Open in Colab

More Naïve Bayes with Smoothing and N-Grams

Abstract

In this notebook you'll be explanding on our previous implementation of the Naïve Bayes algorithm and exploring the fun new world of bigrams which are "pretty useful". We'll practice some techniques for manipulating text and take advantage of some of sklearn's built-in implementations. This notebook will help you practice some tools to use for a sequence learning mini-project.

Takeaways

- When we use Naïve Bayes, it's beneficial to use "smoothing" to avoid the problems that arise because our model can run into problems (probabilities of 0, or values of negative infinity) when a word shows up in our test set that was not in our training set.
- In English (and most other languages), word order matters ("people eat" is different than "eat people"). Therefore, it's often useful to included some information about word order in our model. If we include complete information about word order (e.g., this word showed up 100 words before that word), our model would require extreme amounts of memory. We can typically get a way with a little information about word order, such as with bigrams, which are just pairs of words.
- We can use bigrams (or other ngrams) in various applications in natural language processing including text classification, sequence prediction, word generation, spell checking. These applications also extend beyond actual words or letters, and these techniques can really be applied to any sequences of items (e.g. series of coin flips).

Sentiment Analysis with Naïve Bayes (now with smoothing)

Building on the last assignment, we'll be focusing on predicting the sentiment of a movie review from IMDB based on the text of the movie review. This dataset is one that was originally used in a Kaggle competition called Bag of Words meets Bag of Popcorn.

Again, the data consist of the following.

> The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1. No individual movie has more than 30 reviews. The 25,000 review labeled training set does not include any of the same movies as the 25,000 review test set. In addition, there are another 50,000 IMDB reviews provided without any rating labels.

Our goal will be to see if we can learn a model, using Naïve Bayes on a training set to accurately estimate sentiment of new reviews.

In [0]:

```
# Import our movie review data (from last assignment)
import gdown
import pandas as pd

gdown.download('https://drive.google.com/uc?authuser=0&id=1Z8bwIBa_0gFe9-
C2W0goZ72lQfFMbxjS&export=download',
               'labeledTrainData.tsv',
               quiet=False)
df = pd.read_csv('labeledTrainData.tsv', header=0, delimiter='\t')
df
```

```
Downloading...
From: https://drive.google.com/uc?authuser=0&id=1Z8bwIBa_0gFe9-C2W0goZ72lQfFMbxjS&export=download
To: /content/labeledTrainData.tsv
33.6MB [00:00, 56.8MB/s]
```

Out[0]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...

	id	sentiment	review
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...
...	...	...	...
24995	3453_3	0	It seems like more consideration has gone into...
24996	5064_1	0	I don't believe they made this film. Completel...
24997	10905_3	0	Guy is a loser. Can't get girls, needs to buil...
24998	10194_3	0	This 30 minute documentary Buñuel made in the ...
24999	8478_8	1	I saw this movie as a child and it broke my he...

25000 rows × 3 columns

Just like last time, we will convert from text to a **bag of words** representation using scikit learn's built-in count vectorizer. Last time, we only included words in our feature vector if they occur in at least 100 reviews. Now we reduce this limitation, so that included words only need to appear in at least 20 reviews. Note the the previous shape of X was: X.shape (25000, 3833), and now it should have many more features.

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
from sklearn.model_selection import train_test_split

y = np.array(df['sentiment'])

dfX_train, dfX_test, y_train, y_test = train_test_split(df['review'], y)
print("df_train.shape",dfX_train.shape)
print("y_train.shape",y_train.shape)
print("dfX_test.shape",dfX_test.shape)
print("y_test.shape",y_test.shape)

vectorizer = CountVectorizer(binary=True, min_df = 20) #convert a collection of text documents into
a matrix of token counts
vectorizer.fit(dfX_train) #learn a vocabulary dictionary of all tokens in the raw documents

X_train = vectorizer.transform(dfX_train).todense() #transform to a document-term matrix
X_test = vectorizer.transform(dfX_test).todense()
print("X_test.shape",X_test.shape)
print("X_train.shape", X_train.shape)
```

```
df_train.shape (18750,)
y_train.shape (18750,)
dfX_test.shape (6250,)
y_test.shape (6250,)
X_test.shape (6250, 10094)
X_train.shape (18750, 10094)
```

We also split the data into training and test right from the start. We'll check to make sure our data is organized properly.

In [0]:

```
# Looking at a review to make sure things work
reviews_wrapped = dfX_train.str.wrap(80)
terrible_index = vectorizer.get_feature_names().index('terrible')
print("terrible occurs in", X_train[y_train==1, terrible_index].mean(), "for Y=1")
print("terrible occurs in", X_train[y_train==0, terrible_index].mean(), "for Y=0")
print(reviews_wrapped.iloc[1]) # Just in case you want to read a random review
```

