Lecture 18: Text Classification (Mitchell Chapter 6, Khemani Chapter 16)

CS 167: Machine Learning

Project 5

Text Classification Project: Due Monday, December 12th (during finals week, but we don't have a final exam)

- find some data
- prep/clean it
- do machine learning (including Naive Bayes)
- explain everything in write-up

IMDB Data Set

Problem: *Sentiment Analysis*, predict whether text has positive or negative sentiment

Exercise: get data set, open it up in Excel (or something) to see what it looks like

data originates here:
https://www.kaggle.com/c/word2vec-nlp-tutorial/data

Import Statements

These are the import statements you'll need. Make sure you have all these packages.

```
import pandas
from bs4 import BeautifulSoup
import re
import nltk
#only do next line once
nltk.download() #download everythin except panlex_lite
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
```

Stripping out the html

make sure to read it in as a tab-delimited file

```
data = pandas.read_csv("imdb_reviews.tsv", delimiter="\t")

#print out the first review
print(data["review"][0])

#us the Beautiful Soup package to remove html mark up
rev_soup = BeautifulSoup(data["review"][0])
print(rev_soup.get_text())
```

Beautiful Soup documentation:
http://www.crummy.com/software/BeautifulSoup/bs4/doc/

Getting rid of spaces and punctuation

Use the regular expressions library to remove anything that isn't a letter.

```
letters_only = re.sub("[^a-zA-Z]"," ",rev_soup.get_text())
print(letters_only)
```

- `sub` means substitute
- `[^somechars]` means match anything that isn't one of the *somechars*

Discuss: Is there any useful punctuation?

Dealing with stop words

Use `split` to tokenize

```
lower_case = letters_only.lower()
words = lower_case.split()
print(words)
```

Stop Words: words without much meaning

```
#print out the nltk stop words list
print(stopwords.words("english"))
```

Programming Exercise: remove all the stop words from the `words` list

Then, convert them back into one string:

```
clean_text = " ".join(words)
print(clean_text)
```


Exercise

We just cleaned one review, but we need to do that for all the reviews.

To save time, do it now for the *first 1000 reviews* (if time, do more)

```
first1000 = data["review"][0:1000]
```

Do the following:

1. write a function that will take in a review and do everything we just did for the first review
2. loop through all of the reviews and call the function with that review
3. append each cleaned review to a running list of all the reviews called `cleaned_reviews`

Creating our attributes: Bag of Words

Bag of Words

- choose vocabulary (say 5000 most common words)
- one column for each word
- row contains counts for each word

Example

Sentence 1: "The cat sat on the hat"
Sentence 2: "The dog ate the cat and the hat"
Vocabulary: { the, cat, sat, on, hat, dog, ate, and }

	the	cat	sat	on	hat	dog	ate	and
Sentence 1	2	1	1	1	1	0	0	0
Sentence 2	3	1	0	0	1	1	1	1

Bag of Words in scikit-learn

```
#Bag of Words with 5000 most common words
vectorizer = CountVectorizer(analyzer='word', \
     max_features = 5000)

word_columns = vectorizer.fit_transform(cleaned_reviews)

#convert to numpy array so we can feed it
#into learning algorithm
word_columns = word_columns.toarray()
print(vectorizer.get_feature_names())
print(word_columns)
```


Feed it into Naive Bayes

```
(train_data, test_data, train_target, test_target) = \
    cv.train_test_split(word_columns, data["sentiment"],\
    test_size = 0.2)

mnb = MultinomialNB()
mnb.fit(train_data,train_target)
preds = mnb.predict(test_data)
print accuracy_score(preds,test_target)
```

Exercise: How does this compare to performance with a support vector machine?
Exercise: How about PCA then SVM?
Exercise: Does PCA improve Naive Bayes at all?

Persistent scikit-learn objects

Training takes a long time. What if I fit a learning algorithm and then want to load it later without having to retrain it?

```
from sklearn.externals import joblib
joblib.dump(mnb, 'imdb_naive_bayes_classifier.pkl')
joblib.dump(vectorizer, 'imdb_vectorizer.pkl')
```

Then, to load it back later

```
mnb = joblib.load('imdb_naive_bayes_classifier.pkl')
vectorizer = joblib.load('imdb_vectorizer.pkl')
```

Exercise

Write a program that can make sentiment predictions on brand new text that you give it (i.e., not from the test set).