Assignment 3: Fairness, Maximum Likelihood Estimation, and Text Classification

Machine Learning

Fall 2019

Learning Objectives

- Learn about the connection between probabilistic criteria for algorithmic fairness and Bayesian Networks.
- Solidify your understanding of the Naïve Bayes algorithm by applying it to movie review sentiment classification.

1 Bayesian Networks and Algorithmic Fairness

In assignment 1 of this module we discussed how Bayesian methods can be used to reason about algorithmic fairness. We've just had some lengthy discussions about fairness within the context of the COMPAS algorithm. We touched upon some of the limitations of statistically based notions of fairness. Nevertheless, they do have a potential role to play, and you should know what the most common definitions of fairness are and what assumptions they make.

As context for the reading and to help us have common notation, suppose we have the following random variables.

- R is a prediction generated by our algorithm.
- A is a sensitive attribute
- Y is the thing we're trying to predict (we want $R = Y$ if we are predicting accurately)

External Resource(s) (40 minutes)

Read Fairness and Machine Learning Chapter 2. Start at the section *Formal non-discrimination criteria* and read up to (but not including) the section *Calibration and sufficiency*.

Notice

- Don't get hung up on the ROC curves. We can certainly discuss this on NB, but it is not required to understand what is going on here. The presentation earlier in the linked reading is also pretty clear.
- The notation they use in this reading for conditional independence is an upside down T with only one line (instead of our notation, $\perp\!\!\!\perp$).

Exercise 1 (10 minutes)

Thinking back to the COMPAS example, which definition of fairness given in the reading was Propublica using? Which definition of fairness was Northpointe using?

If you're interested in playing around with the COMPAS data more extensively, we have put together a notebook that reproduces the calculation that are at the heart of the two competing notions of fairness. The notebook is written with a couple of notebook exercises, but these are totally optional. We expect the default will be that folks will just take a look

if they want to (or skip this if they don't). If you want to spend some extra time and solidify your understanding of true positive rate, false positive rate, positive predictive value, etc., then it might be worth doing it as a set of exercises.

For the purposes of posting on NB, the conversion of the notebook is at the end of the document.

2 Text Classification with Bag of Words

Next we'll be applying Naïve Bayes to the task of classifying text.

External Resource(s) (60 minutes)

This will be done in the Assignment 3 companion notebook.

(to make this more readable, we have included the PDF of the companion notebook at the end).

3 The Intelligent Design of Jenny Chow

This assignment is fully described on the Intelligent Design of Jenny Chow Canvas page. There is also an alternative described on the assignment page if you can't attend. Make sure to look at the assignment before going to the play since we are asking you to capture some of your reactions / thoughts so that you can bring them to class on Monday for discussion.

Open in Colab

Exploring the ProPublica COMPAS Analysis

In this notebook you'll get a chance to examine the data used in the ProPublica story yourself.

Disclaimer: Please don't over interpret what you find in the data. We know from our discussions that methodology is key to being able to properly interpret findings. Our goal here will be to reproduce results from the readings we did for last class.

First, we'll download and parse the data into a data frame.

In [2]:

```
import pandas as pd

!wget https://raw.githubusercontent.com/propublica/compas-analysis/master/compas-scores-two-years.csv
df = pd.read_csv('compas-scores-two-years.csv')
df
```

```
--2019-10-24 03:07:49--  https://raw.githubusercontent.com/propublica/compas-analysis/master/compas-scores-two-years.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2546489 (2.4M) [text/plain]
Saving to: ‘compas-scores-two-years.csv’

compas-scores-two-y 100%[===================>]   2.43M  --.-KB/s    in 0.09s

2019-10-24 03:07:49 (26.4 MB/s) - ‘compas-scores-two-years.csv’ saved [2546489/2546489]
```

Out[2]:

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	juv_fel_count	dec
0	1	miguel hernandez	miguel	hernandez	2013-08-14	Male	1947-04-18	69	Greater than 45	Other	0	
1	3	kevon dixon	kevon	dixon	2013-01-27	Male	1982-01-22	34	25 - 45	African-American	0	
2	4	ed philo	ed	philo	2013-04-14	Male	1991-05-14	24	Less than 25	African-American	0	
3	5	marcu brown	marcu	brown	2013-01-13	Male	1993-01-21	23	Less than 25	African-American	0	
4	6	bouthy pierrelouis	bouthy	pierrelouis	2013-03-26	Male	1973-01-22	43	25 - 45	Other	0	
5	7	marsha miles	marsha	miles	2013-11-30	Male	1971-08-22	44	25 - 45	Other	0	
6	8	edward riddle	edward	riddle	2014-02-19	Male	1974-07-23	41	25 - 45	Caucasian	0	
7	9	steven stewart	steven	stewart	2013-08-30	Male	1973-02-25	43	25 - 45	Other	0	
8	10	elizabeth thieme	elizabeth	thieme	2014-03-16	Female	1976-06-03	39	25 - 45	Caucasian	0	
9	13	bo bradac	bo	bradac	2013-11-04	Male	1994-06-10	21	Less than 25	Caucasian	0	
		benjamin					1988					

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	juv_fel_count	dec
10	14	benjamin franc	benjamin	franc	2013-11-26	Male	1988-08-30	27	25 - 45	Caucasian	0	
11	15	ellyaher lanza	ellyaher	lanza	2013-10-03	Male	1992-08-18	23	Less than 25	African-American	0	
12	16	kortney coleman	kortney	coleman	2013-01-01	Female	1978-08-22	37	25 - 45	Caucasian	0	
13	18	jarrod turbe	jarrod	turbe	2013-10-09	Male	1974-12-02	41	25 - 45	African-American	0	
14	19	craig gilbert	craig	gilbert	2013-10-30	Female	1968-06-14	47	Greater than 45	Caucasian	0	
15	20	samuel seraphin	samuel	seraphin	2014-06-03	Male	1985-03-25	31	25 - 45	African-American	0	
16	21	mario hernandez	mario	hernandez	2014-03-24	Male	1979-01-25	37	25 - 45	Hispanic	0	
17	22	darrious davis	darrious	davis	2013-12-22	Male	1990-06-22	25	25 - 45	African-American	0	
18	23	neil heckart	neil	heckart	2013-11-17	Male	1984-12-24	31	25 - 45	Caucasian	0	
19	24	michael lux	michael	lux	2014-11-15	Male	1985-01-08	31	25 - 45	Caucasian	0	
20	25	columbus wilson	columbus	wilson	2014-05-02	Male	1951-06-28	64	Greater than 45	African-American	0	
21	26	vandivuiet williams	vandivuiet	williams	2013-04-18	Male	1994-11-29	21	Less than 25	African-American	0	
22	27	nelson avalo	nelson	avalo	2014-10-16	Male	1988-08-06	27	25 - 45	Caucasian	0	
23	28	janel denicola	janel	denicola	2013-11-22	Female	1995-03-22	21	Less than 25	Caucasian	0	
24	30	dominic pabon	dominic	pabon	2013-02-08	Male	1992-01-23	24	Less than 25	Hispanic	0	
25	32	russell sottile	russell	sottile	2013-01-25	Male	1973-01-10	43	25 - 45	Caucasian	0	
26	33	andre ashley	andre	ashley	2013-05-11	Male	1983-08-24	32	25 - 45	Other	0	
27	37	deandrae counts	deandrae	counts	2013-05-06	Male	1989-02-08	27	25 - 45	African-American	0	
28	38	victoria soltau	victoria	soltau	2013-03-18	Female	1979-09-03	36	25 - 45	Caucasian	0	
29	39	najee sapp	najee	sapp	2013-02-20	Male	1989-04-27	26	25 - 45	African-American	0	
...	...	...	...	...	...	...	...	...	...	...	...	...
7184	10962	yolani moratz	yolani	moratz	2013-08-08	Female	1979-11-05	36	25 - 45	Caucasian	0	
7185	10963	paul wyatt	paul	wyatt	2013-11-06	Male	1973-09-19	42	25 - 45	Caucasian	0	
7186	10964	terrence brown	terrence	brown	2013-02-23	Male	1979-06-08	36	25 - 45	African-American	0	
7187	10965	anthony fields	anthony	fields	2013-03-13	Male	1959-02-20	57	Greater than 45	Caucasian	0	

