Deep Learning for Natural Language Processing

Develop Deep Learning Models for Natural Language in Python

Jason Brownlee

MACHINE
LEARNING
MASTERY

Disclaimer

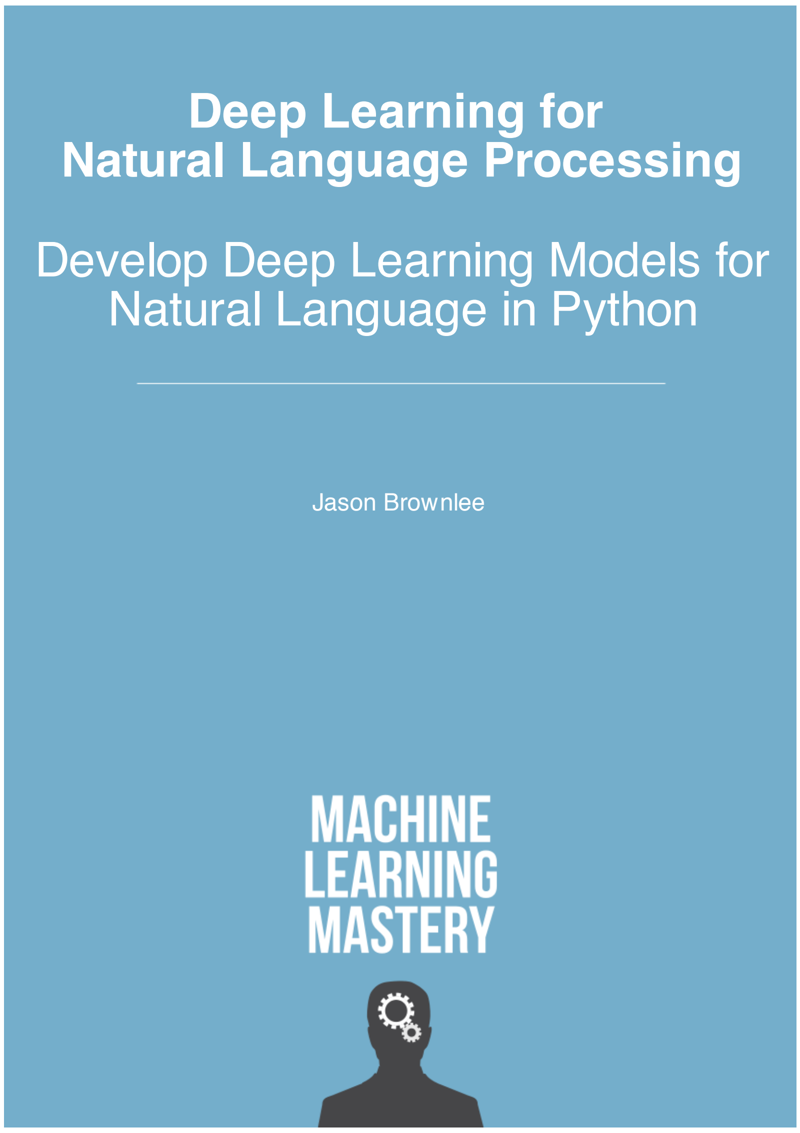
Acknowledgements

Special thanks to my copy editor Sarah Martin and my technical editors Arun Koshy and Andrei Cheremskoy.

Copyright

This is Just a Sample

Thank-you for your interest in **Deep Learning for Natural Language Processing**. This is just a sample of the full text. You can purchase the complete book online from: https://machinelearningmastery.com/deep-learning-for-nlp/


Deep Learning for Natural Language Processing
Develop Deep Learning Models for Natural Language in Python
Jason Brownlee
MACHINE LEARNING MASTERY

Contents

Preface

We are awash with text, from books, papers, blogs, tweets, news, and increasingly text from spoken utterances. Every day, I get questions asking how to develop machine learning models for text data. Working with text is hard as it requires drawing upon knowledge from diverse domains such as linguistics, machine learning, statistical natural language processing, and these days, deep learning.

I have done my best to write blog posts to answer frequently asked questions on the topic and decided to pull together my best knowledge on the matter into this book. I designed this book to teach you step-by-step how to bring modern deep learning methods to your natural language processing projects. I chose the programming language, programming libraries, and tutorial topics to give you the skills you need.

Python is the go-to language for applied machine learning and deep learning, both in terms of demand from employers and employees. This is not least because it could be a renaissance for machine learning tools. I have focused on showing you how to use the best of breed Python tools for natural language processing such as Gensim and NLTK, and even a little scikit-learn. Key to getting results is speed of development, and for this reason, we use the Keras deep learning library as you can define, train, and use complex deep learning models with just a few lines of Python code.

There are three key areas that you must know when working with text:

1. How to clean text. This includes loading, analyzing, filtering and cleaning tasks required prior to modeling.

2. How to represent text. This includes the classical bag-of-words model and the modern and powerful distributed representation in word embeddings.

3. How to generate text. This includes the range of most interesting problems, such as image captioning and translation.

These key topics provide the backbone for the book and the tutorials you will work through. I believe that after completing this book, you will have the skills that you need to both work through your own natural language processing projects and bring modern deep learning methods to bare.

