Completed • Knowledge • 578 teams

Bag of Words Meets Bags of Popcorn

Tue 9 Dec 2014 – Tue 30 Jun 2015 (19 months ago)

Part 1: For Beginners - Bag of Words

What is NLP?

NLP (Natural Language Processing) is a set of techniques for approaching text problems. This page will help you get started with loading and cleaning the IMDB movie reviews, then applying a simple Bag of Words model to get surprisingly accurate predictions of whether a review is thumbs-up or thumbs-down.

Before you get started

This tutorial is in Python. If you haven't used Python before, we suggest heading over to the Titanic Competition Python Tutorials to get your feet wet (check out the Random Forest intro while you're there).

If you are already comfortable with Python and with basic NLP techniques, you may want to skip to Part 2.

This part of the tutorial is not platform dependent. Throughout this tutorial we'll be using various Python modules for text processing, deep learning, random forests, and other applications. See the **Setting Up Your System** page for more details.

There are many good tutorials, and indeed entire books written about NLP and text processing in Python. This tutorial is in no way meant to be exhaustive - just to help get you started with the movie reviews.

Code

The tutorial code for Part 1 lives here.

Reading the Data

The necessary files can be downloaded from the Data page. The first file that you'll need is **unlabeledTrainData.tsv**, which contains 25,000 IMDB movie reviews, each with a positive or negative sentiment label.

Next, read the tab-delimited file into Python. To do this, we can use the **pandas** package, introduced in the Titanic tutorial, which provides the `read_csv` function for easily reading and writing data files. If you haven't used pandas before, you may need to install it.

```
# Import the pandas package, then use the "read_csv" function to read
# the labeled training data
import pandas as pd
train = pd.read_csv("labeledTrainData.tsv", header=0, \
                    delimiter="\t", quoting=3)
```

Here, "header=0" indicates that the first line of the file contains column names, "delimiter=\t" indicates that the fields are separated by tabs, and quoting=3 tells Python to ignore doubled quotes, otherwise you may encounter errors trying to read the file.

We can make sure that we read 25,000 rows and 3 columns as follows:

```
>>> train.shape
(25000, 3)

>>> train.columns.values
array([id, sentiment, review], dtype=object)
```

The three columns are called "id", "sentiment", and "array." Now that you've read the training set, take a look at a few reviews:

```
print train["review"][0]
```

As a reminder, this will show you the first movie review in the column named "review." You should see a review that starts like this:

```
"With all this stuff going down at the moment with MJ i've started listening to his
music, watching the odd documentary here and there, watched The Wiz and watched
Moonwalker again. Maybe i just want to get a certain insight into this guy who i
thought was really cool in the eighties just to maybe make up my mind whether he is
guilty or innocent. Moonwalker is part biography, part feature film which i remember
going to see at the cinema when it was originally released. Some of it has subtle
messages about MJ's feeling towards the press and also the obvious message of drugs are
bad m'kay. <br/><br/>..."
```

There are HTML tags such as "`<br/>`", abbreviations, punctuation - all common issues when processing text from online. Take some time to look through other reviews in the training set while you're at it - the next section will deal with how to tidy up the text for machine learning.

Data Cleaning and Text Preprocessing

Removing HTML Markup: The BeautifulSoup Package

First, we'll remove the HTML tags. For this purpose, we'll use the **Beautiful Soup** library. If you don't have Beautiful soup installed, do:

```
$ sudo pip install BeautifulSoup4
```

from the command line (NOT from within Python). Then, from within Python, load the package and use it to extract the text from a review:

```
# Import BeautifulSoup into your workspace
from bs4 import BeautifulSoup

# Initialize the BeautifulSoup object on a single movie review
example1 = BeautifulSoup(train["review"][0])

# Print the raw review and then the output of get_text(), for
# comparison
print train["review"][0]
print example1.get_text()
```

Calling `get_text()` gives you the text of the review, without tags or markup. If you browse the BeautifulSoup documentation, you'll see that it's a very powerful library - more powerful than we need for this dataset. However, it is not considered a reliable practice to remove markup using regular expressions, so even for an application as simple as this, it's usually best to use a package like BeautifulSoup.

Dealing with Punctuation, Numbers and Stopwords: NLTK and regular expressions

When considering how to clean the text, we should think about the data problem we are trying to solve. For many problems, it makes sense to remove punctuation. On the other hand, in this case, we are tackling a sentiment analysis problem, and it is possible that "!!!" or ":-(" could carry sentiment, and should be treated as words. In this tutorial, for simplicity, we remove the punctuation altogether, but it is something you can play with on your own.

Similarly, in this tutorial we will remove numbers, but there are other ways of dealing with them that make just as much sense. For example, we could treat them as words, or replace them all with a placeholder string such as "NUM".

