Hacettepe University Department of Computer Engineering

BBM 409: Introduction to Machine Learning Lab.

Name: Denizkaan Araci

Number: 21426606

Subject: Sentiment Analysis

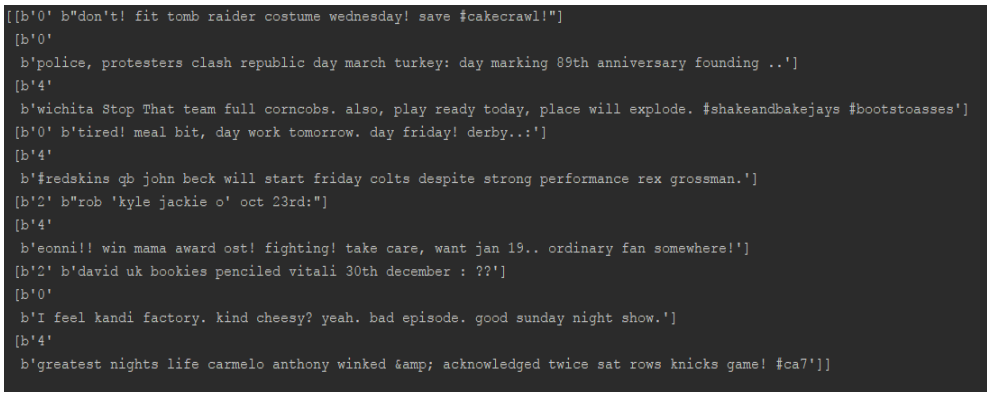
Advisors: Aykut Erdem, Levent Karacan, Tuğba Erdoğan

Problem

In this project, we will try to predict the sentiment of tweets with implementing Naive Bayes algorithm. And while implementing, we will use Bag of Word model with three options. These are unigram, bigram and both. We will train the machine with training data. Then machine predicts given test tweet's sentiment.

Data Analysis

Our training dataset contains 4654 classified tweets with 2 fields. Our validation(test) dataset contains 1000 classified tweets with 2 fields. First field shows tweet's sentiments (b'0' = negative, b'2' = neutral, b'4' = positive). Second field shows tweet part.



```
[[b'0' b"don't! fit tomb raider costume wednesday! save #cakecrawl!"]
 [b'0'
  b'police, protesters clash republic day march turkey: day marking 89th anniversary founding ..']
 [b'4'
  b'wichita Stop That team full corncobs. also, play ready today, place will explode. #shakeandbakejays #bootstoasses']
 [b'0' b'tired! meal bit, day work tomorrow. day friday! derby..:']
 [b'4'
  b'#redskins qb john beck will start friday colts despite strong performance rex grossman.']
 [b'2' b"rob 'kyle jackie o' oct 23rd:"]
 [b'4'
  b'eonni!! win mama award ost! fighting! take care, want jan 19.. ordinary fan somewhere!']
 [b'2' b'david uk bookies penciled vitali 30th december : ??']
 [b'0'
  b'I feel kandi factory. kind cheesy? yeah. bad episode. good sunday night show.']
 [b'4'
  b'greatest nights life carmelo anthony winked &amp; acknowledged twice sat rows knicks game! #ca7']]
```

First 10 lines of the dataset are represented like this.

Solution

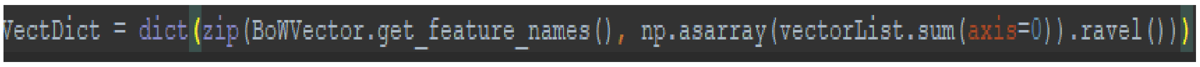
At first, I tried to create a proper dictionary for training dataset. For creating a Bag of Words dictionary, I used CountVectorizer from the scikit-learn library as mentioned in the PDF.

CountVectorizer helps us for creating bag of words implementation. It has a token pattern, minimum document frequency, maximum document frequency, stop words, convert lowercase etc. parameters. I used these parameters in CountVectorizer.

For decreasing the noise of data, I made some regulations. I deleted whitespaces, punctuations from tweets. Then I convert every word to lowercase.

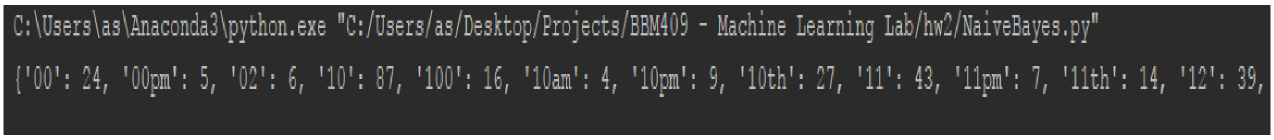
With these parameters, CountVectorizer runs with my tweet list. After that, I used CountVectorizer's fit_and_transform function to make a matrix. Each matrix row represents one tweet. Each matrix column represents each feature name(splitted word). By the way, each column sum becomes our word count. If the tweet has that word, the column is 1 one. If not, it's 0.