Jason Brownlee
2019

Part I

Introductions

Welcome

Welcome to *Deep Learning for Natural Language Processing.* Natural language processing is the area of study dedicated to the automatic manipulation of speech and text by software. It is an old field of study, originally dominated by rule-based methods designed by linguists, then statistical methods, and, more recently, deep learning methods that show great promise in the field. So much so that the heart of the Google Translate service uses a deep learning method, a topic that you will learn more about in this book.

This book was designed to teach you step-by-step how to bring modern deep learning models to your own natural language processing projects.

Who Is This Book For?

Before we get started, let's make sure you are in the right place. This book is for developers that know some applied machine learning and some deep learning. Maybe you want or need to start using deep learning for text on your research project or on a project at work. This guide was written to help you do that quickly and efficiently by compressing years worth of knowledge and experience into a laser-focused course of hands-on tutorials. The lessons in this book assume a few things about you, such as:

- You know your way around basic Python for programming.
- You know your way around basic NumPy for array manipulation.
- You know your way around basic scikit-learn for machine learning.
- You know your way around basic Keras for deep learning.

For some bonus points, perhaps some of the below points apply to you. Don't panic if they don't.

- You may know how to work through a predictive modeling problem end-to-end.
- You may know a little bit of natural language processing.
- You may know a little bit of natural language libraries such as NLTK or Gensim.

This guide was written in the top-down and results-first machine learning style that you're used to from MachineLearningMastery.com.

About Your Outcomes

This book will teach you how to get results as a machine learning practitioner interested in using deep learning on your natural language processing project. After reading and working through this book, you will know:

- What natural language processing is and why it is challenging.
- What deep learning is and how it is different from other machine learning methods.
- The promise of deep learning methods for natural language processing problems.
- How to prepare text data for modeling using best-of-breed Python libraries.
- How to develop distributed representations of text using word embedding models.
- How to develop a bag-of-words model, a representation technique that can be used for machine learning and deep learning methods.
- How to develop a neural sentiment analysis model for automatically predicting the class label for a text document.
- How to develop a neural language model, required for any text generating neural network.
- How to develop a photo captioning system to automatically generate textual descriptions of photographs.
- How to develop a neural machine translation system for translating text from one language to another.

This book will NOT teach you how to be a research scientist and all the theory behind why specific methods work. For that, I would recommend good research papers and textbooks. See the *Further Reading* section at the end of each tutorial for a good starting point.

How to Read This Book

This book was written to be read linearly, from start to finish. That being said, if you know the basics and need help with a specific problem type or technique, then you can flip straight to that section and get started. This book was designed for you to read on your workstation, on the screen, not on an eReader. My hope is that you have the book open right next to your editor and run the examples as you read about them.

This book is not intended to be read passively or be placed in a folder as a reference text. It is a playbook, a workbook, and a guidebook intended for you to learn by doing and then apply your new understanding to your own natural language projects. To get the most out of the book, I would recommend playing with the examples in each tutorial. Extend them, break them, then fix them. Try some of the extensions presented at the end of each lesson and let me know how you do.

About the Book Structure

This book was designed around major activities, techniques, and natural language processing problems. There are a lot of things you could learn about deep learning and natural language processing, from theory to applications to APIs. My goal is to take you straight to getting results with laser-focused tutorials. I designed the tutorials to focus on how to get things done. They give you the tools to both rapidly understand and apply each technique to your own natural language processing prediction problems.

Each of the tutorials are designed to take you about one hour to read through and complete, excluding the extensions and further reading. You can choose to work through the lessons one per day, one per week, or at your own pace. I think momentum is critically important, and this book is intended to be read and used, not to sit idle. I would recommend picking a schedule and sticking to it. The tutorials are divided into eight parts:

- **Part 1: Foundations**. Discover a gentle introduction to natural language processing, deep learning, and the promise of combining the two, as well as tutorials on how to get started with Keras.
- **Part 2: Data Preparation**: Discover tutorials that show how to clean, prepare and encode text ready for modeling with neural networks.
- **Part 3: Bag-of-Words**. Discover the bag-of-words model, a staple representation for machine learning and a good starting point for neural networks for sentiment analysis.
- **Part 4: Word Embeddings**. Discover a more powerful word representation in word embeddings, how to develop them as standalone models, and how to learn them as part of neural network models.
- **Part 5: Text Classification**. Discover how to leverage word embeddings and convolutional neural networks to learn spatial invariant models of text for sentiment analysis, a successor to the bag-of-words model.
- **Part 6: Language Modeling**. Discover how to develop character-based and word-based language models, a technique that is required as part of any modern text generating model.
- **Part 7: Image Captioning**. Discover how to combine a pre-trained object recognition model with a language model to automatically caption images.
- **Part 8: Machine Translation**. Discover how to combine two language models to automatically translate text from one language to another.

Each part targets a specific learning outcome, and so does each tutorial within each part. This acts as a filter to ensure you are only focused on the things you need to know to get to a specific result and do not get bogged down in the math or near-infinite number of configuration parameters. The tutorials were not designed to teach you everything there is to know about each of the techniques or natural language processing problems. They were designed to give you an understanding of how they work, how to use them on your projects the fastest way I know how: to learn by doing.

