# Department of Computing

**CS471: Machine Learning**

**Class: BESE-7AB**

# Lab 09: Sentiment Analysis (Part 2)

**Date: 05-04-2019**

**Time: 10:00 am– 1:00 pm & 2:00 pm-5:00 pm**

**Instructor: Dr. Pakeeza Akram**

# Lab 6: Sentiment Analysis (Part 2)

**Introduction**

Sentiment Analysis, also known as opinion mining refers to the use of natural language processing, text analysis to identify and extract subjective information in source materials. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. (Wikipedia)

**Objective**

In this lab you will use cleaned IMDB reviews from lab 8 to train a classifier. Each movie review is labeled with a positive or a negative sentiment. This is your training dataset. You will use cleaned reviews with labels to train a Naïve Bayes’ classifier. The trained classifier when presented with a new review will predict if the review is positive or negative.

**Tools/Software Requirement**

Python, scikit-learn

Reference:

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

**Description**

First we are going to implement an easy task so that you know what is going on. Lets say you want to classify emails as spam and non-spam using machine learning.

The first thing is to obtain training data and training labels. In this case, we are going to start with just 8 emails and their labels. A label of 0 means that the corresponding email is non-spam and a label of 1 means spam. Create the training data in python as follows.

X = ['offer secret', 'click secret link', 'secret sports link', 'play sports today', 'went play sports', 'secret sports event', 'sports today', 'sports costs money']

Y = [1,1,1,0,0,0,0,0] # review labels. 1 indicate spam, 0 non-spam

The training data is already cleaned, so we don’t need to clean them as we did in the last lab. However, we must convert them to some kind of numeric representation for machine learning? One common approach is called [Bag of Words](http://en.wikipedia.org/wiki/Bag-of-words_model) (BoW).The Bag of Words model learns a vocabulary from the training data (in this case from the list X), and 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. The vocabulary and the BoW representation for sentence 1 and 2 are given below.

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

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

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

In python, we'll be using the feature\_extraction module from scikit-learn to create bag-of-words features. Install scikit-learn, if its not installed. Scikit-learn is a python machine learning package.

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 = 100)

# 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.

X = vectorizer.fit\_transform(X)

# Numpy arrays are easy to work with, so convert the result to an

# array

X = X.toarray()

To see how training data now looks like:

print X.shape (8, 11)

>>>(8, 11)

Each review has 11 features, one for each vocabulary word (each feature counts the number of times the vocabulary word appears in the review).

The vocabulary words are:

print vectorizer.vocabulary\_

>>> {u'click': 0, u'costs': 1, u'event': 2, u'link': 3, u'money': 4, u'offer': 5, u'play': 6, u'secret': 7, u'sports': 8, u'today': 9, u'went': 10}

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. However, we wanted to write our own function for data cleaning in the previous lab so that you know how it is done.

Now we are going to train the Naïve Bayes’ classifier using our training data. You will learn the details of it in next week.

<http://scikit-learn.org/stable/modules/naive_bayes.html>

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB(alpha=0.000001) # alpha=0 means no laplace smoothing

clf.fit(X, np.array(Y))

The alpha parameter specifies the amount of Laplace smoothing. If alpha is 0, then there is no smoothing and maximum likelihood estimators are used. (Note: use very small number instead of 0 to simulate no smoothing as setting it to 0 may give you –inf). Play with different values of alpha to understand the effect.

Now we test the classifier on test data.

test\_reviews = ['sports', 'secret secret', 'today secret']

# bag of word representation

tX = vectorizer.transform(test\_reviews).toarray()

# prediction

print(clf.predict(tX))

We can also compute the probabilities:

print clf.predict\_proba(tX)

>>>[[ 8.26446281e-01 1.73553719e-01] [ 5.70206700e-02 9.42979330e-01] [ 1.00000000e+00 2.75625000e-10]]

**Lab Task**

In the previous section, you learnt how to train a Naïve Bayes classifier. Your lab task is to train a Naïve Bayes’ classifier using IMDB movie reviews (25000 movie reviews). Remember to clean the reviews before building bag of word representation. You can either use the function you wrote in the last lab to clean the reviews or use review\_to\_words.py.

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. You can start with 5000 most frequent words

vectorizer = CountVectorizer(analyzer = "word", \

tokenizer = None, \

preprocessor = None, \

stop\_words = None, \

max\_features = **5000**)

Partition the 25000 movie reviews into training (80%) and validation set (20%). Train on 80% reviews and then report accuracy on the remaining 20%.

Fill the following table for different values of vocabulary size and Laplace smoothing.

|  |  |  |
| --- | --- | --- |
| Vocabulary size | Alpha (laplace smoothing) | Accuracy on 20% validation set |
| 3000 | 0.00001 |  |
| 3000 | 5 |  |
| 5000 | 0.00001 |  |
| 5000 | 5 |  |

**Deliverables**

Upload the jupyter notebook file with your name along with Word file containing the task.