To remove punctuation and numbers, we will use a package for dealing with regular expressions, called **`re`**. The package comes built-in with Python; no need to install anything. For a detailed description of how regular expressions work, see the package documentation. Now, try the following:

```
import re
# Use regular expressions to do a find-and-replace
letters_only = re.sub("[^a-zA-Z]",           # The pattern to search for
                      " ",                   # The pattern to replace it with
```

```
                      example1.get_text() )  # The text to search
print letters_only
```

A full overview of regular expressions is beyond the scope of this tutorial, but for now it is sufficient to know that [] indicates group membership and ^ means "not". In other words, the re.sub() statement above says, "Find anything that is NOT a lowercase letter (a-z) or an upper case letter (A-Z), and replace it with a space."

We'll also convert our reviews to lower case and split them into individual words (called "tokenization" in NLP lingo):

```
lower_case = letters_only.lower()        # Convert to lower case
words = lower_case.split()               # Split into words
```

Finally, we need to decide how to deal with frequently occurring words that don't carry much meaning. Such words are called "stop words"; in English they include words such as "a", "and", "is", and "the". Conveniently, there are Python packages that come with stop word lists built in. Let's import a stop word list from the Python **Natural Language Toolkit** (NLTK). You'll need to install the library if you don't already have it on your computer; you'll also need to install the data packages that come with it, as follows:

```
import nltk
nltk.download()  # Download text data sets, including stop words
```

Now we can use nltk to get a list of stop words:

```
from nltk.corpus import stopwords # Import the stop word list
print stopwords.words("english")
```

This will allow you to view the list of English-language stop words. To remove stop words from our movie review, do:

```
# Remove stop words from "words"
words = [w for w in words if not w in stopwords.words("english")]
print words
```

This looks at each word in our "words" list, and discards anything that is found in the list of stop words. After all of these steps, your review should now begin something like this:

```
[u'stuff', u'going', u'moment', u'mj', u've', u'started', u'listening', u'music',
u'watching', u'odd', u'documentary', u'watched', u'wiz', u'watched', u'moonwalker',
u'maybe', u'want', u'get', u'certain', u'insight', u'guy', u'thought', u'really',
u'cool', u'eighties', u'maybe', u'make', u'mind', u'whether', u'guilty', u'innocent',
u'moonwalker', u'part', u'biography', u'part', u'feature', u'film', u'remember',
u'going', u'see', u'cinema', u'originally', u'released', u'subtle', u'messages', u'mj',


u'feeling', u'towards', u'press', u'also', u'obvious', u'message', u'drugs', u'bad',
u'm', u'kay',.....]
```

Don't worry about the "u" before each word; it just indicates that Python is internally representing each word as a unicode string.

There are many other things we could do to the data - For example, Porter Stemming and Lemmatizing (both available in NLTK) would allow us to treat "messages", "message", and "messaging" as the same word, which could certainly be useful. However, for simplicity, the tutorial will stop here.

Putting it all together

Now we have code to clean one review - but we need to clean 25,000 training reviews! To make our code reusable, let's create a function that can be called many times:

```
def review_to_words( raw_review ):
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove HTML
    review_text = BeautifulSoup(raw_review).get_text()
    #
    # 2. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    #
    # 3. Convert to lower case, split into individual words
```

```
    words = letters_only.lower().split()
    #
    # 4. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 5. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 6. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))
```

Two elements here are new: First, we converted the stop word list to a different data type, a set. This is for speed; since we'll be calling this function tens of thousands of times, it needs to be fast, and searching sets in Python is much faster than searching lists.

Second, we joined the words back into one paragraph. This is to make the output easier to use in our Bag of Words, below. After defining the above function, if you call the function for a single review:

```
clean_review = review_to_words( train["review"][0] )
print clean_review
```

it should give you exactly the same output as all of the individual steps we did in preceding tutorial sections. Now let's loop through and clean all of the training set at once (this might take a few minutes depending on your computer):

```
# Get the number of reviews based on the dataframe column size
num_reviews = train["review"].size

# Initialize an empty list to hold the clean reviews
clean_train_reviews = []

# Loop over each review; create an index i that goes from 0 to the length
# of the movie review list
for i in xrange( 0, num_reviews ):
    # Call our function for each one, and add the result to the list of
    # clean reviews
    clean_train_reviews.append( review_to_words( train["review"][i] ) )
```

Sometimes it can be annoying to wait for a lengthy piece of code to run. It can be helpful to write code so that it gives status updates. To have Python print a status update after every 1000 reviews that it processes, try adding a line or two to the code above:

```
print "Cleaning and parsing the training set movie reviews...\n"
clean_train_reviews = []
for i in xrange( 0, num_reviews ):
    # If the index is evenly divisible by 1000, print a message
    if( (i+1)%1000 == 0 ):
        print "Review %d of %d\n" % ( i+1, num_reviews )
    clean_train_reviews.append( review_to_words( train["review"][i] ))
```

Creating Features from a Bag of Words (Using scikit-learn)

Now that we have our training reviews tidied up, how do we convert them to some kind of numeric representation for machine learning? One common approach is called a Bag of Words. The Bag of Words model learns a vocabulary from all of the documents, then models each document by counting the number of times each word appears. For example, consider the following two sentences:

Sentence 1: "The cat sat on the hat"

Sentence 2: "The dog ate the cat and the hat"

From these two sentences, our vocabulary is as follows:

{ the, cat, sat, on, hat, dog, ate, and }

To get our bags of words, we count the number of times each word occurs in each sentence. In Sentence 1, "the" appears twice, and "cat", "sat", "on", and "hat" each appear once, so the feature vector for Sentence 1 is:

{ the, cat, sat, on, hat, dog, ate, and }

Sentence 1: { 2, 1, 1, 1, 1, 0, 0, 0 }

Similarly, the features for Sentence 2 are: { 3, 1, 0, 0, 1, 1, 1, 1}

In the IMDB data, we have a very large number of reviews, which will give us a large vocabulary. To limit the size of the feature vectors, we should choose some maximum vocabulary size. Below, we use the 5000 most frequent words (remembering that stop words have already been removed).

We'll be using the feature_extraction module from **scikit-learn** to create bag-of-words features. If you did the Random Forest tutorial in the Titanic competition, you should already have scikit-learn installed; otherwise you will need to install it.

```
print "Creating the bag of words...\n"
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the "CountVectorizer" object, which is scikit-learn's
# bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = None,   \
                             max_features = 5000)

# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of
# strings.
train_data_features = vectorizer.fit_transform(clean_train_reviews)

# Numpy arrays are easy to work with, so convert the result to an
# array
train_data_features = train_data_features.toarray()
```

To see what the training data array now looks like, do:

```
>>> print train_data_features.shape
(25000, 5000)
```

It has 25,000 rows and 5,000 features (one for each vocabulary word).

Note that CountVectorizer comes with its own options to automatically do preprocessing, tokenization, and stop word removal -- for each of these, instead of specifying "None", we could have used a built-in method or specified our own function to use. See the function documentation for more details. However, we wanted to write our own function for data cleaning in this tutorial to show you how it's done step by step.

Now that the Bag of Words model is trained, let's look at the vocabulary:

```
# Take a look at the words in the vocabulary
vocab = vectorizer.get_feature_names()
print vocab
```

If you're interested, you can also print the counts of each word in the vocabulary:

```
import numpy as np

# Sum up the counts of each vocabulary word
dist = np.sum(train_data_features, axis=0)

# For each, print the vocabulary word and the number of times it
# appears in the training set
for tag, count in zip(vocab, dist):
    print count, tag
```

Random Forest

At this point, we have numeric training features from the Bag of Words and the original sentiment labels for each feature vector, so let's do some supervised learning! Here, we'll use the Random Forest classifier that we introduced in the Titanic tutorial. The Random Forest algorithm is included in scikit-learn (Random Forest uses many tree-based classifiers to make predictions, hence the "forest"). Below, we set the number of trees to 100 as a reasonable default value. More trees may (or may not) perform better, but will certainly take longer to run. Likewise, the more features you include for each review, the longer this will take.

```
print "Training the random forest..."
```

```
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators = 100)

# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
#
# This may take a few minutes to run
forest = forest.fit( train_data_features, train["sentiment"] )
```

Creating a Submission

All that remains is to run the trained Random Forest on our test set and create a submission file. If you haven't already done so, download **testData.tsv** from the Data page. This file contains another 25,000 reviews and ids; our task is to predict the sentiment label.

Note that when we use the Bag of Words for the test set, we only call "transform", not "fit_transform" as we did for the training set. In machine learning, you shouldn't use the test set to fit your model, otherwise you run the risk of overfitting. For this reason, we keep the test set off-limits until we are ready to make predictions.

```
# Read the test data
test = pd.read_csv("testData.tsv", header=0, delimiter="\t", \
                   quoting=3 )

# Verify that there are 25,000 rows and 2 columns
print test.shape

# Create an empty list and append the clean reviews one by one
num_reviews = len(test["review"])
clean_test_reviews = []

print "Cleaning and parsing the test set movie reviews...\n"
for i in xrange(0,num_reviews):
    if( (i+1) % 1000 == 0 ):
        print "Review %d of %d\n" % (i+1, num_reviews)
    clean_review = review_to_words( test["review"][i] )
    clean_test_reviews.append( clean_review )

# Get a bag of words for the test set, and convert to a numpy array
test_data_features = vectorizer.transform(clean_test_reviews)
test_data_features = test_data_features.toarray()

# Use the random forest to make sentiment label predictions
result = forest.predict(test_data_features)

# Copy the results to a pandas dataframe with an "id" column and
# a "sentiment" column
output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )

# Use pandas to write the comma-separated output file
output.to_csv( "Bag_of_Words_model.csv", index=False, quoting=3 )
```

Congratulations, you are ready to make your first submission! Try different things and see how your results change. You can clean the reviews differently, choose a different number of vocabulary words for the Bag of Words representation, try Porter Stemming, a different classifier, or any number of other things. To try out your NLP chops on a different data set, you can also head over to our Rotten Tomatoes competition. Or, if you're ready for something completely different, move along to the Deep Learning and Word Vector pages.