Sentiment Analysis: Python and Scikit-learn

If you have python installed on your OS and you have not installed scikit-learn then follow this link: http://scikit-learn.org/stable/install.html

If you haven't got an installation of python there are a few options:

- Create a beginner account here: https://www.pythonanywhere.com/pricing/ and you can upload and run python files in a selected console.
- Use a distribution https://www.enthought.com/products/canopy/ this is free and works well with a package manager but you can choose whatever you like. If you can install it reasonably quickly.
- Install python and the dependencies you need for machine learning as you go.

If you've never used python before don't worry, it's easy to learn.

The first thing to do is get our data set, it's been split into three: Training data, testing data and validation data. Each are a CSV file with two columns Polarity (positive and negative) and Text. We're using the (Pang and Lee, 2004) IMDB movie review dataset. This is just an initial dataset, quite small, to make this process a bit easier. You can get the dataset here: http://arg.tech/SentimentTutorial or it's on my USB stick.

The first thing we need to do is read our dataset into a data structure. Pandas in python is perfect for this we can read straight from CSV files into a data structure and it also works well with sklearn.

We need to import pandas into python and read from our csv file:

```
import pandas as pd     // pd is the name we have given to our import

trainData = pd.read_csv("FILEPATH/.csv", header=0, delimiter=",", encoding='utf-8')
```

We don't need to declare variable names in this case 'trainData', we use the pandas import to read the CSV file. Header is set to 0 as the headers on our datasets are on the first line, delimiter is a comma as this is what makes up CSV files and encoding is set to utf-8 to make sure all characters are represented from the text. **Read in the test set too**.

Our text is already all lowercase with punctuation removed. The easiest form of machine learning is the bag of words (e.g. the cat sat on the mat = 0 - 2, 1 - 1, 2 - 1, 3 - 1, 4 - 1) approach so we'll use that.

In sklearn we have the CountVectorizer module that can do this for us.

```
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB
```

What will also make our life easier is using a pipeline in sklearn. This just means we can extract features, determine our classifier and then train our classifier in two steps rather than more and it makes it easier to perform the same feature extraction on our test set. We also need to select our classifier, there are lots to choose from in sklearn. We'll start with Naïve Bayes as this is good for text tasks.

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([

        ('BagOfWords', CountVectorizer()),

        ('classifier', MultinomialNB()) ])
```

The remaining steps are to train the classifier using our training data creating a model and then use this model to predict our test data labels and finally evaluate our scores.

```
model = pipeline.fit(trainData["Text"], trainData["Polarity"]) // train the classifier
predictions = model.predict(testData["Text"]) // predict test set
```

We also need to set our target labels.

```
target_names = ['0', '1']
```

Finally, we import our confusion matrix to show our predictions and then use the accuracy to evaluate our model.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
print(confusion_matrix(testData["Polarity"], predictions))
print(accuracy_score(testData["Polarity"], predictions))
```

Some other things to try (use the sklearn documentation):

- different classifiers (Decision Trees, SVM, Logistical Regression, Random forest)
- using n-grams or removing stopwords in CountVectorizer
- replacing count vectorizer with TF-IDF vectorizer – in basic terms it is the opposite.
- playing with the classifier parameters e.g. different kernels for classifiers and values
- use NLTK to tag words with POS or lemmatize words
- adding your own features (this is more difficult see below)

Creating your own features requires using a feature union within the pipeline as well as creating a class which grabs your column of interest from the pandas dataframe (ItemSelector). Then we can create a class which does something with this data i.e. does the length of text indicate the sentiment. We also need to edit the way we train and test our data as we no longer need to specify the column we are training on.

```
from sklearn.base import TransformerMixin, BaseEstimator
class ItemSelector(BaseEstimator, TransformerMixin):
    def __init__(self, key):
        self.key = key
    def fit(self, x, y=None):
        return self
    def transform(self, data_dict)
        return data_dict[self.key]
```

```
class LengthStats(BaseEstimator, TransformerMixin): // this is our length of text class
    def fit(self, x, y=None):
        return self
    def transform(self, text):
        return [{'length': len(t)} for t in text]
```

```
pipeline = Pipeline([
    ('features', FeatureUnion([
        ('BagOfWords', Pipeline([
            ('selector', ItemSelector(key="Text")),
            ('vectorizer', CountVectorizer()),
        ])),
        ('length_stats', Pipeline([
            ('selector', ItemSelector(key="Text")),
            ('stats', LengthStats()),  # returns a list of dicts
            ('vector', DictVectorizer()),  # list of dicts -> feature matrix
        ])),
    ],
    )),
    ('classifier', MultinomialNB()) ])
```

```
model = pipeline.fit(trainData, trainData["Polarity"]) // train the classifier
predictions = model.predict(testData) // predict test set
```

If you have exhausted everything and want a much more challenging task I have another dataset you can download from here: http://arg.tech/EthosDataset or it is on my USB stick. This is a basic ethos dataset (attacks and supports on people or entities) with raw data so all pre-processing must be done by you.