offers as a Python library.

After the warm-up, you will quickly get started with spaCy by downloading the library and loading the models. You will then explore spaCy's popular visualizer displaCy by visualizing several features of spaCy.

By the end of this chapter, you will know what you can achieve with spaCy and how to plan your journey with spaCy code. You will be also settled with your development environment, having already installed all the necessary packages for NLP tasks in the upcoming sections.

We're going to cover the following main topics in this chapter:

- Overview of spaCy
- Installing spaCy
- Installing spaCy's statistical models
- Visualization with displaCy
Technical requirements

The chapter code can be found at the book's GitHub repository: https://github.com/PacktPublishing/Mastering-spaCy/tree/main/Chapter01

Overview of spaCy

Before getting started with the spaCy code, we will first have an overview of NLP applications in real life, NLP with Python, and NLP with spaCy. In this section, we'll find out the reasons to use Python and spaCy for developing NLP applications. We will first see how Python goes hand-in-hand with text processing, then we'll understand spaCy's place in the Python NLP libraries. Let's start our tour with the close-knit relationship between Python and NLP.

Rise of NLP

Over the past few years, most of the branches of AI created a lot of buzz, including NLP, computer vision, and predictive analytics, among others. But just what is NLP? How can a machine or code solve human language?

NLP is a subfield of AI that analyzes text, speech, and other forms of human-generated language data. Human language is complicated – even a short paragraph contains references to the previous words, pointers to real-world objects, cultural references, and the writer's or speaker's personal experiences. Figure 1.1 shows such an example sentence, which includes a reference to a relative date (recently), phrases that can be resolved only by another person who knows the speaker (regarding the city that the speaker's parents live in) and who has general knowledge about the world (a city is a place where human beings live together):

Figure 1.1 – An example of human language, containing many cognitive and cultural aspects
How do we process such a complicated structure then? We have our weapons too; we model natural language with statistical models, and we process linguistic features to turn the text into a well-structured representation. This book provides all the necessary background and tools for you to extract the meaning out of text. By the end of this book, you will possess statistical and linguistic knowledge to process text by using a great tool – the spaCy library.

Though NLP gained popularity recently, processing human language has been present in our lives via many real-world applications, including search engines, translation services, and recommendation engines.

Search engines such as Google Search, Yahoo Search, and Microsoft Bing are an integral part of our daily lives. We look for homework help, cooking recipes, information about celebrities, the latest episodes of our favorite TV series; all sorts of information that we use in our daily lives. There is even a verb in English (also in many other languages), to google, meaning to look up some information on the Google search engine.

Search engines use advanced NLP techniques including mapping queries into a semantic space, where similar queries are represented by similar vectors. A quick trick is called autocomplete, where query suggestions appear on the search bar when we type the first few letters. Autocomplete looks tricky but indeed the algorithm is a combination of a search tree walk and character-level distance calculation. A past query is represented by a sequence of its characters, where each character corresponds to a node in the search tree. The arcs between the characters are assigned weights according to the popularity of this past query.

Then, when a new query comes, we compare the current query string to past queries by walking on the tree. A fundamental Computer Science (CS) data structure, the tree, is used to represent a list of queries, who would have thought that? Figure 1.2 shows a walk on the character tree:

![Autocomplete Example](image)

Figure 1.2 – An autocomplete example
This is a simplified explanation; the real algorithms blend several techniques usually. If you want to learn more about this subject, you can read the great articles about the data structures: http://blog.notdot.net/2010/07/Damn-Cool-Algorithms-Levenshtein-Automata and http://blog.notdot.net/2007/4/Damn-Cool-Algorithms-Part-1-BK-Trees.

Continuing with search engines, search engines also know how to transform unstructured data to structured and linked data. When we type Diana Spencer into the search bar, this is what comes up:

![Search results](image)

Figure 1.3 – Search results for the query "Diana Spencer"

How did the search engine link Diana Spencer to her well-known name Princess Diana? This is called entity linking. We link entities that mention the same real-world entity. Entity-linking algorithms concern representing semantic relations and knowledge in general. This area of NLP is called the Semantic Web. You can learn more about this at https://www.cambridgesemantics.com/blog/semantic-university/intro-semantic-web/. I worked as a knowledge engineer at a search engine company at the beginning of my career and really enjoyed it. This is a fascinating subject in NLP.
There is really no limit to what you can develop: search engine algorithms, chatbots, speech recognition applications, and user sentiment recognition applications. NLP problems are challenging yet fascinating. This book's mission is to provide you a toolbox with all the necessary tools. The first step of NLP development is choosing the programming language we will use wisely. In the next section, we will explain why Python is the weapon of choice. Let's move on to the next section to see the string bond of NLP and Python.

