Text classification using python (without priors.. and without tears… hopefully)

Abeed Sarker (v 1)

-3 Objectives

(i) To learn basic python techniques such as loading files and reading from them (for text classification)
(ii) Basic preprocessing of text using python
(iii) Using sklearn to perform very simple text classification in python

-2 Prerequisites

(i) Understand what text classification is and its applications (most, if not all, of us have come across text classification at some point!
(ii) Have python installed with the libraries (more on this later)

-1 Expected outcomes

This will be a quick run through of the techniques needed for performing basic text classification with close to zero prior knowledge. **The expected outcome is that you will be able to use the template I provided to perform other text classification tasks.** Once you learn to use the techniques, you can apply many algorithms and solve many problems Plus, you can add to the script various other preprocessing/feature extraction tasks… but let's not talk about it now.

0 Technical Prerequisites (installation)

Recommended Installation

Everything is included in the Anaconda Python distribution. So, the best thing to do is to install anaconda **http://continuum.io/downloads** (if you have previous python installations, it may be useful to uninstall them before installing anaconda).

If you want to install manually (not recommended for beginners… because of the dependencies):

(i) Python (2.x.x) - DON'T INSTALL PYTHON 3.x.x!!: https://www.python.org/downloads/
(ii) The scikit-learn package (sklearn): http://scikit-learn.org/stable/install.html
(iii) The dependencies: scipy, numpy etc.

Note: There are many python libraries for machine learning and NLP. We are trying to take a minimalist approach.

Henceforth, I'll assume that you have installed anaconda.

To test if python is working, (i) launch anaconda, (ii) once anaconda opens, launch Spyder.

You need to create a workspace (where all your projects will be stored).

Do:

File-> new project

Then, follow the prompts to create a workspace (it's only asked the first time). Then create a new project. Let's call the project **TextClassification.** The project will appear in the left panel. Right-click on the project, and **choose New-> Module**

Let's call this module (rather thoughtlessly) **Main**

Now type:

print 'hello world' and click on the green play button at the top to run your module.

——— all this while, we have only been setting things up to run our text classification approach. **If you are comfortable with running and using python, don't bother about these**. Also, you can use your own IDEs, editors etc. (I use PyDev + eclipse now.. and I will be running the workshop on that IDE… but these details are uninteresting).

1 Introduction

Automatic text classification has useful applications in many tasks. Knowing how to quickly run some classification algorithms can be a handy capability. In applied domains, such as biomedical informatics, researchers generally don't have the time to dive deep into machine learning algorithms. As such, many refrain from using these strong tools despite their promise. The intent of this tutorial is to help researchers who have been exposed to the idea of text classification, but have never done it themselves.

2 The problem

We will use part of our Twitter Adverse Drug Reaction (ADR) data for this tutorial.[1] This is an internal data set, and is not available outside the organization. So DON'T share the data without following the standard Twitter data sharing policies. **Downloadable version of a large chunk of the data is available at** diego.asu.edu/downloads/.

The data consists of 4 columns:

ID	a unique ID for the instance (tweet)
Tweet	the text of the tweet
Drug	the drug associated with the tweet
Class	1- the tweet mentions an ADR, 0 - the tweet does not contain ADR.

The columns are tab separated. Each row represents an instance. We will use a very small set of N tweets.[2]

[1] You can use any other data if you want. If this is your first time, it may be simpler if your data file is formatted the same way.

[2] I have artificially balanced the data to create the (false) impression that our learning algorithms are performing well.

Filename: sample_adrs.txt   downloadable from-> diego.asu.edu/abeedtest/

Given these tweets, we want to train a system to automatically identify if a tweet mentions an ADR or not (when dealing with millions of tweets for analysis, it can be very important to get rid of the noise).

3 Loading and preprocessing

We will not attempt to learn python in this tutorial. **Instead, we will see python + sklearn as a tool which we can configure to do some learning. We will build this tool ourselves… by copy-pasting code.**

PLEASE REMEMBER: while we are taking such a simplistic approach to text classification, it is essential to have a better understanding of these techniques before applying them to scientific studies.

First let's copy-paste some `imports`:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
```

These provide us with the functionality we need to learn.

Next we load the file. Place the file somewhere on your hard drive and note the path. For example, on windows, you can place it in your C drive and the path will be- C:\\
I placed it in the documents folder on my mac, so the path is: /Users/abeedsarker/Documents

Now let's load the file.

```
data = pd.read_csv('/Users/abeedsarker/Documents/
ADR_tutorial_data.tsv', header=0, delimiter="\t", quoting=3)
```

And we are done loading it! Now, the **data** variable holds all the information. We can try printing different instances, from zero to N-1. Try the following for example..

