

Steven Cifarelli
Prof. Feng Chen
ICSI431 Data Mining
Final Project Technical Report
May 13th, 2016

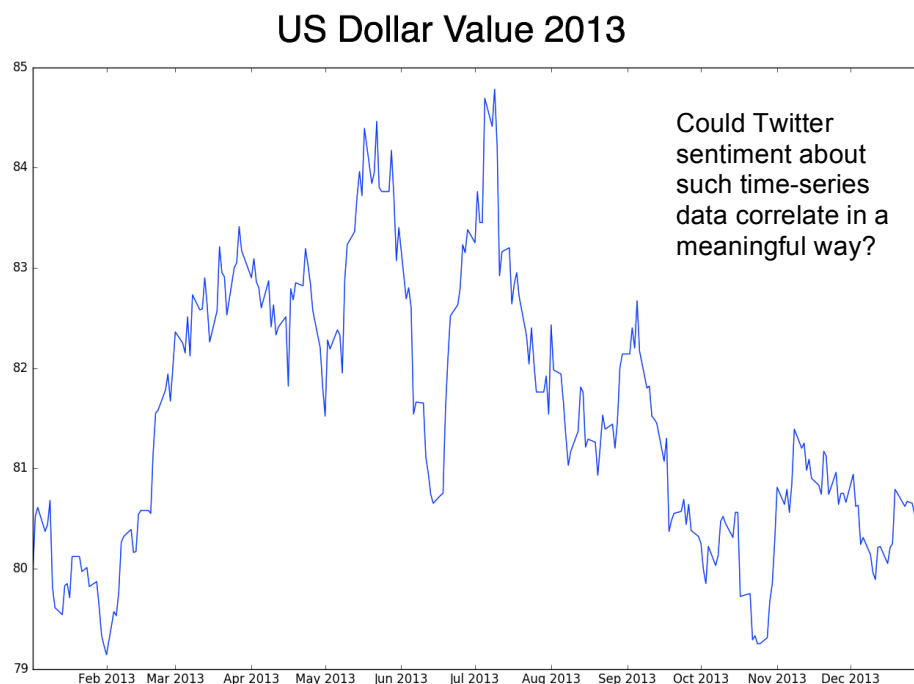
So for my part in this project, I was tasked with constructing the main Python program for the Naive Bayes algorithm. I had learned a bit about these algorithms in a class I took last semester with Prof. Kevin Knuth called "Bayesian Data Analysis" but it went more into the deep mathematical basis behind some of this stuff and when we did apply the algorithms it was all in MatLab, so I had to do this thing from the ground up— no old code was used.

The screenshot shows the scikit-learn documentation page for Naive Bayes. The header includes the scikit-learn logo, navigation links (Home, Installation, Documentation, Examples), a Google Custom Search bar, and a search button. A sidebar on the left contains links for 'Previous' and 'Up' versions, a note about the documentation version (0.17.1), and a list of sub-topics under '1.9. Naive Bayes'. The main content area is titled '1.9. Naive Bayes' and contains a paragraph explaining that Naive Bayes methods are based on applying Bayes' theorem with the 'naive' assumption of independence between every pair of features. It then presents the relationship:
$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$
 followed by the text 'Using the naive independence assumption that' and the equation:
$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$
 and the text 'for all i , this relationship is simplified to' and the equation:
$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)}$$
 followed by the text 'Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:' and the equation:
$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$
 followed by a downward arrow and the equation:
$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

I begin by researching what Bayesian methods were available through SKLearn, and I was happy to learn that that library does have a Naive Bayes component built in. However, using it proved to be a more difficult task, and I was having trouble understanding what it was doing, because so much of the functionality in that library of algorithms is buried under the hood. So after some time of trying to plug our data into the algorithm (even starting with examples online), I decided to