**NLP with Python**

As we remarked before, NLP is a subfield of AI that analyzes text, speech, and other forms of human-generated language data. As an industry professional, my first choice for manipulating text data is Python. In general, there are many benefits to using Python:

- It is easy to read and looks very similar to pseudocode.
- It is easy to produce and test code with.
- It has a high level of abstraction.

Python is a great choice for developing NLP systems because of the following:

- **Simplicity**: Python is easy to learn. You can focus on NLP rather than the programming language details.
- **Efficiency**: It allows for easier development of quick NLP application prototypes.
- **Popularity**: Python is one of the most popular languages. It has huge community support, and installing new libraries with pip is effortless.
- **AI ecosystem presence**: A significant number of open source NLP libraries are available in Python. Many machine learning (ML) libraries such as PyTorch, TensorFlow, and Apache Spark also provide Python APIs.
- **Text methods**: String and file operations with Python are effortless and straightforward. For example, splitting a sentence at the whitespaces requires only a one-liner, `sentence.split()`, which can be quite painful in other languages, such as C++, where you have to deal with stream objects for this task.
When we put all the preceding points together, the following image appears – Python intersects with string processing, the AI ecosystem, and ML libraries to provide us the best NLP development experience:

![Figure 1.4 - NLP with Python overview](image)

We will use Python 3.5+ throughout this book. Users who do not already have Python installed can follow the instructions at [https://realpython.com/installing-python/](https://realpython.com/installing-python/). We recommend downloading and using the latest version of Python 3.

In Python 3.x, the default encoding is **Unicode**, which means that we can use Unicode text without worrying much about the encoding. We won't go into details of encodings here, but you can think of Unicode as an extended set of ASCII, including more characters such as German-alphabet umlauts and the accented characters of the French alphabet. This way we can process German, French, and many more languages other than English.

**Reviewing some useful string operations**

In Python, the text is represented by **strings**, objects of the `str` class. Strings are immutable sequences of characters. Creating a string object is easy – we enclose the text in quotation marks:

```python
word = 'Hello World'
```
Now the `word` variable contains the string `Hello World`. As we mentioned, strings are sequences of characters, so we can ask for the first item of the sequence:

```python
print (word [0])
H
```

Always remember to use parentheses with `print`, since we are coding in Python 3.x. We can similarly access other indices, as long as the index doesn't go out of bounds:

```python
word [4]
'o'
```

How about string length? We can use the `len` method, just like with `list` and other sequence types:

```python
len(word)
11
```

We can also iterate over the characters of a string with sequence methods:

```python
for ch in word:
    print(ch)
H e l l o
W o r l d
```

**Pro tip**

Please mind the indentation throughout the book. Indentation in Python is the way we determine the control blocks and function definitions in general, and we will apply this convention in this book.
Now let's go over the more string methods such as counting characters, finding a substring, and changing letter case.

`count` counts the number of occurrences of a character in the string, so the output is 3 here:

```python
word.count('l')
3
```

Often, you need to find the index of a character for a number of substring operations such as cutting and slicing the string:

```python
word.index('e')
1
```

Similarly, we can search for substrings in a string with the `find` method:

```python
word.find('World')
6
```

`find` returns -1 if the substring is not in the string:

```python
word.find('Bonjour')
-1
```

Searching for the last occurrence of a substring is also easy:

```python
word.rfind('l')
9
```

We can change letter case by the `upper` and `lower` methods:

```python
word.upper()
'HELLO WORLD'
```

The `upper` method changes all characters to uppercase. Similarly, the `lower` method changes all characters to lowercase:

```python
word.lower()
'hello world'
```
The **capitalize** method capitalizes the first character of the string:

```python
'hello madam'.capitalize()
Hello madam
```

The **title** method makes the string **title case**. Title case literally means *to make a title*, so each word of the string is capitalized:

```python
'hello madam'.title()
Hello Madam
```

Forming **new strings from other strings** can be done in several ways. We can **concatenate two strings** by adding them:

```python
'Hello Madam!' + 'Have a nice day.'
Hello Madam!Have a nice day.
```

We can also **multiply a string with an integer**. The output will be the string concatenated to itself by the number of times specified by the integer:

```python
'sweet ' * 5
sweet sweet sweet sweet '
```

**join** is a frequently used method; it takes a list of strings and joins them into one string:

```python
' '.join(['hello', 'madam'])
'hello madam'
```

There is a variety of **substring methods**. Replacing a substring means changing all of its occurrences with another string:

```python
'hello madam'.replace('hello', 'good morning')
good morning madam
```

Getting a substring by index is called **slicing**. You can slice a string by specifying the start index and end index. If we want only the second word, we can do the following:

```python
word = 'Hello Madam Flower'
word [6:11]
'Madam'
```
Getting the first word is similar. Leaving the first index blank means the index starts from zero:

```python
word [:5]
'Hello'
```

Leaving the second index blank has a special meaning as well – it means the rest of the string:

```python
word [12:]
'Flower'
```

We now know some of the Pythonic NLP operations. Now we can dive into more of spaCy.