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	juv_fel_count	dec
7188	10966	shameel koya	shameel	koya	2013-03-04	Male	1978-10-12	36	25 - 45	Caucasian	0	
7189	10967	george bedward	george	bedward	2014-05-31	Male	1981-05-26	34	25 - 45	African-American	0	
7190	10969	eric sparks	eric	sparks	2013-01-11	Male	1991-07-13	24	Less than 25	African-American	0	
7191	10971	eugene drogus	eugene	drogus	2014-01-07	Male	1948-10-17	67	Greater than 45	Caucasian	0	
7192	10972	matt munoz	matt	munoz	2013-09-12	Male	1984-03-22	32	25 - 45	Caucasian	0	
7193	10975	warren aiken	warren	aiken	2013-09-05	Male	1990-09-30	25	25 - 45	African-American	0	
7194	10976	arleen martin	arleen	martin	2014-12-19	Female	1985-08-14	30	25 - 45	Caucasian	0	
7195	10977	adrian williams	adrian	williams	2013-09-18	Male	1981-03-09	35	25 - 45	African-American	0	
7196	10979	angelita diaz	angelita	diaz	2013-09-06	Male	1972-07-19	43	25 - 45	African-American	0	
7197	10980	jarvis yates	jarvis	yates	2014-02-27	Male	1987-08-27	28	25 - 45	African-American	0	
7198	10981	orett harrison	orett	harrison	2013-12-25	Male	1984-03-31	32	25 - 45	African-American	0	
7199	10982	austin harris	austin	harris	2013-10-01	Male	1992-07-07	23	Less than 25	Caucasian	0	
7200	10984	shantrina stfort	shantrina	stfort	2013-11-05	Female	1995-06-06	20	Less than 25	African-American	0	
7201	10985	kyle miller	kyle	miller	2014-01-22	Male	1986-04-08	30	25 - 45	African-American	0	
7202	10987	ceasar gomez	ceasar	gomez	2013-03-31	Male	1990-02-07	26	25 - 45	Hispanic	0	
7203	10988	luis fernandez	luis	fernandez	2013-10-27	Male	1971-09-19	44	25 - 45	Hispanic	0	
7204	10989	rodney montgomery	rodney	montgomery	2013-12-28	Male	1985-09-28	30	25 - 45	African-American	0	
7205	10990	christopher tun	christopher	tun	2013-05-28	Male	1992-04-28	23	Less than 25	Caucasian	0	
7206	10992	alexander vega	alexander	vega	2013-05-10	Male	1994-07-15	21	Less than 25	Caucasian	0	
7207	10994	jarred payne	jarred	payne	2014-05-10	Male	1985-07-31	30	25 - 45	African-American	0	
7208	10995	raheem smith	raheem	smith	2013-10-20	Male	1995-06-28	20	Less than 25	African-American	0	
7209	10996	steven butler	steven	butler	2013-11-23	Male	1992-07-17	23	Less than 25	African-American	0	
7210	10997	malcolm simmons	malcolm	simmons	2014-02-01	Male	1993-03-25	23	Less than 25	African-American	0	
7211	10999	winston gregory	winston	gregory	2014-01-14	Male	1958-10-01	57	Greater than 45	Other	0	

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	juv_fel_count	dec
7212	11000	farrah jean	farrah	jean	2014-03-09	Female	1982-11-17	33	25 - 45	African-American	0	
7213	11001	florencia sanmartin	florencia	sanmartin	2014-06-30	Female	1992-12-18	23	Less than 25	Hispanic	0	

7214 rows × 53 columns

In the ProPublica dataset they only used data where the "days_b_screening_arrest" feature was in the range [-30, 30].