About Python Code Examples

The code examples were carefully designed to demonstrate the purpose of a given lesson. For this reason, the examples are highly targeted.

- Models were demonstrated on real-world datasets to give you the context and confidence to bring the techniques to your own natural language processing problems.
- Model configurations used were discovered through trial and error are skillful, but not optimized. This leaves the door open for you to explore new and possibly better configurations.
- Code examples are complete and standalone. The code for each lesson will run as-is with no code from prior lessons or third parties required beyond the installation of the required packages.

A complete working example is presented with each tutorial for you to inspect and copy-and-paste. All source code is also provided with the book and I would recommend running the provided files whenever possible to avoid any copy-paste issues. The provided code was developed in a text editor and intended to be run on the command line. No special IDE or notebooks are required. If you are using a more advanced development environment and are having trouble, try running the example from the command line instead.

Neural network algorithms are stochastic. This means that they will make different predictions when the same model configuration is trained on the same training data. On top of that, each experimental problem in this book is based around generating stochastic predictions. As a result, this means you will not get exactly the same sample output presented in this book. This is by design. I want you to get used to the stochastic nature of the neural network algorithms. If this bothers you, please note:

- You can re-run a given example a few times and your results should be close to the values reported.
- You can make the output consistent by fixing the NumPy random number seed.
- You can develop a robust estimate of the skill of a model by fitting and evaluating it multiple times and taking the average of the final skill score (highly recommended).

All code examples were tested on a POSIX-compatible machine with Python 3 and Keras 2. All code examples will run on modest and modern computer hardware and were executed on a CPU. No GPUs are required to run the presented examples, although a GPU would make the code run faster. I am only human and there may be a bug in the sample code. If you discover a bug, please let me know so I can fix it and update the book and send out a free update.

About Further Reading

Each lesson includes a list of further reading resources. This may include:

- Research papers.

- Books and book chapters.
- Webpages.
- API documentation.

Wherever possible, I try to list and link to the relevant API documentation for key objects and functions used in each lesson so you can learn more about them. When it comes to research papers, I try to list papers that are first to use a specific technique or first in a specific problem domain. These are not required reading, but can give you more technical details, theory, and configuration details if you're looking for it. Wherever possible, I have tried to link to the freely available version of the paper on `arxiv.org`. You can search for and download any of the papers listed on Google Scholar Search `scholar.google.com`. Wherever possible, I have tried to link to books on Amazon. I don't know everything, and if you discover a good resource related to a given lesson, please let me know so I can update the book.

About Getting Help

You might need help along the way. Don't worry, you are not alone.

- *Help with a Technique?* If you need help with the technical aspects of a specific model or method, see the *Further Reading* sections at the end of each lesson.
- *Help with Keras?* If you need help with using the Keras library, see the list of resources in *Appendix A*.
- *Help with your workstation?* If you need help setting up your environment, I would recommend using Anaconda and following my tutorial in *Appendix B*.
- *Help running large models?* I recommend renting time on Amazon Web Service (AWS) to run large models. If you need help getting started on AWS, see the tutorial in *Appendix C*.
- *Help in general?* You can shoot me an email. My details are in *Appendix A*.

Summary

Are you ready? Let's dive in!

Next up you will discover a concrete idea of what natural language processing actually means.

Part II

Foundations

Part III

Data Preparation

Part IV

Bag-of-Words

Chapter 1

How to Prepare Movie Review Data for Sentiment Analysis

Text data preparation is different for each problem. Preparation starts with simple steps, like loading data, but quickly gets difficult with cleaning tasks that are very specific to the data you are working with. You need help as to where to begin and what order to work through the steps from raw data to data ready for modeling. In this tutorial, you will discover how to prepare movie review text data for sentiment analysis, step-by-step. After completing this tutorial, you will know:

- How to load text data and clean it to remove punctuation and other non-words.
- How to develop a vocabulary, tailor it, and save it to file.
- How to prepare movie reviews using cleaning and a pre-defined vocabulary and save them to new files ready for modeling.

Let's get started.

1.1 Tutorial Overview

This tutorial is divided into the following parts:

1. Movie Review Dataset
2. Load Text Data
3. Clean Text Data
4. Develop Vocabulary
5. Save Prepared Data

1.2 Movie Review Dataset

The Movie Review Data is a collection of movie reviews retrieved from the `imdb.com` website in the early 2000s by Bo Pang and Lillian Lee. The reviews were collected and made available as part of their research on natural language processing. The reviews were originally released in 2002, but an updated and cleaned up version was released in 2004, referred to as *v2.0*. The dataset is comprised of 1,000 positive and 1,000 negative movie reviews drawn from an archive of the rec.arts.movies.reviews newsgroup hosted at IMDB. The authors refer to this dataset as the *polarity dataset*.

> Our data contains 1000 positive and 1000 negative reviews all written before 2002, with a cap of 20 reviews per author (312 authors total) per category. We refer to this corpus as the polarity dataset.

