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

Abeed Sarker (v2)
Thoughts and comments to: abeed@pennmedicine.upenn.edu

Last update: May, 2018

Note: v1 of this workshop was first conducted at Arizona State University in 2015, and is available here: (https://sarkerabeed.files.wordpress.com/2018/05/text_classification_workshop-v1.pdf). v2 was first conducted at the University of Pennsylvania in 2016, using the ADR dataset [1].

-3 Objectives

(i) Python techniques such as loading files and reading from them (for text classification)
(ii) Basic preprocessing of text using python (using `nltk`)
(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 methods 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 accompanying these notes to perform other text classification tasks. Once you learn to use the basic techniques, you can apply many different classification algorithms. You can also add to the script various other preprocessing/feature extraction code.

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 beginner because of the dependencies):
(i) Python (2.x.x) – don't install python 3.x.x for this tutorial!: 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 `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 PyCharm these days 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. Therefore, 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 in-house Twitter Birth Defect Dataset for this tutorial.[1] This is an internal data set, and is not available outside the organization. We are only using 1000 instances from this dataset.

The data consists of 5 columns:

User ID	a unique ID for the user
Tweet	the text of the tweet
Tweet ID	a unique ID for the tweet
Timestamp	the time and date for the tweet
Class	1- the tweet indicates a birth defect in the user's child, 2 - the tweet indicates a possible birth defect in the user's child, 3 – the tweet just mentions a birth defect.

[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.

The columns are comma separated (csv format). Each row represents an instance.

Filename: `birth_defects_sample.csv`

Given the texts in this file, we want to train a system that can perform automatic classification. Training the system will enable us to run it on millions of unlabeled instances.

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 we need to copy-paste the imports.[2]

```
import csv
from nltk.stem.porter import *
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn.metrics import accuracy_score
from nltk.corpus import stopwords


stemmer = PorterStemmer()
st =  stopwords.words('english')
```

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 inside the project folder, so I can access it as: `./birth_defects_sample.csv`

We load the file using the `csv` library inside '__main__'

```
if __name__=='__main__':
    #to hold the texts and the classes.
    texts = []
    classes = []

    with open('birth_defects_sample.csv') as csvfile:
        birthdefectfilereader = csv.reader(csvfile,delimiter=',')
        for row in birthdefectfilereader:
            userid = row[0]
```

[2] In v1 of this workshop, I used pandas for loading the csv. Pandas is a great library. This is just another way of loading and working with the data.

```
            tweettext = preprocess_text(row[1])
            tweetid = row[2]
            datetime = '\"'+row[3]+'\"'
            class_ = row[-1]

            texts.append(tweettext)
            classes.append(class_)
```

- We are saving all the texts in the `texts` list and all the corresponding classes in the `classes` list.
- We can (and perhaps should) add print statements within the loop to check that the texts are loading fine.
- For this tutorial, we will not worry about the other columns in the file.

Preprocessing

The code, as shown above, will show an error. This is because we are trying to pass the text to a function called `preprocess_text()`,which we have not written yet. The following is a possible (simple) preprocessing function, that allows us to remove non alpha characters, lowercase text, stem and remove stopwords. Of course we can add many more preprocessing methods.

```
def preprocess_text(tweet_text):
    '''
    :param tweet_text: the tweet to preprocess
    :return: a string containing the preprocessed text
    '''
    # Remove non-letters?
    tweet_text = re.sub("[^a-zA-Z]", " ", tweet_text)

    #lowercase?
    tweet_text = tweet_text.lower()

    #stem?
    tweet_text = ' '.join([stemmer.stem(w) for w in tweet_text.split()])

    # remove stopwords?
    tweet_text = ' '.join([w for w in tweet_text.split() if not w in st])


    return tweet_text
```

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.

```
#(i)
```

```
bound = int(0.8 * len(texts))
train_texts = texts[:bound]
test_texts = texts[bound:]

train_classes = classes[:bound]
test_classes = classes[bound:]
```

(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.

```
# (ii)
# a
train_classes = classes[:bound]
test_classes = classes[bound:]
# b
vectorizer = CountVectorizer(analyzer="word", tokenizer=None, preprocessor=None,
stop_words=None, max_features=5000)
#c
train_data_features = vectorizer.fit_transform(train_texts)
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, train_classes)
```

(iv) Now our algorithm has trained on the annotated data. 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 the predictions and then check the accuracy using `accuracy_score()`

And that's it! Our relatively unintelligent machine is ready for the real world (#notreally). Output should be something like this:

```
.... and the Accuracy is:
0.865
```

5 Further tasks

During the workshop, we will look at two more things:

(i) How to slightly optimize the SVM
(ii) Other possible evaluation metrics
(iii) Adding features

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

Feeling confident?
If you want to add more features and know more about what features have worked well for similar datasets, please have a look at our past papers on adverse drug reaction classification [1], drug abuse classification [2], or sentiment analysis [3].

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

References

[1] A. Sarker and G. Gonzalez, "Portable automatic text classification for adverse drug reaction detection via multi-corpus training.," *J. Biomed. Inform.*, vol. 53, pp. 196–207, Nov. 2014.

[2] A. Sarker *et al.*, "Social Media Mining for Toxicovigilance: Automatic Monitoring of Prescription Medication Abuse from Twitter," *Drug Saf.*, vol. 39, no. 3, pp. 231–40, Jan. 2016.

[3] A. Sarker and G. Gonzalez, "HLP@UPenn at SemEval-2017 Task 4A: A simple, self-optimizing text classification system combining dense and sparse vectors," in *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, 2017, pp. 640–643.