**Getting a high-level overview of the spaCy library**

spaCy is an open source Python library for modern NLP. The creators of spaCy describe their work as **industrial-strength** NLP, and as a contributor I can assure you it is true. spaCy is shipped with pretrained language models and word vectors for 60+ languages.

spaCy is focused on production and shipping code, unlike its more academic predecessors. The most famous and frequently used Python predecessor is NLTK. NLTK's main focus was providing students and researchers an idea of language processing. It never put any claims on efficiency, model accuracy, or being an industrial-strength library. spaCy focused on providing production-ready code from the first day. You can expect models to perform on real-world data, the code to be efficient, and the ability to process a huge amount of text data in a reasonable time. The following table is an efficiency comparison from the spaCy documentation (https://spacy.io/usage/facts-figures#speed-comparison):

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>ABSOLUTE (MS PER DOC)</th>
<th>RELATIVE (TO SPACY)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TOKENIZE</td>
<td>TAG</td>
</tr>
<tr>
<td>spaCy</td>
<td>0.2ms</td>
<td>1ms</td>
</tr>
<tr>
<td>CoreNLP</td>
<td>0.18ms</td>
<td>10ms</td>
</tr>
<tr>
<td>ZPar</td>
<td>1ms</td>
<td>8ms</td>
</tr>
<tr>
<td>NLTK</td>
<td>4ms</td>
<td>443ms</td>
</tr>
</tbody>
</table>

Figure 1.5 – A speed comparison of spaCy and other popular NLP frameworks
The spaCy code is also maintained in a professional way, with issues sorted by labels and new releases covering as many fixes as possible. You can always raise an issue on the spaCy GitHub repo at https://github.com/explosion/spaCy, report a bug, or ask for help from the community.

Another predecessor is CoreNLP (also known as StanfordNLP). CoreNLP is implemented in Java. Though CoreNLP competes in terms of efficiency, Python won by easy prototyping and spaCy is much more professional as a software package. The code is well maintained, issues are tracked on GitHub, and every issue is marked with some labels (such as bug, feature, new project). Also, the installation of the library code and the models is easy. Together with providing backward compatibility, this makes spaCy a professional software project. Here is a detailed comparison from the spaCy documentation at https://spacy.io/usage/facts-figures#comparison:

**Feature comparison**

Here's a quick comparison of the functionalities offered by spaCy, NLTK, and CoreNLP.

<table>
<thead>
<tr>
<th>Feature</th>
<th>spaCy</th>
<th>NLTK</th>
<th>CoreNLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming language</td>
<td>Python</td>
<td>Python</td>
<td>Java / Python</td>
</tr>
<tr>
<td>Neural network models</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Integrated word vectors</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Multi-language support</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tokenization</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Part-of-speech tagging</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Sentence segmentation</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Dependency parsing</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Entity recognition</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Entity linking</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Coreference resolution</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

Figure 1.6 - A feature comparison of spaCy, NLTK, and CoreNLP
Throughout this book, we will be using spaCy’s latest releases, v2.3 and v3.0 (the versions used at the time of writing this book) for all our computational linguistics and ML purposes. The following are the features in the latest release:

- Original data preserving tokenization.
- Statistical sentence segmentation.
- Named entity recognition.
- **Part-of-Speech (POS) tagging.**
- Dependency parsing.
- Pretrained word vectors.
- Easy integration with popular deep learning libraries. spaCy’s ML library Thinc provides thin wrappers around PyTorch, TensorFlow, and MXNet. spaCy also provides wrappers for HuggingFace Transformers by spacy-transformers library. We’ll see more of the Transformers in Chapter 9, *spaCy and Transformers.*
- Industrial-level speed.
- A built-in visualizer, displaCy.
- Support for 60+ languages.
- 46 state-of-the-art statistical models for 16 languages.
- Space-efficient string data structures.
- Efficient serialization.
- Easy model packaging and usage.
- Large community support.