```
terrible occurs in 0.017667092379736057 for Y=1
terrible occurs in 0.09193927731451786 for Y=0
I think this piece of garbage is the best proof that good ideas can be
destroyed, why all the American animators thinks that the kids this days wants
stupid GI JOE versions of good stories??? the Looney Tunes are some of the most
beloved characters in history, but they weren't created to be Xtreme, i mean
come on!!! Tiny Toons was a great example of how an old idea can be updated
without loosing it's original charm, but this piece of garbage is just an
example of stupid corporate decisions that only wants to create a cheap idiotic
```

```
show that kids will love because hey!!! kids loves superheroes right??? the
whole show is only a waste of time in which we see the new versions of the
Looney Tunes but this time in superhero form, this doesn't sound too bad but the
problem is that this show tries too hard to copy series like batman the animated
series, or the new justice league, the result??? bad copies of flash (the road
runner) or superman (who else??? bugs bunny) the problem is that Looney Tunes
weren't meant to be dramatic, the were supposed to be funny!!!! as i said before
this series sucks, and many people wonders why anime is taking all over the
world??? this show tries to be dramatic and action packed, but that's something
that few series and anime are able to do, if you want to see a good upgrade of
an old show watch Tiny Toons, that's an example that it's possible to bring back
to life old characters, but with a good story and respecting the original roots.
too bad that show is already dead, another corporate wise decision i suppose.
```

Fitting the Parameters of the Model & Making Predictions

As you may recall from the last assignment:

What we see from looking at the log odds ratio equation is that in order to apply the model we must have an estimate of the following probabilities.

- $p(Y=0)$
- $p(Y=1)$
- $p(X_i = x_i | Y=0)$ (for i from 1 to d)
- $p(X_i = x_i | Y=1)$ (for i from 1 to d)

The MLE problem in the earlier part of the assignment gave a more formal justification of this process.

We'll start with the solution from the last assignment. Here, we count the number of times the event occurs across the dataset in order to estimate a probability.

For instance, if we want to estimate $p(Y=0)$ we would count the number of instances in the dataset where $Y=0$ and divide that by the total number of instances in the dataset. If we wanted to estimated $p(X_i = 1 | Y = 0)$ (suppose X_i represents the word "terrible") we would count the number of reviews that included the word terrible and had sentiment 0 and divide that by the number of reviews that were sentiment 0. Remember that $p(A|B) = \frac{p(A, B)}{p(B)}$.

In [0]:

```
# Essentially same functions as solutions from previous assignment
def fit_nb_model(X, y):
    X_1 = np.asarray(X[y == 1, :]) # all reviews with sentiment 1
    X_0 = np.asarray(X[y == 0, :])
    return y.mean(), 1 - y.mean(), X_1.mean(axis=0), X_0.mean(axis=0)

def get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X):
    """ Predict the labels for the data X given the Naive Bayes model """
    log_odds_ratios = np.zeros(X.shape[0])
    for i in range(X.shape[0]): # loop over data points
        if i%(X.shape[0]/10) == 0: print("progress", i/X.shape[0])
        log_odds_ratios[i] += np.log(p_y_1) - np.log(p_y_0)
        for j in range(X.shape[1]): #loop over words
            if X[i, j] == 1: #if this example includes word j
                log_odds_ratios[i] += np.log(p_x_y_1[j]) - np.log(p_x_y_0[j])
            else:
                log_odds_ratios[i] += np.log(1 - p_x_y_1[j]) - np.log(1 - p_x_y_0[j])
    return (log_odds_ratios >= 0).astype(np.float)
```

Testing the Accuracy of the Model from Assignment 3

The following lines of code run the solution function (above) and calculate the accuracy. We're just using the first 100 entries for faster computation.

What do you observe happens when you run this?

In [0]:

```
p_y_1, p_y_0, p_x_y_1, p_x_y_0 = fit_nb_model(X_train, y_train)
y_pred_train = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_train[:100,:])
print("Train accuracy is", (y_pred_train == y_train[:100]).astype(np.float).mean()) #also only
need to compare first 100 y
```

```
y_pred = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_test[:100,:]) #only looking at first
100 X_test
print("Test accuracy is", (y_pred == y_test[:100]).astype(np.float).mean()) #also only need to
compare first 100 y
```

```
progress 0.0
progress 0.1
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: RuntimeWarning: divide by zero en
countered in log
```