```
print data["ID"][4]
print data["Tweet"][10]
```

It should print something like this:

```
lamotrigine-518b9d7aac6ab35b4d4803c2
At the Cypriot art and archaeology exhibition at the @XXXX. Love the
Cypro-Archaic bird jugs! Never seen anything like them.
```

Preprocessing
We will skip the crucial step of preprocessing for now and revisit it later. Let's see what we can learn with the texts we loaded so far.

4 Classification

Now that we have loaded the data and seen that it is actually loaded, let's run some classification!

We will do it in as little as 4 steps!

(i) First we divide the data into train and test sets. We are using 80% for training, 20% for testing. len(data) gives the size of the full set.

```
bound = int(0.8*len(data))
train = data[:bound]
test = data [bound:]
print len(train)
print len(test)
```

(ii) (a)Then we take out the training tweets and the classes, and (b) create a vectorizer (which converts the text in the tweets into feature vectors that our training algorithm can understand). (c) we transform the tweets into feature array.

```
#a
training_tweets = train["Tweet"]
training_classes = train["Class"]
#b
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = None,   \
                             max_features = 5000)
#c
train_data_features = vectorizer.fit_transform(training_tweets)
train_data_features = train_data_features.toarray()
```

(iii) Initialize our learning algorithm (SVMs) and then use the feature array (and the set of classes) to train it.

```
svms = svm.SVC()
svms = svms.fit( train_data_features, training_classes)
```

(iv) Now, our algorithm has trained itself to figure out if a tweet contains an ADR or not. So now we can test its performance on the test set. (a) prepare the test data in the same way as the training data. (b) run the predict() function to get predictions and then check the accuracy.

```
#a
testing_tweets = test["Tweet"]
test_data_features = vectorizer.transform(testing_tweets)
test_data_features = test_data_features.toarray()
#b
print "Predicting test labels...\n"
result = svms.predict(test_data_features)
print ".... and the Accuracy is: "
print accuracy_score(test["Class"],result)
```

And that's it! Our relatively unintelligent machine is ready for the real world (#notreally).

Output should be something like this:

```
Training the SVMs (this may take a while)...
Predicting test labels...

.... and the Accuracy is:
0.808823529412
```

So it seems that our learning algorithm is good.. with around 80% accuracy.[3]

5 Preprocessing revisited
We skipped the important aspect of preprocessing. Preprocessing is crucial, because it prepares the data that will be used to train the machine. It involves steps such as stemming, lowercasing, stopword removal, url removal etc. depending on the problem. It is expected that you'll have some idea of the data (and hence what to preprocess) when you do some classification.

[3] Actually the performance of this system is very bad. But if you want to understand why, you need to have a deeper knowledge of the problem domain.

We will again, take a simplistic approach. Ideally, we want the tweet text to go through some function which will preprocess them and get them ready! So we'll write a function called preprocess().

```
def preprocess(text):
  text = text.lower()
  return text
```

Let's say we want to use a simple preprocessing function to convert all the strings to lowercase.

We can now plug in this function before passing the tweets for learning. REMEMBER, the train and test tweets must be preprocessed the same way! We just add the code to our previous code:

```
training_tweets = train["Tweet"]
processed_training_tweets = []
for t in training_tweets:
    t = preprocess(t)
    processed_training_tweets.append(t)
```

Now, we train using the processed_training_tweets instead of the training_tweets.

That's it!

File Location
The python file and the text file are here:
diego.asu.edu/abeedtest/

Further Readings
The intent was to get everyone started. There are many tools available. Here are some recommended sources:

NLTK book: A fantastic resource for NLP using python
http://www.nltk.org/book/

Kaggle Tutorials: Kaggle is one of the most innovative data mining websites. They have various tutorials available (and scripts.. and I copy-pasted some of the code from Angela Chapman's tutorial). They can be found at:

https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words

If you're just getting started with python, here's a quickstart:
https://www.kaggle.com/c/titanic/details/getting-started-with-python
https://www.kaggle.com/c/titanic/details/getting-started-with-python-ii

Acknowledgements and inspiration

There are many 'without tears' tutorials. I had a particularly hard (and tearful) introduction to bayesian inference. Kevin Knight seemed to understand:
http://www.isi.edu/natural-language/people/bayes-with-tears.pdf

I found the tutorial by Angela Chapman (Kaggle) particularly useful. As one would expect, I copiedpasted some of the code. Her work resides here:
https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words