School of EECS

Machine Learning: CS534

Naive Bayes classifier

Professor: Xiaoli Fern

Authors: Group_JEK
Jialin Yuan * 932-983-933 * 37%
Eugene Soe * 932-981-978 * 38%
Kaibo Liu * 932-976-427 * 26%

Oct. 17th, 2016

Overview

File List & Introduction:

- 1. RunMain.py #excute **python RunMain.py** to run
- 2. LoadData.py & OutputData.py # input and output interface.
- 3. NaiveBayes.py # building model to learn P(y) and P(x|y) from a given training set, and predict P(y|x) for a test data.
- 4. Improvement.py # Identify important features.
- 5. Bonus.py # the implementation to improve performance of classifier.
- 6. clintontrump-data # training and test data
- 7. Result # experiment result, for all of the project.
- 8. report.pdf

This report is divided to three part according the assignment: Baic Implemendation, Prior and Overfitting, and Boners.

- In the first part, we gave answer to how we use log to do the classification in our implementation, what's the test accuracy of Bernoulli and Multinomial.and which tweets are more confused than the other and why. We also ranked the top 10 words with the highest probability in each class (Trump or Clinton), which turns out to be common normal words being helpless for classification, and gives an explanation.
- In the second part, we focused on Multinomial model and explored the influence of the *Dirichlet* in Laplace smoothing, found that a too small or too big value are both not good choice in learning. To achieve the best test accuracy, we suggested to set the $\alpha = 0.1$. Then we also tried some methods to identify important features, and we run some experiments to test their ability to measure the discrimitive power of words. Our experiments shows that PCA is a good method to describe the discrimitive power of words. Also we found that using important words to do classify could not guarantee a better test accuracy performance than that before we remove words, but it do guarantee the improvement of time cost performance.
- In the final part, we explored ways to improve the performance of our classifier. Learning from those misclassified tweets, we forced it to improve the conditional probability of some words, we achieved to improve the accuracy performance.

Basic Implementation

1. How to use Log

We take Naive Bayes Rule to do the classification. So, at first, we would learn the prior possibility P_y and the conditional probability P_{x_i}.
For Bernoulli:

$$P(\mathbf{x}|y) = \prod_{i}^{|V|}(P_{i|y}^{x_i}(1-p_{i|y})^{1-x_i})$$
$$\Rightarrow \log P(\mathbf{x}|y) = \sum_{i}^{|V|}(\log P_{i|y})$$

For Multinomial:

$$P(\mathbf{x}|y) = \prod_{i}^{|V|}(P_{i|y}^{x_i})$$
$$\Rightarrow \log P(\mathbf{x}|y) = \sum_{i}^{|V|}(x_i \log P_{i|y})$$

We can see from these two equations, taking log can help us to avoid the problem mentioned in the assignment: $P(\mathbf{x}|y)$ becomes too small after the production. What's more, taking log can help to simplify the implementation while we are doing the prediction.

$$P(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \tag{1}$$
$$\Rightarrow \log P(y|\mathbf{x}) = \log p(\mathbf{x}|y) + \log p(y) - \log p(\mathbf{x}) \tag{2}$$

Where $p(\mathbf{x})$ is the same for all classes. So we don't need to consider about it in prediction. Since Log is a **monotone increasing function**, it wouldn't change the relationship between class and the calculated possibility

2. Test Accuracy

In the experiment, we use MLE estimation to learn $P(y)$, and Laplace smoothing for learning $P_{i|y}$:
For Bernoulli:

$$P_{i|y} = \frac{numberoftweetsthatword_i appearsingroupy + 1}{numberoftweetsingroupy + 2}$$

For Multinomial:

$$P_{i|y} = \frac{thetimeofword_i appearsingroupy + 1}{numberofwordsingrpy + |\mathbf{V}|}$$

After we learned P(y), and $P(\mathbf{x}|y)$ for both Bernoulli and Multinomial, We use equetion(2) to do the prediction. Both for Bernoulli and Multinomial, we do the prediction for test set as **Algorithm 1**. We test *clintontrump.tweets.dev* both by Bernoulli Model and Multinomial Model, and save the predict result in $Predict.Bernoulli.1.dev$ and$Predict.Multinomial.1.dev$. The test accuracy is:

$$\text{Accuracy\_bernoulli} = 94.72\%$$
$$\text{Accuracy\_multinomial} = 94.10\%$$