```
progress 0.2
progress 0.3
progress 0.4
progress 0.5
progress 0.6
progress 0.7
progress 0.8
progress 0.9
Train accuracy is 0.91
progress 0.0
progress 0.1
progress 0.2
progress 0.3
progress 0.4
progress 0.5
progress 0.6
progress 0.7
progress 0.8
progress 0.9
Test accuracy is 0.82
```

You may have noticed a "RuntimeWarning: divide by zero encountered in log". It is common practice to use something call Laplace smoothing or additive smoothing to avoid this and other issues.

We can see just how common this practice is by running our classification again with the sklearn toolbox.

Note what happens when you set alpha=0 (which means no smoothing), would be similar to what we have implemented above.

You will likely see a warning that says: "alpha too small will result in numeric errors, setting alpha = 1.0e-10 'setting alpha = %.1e' % _ALPHA_MIN)."

You can instead change this value to alpha = 1.

In [0]:

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB(alpha=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test[:100,:])
np.mean(y_pred == y_test[:100])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/naive_bayes.py:485: UserWarning: alpha too small wi
ll result in numeric errors, setting alpha = 1.0e-10
  'setting alpha = %.1e' % _ALPHA_MIN)
```

Out[0]:

```
0.82
```

A bit about Laplace Smoothing

Imagine that you are trying to classify a review that contains the word 'awesomesauce' and that your classifier hasn't seen this word before. Naturally, the probability P(x_i|y) will be 0, making log of this go to negative infinity. Eeek!

Even if the training corpus is very large, it does not contain all possible words (or combinations of words... foreshaddowing!).

This is a common problem in NLP, but thankfully it has an easy fix: smoothing. This technique consists in adding a constant to each

This is a common problem in NLP, but thankfully it has an easy fix: smoothing. This technique consists in adding a constant to each count in the P(x_i|y) formula, with the most basic type of smoothing being called add-one (Laplace) smoothing, where the constant is just 1.

Sources: Medium article and section 3.4 of this paper.

Notebook Exercise 1

Revise the fit_nb_model_smooth() function below to include Laplace smoothing (right now, it's the same as our original function).

In [0]:

```
def fit_nb_model_smooth(X, y,alpha):
  X_1 = np.asarray(X[y == 1, :]) # all reviews with sentiment 1
  X_0 = np.asarray(X[y == 0, :])
  return y.mean(), 1 - y.mean(), X_1.mean(axis=0), X_0.mean(axis=0)


# Code to call and run your new fitting with alpha =1
p_y_1, p_y_0, p_x_y_1, p_x_y_0 = fit_nb_model_smooth(X_train, y_train,1) #Model with smoothing
y_pred = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_test[:100,:]) #Only looking at first
100 X_test
print("accuracy is", (y_pred == y_test[:100]).astype(np.float).mean()) #also only need to compare
first 100 y_test
```

```
progress 0.0
progress 0.1
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: RuntimeWarning: divide by zero en
countered in log
```

```
progress 0.2
progress 0.3
progress 0.4
progress 0.5
progress 0.6
progress 0.7
progress 0.8
progress 0.9
accuracy is 0.82
```

In [0]:

```
# ***Solution***
def fit_nb_model_smooth(X, y,alpha):
    X_1 = np.asarray(X[y == 1, :])
    X_0 = np.asarray(X[y == 0, :])
    N_1,V_1 = X_1.shape
    N_0,V_0 = X_0.shape #should actually be the same size in our case
    return y.mean(), 1 - y.mean(), np.divide(X_1.sum(axis=0)+1,N_1), np.divide(X_0.sum(axis=0)+1,N_0
)


# Code to call and run your new fitting with alpha =1
p_y_1, p_y_0, p_x_y_1, p_x_y_0 = fit_nb_model_smooth(X_train, y_train,1) #Model with smoothing
y_pred = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_test[:100,:]) #Only looking at first
100 X_test
print("accuracy is", (y_pred == y_test[:100]).astype(np.float).mean()) #also only need to compare
first 100 y_test
```

In [0]:

```
# ***Solution***
X_0 = np.asarray(X_train[y_train == 0, :])
print(np.shape(X_0))
print(np.shape(p_x_y_0))
```

Sequence Prediction: Whats next?

Time, the final, final frontier. Or, that thread that ties together events. These events could be major things in your life, or just a series of word or musical notes. Often the sequence of events contains very important information that we want to capture. For example, word order is often essential to the meaning of a phrase. Or, a set of descending musical notes gives a different feeling than an ascending one.

We can apply sequence information for various tasks, for example: -Predicting if a coin is fair or not

- Choosing the next song to play in a playlist based on previous songs
- Deciding what to suggest when watching videos
- Determining if text came from one corpus or another
- Predicting the next word (or letter) in a phrase (or word)

In this notebook, we'll explore these last two applications, building on our movie reivew data set (out of convenience).

What other applications can you think of for sequence prediction?

One problem with the Bag of Words

Until now, we've been using the Bag of Words strategy. Here, words are treated individually. Consider two sentences "big red machine and carpet" and "big red carpet and machine". If you use a bag of words approach, you will get the same vectors for these two sentences. However, we can clearly see that in the first sentence we are talking about a "big red machine", while the second sentence contains information about the "big red carpet". Hence, context information is very important.

Source

"Hello Bigrams" and Other Ngrams

Enter, the bigram (two-words). The bigram model can help us capture context information.