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

The data has been cleaned up somewhat, for example:

- The dataset is comprised of only English reviews.
- All text has been converted to lowercase.
- There is white space around punctuation like periods, commas, and brackets.
- Text has been split into one sentence per line.

The data has been used for a few related natural language processing tasks. For classification, the performance of classical models (such as Support Vector Machines) on the data is in the range of high 70% to low 80% (e.g. 78%-to-82%). More sophisticated data preparation may see results as high as 86% with 10-fold cross-validation. This gives us a ballpark of low-to-mid 80s if we were looking to use this dataset in experiments on modern methods.

> ... depending on choice of downstream polarity classifier, we can achieve highly statistically significant improvement (from 82.8% to 86.4%)

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

You can download the dataset from here:

- Movie Review Polarity Dataset (`review_polarity.tar.gz`, 3MB).
  http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz

After unzipping the file, you will have a directory called `txt_sentoken` with two sub-directories containing the text *neg* and *pos* for negative and positive reviews. Reviews are stored one per file with a naming convention from *cv000* to *cv999* for each of *neg* and *pos*. Next, let's look at loading the text data.

1.3 Load Text Data

In this section, we will look at loading individual text files, then processing the directories of files. We will assume that the review data is downloaded and available in the current working directory in the folder `txt_sentoken`. We can load an individual text file by opening it, reading in the ASCII text, and closing the file. This is standard file handling stuff. For example, we can load the first negative review file `cv000_29416.txt` as follows:

```
# load one file
filename = 'txt_sentoken/neg/cv000_29416.txt'
# open the file as read only
file = open(filename, 'r')
# read all text
text = file.read()
# close the file
file.close()
```

Listing 1.1: Example of loading a single movie review.

This loads the document as ASCII and preserves any white space, like new lines. We can turn this into a function called `load_doc()` that takes a filename of the document to load and returns the text.

```
# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text
```

Listing 1.2: Function to load a document into memory.

We have two directories each with 1,000 documents each. We can process each directory in turn by first getting a list of files in the directory using the `listdir()` function, then loading each file in turn. For example, we can load each document in the negative directory using the `load_doc()` function to do the actual loading.

```
from os import listdir

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# specify directory to load
directory = 'txt_sentoken/neg'
# walk through all files in the folder
for filename in listdir(directory):
  # skip files that do not have the right extension
  if not filename.endswith(".txt"):
    next
  # create the full path of the file to open
  path = directory + '/' + filename
  # load document
  doc = load_doc(path)
  print('Loaded %s' % filename)
```

Listing 1.3: Example of loading a all movie reviews.

Running this example prints the filename of each review after it is loaded.

```
...
Loaded cv995_23113.txt
Loaded cv996_12447.txt
Loaded cv997_5152.txt
Loaded cv998_15691.txt
Loaded cv999_14636.txt
```

Listing 1.4: Example output of loading all movie reviews.

We can turn the processing of the documents into a function as well and use it as a template later for developing a function to clean all documents in a folder. For example, below we define a `process_docs()` function to do the same thing.

```
from os import listdir

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# load all docs in a directory
def process_docs(directory):
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # load document
    doc = load_doc(path)
    print('Loaded %s' % filename)

# specify directory to load
directory = 'txt_sentoken/neg'
process_docs(directory)
```

Listing 1.5: Example of loading a all movie reviews with functions.

Now that we know how to load the movie review text data, let's look at cleaning it.

1.4 Clean Text Data

In this section, we will look at what data cleaning we might want to do to the movie review data. We will assume that we will be using a bag-of-words model or perhaps a word embedding that does not require too much preparation.

1.4.1 Split into Tokens

First, let's load one document and look at the raw tokens split by white space. We will use the `load_doc()` function developed in the previous section. We can use the `split()` function to split the loaded document into tokens separated by white space.

```
# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# load the document
filename = 'txt_sentoken/neg/cv000_29416.txt'
text = load_doc(filename)
# split into tokens by white space
tokens = text.split()
print(tokens)
```

Listing 1.6: Load a movie review and split by white space.

Running the example gives a nice long list of raw tokens from the document.

```
...
'years', 'ago', 'and', 'has', 'been', 'sitting', 'on', 'the', 'shelves', 'ever', 'since',
    '.', 'whatever', '.', '.', '.', 'skip', 'it', '!', "where's", 'joblo', 'coming',
    'from', '?', 'a', 'nightmare', 'of', 'elm', 'street', '3', '(', '7/10', ')', '-',
    'blair', 'witch', '2', '(', '7/10', ')', '-', 'the', 'crow', '(', '9/10', ')', '-',
    'the', 'crow', ':', 'salvation', '(', '4/10', ')', '-', 'lost', 'highway', '(',
    '10/10', ')', '-', 'memento', '(', '10/10', ')', '-', 'the', 'others', '(', '9/10',
    ')', '-', 'stir', 'of', 'echoes', '(', '8/10', ')']
```

Listing 1.7: Example output of spitting a review by white space.