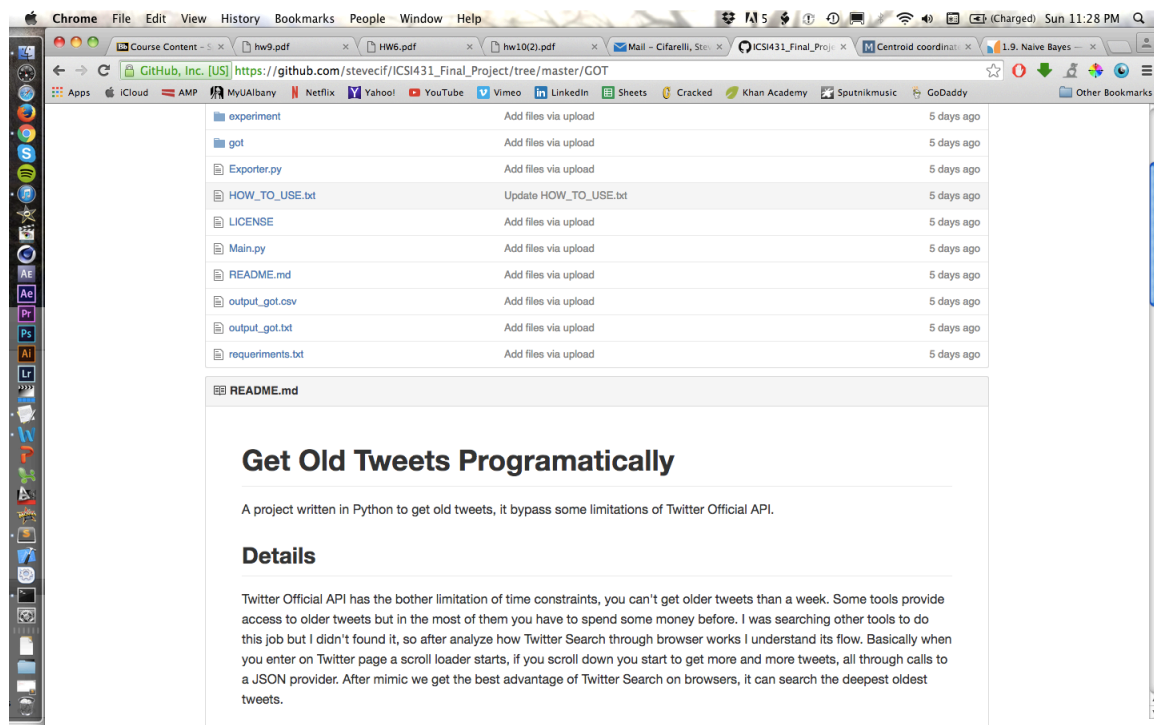
look around online to see if there were any tutorials about designing Bayesian algorithms from scratch (the more raw Python the better).

To my great luck, I immediately discovered just such a tutorial, and proceeded to download and play around with some of their example code and data-sets. After figuring out what most of their code was doing, I decided to try plugging our data sets into it to see what happens. Of course, it didn't like our data sets and the program suddenly became error prone. I decided to take a lot of what I'd learned from these tutorial code sets and apply them as I designed a new program from the ground up. This new program was initially intended to match the days in our data sets where the rise in stock price correlated with the rise in Twitter sentiment about the subject.



The group and I sat down to have a conversation about how best to structure the data for this project. Brian wanted to stick to the JSON format that we had used for tweets in previous assignments, but the issue with this was the Twitter REST API couldn't return tweets older than a week, which could not serve our purposes. In an attempt to resolve this issue, I found a program on GitHub that

an independent developer was freely distributing called Get Older Tweets (GOT), which does exactly what its title implies. The issue however was the program was designed to output the tweets in a CSV format, which does not lend itself to JSON functionality. I attempt to modify the code to output the tweets in a .txt file, line by line, in a JSON/dictionary-like format. Outputting this file was easily enough accomplished, but JSON stubbornly refused to read the tweets no matter what we tried, because each tweet was technically a string. An eventual solution was achieved with the help of a Python command called "ast.literal_eval(x)", which can convert a string of text with the right syntax into an actual dictionary format. Although we were happy with the text file output of the GOT tweet-getter, we ultimately chose to write a few extra lines of code that shaved the tweets down to their dates and sentiment scores imported as a CSV file, since the stock price changes were already in a CSV format.