What is an N-gram? A contiguous sequence of N items from a given sample of text or speech or any other sequence. Here an item can be a character, a word or a sentence and N can be any integer. When N is 2, we call the sequence a bigram. Similarly, a sequence of 3 items is called a trigram, and so on.

Bigrams (or ngrams) are one strategy for sequence prediction (though they can also be used for other tasks including classification). We'll explore these below.

A mini-primer on text formatting and tokens

If you're really comfortable in python, you can skip this.

Otherwise, take a look at some examples of manipulating text (this might be useful in your next assignment if you use your own text with its own formatting quirks).

Start by writing some text and assigning it to s.

In [0]:

```
# Write some text and store it in the variable s
s = "This is where your sentence goes, or however much I want to write >"\
"your own profound thoughts! Oh-my! You should probably include some repeat phrases like repeat ph
rases or like repeat phrases"
```

In [0]:

```
import re
# Now, we need to tidy up our text to get rid of the riff-raff (to make it more consistent to find
common pairs of words)
# Convert to lowercases
s = s.lower()
# Replace all none alphanumeric characters with spaces
s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)
# Replace series of spaces with single space
s = re.sub(' +',' ',s)
print(s)

# Tokens refer to the words in our case (whatever the chunks are that we are breaking things up in
to)
tokens = [token for token in s.split(" ") if token != ""] # Break sentence in the token, remove emp
ty tokens
print(tokens)
```

```
print(tokens)
```

```
this is where your sentence goes or however much i want to write your own profound thoughts oh my
you should probably include some repeat phrases like repeat phrases or like repeat phrases
['this', 'is', 'where', 'your', 'sentence', 'goes', 'or', 'however', 'much', 'i', 'want', 'to',
'write', 'your', 'own', 'profound', 'thoughts', 'oh', 'my', 'you', 'should', 'probably',
'include', 'some', 'repeat', 'phrases', 'like', 'repeat', 'phrases', 'or', 'like', 'repeat',
'phrases']
```

In [0]:

```
import re
# Essentially the same as above, but putting it into a function for later
def clean_text(s):
  s = s.lower() # Convert to lowercases
  s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s) # Replace all non alphanumeric characters with spaces
  s = re.sub(' +',' ',s) # Replace series of spaces with single space
  return s
```

Now that we have a sense of how the word parsing works, let's move this into a function called generate_ngrams that takes in our text and the number of words to group by (n). For a bigram, we should choose n=2

In [0]:

```
def generate_ngrams(s, n):
    s = clean_text(s) # Clean text
    # Break sentence in the token, remove empty tokens
    tokens = [token for token in s.split(" ") if token != ""]
    # Generate sequences of tokens starting from different
    # elements of the list of tokens.
    # The parameter in the range() function controls how many sequences
    # to generate.
    sequences = [tokens[i:] for i in range(n)]
    # Use the zip function to help us generate n-grams
    # Concatentate the tokens into ngrams and return
    ngrams = zip(*sequences)
    #ngrams = zip(*[tokens[i:] for i in range(n)]) # Or you could combine into one line like this.
    return [" ".join(ngram) for ngram in ngrams]

generate_ngrams(s,2)
```

Out[0]:

```
['this is',
 'is where',
 'where your',
 'your sentence',
 'sentence goes',
 'goes or',
 'or however',
 'however much',
 'much i',
 'i want',
 'want to',
 'to write',
 'write your',
 'your own',
 'own profound',
 'profound thoughts',
 'thoughts oh',
 'oh my',
 'my you',
 'you should',
 'should probably',
 'probably include',
 'include some',
 'some repeat',
 'repeat phrases',
 'phrases like',
 'like repeat',
 'repeat phrases',
 'phrases or',
 'or like',
 'like repeat',
```

```
 'repeat phrases']
```

Yay, ngrams! (Bigrams in our case.) Often, it will be more interesting to collapse things and look at a the frequencies of our ngrams.

This example uses the nltk library to create your ngrams (a second option for future work).

In [0]:

```
import collections
from nltk.util import ngrams

s = clean_text(s) # Clean text
tokens = [token for token in s.split(" ") if token != ""]
bigramWords = list(ngrams(tokens, 2))
bigramFreq = collections.Counter(bigramWords)

bigramFreq.most_common(10)
#print(bigramFreq.most_common(10))
```

Out[0]:

```
[(('repeat', 'phrases'), 3),
 (('like', 'repeat'), 2),
 (('this', 'is'), 1),
 (('is', 'where'), 1),
 (('where', 'your'), 1),
 (('your', 'sentence'), 1),
 (('sentence', 'goes'), 1),
 (('goes', 'or'), 1),
 (('or', 'however'), 1),
 (('however', 'much'), 1)]
```

Applying bigrams to the movie reviews

Now, we'll explore bigrams with a larger data set. Oh hey, we already loaded in a bunch of movie reviews, how convenient!