At this point, I create a Bag of Words dictionary with this created matrix and feature name. It becomes really easy to find how many words in the file and word count. For this, I combined CountVectorizer.get_feature_names and transformed matrix. Then it creates a dictionary.



```
VectDict = dict(zip(BoWVector.get_feature_names(), np.asarray(vectorList.sum(axis=0)).ravel()))
```

I split our training data according to their sentiments and save their vectorized forms one by one. Then I create a negative, neutral and positive dictionary to make calculations easy.



```
C:\Users\as\Anaconda3\python.exe "C:/Users/as/Desktop/Projects/BBM409 - Machine Learning Lab/hw2/NaiveBayes.py"
{'00': 24, '00pm': 5, '02': 6, '10': 87, '100': 16, '10am': 4, '10pm': 9, '10th': 27, '11': 43, '11pm': 7, '11th': 14, '12': 39,
```

Dictionary becomes like this

I smoothed algorithm using min_df = 4 and max_df = 150. It deletes too rare or too common words from the matrix. Then dictionary has 2623 different words and 32425 words in total.

After creating Bag of Words dictionaries, I started to calculate accuracy. For that, I used the formula from lecture slides which we used in Naive Bayes example.

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

I parsed each validation tweet with CountVectorizer again. Then I create a dictionary for each tweet as I used in training data. It helps with parsing and counting validation tweet data.

After creating a dictionary, I calculate the negative, neutral and positive probabilities of each tweet. For this calculation, i calculate every word conditional probability in the tweet. Then multiplied them with their counts in the word.

After that, if tweet's negative probability is maximum, then machine says it should be a negative tweet. If tweet's neutral probability is maximum, then machine says it should be a neutral tweet. if tweet's positive probability is maximum, then machine says it should be a positive tweet.

In the end, comparing all calculated probabilities with given validation test data, I get some results for Unigram, Bigram and Both.

Note: Due to the train data size, I couldn't use stop words or some delimiters. Because when I used them, accuracy extremely decreases.

Results

For calculations, I used some delimiters, parameters and stop words. It affects the accuracy because of decreasing the noise of data.

Results with changing options. Options are the range of min_df and max_df.

min_df: ignore terms that have a document frequency strictly lower than the given threshold.

max_df: ignore terms that have a document frequency strictly higher than the given threshold.

Results of Unigram

Limit	Accuracy	Time
Min_Df = None | Max_Df = None	49.0%	19.55
Min_Df = None | Max_Df = 150	55.60%	18.8
Min_Df = 2 | Max_Df = None	51.1%	10.2
Min_Df = 2 | Max_Df = 150	55.2%	9.86
Min_Df = 4 | Max_Df = None	51.8%	5.6
Min_Df = 4 | Max_Df = 150	55.4%	5.37
Min_Df = 10 | Max_Df = None	49.3%	2.51
Min_Df = 10 | Max_Df = 150	51.7%	2.50

Best accuracy of unigram: 55.4%

Best limits: 4 - 150

Results of Bigram

Limit	Accuracy	Time
Min_Df = None | Max_Df = None	45.0%	54.56
Min_Df = None | Max_Df = 150	45.2%	54.66
Min_Df = 2 | Max_Df = None	44.57%	11.57
Min_Df = 2 | Max_Df = 150	44.0%	11.92
Min_Df = 4 | Max_Df = None	40.3%	2.09
Min_Df = 4 | Max_Df = 150	40.3%	2.14
Min_Df = 10 | Max_Df = None	35.9%	0.8
Min_Df = 10 | Max_Df = 150	35.9%	0.92

Best accuracy of bigram: 45.2%

Best limits: None – 150

Results of Both(Unigram and Bigram)

Limit	Accuracy	Time
Min_Df = None | Max_Df = None	55.3%	154.08
Min_Df = None | Max_Df = 150	55.5%	149.8
Min_Df = 2 | Max_Df = None	49.7%	43.29
Min_Df = 2 | Max_Df = 150	55.4%	41.59
Min_Df = 4 | Max_Df = None	52.5%	12.94
Min_Df = 4 | Max_Df = 150	56.01%	12.18
Min_Df = 10 | Max_Df = None	49.4%	4.42
Min_Df = 10 | Max_Df = 150	52.0%	4.27

Best accuracy of Both(Unigram and Bigram): 56.1%

Best limits: 4 – 150

After all calculations, we can see changing the limitations of frequency effects results and runtime of the algorithm. Due to the training data size, accuracy is less than my estimates. But if the training data size increases, the algorithm can work accurately.

References

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

http://nbviewer.jupyter.org/github/twistedhardware/mltutorial/blob/master/notebooks/Lesson%206%20-%20Features%20Extraction.ipynb

http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html