The initial block of code is a comment section listing our group and names, and explaining what our goal was with this program (lines 1-6). The second block of code is the importing of necessary libraries (lines 8-21). The third significant block of code is where all the functions are built for the Naive Bayes algorithm (lines 23-104). A few of these functions are written very closely to how I found

them in different tutorials, due to my not wanting to screw up their functionality in the process of recruiting them for the purposes of generating the central algorithm of our project (it was also interesting to see how one can build the equation for Standard Deviation right in Python without importing the "Math" library). I did have to make a few adjustments to these functions late in the programming process, as some of our data, when fed through the functions, were generating instances where floats were trying to be divided by 0. The solution proved to be a very tiny addition of "0.0000001" to the zero-numbers that were clogging up the works (a double checking of the outputs revealed this step to have no significant impact on the output predictions made by the program).

```
106 # Pulls in the stock market data CSV (excel file) and stores the date information into "dates" array, and the closing price information into "prices" array.
107 df = pd.read_csv('./data/stocks.csv')
108 dates = df['Date']
109 prices = df['Price']
110 stock_data = [list(i) for i in zip(dates, prices)]
111 stock_data = stock_data
112 # Pulls in Tweet data, which is strictly the date and sentiment score.
113 tweet_data = []
114 for line in open('./data/tweets.txt'):
115     tweet_data.append(line)
116 tweet_data = tweet_data
117
118 # Refines imported data and splits it into four arrays by type, which are "t_dates", "s_dates", "prices", and "scores".
119 s_dates = []
120 prices = []
121 for x in range(len(stock_data)):
122     date = stock_data[x][0]
123     s_dates.append(date)
124     price = stock_data[x][1]
125     prices.append(price)
126 t_dates = []
127 scores = []
128 for line in tweet_data:
129     tweet = ast.literal_eval(line)
130     date = tweet[0]
131     t_dates.append(date)
132     score = tweet[1]
133     scores.append(score)
134 t_dates.reverse()
135 scores.reverse()
136
137 print "
```

The next block of code imports all the data (via lines of code 106-116), and then the block of code after that (lines 118-135) breaks the four variables of interest apart into four separate arrays: "s_dates", the stock dates; "prices", the stock prices; "t_dates", the Tweet dates; and "scores", the respective Tweet sentiment scores. Because the GOT program stored tweets starting at December 31st and then working back up to January 1st, a simple "t_dates.reverse()" and "scores.reverse()" set of functions was implemented to reverse the orders of the array so as to match the stock market arrays in date order. The block of code after this (lines 139-154), stores the days where the stock price goes up in an array ("sto_ups"), as well as the days when the stock price goes down ("sto_downs"), and the changes in stock value ("sto_changes"), which were averaged on line 154 ("avg_sto_change").

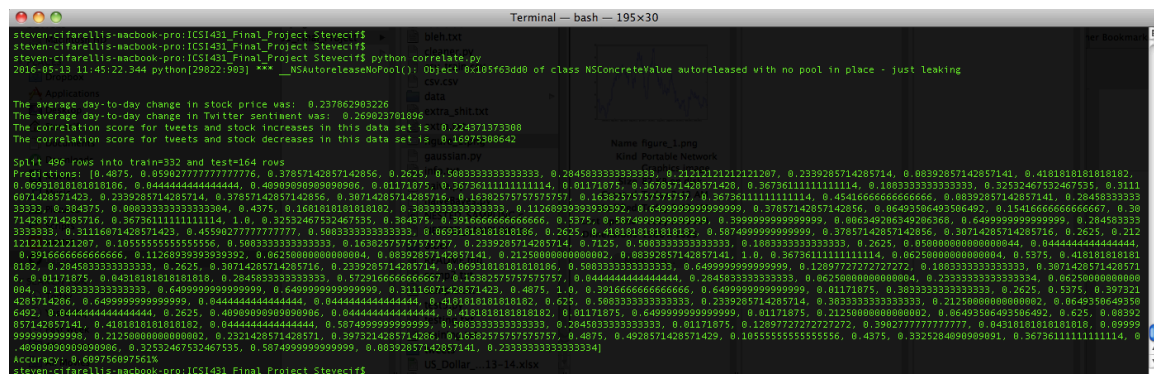
An exactly similar operation is carried out for the tweets in the next block (lines of code 156-171), where "twe_ups" records the days where the Twitter subject sentiment score went up, "twe_downs" the days where it went down, and "twe_changes" the amounts of the respective changes. The chunk of code after that, lines 173-179, creates two lists recording the days where either both the stock price and the subject sentiment score went up, or both the stock price and the subject sentiment score went down. Lines of code 181-183 calculate the "correlation scores" for the ups and downs, meaning the total number of days where the fluctuations matched out of all the dates used in the data set. Lines 191-195 constitute another larger comment break in the code wherein I recap the initial correlation scores, and what they mean for us as we moved forward with designing the program.