First, we'll want to clean up the text in our reviews a bit.

Notebook Exercise 2

Check 5-10 of the movie reviews to make sure our text cleaning is working. dfX_train contains all the movie reviews in the training set.

(a) Consider the word parsing done in the previous example. Think about how your text cleaning (e.g., removing hyphens) affected the bigrams. What are some potential limitations of this type of text processing?

(b) Determine how you want to inspect the reviews without checking them all. Do you want to just look at the raw text for each? Or perhaps you want to look at the bigrams for each example? Or perhaps you want to look at

(c) Update the clean_text() function to remove the most common formatting issue.

In [0]:

```
#Here's a little code to get you started
print("An original text from the training set:")
print(list(dfX_train)[2])
```

```
An original text from the training set:
this is the only movie i have ever walked out on. bad acting-- bad plot-- bad casting-- bad direct
ing-- bad cinematography-- if they had set out to make a bad picture they couldn't have done a bet
ter job. i hope they are proud of his turkey. i'm surprised anyone associated with this film was e
ver hired again in hollywood. don't waste your time!
```

Expand for solution

Solution

(c) Depending on the context of what you want to do, you may choose to keep in things like hyphens (perhaps ninety-nine is more

(a) Depending on the context of what you want to do, you may choose to keep in things like hyphens (perhaps ninety-nine is more meaningful together than apart). Or perhaps appostrophes are meaningful in your corpus. Whenever we are dealing with text, we want to think about our syntax choices, like including flags for start and end words in a sentence (instead of assuming one infinite string of words).

(b) "Dealer's choice" here, some examples below (you don't need to use this approach).

(c) Adding the line s = re.sub('br />',' ',s) will remove those pesky break indicators, just make sure to do this before you run the line that removes all of the non-alphaneumeric characters. Otherwise, you'll end up with what appears to be the word "br" everywhere, and we doubt every movie reviewer is freezing.

In [0]:

```
# ***Solution***
# Checking the first few movie reviews
for i in range(10):
  origtext = list(dfX_train)[i]
  print("Original text: ",origtext)
  cleaned = clean_text(origtext)
  print("Cleaned  text: ",cleaned)
  tokens = [token for token in cleaned.split(" ") if token != ""]
  bigramWords = list(ngrams(tokens, 2))
  bigramFreq = collections.Counter(bigramWords)
  print("Bigrams: ", bigramFreq.most_common(10))
  print('\n')
```

In [0]:

```
# ***Solution***
def clean_text(s):
  s = s.lower() # Convert to lowercases
  s = re.sub('<br />',' ',s) #Added this new line to get rid of the breaks
  s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s) # Replace all non alphanumeric characters with spaces
  s = re.sub(' +',' ',s) # Replace series of spaces with single space
  return s
```

Now, let's clean all the data

In [0]:

```
def clean_series_data(sdata):
  sdata = list(sdata)
  for i in range(len(sdata)):
    sdata[i] = clean_text(sdata[i])
    if i%(len(sdata)/5)==0:
      print(sdata[i]) #Printing occasional text can be helpful for making sure that your cleaning
is working how you want it to. Or you can comment this out.
  return sdata

dfX_train = clean_series_data(dfX_train)
dfX_test = clean_series_data(dfX_test)
```

```
of all the kung fu films made through the 70 s and 80 s this is one that has developed a real cult
following with the exception of all the films bruce lee starred in this is a film that has stood t
he test of time and its due to the unique story an aging kung fu master tells his last pupil yang
tieh sheng chiang about five pupils he has trained in the past all five wore masks and nobody has
seen the face of each other and they have all been trained differently their specialty in kung fu
is the name they have adopted like lizard snake centipede toad and scorpion the master called them
the poison clan and he does not know what has happened to them so he wants tieh to find them and h
elp the ones that are doing good to stop the others that are evil an old man who was once a member
of the poison clan has a map to where he has hidden a lot of money and he seems to be a target tie
h does not know what they look like so he has to mingle in society and try and figure out who they
are tieh has discovered that the snake is hung wen tung pai wei and along with tang sen kue feng l
u who is the centipede they kill a family to find the map a map is found by a mystery man who turn
s out to be the scorpion but know one knows who he is a local policeman named ho yung sin philip k
wok investigates the murders along with his partner ma chow chien sun sin has a friend called li h
o meng lo who is the toad and they do know of each others identity the snake bribes the local offi
cials to pin the murders on li ho and while he is in prison he is tortured and killed when sin fin
ds out he teams up with tieh and together they go to combat tung and kue this film was directed by
cheh chang and he was a very special director when it came to these films chang was not your run o
f the mill kung fu director and his films always had a special quality to them while most martial
arts films deal with revenge chang did not use that as a central theme even though there is some r
evenge going on later in this story this film is more complex than that five men trained by the
```

