# Department of Computing

# CS471: Machine Learning

# Class: BSCS-6C

# Lab 11: Sentiment Analysis (Part 2)

# Date: 5 April 2019

# Time: 10:00-1:00

# Instructor: Dr. Omar Arif

# Lab 6: Sentiment Analysis (Part 2)

**Name:** Ahmad Amjad Mughal

**Reg No:** 121672

**Class:** BSCS-6C

**Description**

First we are going to implement assignment 2 in python.

Start with training data and training labels.

reviews = ['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 reviews are 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 a [Bag of Words](http://en.wikipedia.org/wiki/Bag-of-words_model). 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}

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(reviews)

# 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 this tutorial to show you how it's done step by step.

Now we are going to train the Naïve Bayes’ classifier using our training data.

<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).

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]]

The preceding code is contained in the file assignment2.py

**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 | 83.81 |
| 3000 | 5 | 83.76 |
| 5000 | 0.00001 | 84.84 |
| 5000 | 5 | 84.84 |

**Deliverables:** Upload Word file containing the task

**Time**: End of lab.