Just looking at the raw tokens can give us a lot of ideas of things to try, such as:

- Remove punctuation from words (e.g. 'what's').
- Removing tokens that are just punctuation (e.g. '-').
- Removing tokens that contain numbers (e.g. '10/10').
- Remove tokens that have one character (e.g. 'a').
- Remove tokens that don't have much meaning (e.g. 'and').

Some ideas:

- We can filter out punctuation from tokens using regular expressions.
- We can remove tokens that are just punctuation or contain numbers by using an `isalpha()` check on each token.
- We can remove English stop words using the list loaded using NLTK.
- We can filter out short tokens by checking their length.

Below is an updated version of cleaning this review.

```
from nltk.corpus import stopwords
import string
import re

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# load the document
filename = 'txt_sentoken/neg/cv000_29416.txt'
text = load_doc(filename)
# split into tokens by white space
tokens = text.split()
# prepare regex for char filtering
re_punc = re.compile('[%s]' % re.escape(string.punctuation))
# remove punctuation from each word
tokens = [re_punc.sub('', w) for w in tokens]
# remove remaining tokens that are not alphabetic
tokens = [word for word in tokens if word.isalpha()]
# filter out stop words
stop_words = set(stopwords.words('english'))
tokens = [w for w in tokens if not w in stop_words]
# filter out short tokens
tokens = [word for word in tokens if len(word) > 1]
print(tokens)
```

Listing 1.8: Load and clean one movie review.

Running the example gives a much cleaner looking list of tokens.

```
...
'explanation', 'craziness', 'came', 'oh', 'way', 'horror', 'teen', 'slasher', 'flick',
    'packaged', 'look', 'way', 'someone', 'apparently', 'assuming', 'genre', 'still',
    'hot', 'kids', 'also', 'wrapped', 'production', 'two', 'years', 'ago', 'sitting',
    'shelves', 'ever', 'since', 'whatever', 'skip', 'wheres', 'joblo', 'coming',
    'nightmare', 'elm', 'street', 'blair', 'witch', 'crow', 'crow', 'salvation', 'lost',
    'highway', 'memento', 'others', 'stir', 'echoes']
```

Listing 1.9: Example output of cleaning one movie review.

We can put this into a function called `clean_doc()` and test it on another review, this time a positive review.

```
from nltk.corpus import stopwords
import string
import re

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# turn a doc into clean tokens
def clean_doc(doc):
  # split into tokens by white space
  tokens = doc.split()
  # prepare regex for char filtering
  re_punc = re.compile('[%s]' % re.escape(string.punctuation))
  # remove punctuation from each word
  tokens = [re_punc.sub('', w) for w in tokens]
  # remove remaining tokens that are not alphabetic
  tokens = [word for word in tokens if word.isalpha()]
  # filter out stop words
  stop_words = set(stopwords.words('english'))
  tokens = [w for w in tokens if not w in stop_words]
  # filter out short tokens
  tokens = [word for word in tokens if len(word) > 1]
  return tokens

# load the document
filename = 'txt_sentoken/pos/cv000_29590.txt'
text = load_doc(filename)
tokens = clean_doc(text)
print(tokens)
```

Listing 1.10: Function to clean movie reviews.

Again, the cleaning procedure seems to produce a good set of tokens, at least as a first cut.

```
...
'comic', 'oscar', 'winner', 'martin', 'childs', 'shakespeare', 'love', 'production',
    'design', 'turns', 'original', 'prague', 'surroundings', 'one', 'creepy', 'place',
    'even', 'acting', 'hell', 'solid', 'dreamy', 'depp', 'turning', 'typically', 'strong',
    'performance', 'deftly', 'handling', 'british', 'accent', 'ians', 'holm', 'joe',
    'goulds', 'secret', 'richardson', 'dalmatians', 'log', 'great', 'supporting', 'roles',
    'big', 'surprise', 'graham', 'cringed', 'first', 'time', 'opened', 'mouth',
    'imagining', 'attempt', 'irish', 'accent', 'actually', 'wasnt', 'half', 'bad', 'film',
    'however', 'good', 'strong', 'violencegore', 'sexuality', 'language', 'drug', 'content']
```

Listing 1.11: Example output of a function to clean movie reviews.

There are many more cleaning steps we could take and I leave them to your imagination. Next, let's look at how we can manage a preferred vocabulary of tokens.

1.5 Develop Vocabulary

When working with predictive models of text, like a bag-of-words model, there is a pressure to reduce the size of the vocabulary. The larger the vocabulary, the more sparse the representation of each word or document. A part of preparing text for sentiment analysis involves defining and tailoring the vocabulary of words supported by the model. We can do this by loading all of the documents in the dataset and building a set of words. We may decide to support all of these words, or perhaps discard some. The final chosen vocabulary can then be saved to file for later use, such as filtering words in new documents in the future.

We can keep track of the vocabulary in a `Counter`, which is a dictionary of words and their count with some additional convenience functions. We need to develop a new function to process a document and add it to the vocabulary. The function needs to load a document by calling the previously developed `load_doc()` function. It needs to clean the loaded document using the previously developed `clean_doc()` function, then it needs to add all the tokens to the `Counter`, and update counts. We can do this last step by calling the `update()` function on the counter object. Below is a function called `add_doc_to_vocab()` that takes as arguments a document filename and a `Counter` vocabulary.

```
# load doc and add to vocab
def add_doc_to_vocab(filename, vocab):
  # load doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # update counts
  vocab.update(tokens)
```

Listing 1.12: Function add a movie review to a vocabulary.