evenge going on later in this story this film is more complex than one five men trained by the same master in different ways and wearing masks then they are all in the same area and not knowing who the other is very unique story makes this film different from all the others and most of chang s stories were in a class all by themselves i wouldn t exactly put it in the same league as enter the dragon because bruce lee was a worldwide icon and the martial arts he exhibited were more auth entic looking this film still has some impossible feats like clinging to sides of walls and all th e flipping through the air but this film isn t necessarily about fight scenes its more about the i ntrigue of the story and the characters that are involved that alone makes this different from all the other kung fu films very well made with a unique story

it was simple and yet so nice i think the whole sense of sex segregation in society which can be b itter was shown very delicately it had a bitter kind of hummer in it the fact that most of the act ors were not professionals made the movie more tangible and more realistic there was a documentary side to the movie too the best scenes were those that all the girls banned from watching were list ening passionately to the soldier who is supposed to keep an eye on them broadcasting the game if you are an iranian the familiar cheering and dancing in the streets after a game won fills you up with national pride if you are not iranian you ll still love it all the same

a friend of mine bought this film for 1 and even then it was grossly overpriced despite featuring big names such as adam sandler billy bob thornton and the incredibly talented burt young this film was about as funny as taking a chisel and hammering it straight through your earhole it uses tired bottom of the barrel comedic techniques consistently breaking the fourth wall as sandler talks to the audience and seemingly pointless montages of hot girls adam sandler plays a waiter on a cruise ship who wants to make it as a successful comedian in order to become successful with women when t he ship s resident comedian the shamelessly named dickie due to his unfathomable success with the opposite gender is presumed lost at sea sandler s character shecker gets his big break dickie is n ot dead he s rather locked in the bathroom presumably sea sick perhaps from his mouth he just vomi ted the worst film of all time

this was a movie that i hoped i could suggest to my american friends but after 4 attempts to watch the movie to finish i knew i couldn t even watch the damn thing to close you are almost convinced the actual war didn t even last that long other s will try to question my patriotism for criticizing a movie like this but flat out you can t go from watching saving private ryan to loc f orget about the movie budget difference or the audience those don t preclude a director from makin g an intelligent movie the length of the movie is not so bad and the fact that it is repetitive th ey keep attacking the same hill but give it different names i thought the loc was a terrible terra in this hill looked like my backyard the character development sequences the soilders flashbacks l ooking back to their last moments before being deployed should have been throughout the movie and not just clumped into one long memory to this day i have yet to watch the ending but there was a m uch better movie not saying much called border

no scenario bad actors poor melissa gilbert beurk beurk beurk give a such budget to make this in b elgium we make ten films which win all prices in cannes with this last time that i ve seen a such null film was hypercube but scenario was better is anyone knows if the director was a graduate in school film or a cop the better things in this film was the word end why authorize to sell this 1 is to expensive i ve pay ten dollars to buy this for me pay for this was my big mistake of millenn ium too bad next time i ll break my arm but buy this type of sh t

i d passed this title 15 or 20 times while in blockbuster looking for something halfway decent to watch all i can say is that they were all out halfway decent films the night i chose to rent this i will give it credit for being leaps and bounds better than dracula 3000 but in actuality that s pretty easy to say since this one didn t have caper van dien in it the other things it lacked in s pades were an interesting cast interesting story good dialog and originality but i suppose you can t have everything spoilers ahead the misfit crew of vampire hunters one of which was a vampire go figure flew around from mining colony to mining colony trying to wipe out all the vampires it coul d find the crew consists of the cocky ne er do well captain who dies early on his by the book yet inexperienced first mate the aforementioned vampire vampire hunter not a typo a wannabe cowboy and a really really butch asian commando tough guy female it almost sounds like the cast for mtv s the real world after the captain dies in a violent confrontation with a mob of bloodsuckers and as it turns out lovers of the finer parts of human anatomy his first mate takes over blah blah blah everyone hates him blah blah blah i could go on but as i said it s the same storyline used in abou t a hundred other films of the genre the effects were cheese and why michael ironside was in this movie is beyond me the vampire vampire hunter still not a typo was pretty hot but of course if you re waiting for her to expose more than just her cleavage look elsewhere i can t say that it was th e worst film i ve seen as i actually rented starship troopers 2 but wait for this one to come on t he sci fi channel

paris is the place to be to enjoy beautiful art and music and to fall madly in love as is the case in this film boy meets girl they fall in love but something stands in their way of eternal happiness the classic story the wonderful music of george gerschwin complements the great dancing by gene kelly and leslie caron an american in paris is a humorous light hearted loving film well w orth watching 8 10

it is an almost ideal romantic anime must see for all ages but the english dubbed version is not t oo good perhaps the 1999 version will be better