The next block of code (lines 197-215) converts the string contents of the "s_dates" and "t_dates" lists into actual Python datetime float objects, so that subsequent mathematical operations within the algorithm could be performed on the dates. The module immediately following (lines of code 217-223) creates an ID for the stocks based on their datetime objects. Then, the module after this one, lines of code 225-235, grabs the first "C" number of items in the lists of stock price changes and Twitter sentiment score changes and writes them to a CSV to be processed in the actual prediction function of the algorithm. Originally it was assumed that the algorithm would need the corresponding datetime IDs for this data, but after some practice with the algorithm on a simpler dataset, I realized it was only necessary to write the stock price and twitter score changes to the CSV. Finally, the last module of the naiveBayes.py program, lines of code 237-249, imports the CSV that was exported in the previous module, uses a "splitRatio" function to store about 2/3rd's of the data in the array "trainingSet", and 1/3rd of the data in the array "testSet", as it's generally considered good practice to make the training data set twice as large as the testing data set. The

model is then prepared with the "summarizeClass(x)" function, and predictions are tested using the "getPredictions(y,z)" function.

```
197 # This section revises the dates (which are stored as strings in their respective arrays), and converts them into datetime objects (which are floats).
198 new_sd = []
199 for x in s_dates:
200     date_obj = datetime.strptime(x, '%Y-%m-%d')
201     new_sd.append(date_obj)
202
203 new_td = []
204 for x in t_dates:
205     date_obj = datetime.strptime(x, '%Y-%m-%d')
206     new_td.append(date_obj)
207
208 sc_class = []
209 pos = 1
210 neg = 0
211 for x in scores:
212     if x > avg_sto_change:
213         sc_class.append(pos)
214     else:
215         sc_class.append(neg)
216
217 # This module creates an ID for the stocks based on their corresponding date.
218 def to_integer(dt_time):
219     return 1000*dt_time.year + 100*dt_time.month + dt_time.day
220
221 ids = []
222 for x in new_sd:
223     fixed = to_integer(x)
224     ids.append(fixed)
225
226 # This module writes the testing stock date/ID, the day to day change from each date, and the day-to-day changes of the sentiment scores to a CSV.
227 # The value of "C" is used to cull the datasets down to a manageable size.
228 C = 500
229 list1 = ids[:C]
230 list2 = sto_changes[:C]
231 list3 = twe_changes[:C]
232 rows = zip(list1, list2, list3)
233 with open('csv_train.csv', 'w') as fp:
234     a = csv.writer(fp, delimiter=',')
235     for row in rows:
236         a.writerow(row)
237
238 # Now that we've constructed a CSV from the day-to-day changes
239 filename = 'csv_train.csv'
240 splitRatio = 0.67
241 dataset = csvLoad(filename)
242 trainingSet, testSet = splitData(dataset, splitRatio)
243 print('Split {} rows into train={} and test={} rows'.format(len(dataset), len(trainingSet), len(testSet)))
244 # prepare model
245 summaries = summarizeClass(trainingSet)
246 # test model
247 predictions = getPredictions(summaries, testSet)
248 print('Predictions: {}').format(predictions)
249 accuracy = getAccuracy(testSet, predictions)
250 print('Accuracy: {}%').format(accuracy)
```