Finally, we can use our template above for processing all documents in a directory called `process_docs()` and update it to call `add_doc_to_vocab()`.

```
# load all docs in a directory
def process_docs(directory, vocab):
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # add doc to vocab
    add_doc_to_vocab(path, vocab)
```

Listing 1.13: Updated process documents function.

We can put all of this together and develop a full vocabulary from all documents in the dataset.

```
import string
import re
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
```

```
# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# turn a doc into clean tokens
def clean_doc(doc):
  # split into tokens by white space
  tokens = doc.split()
  # prepare regex for char filtering
  re_punc = re.compile('[%s]' % re.escape(string.punctuation))
  # remove punctuation from each word
  tokens = [re_punc.sub('', w) for w in tokens]
  # remove remaining tokens that are not alphabetic
  tokens = [word for word in tokens if word.isalpha()]
  # filter out stop words
  stop_words = set(stopwords.words('english'))
  tokens = [w for w in tokens if not w in stop_words]
  # filter out short tokens
  tokens = [word for word in tokens if len(word) > 1]
  return tokens

# load doc and add to vocab
def add_doc_to_vocab(filename, vocab):
  # load doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # update counts
  vocab.update(tokens)

# load all docs in a directory
def process_docs(directory, vocab):
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # add doc to vocab
    add_doc_to_vocab(path, vocab)

# define vocab
vocab = Counter()
# add all docs to vocab
process_docs('txt_sentoken/neg', vocab)
process_docs('txt_sentoken/pos', vocab)
# print the size of the vocab
print(len(vocab))
```

```
# print the top words in the vocab
print(vocab.most_common(50))
```

Listing 1.14: Example of cleaning all reviews and building a vocabulary.

Running the example creates a vocabulary with all documents in the dataset, including positive and negative reviews. We can see that there are a little over 46,000 unique words across all reviews and the top 3 words are *film*, *one*, and *movie*.

```
46557
[('film', 8860), ('one', 5521), ('movie', 5440), ('like', 3553), ('even', 2555), ('good',
    2320), ('time', 2283), ('story', 2118), ('films', 2102), ('would', 2042), ('much',
    2024), ('also', 1965), ('characters', 1947), ('get', 1921), ('character', 1906),
    ('two', 1825), ('first', 1768), ('see', 1730), ('well', 1694), ('way', 1668), ('make',
    1590), ('really', 1563), ('little', 1491), ('life', 1472), ('plot', 1451), ('people',
    1420), ('movies', 1416), ('could', 1395), ('bad', 1374), ('scene', 1373), ('never',
    1364), ('best', 1301), ('new', 1277), ('many', 1268), ('doesnt', 1267), ('man', 1266),
    ('scenes', 1265), ('dont', 1210), ('know', 1207), ('hes', 1150), ('great', 1141),
    ('another', 1111), ('love', 1089), ('action', 1078), ('go', 1075), ('us', 1065),
    ('director', 1056), ('something', 1048), ('end', 1047), ('still', 1038)]
```

Listing 1.15: Example output of building a vocabulary.

Perhaps the least common words, those that only appear once across all reviews, are not predictive. Perhaps some of the most common words are not useful too. These are good questions and really should be tested with a specific predictive model. Generally, words that only appear once or a few times across 2,000 reviews are probably not predictive and can be removed from the vocabulary, greatly cutting down on the tokens we need to model. We can do this by stepping through words and their counts and only keeping those with a count above a chosen threshold. Here we will use 5 occurrences.

```
# keep tokens with > 5 occurrence
min_occurrence = 5
tokens = [k for k,c in vocab.items() if c >= min_occurrence]
print(len(tokens))
```

Listing 1.16: Example of filtering the vocabulary by an occurrence count.

This reduces the vocabulary from 46,557 to 14,803 words, a huge drop. Perhaps a minimum of 5 occurrences is too aggressive; you can experiment with different values. We can then save the chosen vocabulary of words to a new file. I like to save the vocabulary as ASCII with one word per line. Below defines a function called `save_list()` to save a list of items, in this case, tokens to file, one per line.

```
def save_list(lines, filename):
  data = '\n'.join(lines)
  file = open(filename, 'w')
  file.write(data)
  file.close()
```

Listing 1.17: Function to save the vocabulary to file.