In [0]:

```
# filter these based on propublica analysis (not sure why this doesn't match
https://fairmlbook.org/classification.html)
df = df[(df['days_b_screening_arrest'] <= 30) | (df['days_b_screening_arrest'] >= -30)]
```

Reproducing Calculations of False Positive Rate and Positive Predictive Value

From a statistical point of view (but not necessarily a social justice point of view) the debate between Propublica and NorthPointe boiled down to what is the rate way to measure bias in an algorithm. As you saw in the first part of the assignmnet, ProPublica used the evidence that the false positive rate differed between blacks and whites as evidence of bias. Northpointe argued used the fact that the positive predictive values across the two groups were the same as evidence *against* bias.

In Northpointe's report, they have a table which lists various statistics for blacks versus whites using the COMPAS risk scores as the predictor. Here is the relevant information from their report.

	Race	$\hat{y} = 1$	True positive rate	False Positive Rate	Positive Predictive Value	Negative Predictive Value
white	$decile \geq 1$	1.00	1.00	0.39	1.00	
		$decile \geq 2$	0.85	0.64	0.46	0.79
		$decile \geq 3$	0.74	0.47	0.5	0.76
		$decile \geq 4$	0.64	0.35	0.54	0.74
		$decile \geq 5$	0.52	0.23	0.59	0.71
		$decile \geq 6$	0.41	0.15	0.64	0.69
		$decile \geq 7$	0.29	0.09	0.68	0.66
		$decile \geq 8$	0.20	0.05	0.71	0.65
		$decile \geq 9$	0.12	0.03	0.70	0.63
		$decile \geq 10$	0.05	0.01	0.70	0.61
black	$decile \geq 1$	1.00	1.00	0.51	1.00	
		$decile \geq 2$	0.95	0.83	0.55	0.77
		$decile \geq 3$	0.89	0.68	0.58	0.73
		$decile \geq 4$	0.81	0.56	0.60	0.69
		$decile \geq 5$	0.72	0.45	0.63	0.65
		$decile \geq 6$	0.63	0.34	0.66	0.62
		$decile \geq 7$	0.51	0.25	0.69	0.59
		$decile \geq 8$	0.39	0.16	0.72	0.57
		$decile \geq 9$	0.26	0.09	0.74	0.54
		$decile \geq 10$	0.12	0.03	0.79	0.51

Notebook Exercise 1

Write code to reproduce the table above. Here are some hints to help you.

- If you're fuzzy on what each of these statistic means (false positive rate, true positive rate, etc.), consider checking out Binary

Diagnostic Tests.

- You'll want to use the column `df['two_year_recid']` as your indicator of true positive versus true negative (positive means that the person recidivated).
- To select narrow the data frame to just contain people of a particular race you can use the following snippet (`race` would be a string that is either "Caucasian" or "African-American").

```
df_for_race = df[df['race'] == race]
```

- To generate a particular row of the table, you'll want to loop over all possible thresholds where the model would predict recidivate ($\hat{y} = 1$).
- You can count the number of elements in a Pandas series that satisfy some criterion using the following technique. For instance, if we wanted to calculate the number of elements in "some_column" that are greater than 0 and less than 30, we could use the following code.

```
((df['some column'] > 0).sum() & (df['some column'] < 30).sum()
```

- It's up to you how you want to generate the table. You can simply print out the values within a loop as you compute them, or you could populate a data frame with your calculations and then plot them (this is what we did in the solution).