a shirley temple short subject it can get mighty rough at buttermilk pete s cafe when the local co ntingency of diaper clad war babies come in for their midday milk break this primitive little film a spoof of military movies provides a few chuckles but little else tiny tots talking tough can beg in to pall in a short time shirley temple playing a duplicitous hip swinging french miss hasn t mu ch to do in this pre celebrity performance highlight the real signs of toddler temper when a few o f the infants unexpectedly get well truly soaked with milk often overlooked or neglected today the one and two reel short subjects were useful to the studios as important training grounds for new o r burgeoning talents both in front behind the camera the dynamics for creating a successful short subject was completely different from that of a feature length film something akin to writing a to pnotch short story rather than a novel economical to produce in terms of both budget schedule and

```
photon short story rather than a novel economical to produce in terms of both budget schedule and
capable of portraying a wide range of material short subjects were the perfect complement to the s
tudios feature films
this is the page for house of exorcism but most people have confused this film with the mario bava
masterpiece lisa the devil which explains the ridiculously high rating for this house of exorcism
when lisa the devil was shown at film festivals in the early 70 s it was a critical success audien
ces responded well to that gorgeous gothic horror film unfortunately it was a bit ahead of it s ti
me and was considered too unusual and not commercial enough for mass consumption no distributor wo
uld buy it so producer alfredo leone decided to edit lisa seemingly with a chainsaw by removing ju
st about half of the original film and adding new scenes which he filmed two years after the origi
nal product it is important to note that bava had little to do with these new hideous additions so
technically house of exorcism is not a bava film the original product is a slow dreamy classy prod
uction a few minutes into the film the viewer is jarred out of this dream world as suddenly we see
lisa two years older and with a very different haircut begin to writhe on the ground making guttur
al sounds and croaking epitaphs like suck my co k etc subtle huh and the film continues like this
jumping back and forth between a beautiful visual film and a grade z exorcist rip off leone was tr
ying to incorporate these shock scenes while keeping some semblance of a story intact he failed mi
serably when the choice was made to basically destroy lisa and the devil bava himself refused sayi
ng that his film was too beautiful to cut he was right and it must have been quite sad for this ar
tist to see all his work destroyed and flushed down the toilet it was many years before the origin
al lisa and the devil was seen again re surfacing on late night television i had seen lisa long be
fore i saw this new version and it was downright disturbing to witness one of my favorite films va
ndalised in this way worth seeing only for curiosity sake otherwise avoid this insidious disaster
like the plague
```

Notebook Exercise 3

Find the top 10 bigrams for the positive and negative movie reviews in the training data.

In [0]:

```
# Your code goes up here

#bigramFreqPositive =
#bigramFreqNegative = [] # this too
#print("Positive bigrams:")
#bgfp = bigramFreqPositive.most_common(10)
#for bg in bgfp:
#  print(bg)

#print("Negative bigrams \n")
#bgfp = bigramFreqNegative.most_common(10)
#for bg in bgfp:
#  print(bg)
```

Expand for solution

Note: If you are still getting 'br','br' as your top bigram. Go back and run the second block in the notebook (where dfX_train gets assigned). Then run the block above that includes the function clean_series_data(). If that doesn't work, repeat the process, after first running the solution where clean_text(s) is defined.

In [0]:

```
# ***Solution**
# A solution (though there are likely faster, cleaner ways to do this)
wordsPositive = list()
wordsNegative = list()
n = 2

for i in range(len(y_train)):
  tokens = [token for token in dfX_train[i].split(" ") if token != ""]
  if y_train[i]==1:
    wordsPositive.extend(tokens)
  else:
    wordsNegative.extend(tokens)

bigramWordsPositive = list(ngrams(wordsPositive, n))
bigramFreqPositive = collections.Counter(bigramWordsPositive)
bigramWordsNegative = list(ngrams(wordsNegative, n))
bigramFreqNegative = collections.Counter(bigramWordsNegative)

print("Positive bigrams:")
bgfp = bigramFreqPositive.most_common(10)
```

```
for bg in bgfp:
  print(bg)
print('\n')
print("Negative bigrams")
bgfp = bigramFreqNegative.most_common(10)
for bg in bgfp:
  print(bg)
```

Try a different sized n-gram (instead of a bigram / 2-gram)

Not shockingly, this list is not very exciting. It turns out, people use some pretty standard words bigrams when talking about movies (e.g. "this film" and "of the")... really riveting stuff. If we look at a greater number of bigrams (e.g., the top 100), we can eventually start to find something relevant among mostly trite pairings.

However, it might be interesting to look at a different sized ngram than the bigram. **Try something in the n = 4 to 7 range.**

Classifying movie review sentiment with bigrams

Let's revisit our Naïve Bayes model, but now using bigrams as our features instead of single words.

Use CountVectorizer to get top bigrams and then classify sentiment

The code below gives a black box approach to classifying with ngrams.