The complete example for defining and saving the vocabulary is listed below.

```
import string
import re
from os import listdir
from collections import Counter
from nltk.corpus import stopwords

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# turn a doc into clean tokens
def clean_doc(doc):
  # split into tokens by white space
  tokens = doc.split()
  # prepare regex for char filtering
  re_punc = re.compile('[%s]' % re.escape(string.punctuation))
  # remove punctuation from each word
  tokens = [re_punc.sub('', w) for w in tokens]
  # remove remaining tokens that are not alphabetic
  tokens = [word for word in tokens if word.isalpha()]
  # filter out stop words
  stop_words = set(stopwords.words('english'))
  tokens = [w for w in tokens if not w in stop_words]
  # filter out short tokens
  tokens = [word for word in tokens if len(word) > 1]
  return tokens

# load doc and add to vocab
def add_doc_to_vocab(filename, vocab):
  # load doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # update counts
  vocab.update(tokens)

# load all docs in a directory
def process_docs(directory, vocab):
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # add doc to vocab
    add_doc_to_vocab(path, vocab)

# save list to file
def save_list(lines, filename):
  data = '\n'.join(lines)
  file = open(filename, 'w')
  file.write(data)
  file.close()

# define vocab
vocab = Counter()
# add all docs to vocab
process_docs('txt_sentoken/neg', vocab)
process_docs('txt_sentoken/pos', vocab)
# print the size of the vocab
print(len(vocab))
# print the top words in the vocab
print(vocab.most_common(50))
# keep tokens with > 5 occurrence
min_occurrence = 5
tokens = [k for k,c in vocab.items() if c >= min_occurrence]
print(len(tokens))
# save tokens to a vocabulary file
save_list(tokens, 'vocab.txt')
```

Listing 1.18: Example building and saving a final vocabulary.

Running this final snippet after creating the vocabulary will save the chosen words to file. It is a good idea to take a look at, and even study, your chosen vocabulary in order to get ideas for better preparing this data, or text data in the future.

```
hasnt
updating
figuratively
symphony
civilians
might
fisherman
hokum
witch
buffoons
...
```

Listing 1.19: Sample of the saved vocabulary file.

Next, we can look at using the vocabulary to create a prepared version of the movie review dataset.

1.6 Save Prepared Data

We can use the data cleaning and chosen vocabulary to prepare each movie review and save the prepared versions of the reviews ready for modeling. This is a good practice as it decouples the data preparation from modeling, allowing you to focus on modeling and circle back to data prep if you have new ideas. We can start off by loading the vocabulary from `vocab.txt`.

```
# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# load vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
```

Listing 1.20: Load the saved vocabulary.

Next, we can clean the reviews, use the loaded vocab to filter out unwanted tokens, and save the clean reviews in a new file. One approach could be to save all the positive reviews in one file and all the negative reviews in another file, with the filtered tokens separated by white space for each review on separate lines. First, we can define a function to process a document, clean it, filter it, and return it as a single line that could be saved in a file. Below defines the `doc_to_line()` function to do just that, taking a filename and vocabulary (as a set) as arguments. It calls the previously defined `load_doc()` function to load the document and `clean_doc()` to tokenize the document.

```
# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
  # load the doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # filter by vocab
  tokens = [w for w in tokens if w in vocab]
  return ' '.join(tokens)
```

Listing 1.21: Function to filter a review by the vocabulary

Next, we can define a new version of `process_docs()` to step through all reviews in a folder and convert them to lines by calling `doc_to_line()` for each document. A list of lines is then returned.

```
# load all docs in a directory
def process_docs(directory, vocab):
  lines = list()
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # load and clean the doc
    line = doc_to_line(path, vocab)
    # add to list
    lines.append(line)
  return lines
```

Listing 1.22: Updated process docs function to filter all documents by the vocabulary.

We can then call `process_docs()` for both the directories of positive and negative reviews, then call `save_list()` from the previous section to save each list of processed reviews to a file.

The complete code listing is provided below.

```
import string
import re
from os import listdir
from nltk.corpus import stopwords

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# turn a doc into clean tokens
def clean_doc(doc):
  # split into tokens by white space
  tokens = doc.split()
  # prepare regex for char filtering
  re_punc = re.compile('[%s]' % re.escape(string.punctuation))
  # remove punctuation from each word
  tokens = [re_punc.sub('', w) for w in tokens]
  # remove remaining tokens that are not alphabetic
  tokens = [word for word in tokens if word.isalpha()]
  # filter out stop words
  stop_words = set(stopwords.words('english'))
  tokens = [w for w in tokens if not w in stop_words]
  # filter out short tokens
  tokens = [word for word in tokens if len(word) > 1]
  return tokens

# save list to file
def save_list(lines, filename):
  data = '\n'.join(lines)
  file = open(filename, 'w')
  file.write(data)
  file.close()

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
  # load the doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # filter by vocab
  tokens = [w for w in tokens if w in vocab]
  return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab):
  lines = list()
  # walk through all files in the folder
  for filename in listdir(directory):
    # skip files that do not have the right extension
    if not filename.endswith(".txt"):
      next
    # create the full path of the file to open
    path = directory + '/' + filename
    # load and clean the doc
    line = doc_to_line(path, vocab)
    # add to list
    lines.append(line)
  return lines

# load vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
# prepare negative reviews
negative_lines = process_docs('txt_sentoken/neg', vocab)
save_list(negative_lines, 'negative.txt')
# prepare positive reviews
positive_lines = process_docs('txt_sentoken/pos', vocab)
save_list(positive_lines, 'positive.txt')
```

Listing 1.23: Example of cleaning and filtering all reviews by the vocab and saving the results to file.