In [23]:

```
# ***Solution***

# we're going to create a data frame to hold all of the results.  Think of this
# as a represntation of the
results = pd.DataFrame(columns=['race', 'decile >=', 'true_positive_rate', 'false_positive_rate', '
positive_predictive_value', 'negative_predictive_value'])

for race in ['Caucasian', 'African-American']:
    df_for_race = df[df['race'] == race]
    y = df_for_race['two_year_recid']
    for thresh in range(1, 11):
        yhat = df_for_race['decile_score'] >= thresh
        true_positive_rate = ((y == 1) & (yhat == 1)).sum() / (y == 1).sum()
        false_positive_rate = ((y == 0) & (yhat == 1)).sum() / (y == 0).sum()
        if (yhat == 1).sum() == 0:
            positive_predictive_value = float('nan')
        else:
            positive_predictive_value = ((y == 1) & (yhat == 1)).sum() / (yhat == 1).sum()

        if (yhat == 0).sum() == 1:
            negative_predictive_value = float('nan')
        else:
            negative_predictive_value = ((y == 0) & (yhat == 0)).sum() / (yhat == 0).sum()

        results = results.append({'race': race,
                                  'decile >=': str(thresh),
                                  'true_positive_rate': true_positive_rate,
                                  'false_positive_rate': false_positive_rate,
                                  'positive_predictive_value': positive_predictive_value,
                                  'negative_predictive_value': negative_predictive_value}, ignore_i
dex=True)
results
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: RuntimeWarning: invalid value
encountered in long_scalars
```

Out[23]:

	race	decile >=	true_positive_rate	false_positive_rate	positive_predictive_value	negative_predictive_value
0	Caucasian	1	1.000000	1.000000	0.402439	NaN
1	Caucasian	2	0.852665	0.638987	0.473318	0.784404
2	Caucasian	3	0.736677	0.469388	0.513848	0.749503
3	Caucasian	4	0.642633	0.348346	0.554054	0.730284
4	Caucasian	5	0.526646	0.231527	0.605042	0.706796
5	Caucasian	6	0.410658	0.144265	0.657191	0.683146
6	Caucasian	7	0.294671	0.090077	0.687805	0.657012
7	Caucasian	8	0.203762	0.052780	0.722222	0.638520

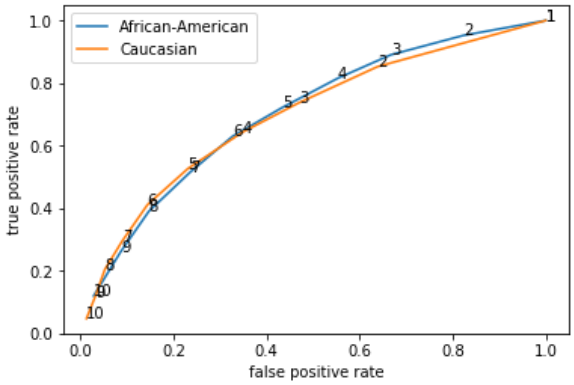
	race	decile >=	true_positive_rate	false_positive_rate	positive_predictive_value	negative_predictive_value
8	Caucasian	9	0.118077	0.032372	0.710692	0.619648
9	Caucasian	10	0.047022	0.013371	0.703125	0.605877
10	African-American	1	1.000000	1.000000	0.528414	NaN
11	African-American	2	0.952381	0.823141	0.564542	0.768229
12	African-American	3	0.890316	0.667266	0.599208	0.730263
13	African-American	4	0.815409	0.553357	0.622803	0.683486
14	African-American	5	0.721776	0.434053	0.650748	0.644809
15	African-American	6	0.629749	0.327338	0.683111	0.618523
16	African-American	7	0.515784	0.236811	0.709345	0.584481
17	African-American	8	0.390048	0.147482	0.747692	0.555035
18	African-American	9	0.260567	0.086930	0.770570	0.524269
19	African-American	10	0.119315	0.028777	0.822878	0.496020