After doing some further reading online, I resolved that it was not enough to merely make predictions. Many other researchers who had conducted a similar experiment were also trying to gauge the accuracy of their predictions, so I went back and did a little research and found a nice way in Python of assessing the accuracy and implemented a function accordingly. This measure of accuracy is the last data-point that gets printed to the console when naiveBayes.py is run.



```
Terminal - bash - 195x30
steven@cfarellis-macbook-pro:~/ICS1481_Final_Project/SteveCi$ python correlate.py
steven@cfarellis-macbook-pro:~/ICS1481_Final_Project/SteveCi$ python correlate.py
2016-05-13 11:45:22.344 python[2902:903] *** _NSAutoreleaseNoPool(): Object 0x105f63d00 of class NSConcreteValue autoreleased with no pool in place - just leaking

The average day-to-day change in stock price was: 0.237062903226
The average day-to-day change in Twitter sentiment was: 0.26023701096
The correlation score for tweets and stock increases in this data set is: 0.224371373308
The correlation score for tweets and stock decreases in this data set is: 0.16975308642

Split 496 rows into train=332 and test=164 rows
Predictions: [0.4875, 0.059027777777777776, 0.37857142857142856, 0.2625, 0.5083333333333333, 0.20450333333333333, 0.2121212121212121, 0.239285714285714, 0.00392857142857141, 0.4181818181818182, 0.0693181818181818, 0.0444444444444444, 0.4899999999999999, 0.01171875, 0.3673611111111111, 0.01171875, 0.3678571428571428, 0.3673611111111111, 0.1803333333333333, 0.3252467532467535, 0.3111807428571428, 0.239285714285714, 0.37857142857142856, 0.30714285714285715, 0.16302575757575757, 0.16383333333333333, 0.3673611111111111, 0.45416666666666666, 0.00392857142857141, 0.2848333333333333, 0.304375, 0.000333333333333334, 0.4375, 0.1681818181818182, 0.3033333333333333, 0.11260000000000002, 0.6499999999999999, 0.37857142857142856, 0.06493506493506492, 0.15416666666666667, 0.30714285714285716, 0.3673611111111111, 1.0, 0.3252467532467535, 0.384375, 0.39166666666666666, 0.5375, 0.5874999999999999, 0.3999999999999999, 0.006349286349286368, 0.6499999999999999, 0.2845833333333333, 0.3116071428571429, 0.45902777777777777, 0.5083333333333333, 0.0693181818181818, 0.2625, 0.4181818181818182, 0.5874999999999999, 0.37857142857142856, 0.30714285714285716, 0.2625, 0.2121212121212121, 0.10155555555555556, 0.5083333333333333, 0.16302575757575757, 0.239285714285714, 0.7125, 0.5083333333333333, 0.1803333333333333, 0.2625, 0.65000000000000004, 0.0444444444444444, 0.39166666666666666, 0.11260000000000004, 0.06250000000000004, 0.08392857142857141, 0.21250000000000002, 0.00392857142857141, 1.0, 0.3673611111111111, 0.06250000000000004, 0.5375, 0.4181818181818182, 0.2845833333333333, 0.2625, 0.30714285714285716, 0.239285714285714, 0.0693181818181818, 0.5083333333333333, 0.6499999999999999, 0.12897272727272727, 0.1803333333333333, 0.30714285714285716, 0.01171875, 0.04318181818181818, 0.2845833333333333, 0.5729166666666667, 0.16302575757575757, 0.0444444444444444, 0.2845833333333333, 0.06250000000000004, 0.23333333333333334, 0.06250000000000004, 0.1803333333333333, 0.6499999999999999, 0.6499999999999999, 0.31118071428571429, 0.4875, 1.0, 0.39166666666666666, 0.6499999999999999, 0.01171875, 0.3833333333333333, 0.2625, 0.5375, 0.3973214285714286, 0.6499999999999999, 0.0444444444444444, 0.0444444444444444, 0.4181818181818182, 0.625, 0.5083333333333333, 0.239285714285714, 0.3833333333333333, 0.21250000000000002, 0.06493506493506492, 0.6492, 0.0444444444444444, 0.2625, 0.4899999999999999, 0.0444444444444444, 0.4181818181818182, 0.01171875, 0.6499999999999999, 0.06493506493506492, 0.625, 0.08392857142857141, 0.4181818181818182, 0.0444444444444444, 0.5874999999999999, 0.5083333333333333, 0.2045033333333333, 0.01171875, 0.12897272727272727, 0.39827777777777777, 0.04318181818181818, 0.09999999999999999, 0.21250000000000002, 0.2321428571428571, 0.3973214285714286, 0.16302575757575757, 0.4875, 0.4928571428571428, 0.10555555555555556, 0.4375, 0.33252848989898991, 0.3673611111111111, 0.48989898989898986, 0.3252467532467535, 0.5874999999999999, 0.08392857142857141, 0.23333333333333334]
Accuracy: 0.609756097561
steven@cfarellis-macbook-pro:~/ICS1481_Final_Project/SteveCi$
```