Running the example saves two new files, `negative.txt` and `positive.txt`, that contain the prepared negative and positive reviews respectively. The data is ready for use in a bag-of-words or even word embedding model.

1.7 Further Reading

This section provides more resources on the topic if you are looking go deeper.

1.7.1 Dataset

- Movie Review Data.
  http://www.cs.cornell.edu/people/pabo/movie-review-data/

- *A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts*, 2004.
  http://xxx.lanl.gov/abs/cs/0409058

- Movie Review Polarity Dataset.
  http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz

- Dataset Readme v2.0 and v1.1.
  http://www.cs.cornell.edu/people/pabo/movie-review-data/poldata.README.2.0.txt
  http://www.cs.cornell.edu/people/pabo/movie-review-data/README.1.1

1.7.2 APIs

- `nltk.tokenize` package API.
  http://www.nltk.org/api/nltk.tokenize.html
- Chapter 2, *Accessing Text Corpora and Lexical Resources*.
  http://www.nltk.org/book/ch02.html
- `os` API Miscellaneous operating system interfaces.
  https://docs.python.org/3/library/os.html
- `collections` API - Container datatypes.
  https://docs.python.org/3/library/collections.html

1.8 Summary

In this tutorial, you discovered how to prepare movie review text data for sentiment analysis, step-by-step. Specifically, you learned:

- How to load text data and clean it to remove punctuation and other non-words.
- How to develop a vocabulary, tailor it, and save it to file.
- How to prepare movie reviews using cleaning and a predefined vocabulary and save them to new files ready for modeling.

1.8.1 Next

In the next chapter, you will discover how you can develop a neural bag-of-words model for movie review sentiment analysis.

Part V

Word Embeddings

Part VI

Text Classification

Part VII

Language Modeling

Part VIII

Image Captioning

Part IX

Machine Translation

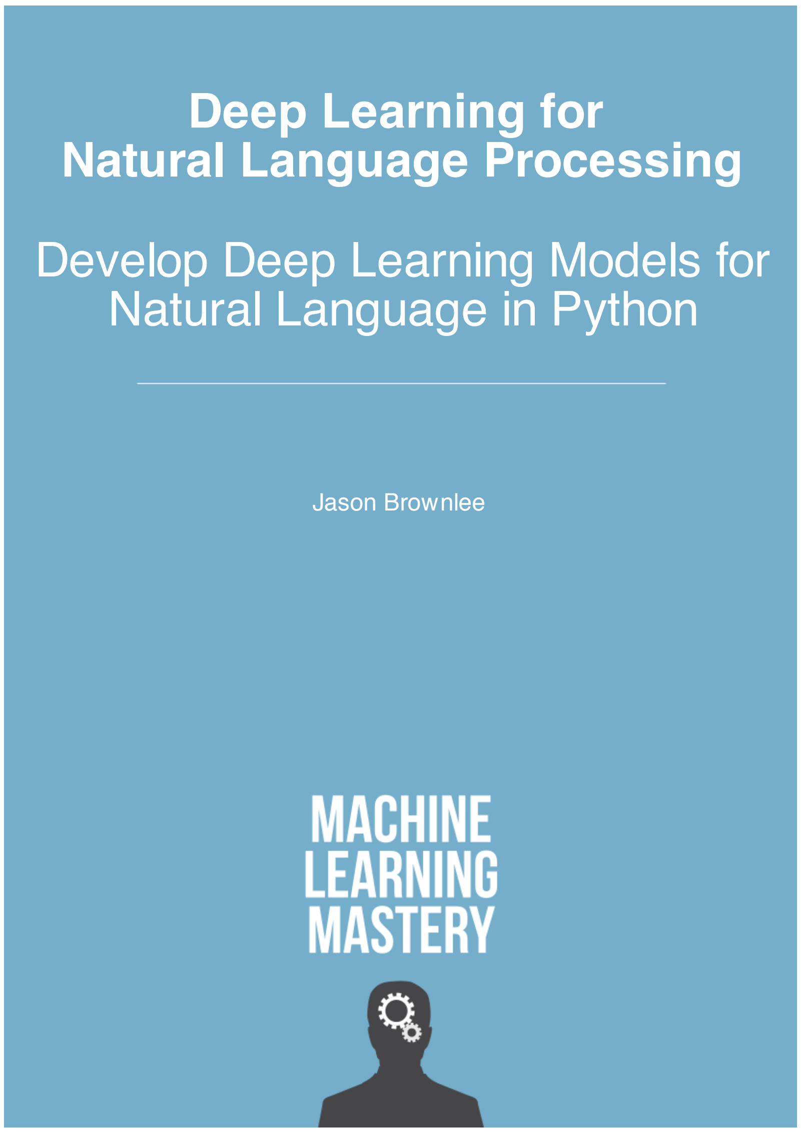
Part X

Appendix

Part XI

Conclusions

This is Just a Sample


Deep Learning for
Natural Language Processing
Develop Deep Learning Models for
Natural Language in Python
Jason Brownlee
MACHINE
LEARNING
MASTERY