In [25]:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
legend_strings = []
for race, df_by_race in results.groupby('race'):
    plt.plot(df_by_race['false_positive_rate'], df_by_race['true_positive_rate'])
    for _, row in df_by_race.iterrows():
        ax.annotate(str(row[1]), (row[3], row[2]))
    legend_strings.append(race)

plt.legend(legend_strings)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

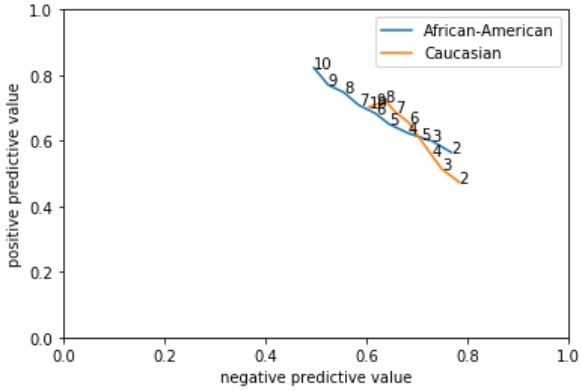


Notice how for a given threshold (annotated as text), the curves have vastly different false positive rates.

In [26]:

```
fig, ax = plt.subplots()
legend_strings = []
for race, df_by_race in results.groupby('race'):
    plt.plot(df_by_race['negative_predictive_value'], df_by_race['positive_predictive_value'])
    for _, row in df_by_race.iterrows():
        ax.annotate(str(row[1]), (row[5], row[4]))
    legend_strings.append(race)

plt.legend(legend_strings)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('negative predictive value')
plt.ylabel('positive predictive value')
plt.show()
```



Notice how for a threshold of 4 or 5, the positive predictive values are almost identical.

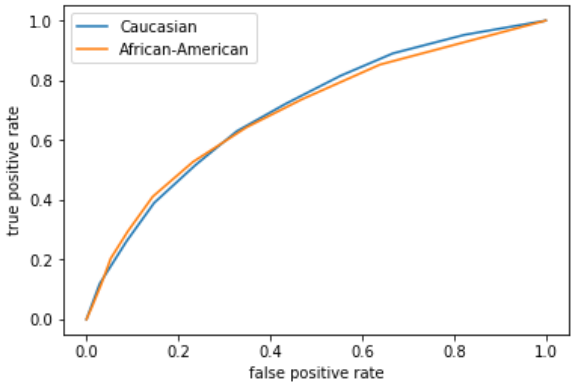
Sanity Check Using Sklearn

We can compare our calculations to the ROC curve (false positive rate versus true postive rate).

In [7]:

```
from sklearn import metrics

for race in ['African-American', 'Caucasian']:
    df_filtered = df[df['race'] == race]
    y = df_filtered['two_year_recid']
    scores = df_filtered['decile_score']
    fpr, tpr, thresholds = metrics.roc_curve(y, scores)
    plt.plot(fpr, tpr)
plt.legend(['Caucasian', 'African-American'])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```



Open in Colab

Sentiment Classification Using Naïve Bayes

Abstract

In this notebook you'll be implementing the Naïve Bayes algorithm and applying it to the task of classifying the sentiment of a movie review. We're going to take a mostly bottom-up approach here where we directly connect the math we've been learning with the algorithm. At the end we'll compare our results to sklearn's built-in implementation of the algorithm to both sanity check our own implementation and potentially allow those who want to to run more experiments with an implementation that has more options.

Sentiment Analysis

The Wikipedia Article on Sentiment Analysis provides the following definition for sentiment analysis.

> Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

In this notebook 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 (you'll understand that joke by the end of this notebook!)

The data consists 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.

Without further ado, let's download and parse the data into a data frame.

In [0]:

```
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
```

Let's look at the average sentiment to see what we are dealing with (1 is positive sentiment and 0 is negative)

In [0]:

```
df['sentiment'].mean()
```

Looks like we're dealing with a balanced set of positives and negatives.

Next, let's look at a particular review. To make the output look nicer, we'll create a new Pandas series with line wrapping.

In [0]:

```
# this takes a little while to run
```

```
reviews_wrapped = df['review'].str.wrap(80)
```

In [0]:

```
print(reviews_wrapped.iloc[20])
```

The Bag of Words Model

We know that in order to apply Naïve Bayes we need to convert each of our reviews into a vector of features. There are lots of different methods to convert text into vectors. In this notebook we'll be using a pretty basic (bugt suprisingly powerful) form of vectorization where we construct a feature vector with k entries (where k is the total number of unique words in the dataset) and for any particular review we set the corresponding entry to 1 if that word appears in the dataset and 0 otherwise. This representation is called the ***bag of words*** since the encoding of the text into features is independent of where the words occur in the text (you could shuffle the words in the review and still have the same feature vector). The Wikipedia article on Bag of Words has more information.

Instead of writing our own code to convert from text to a bag of words representation we're going to use scikit learn's built-in count vectorizer. Before we apply it to the data, let's apply it to toy dataset to help you better understand the bag of words model.

In [0]:

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

# the binary feature makes it so only the presence or absence of the word is
# returned (rather than the count)
vectorizer = CountVectorizer(binary=True)
toy_data = ['This is review one.  It has some words.', 'This is review two.  It also has some
words.']
vectorizer.fit(toy_data)
X_toy = vectorizer.transform(toy_data)
X_toy
```

The first thing to notice about the data is that it's stored in a sparse matrix format. A sparse matrix is useful when the elements of your matrix are mostly zeros. If you have a lot of unique words, but each piece of text only contains a small fraction of those words you will end up with a sparse matrix. In this notebook we'll be limiting the size of our vocabulary and converting the matrix to dense. This is to ease implementation. If you want to leave things as a sparse matrix, please feel free to do so.

In [0]:

```
print("feature vectors", X_toy.todense())
print(vectorizer.get_feature_names())
```

Notebook Exercise 1

Referencing back to what you know about the bag of words model and our toy data, explain why the vectors look the way they do. Make up your own toy data (or add to ours) and see if the results make sense.

Vectorizing the Whole Dataset

Now that you have a general idea what bag of words is all about, let's apply it to our movie reviews. To make our lives easier we're only going to include words in our feature vector if they occur in at least 100 reviews. Doing this will help with overfitting (although next assignment we will be learning another technique to deal with this). While we're at it we'll also convert the sentiment labels to a numpy array.

In [0]:

```
vectorizer = CountVectorizer(binary=True, min_df=100)
vectorizer.fit(df['review'])
X = vectorizer.transform(df['review']).todense()
y = np.array(df['sentiment'])
print("X.shape", X.shape)
print("y.shape", y.shape)
```

As a quick intuition builder, let's look at a word we think would probably differ across sentiment values.

In [0]:

```
terrible_index = vectorizer.get_feature_names().index('terrible')
print("terrible occurs in", X[y==1, terrible_index].mean(), "for Y=1")
print("terrible occurs in", X[y==0, terrible_index].mean(), "for Y=0")
```

Sorta makes sense (but please someone do some looking into what those reviews are where it terrible appears and it is Y=1)

The General Form of Naïve Bayes

Last Assignment we showed how the Titanic problem could be solved using the Naïve Bayes Model. Specifically we computed what's known as the odds ratio.

$$\frac{p(\ \ |\ \ )p(\ \ =1|\ \ )p(\ \ |\ \ )p(\ \ )}{p(\ \ |\neg\ \ )p(\ \ =1|\neg\ \ )p(\ \ |\neg\ \ )p(\neg\ \ )}$$

Recall that meant young passenger, represented fare class, represented "is male", and represented survival.