We had a quick glance around spaCy as an NLP library and as a software package. We will see what spaCy offers in detail throughout the book.
Tips for the reader

This book is a practical guide. In order to get the most out of the book, I recommend readers replicate the code in their own Python shell. Without following and performing the code, it is not possible to get a proper understanding of NLP concepts and spaCy methods, which is why we have arranged the upcoming chapters in the following way:

- Explanation of the language/ML concept
- Application code with spaCy
- Evaluation of the outcome
- Challenges of the methodology
- Pro tips and tricks to overcome the challenges

Installing spaCy

Let's get started by installing and setting up spaCy. spaCy is compatible with 64-bit Python 2.7 and 3.5+, and can run on Unix/Linux, macOS/OS X, and Windows. CPython is a reference implementation of Python in C. If you already have Python running on your system, most probably your CPython modules are fine too – hence you don't need to worry about this detail. The newest spaCy releases are always downloadable via pip (https://pypi.org/) and conda (https://conda.io/en/latest/). pip and conda are two of the most popular distribution packages.

pip is the most painless choice as it installs all the dependencies, so let's start with it.

Installing spaCy with pip

You can install spaCy with the following command:

```bash
$ pip install spacy
```

If you have more than one Python version installed in your system (such as Python 2.8, Python 3.5, Python 3.8, and so on), then select the pip associated with Python you want to use. For instance, if you want to use spaCy with Python 3.5, you can do the following:

```bash
$ pip3.5 install spacy
```
If you already have spaCy installed on your system, you may want to upgrade to the latest version of spaCy. We’re using spaCy 2.3 in this book; you can check which version you have with the following command:

```
$ python -m spacy info
```

This is how a version info output looks like. This has been generated with the help of my Ubuntu machine:

![An example spaCy version output](image)

Suppose you want to upgrade your spaCy version. You can upgrade your spaCy version to the latest available version with the following command:

```
$ pip install -U spacy
```

### Installing spaCy with conda

conda support is provided by the conda community. The command for installing spaCy with conda is as follows:

```
$ conda install -c conda-forge spacy
```

### Installing spaCy on macOS/OS X

macOS and OS X already ship with Python. You only need to install a recent version of the Xcode IDE. After installing Xcode, please run the following:

```
$ xcode-select --install
```

This installs the command-line development tools. Then you will be able to follow the preceding pip commands.
Installing spaCy on Windows

If you have a Windows system, you need to install a version of Visual C++ Build Tools or Visual Studio Express that matches your Python distribution. Here are the official distributions and their matching versions, taken from the spaCy installation guide (https://spacy.io/usage#source-windows):

<table>
<thead>
<tr>
<th>DISTRIBUTION</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python 2.7</td>
<td>Visual Studio 2008</td>
</tr>
<tr>
<td>Python 3.4</td>
<td>Visual Studio 2010</td>
</tr>
<tr>
<td>Python 3.5+</td>
<td>Visual Studio 2015</td>
</tr>
</tbody>
</table>

Figure 1.8 – Visual Studio and Python distribution compatibility table

If you didn't encounter any problems so far, then that means spaCy is installed and running on your system. You should be able to import spaCy into your Python shell:

```python
import spacy
```

Now you successfully installed spaCy – congrats and welcome to the spaCy universe! If you have installation problems please continue to the next section, otherwise you can move on to language model installation.

Troubleshooting while installing spaCy

There might be cases where you get issues popping up during the installation process. The good news is, we’re using a very popular library so most probably other developers have already encountered the same issues. Most of the issues are listed on Stack Overflow (https://stackoverflow.com/questions/tagged/spacy) and the spaCy GitHub Issues section (https://github.com/explosion/spaCy/issues) already. However, in this section, we’ll go over the most common issues and their solutions.

Some of the most common issues are as follows:

- **The Python distribution is incompatible**: In this case please upgrade your Python version accordingly and then do a fresh installation.
• **The upgrade broke spaCy**: Most probably there are some leftover packages in your installation directories. The best solution is to first remove the spaCy package completely by doing the following:

```
pip uninstall spacy
```

Then do a fresh installation by following the installation instructions mentioned.

• **You're unable to install spaCy on a Mac**: On a Mac, please make sure that you don't skip the following to make sure you correctly installed the Mac command-line tools and enabled pip:

```
$ xcode-select -install
```

In general, if you have the correct Python dependencies, the installation process will go smoothly.

We're all set up and ready for our first usage of spaCy, so let's go ahead and start using spaCy's language models.

## Installing spaCy's statistical models

The spaCy installation doesn't come with the statistical language models needed for the spaCy pipeline tasks. spaCy language models contain knowledge about a specific language collected from a set of resources. Language models let us perform a variety of NLP tasks, including **POS tagging** and **named-entity recognition** (NER).