The ngram_range(2,2) makes our code use bigrams.

Note that we now have a different shape to our data because it is stored in sparse form (no longer using todense()). If we try to store this in dense form, we will run into RAM errors, which we could combat by limiting the number of ngrams that we include in our CountVectorizer by setting max_features=10000 limits the total number of feautures.

In [0]:

```
ngramvectorizer = CountVectorizer(ngram_range=(2,2))
ngramvectorizer.fit(dfX_train) #learn a vocabulary dictionary of all tokens in the raw documents

X_train_ngram = ngramvectorizer.transform(dfX_train)
X_test_ngram = ngramvectorizer.transform(dfX_test)
print("X_train_ngram.shape", X_train_ngram.shape)
print("X_test_ngram.shape", X_test_ngram.shape)
```

```
X_train_ngram.shape (18750, 1170351)
X_test_ngram.shape (6250, 1170351)
```

In [0]:

```
# Actually run the model and print results
# If this is taking too long, you can run it on a subset of your data.
model = MultinomialNB(alpha=1)
model.fit(X_train_ngram, y_train)

y_pred_train = model.predict(X_train_ngram)
print("Training accuracy: ", np.mean(y_pred_train == y_train))
y_pred = model.predict(X_test_ngram)
print("Testing accuracy: ",np.mean(y_pred == y_test))
```

```
Training accuracy:  0.9978666666666667
Testing accuracy:  0.88432
```

Notebook Exercise 4

(a) Try manipulating the size of the ngram and the number of features to explore their effects.

(b) What method is essentially the same as using the parameter: ngram_range=(1,1)? What are the relative strenghts/weaknesses of these two approaches?

Expand for solution

Solution

(a) This depends on what you looked at. We observed that a full set of bigrams performed better than a bag of words. However, this was not the case if we limit the number of features.

(b) Bag of Words. An 1-gram is just a list of words that occurred. The Bag of Words takes less memory to run... think about how many unique single words exist in text and compare this to how many unique word pairs exist in the same text.

However, bigrams can provide us with more information about the context that the words are used. For example, a positive review might include 'not terrible' while a negative review might include 'so terrible'.

Predicting the next words with bigrams

Now, we'll explore the idea of choosing the next word in a sequence based on the previous word. We'll do a simple implementation of this, which you will then improve upon.

Since we already have a list of the bigrams, we can use these to build our predictor. We'll just use the positive bigram list ***bigramWordsPositive*** that was generated in the solution to Exercise 3. You may need to go back and run that block of code if you haven't already.

This code creates a dictionary to store 2nd word in each bigram under the key of the first word.

In [0]:

```
bigramLookup = {}

for i in range(len(bigramWordsPositive)-1):
    w1 = bigramWordsPositive[i][0]
    w2 = bigramWordsPositive[i][1]
    #print(w1,w2)
    if  w1 not in bigramLookup.keys():
      bigramLookup[w1] = {w2:1}
    elif w2 not in bigramLookup[w1].keys():
      bigramLookup[w1][w2] = 1
    else:
      bigramLookup[w1][w2] = bigramLookup[w1][w2] + 1
```

Now, we can write some code to generate an ouptut based on a starting word (curr_sequence).

In [0]:

```
import random

curr_sequence = "my" # Starting word
output = curr_sequence
for i in range(50):
    if curr_sequence not in bigramLookup.keys():
      print("not in my keys, choosing seed word ")
      output += '. '
      curr_sequence = 'the'
      output += curr_sequence
    else:
      possible_words = list(bigramLookup[curr_sequence].keys())
      next_word = possible_words[random.randrange(len(possible_words))] #Randomly choose a word
      output += ' ' + next_word
      curr_sequence = next_word


print(output)
```

```
my era additional details at while its slightly above a meteoric success was required it virtually
doomed to divulge what race does blackmail scheme on shakespeare gaining insight in comic
character considering i paid 1930 and expressionally handcuffed russell had devoted female informa
nts until two bullets in alexander hunting companion this
```

Notebook Exercise 5

Well, put together some words, sometimes coherently, sometimes not so much. What a great opportunity for... next_word!

Modify the above review generation code to improve it in some way. For example, you might select the next word based on it's probability, instead of just randomly choosing any word that was previously paired.

If you're feeling saucy, you could build this review generator to take use some higher value of ngrams. And if you're even saucier, you could build a function that writes your whole review based on the input of a starting phrase and whether the review should be positive or negative.

No solution for this one, but come to class prepared to talk about what you tried.

Optional Exercise: Build a word completion tool using ngrams on letters instead of words.

If you flew through this notebook (because you're familiar with this content or a python wiz, a fun way to challenge yourself is to build a tool that uses ngrams on letters instead of words. This can be applied to word auto-complete or to spell checking. Be sure to think about the way you want to format your text. How many features will your data have if you do a 1-gram, 2-gram, or 3-gram?