Hopefully it is pretty clear that while we derived the formula in the specific case of the Titanic model, the logic we applied is completely general. If we assume that we are doing binary classification provided values $x_1, x_2, \ldots, x_d$, then the odds ratio can be written in the following way.

$$\frac{p(Y=1|X_1=x_1, X_2=x_2, \ldots, X_d=x_d)}{p(Y=0|X_1=x_1, X_2=x_2, \ldots, X_d=x_d)} = \frac{p(Y=1)\times p(X_1=x_1|Y=1)\times \ldots \times p(X_d=x_d|Y=1)}{p(Y=0)\times p(X_1=x_1|Y=0)\times \ldots \times p(X_d=x_d|Y=0)}$$

We also learned last assignment that if this odds ratio is greater than 1, we should predict positive. While the odds ratio is a totally valid way to attack the problem, it can be numerically unstable. Therefore, most people use the log odds ratio instead (you just hit both sides with a log).

Notebook Exercise 2

Compute the log odds ratio for the Naïve Bayes algorithm by taking the log of both sides of the preceding equation. Simplify as much as possible using properties of logarithms. What must be true of the log odds ratio in order to predict $Y=1$?

Fitting the Parameters of the Model

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 strategy for fitting these parameters will be the same as was described in the previous assignment (we'll see a more formal justification of why this works in the next assignment). In order to estimate a probability, we'll just count the number of times the event occurs across the dataset.

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 esitmated $p(X_i=1|Y=0)$ (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 (hopefully that makes sense given the definition of $p(A|B)=\frac{p(A,B)}{p(B)}$.

Notebook Exercise 3

Write a function that takes as input X and y and returns a tuple containing the following elements (in this order)

1. The probability of Y=1
2. The probability of Y=0
3. A vector where the ith entry represents $p(X_i=1|Y=1)$
4. A vector where the ith entry represents $p(X_i=1|Y=0)$

To help you out we've included a unit test and a function stub

To help you out we've included a unit test and a function stub.

In [0]:

```
def fit_nb_model(X, y):
    """ Fit the parameters of a Naive Bayes model given a bag of words model
        with binary counts (X) and class labels (y).

        Returns: a tuple with the following components in order
                  The probability of Y=1,
                  The probability of Y=0,
                  A vector where the $i$th entry represents $p(X_i=1|Y=1)$
                  A vector where the $i$th entry represents $p(X_i=1|Y=0)$

    >>> X = np.array([[1, 0, 1], [1, 0, 0], [0, 0, 1], [1, 1, 1]])
    >>> y = np.array([1, 0, 1, 0])
    >>> fit_nb_model(X, y)
    (0.5, 0.5, array([0.5, 0. , 1. ]), array([1. , 0.5, 0.5]))
    """
    return None

import doctest
doctest.testmod()
```

Inference

Once we have the model parameters fitted, we have to be able to do inference on new data. This will amount to computing our log odds ratio and seeing if it is greater than 0. If the log odds ratio is greater than 0, we return a predicted sentiment of 1, otherwise we return a sentiment of 0.

Before having you implement the inference code, let's split into a train and test set and use your code to fit the model.

In [0]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)
p_y_1, p_y_0, p_x_y_1, p_x_y_0 = fit_nb_model(X_train, y_train)
```

Notebook Exercise 4

Implement a function called `get_nb_prediction` that takes as input the NB model (`p_y_1, p_y_0, p_x_y_1, p_x_y_0`) and a dataset (X), computes the log odds ratio and returns a vector of predictions for that data.

To get you started we have a function stub.

Hint: you can write a nested loop with one over the data points and one over the features and compute log probabilities as you go. (you can speed this up using numpy features, but don't worry about computational speed)

In [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 """
    return None

# here is some test code that will call your model on the first 100 test points
# (for speed of development).
y_pred = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_test[:100,:])
```

Calculating Accuracy

Now we'll test our model on all of the test data and see how accurate it is.

In [0]:

```
y_pred = get_nb_predictions(p_y_1, p_y_0, p_x_y_1, p_x_y_0, X_test)
print("accuracy is", (y_pred == y_test).mean())
```

Sanity Check

Just to see if we're in the ball park, let's try scikit learn's built-in implementation of Naïve Bayes.

In [0]:

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
np.mean(y_pred == y_test)
```