Different languages have different models and are language specific. There are also different models available for the same language. We'll see the differences between those models in detail in the *Pro tip* at the end of this section, but basically the training data is different. The underlying statistical algorithm is the same. Some of the currently supported languages are as follows:
<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>CODE</th>
<th>LANGUAGE DATA</th>
<th>MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td>zh</td>
<td>lang/zh</td>
<td>3 models</td>
</tr>
<tr>
<td>Danish</td>
<td>da</td>
<td>lang/da</td>
<td>3 models</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl</td>
<td>lang/nl</td>
<td>3 models</td>
</tr>
<tr>
<td>English</td>
<td>en</td>
<td>lang/en</td>
<td>3 models</td>
</tr>
<tr>
<td>French</td>
<td>fr</td>
<td>lang/fr</td>
<td>3 models</td>
</tr>
<tr>
<td>German</td>
<td>de</td>
<td>lang/de</td>
<td>3 models</td>
</tr>
<tr>
<td>Greek</td>
<td>el</td>
<td>lang/el</td>
<td>3 models</td>
</tr>
<tr>
<td>Italian</td>
<td>it</td>
<td>lang/it</td>
<td>3 models</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja</td>
<td>lang/ja</td>
<td>3 models</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>lt</td>
<td>lang/lt</td>
<td>3 models</td>
</tr>
<tr>
<td>Multi-language</td>
<td>xx</td>
<td>lang/xx</td>
<td>3 models</td>
</tr>
<tr>
<td>Norwegian Bokmal</td>
<td>nb</td>
<td>lang/nb</td>
<td>3 models</td>
</tr>
<tr>
<td>Polish</td>
<td>pl</td>
<td>lang/pl</td>
<td>3 models</td>
</tr>
<tr>
<td>Portuguese</td>
<td>pt</td>
<td>lang/pt</td>
<td>3 models</td>
</tr>
<tr>
<td>Romanian</td>
<td>ro</td>
<td>lang/ro</td>
<td>3 models</td>
</tr>
<tr>
<td>Spanish</td>
<td>es</td>
<td>lang/es</td>
<td>3 models</td>
</tr>
</tbody>
</table>

Figure 1.9 – spaCy models overview

The number of supported languages grows rapidly. You can follow the list of supported languages on the [spaCy Models and Languages page](https://spacy.io/usage/models#languages).
Several pretrained models are available for different languages. For English, the following models are available for download: `en_core_web_sm`, `en_core_web_md`, and `en_core_web_lg`. These models use the following naming convention:

- **Language**: Indicates the language code: `en` for English, `de` for German, and so on.
- **Type**: Indicates the model capability. For instance, `core` means a general-purpose model for the vocabulary, syntax, entities, and vectors.
- **Genre**: The type of text the model recognizes. The genre can be `web` (Wikipedia), `news` (news, media), `Twitter`, and so on.
- **Size**: Indicates the model size: `lg` for large, `md` for medium, and `sm` for small.

Here is what a typical language model looks like:

```
 en_core_web_sm
```

Large models can require a lot of disk space, for example `en_core_web_lg` takes up 746 MB, while `en_core_web_md` needs 48MB and `en_core_web_sm` takes only 11MB. Medium-sized models work well for many development purposes, so we'll use the English `md` model throughout the book.
Pro tip
It is a good practice to match model genre to your text type. We recommend picking the genre as close as possible to your text. For example, the vocabulary in the social media genre will be very different from that in the Wikipedia genre. You can pick the web genre if you have social media posts, newspaper articles, financial news – that is, more language from daily life. The Wikipedia genre is suitable for rather formal articles, long documents, and technical documents. In case you are not sure which genre is the most suitable, you can download several models and test some example sentences from your own corpus and see how each model performs.

Now that we're well-informed about how to choose a model, let's download our first model.

Installing language models
Since v1.7.0, spaCy offers a great benefit: installing the models as Python packages. You can install spaCy models just like any other Python module and make them a part of your Python application. They're properly versioned, so they can go into your requirements.txt file as a dependency. You can install the models from a download URL or a local director manually, or via pip. You can put the model data anywhere on your local filesystem.

You can download a model via spaCy’s download command. download looks for the most compatible model for your spaCy version, and then downloads and installs it. This way you don’t need to bother about any potential mismatch between the model and your spaCy version. This is the easiest way to install a model:

$ python -m spacy download en_core_web_md

The preceding command selects and downloads the most compatible version of this specific model for your local spaCy version. Another option is to do the following:

$ python -m spacy download en
$ python -m spacy download de
$ python -m spacy download fr