Algorithm 1 Predict by Naive Bayes Rule

procedure PREDICT($docList, Py, Pxy$)
  $docCnt \leftarrow 0$
  $pdcHist \leftarrow NULL$
  while $docListisnotempty$ **do**
    $doc \leftarrow docList[docCnt]$
    $P(y = Trump|doc) \leftarrow \sum \log P(w_i indoc|y = Trump) + \log P(y = Trump)$
    $P(y = Clinton|doc) \leftarrow \sum \log P(w_i indoc|y = Clinton) + \log P(y = Clinton)$
    $pdcDoc \leftarrow argmax([P(y = Trump|doc), P(y = Clinton|doc)])$
    $pdcHist.append(pdcDoc)$
    $docCnt \leftarrow (docCnt + 1)$
  end while
  $accNum \leftarrow sum(pdcHist^T . * docLabel)$
  $accuracy \leftarrow float(accNum)/float(docCnt)$
  return $accuracy, pdcHist$
end procedure

3. Who is more confused

In order to find which tweets were more confused than the other, we build the 2×2 confusion matrix for both Bernoulli and Multinomial, showing as following.

(1). Confusion Matrix from Bernoulli

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	298	8
Clinton	26	312

(2). Confusion Matrix from Multinomial

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	296	10
Clinton	28	310

As it can be seen, tweets about Trump are more easily to be confused. In general, we consider there are three reasons:

- **the distribution of training set**. Observing the training data set, we find that we have more Clinton tweets than Trump's, which means $P(y = clinton) > P(y = trump)$. Thus, if a new tweets seems to be Clinton and Trump nearly equally when only considering of $P(\mathbf{x}|y)$, after we consider about the prior probability $P(y)$(referring to Part1-How to use Log), it would finally be classified to Clinton's group.

- **the nature of data**. We all know there could be some kind of 'noise' data in both training set and test set. These 'noise' could affect our learning result or the testing result, to make one group more easily class to another group.

- **drawback of algorithm** Here we use Laplace Smoothing to avoid situations that a new word (not included in training set) appears in test data. If there are lots of these kind of words in a test data, we might being confuse, and get more Clinton than Trump co-considering of the distribution of our training set.

4. Ranking Top 10 with the highest probability

Expection: Top 10 words with the highest probability might be the common words which appear frequently in all the documents, such as 'a', 'the', 'and', or '.'. Since those words do not have a great power to discriminate documents, we are better to seperate the words as considering learning time cost as well as prediction accuracy. Therefore, figuring out what is the most common words from our documents is an important task before learning our model.

Results: We captured the top 10 words with the highest probability estimated from each Bernoulli and Multinomial models.

- As we expected, common words such as 'a', 'the', or '.' are captured as top 10 words with the highest probability.

- Surprisingly, 'Hillary' and 'Trump' words are also one of the top words captured from classes. Since we assumed that the significant words can be the symbol words of each classes like 'Hillary' and 'Trump', this outcome is unexpected.

Explalnation:

- Since the words with high probability have a considerable impact on classifying documents, those words should not be common words appearing all over the documents. To find out those words, we relied on two different models; one is Bernoulli model which counts the number of documents a word appears, another is Multinomial model which counts the number of times a word appears in a class. No matter models, top 10 words are similar for both classes, except the rank order. We now have the most common words including text symbols (. , ! : ; "), preposition(to, and, in, of, in , to, for), and atricles (a, the). Because those words do not have any specific information for documents, we might be better to remove them when learning a model and classifying documents.

- The model type, either Bernoulli or Multinomial, does not play an important role in finding top words. It is because as a word appear all over the documents,

the total number of a word appearance will also increase. Therefore, both model generate quite similar top words for each classes.

- Just like we were wrong that presentative words, like 'Hillary' and 'Trump', will be good indications to classify documents, it actually turns out that presentative words can be common words. It gives us a lesson that effective features of samples may not related with representative features of classes, rather it could be other special words belonging only to a class. In this regard, we tried to improve our classification model.

rank	Trump	Hillary
1	!	.
2	.	to
3	,	,
4	the	the
5	to	a
6	and	”
7	in	and
8	:	of
9	—	Trump
10	a	Hillary

Table 1: Top 10 words from Bernoulli model

rank	Trump	Hillary
1	.	.
2	!	,
3	,	to
4	the	the
5	”	”
6	to	a
7	and	and
8	I	of
9	in	Trump
10	:	for

Table 2: Top 10 words from Multinomial model

Priors and overfitting

1. Different Laplace smoothing

As what we mentioned in **Basic Implementation**, we used Laplace smoothing for our MAP estimation of $P_{i|y}$ for both Bernoulli and Multinomial model. Now let's focus on the Multinomial model, and explore with different *Dirichlet* in order to learn its influence. The new model is as following:

$$P_{i|y} = \frac{thetimeofword_i appearsingroupy + \alpha}{numberofwordsingroupy + \alpha \times |\mathbf{V}|}$$

The α in our experiment comes from $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$, 0.1, 1, 10, 100, 1000, 10000]. And we construct a plot to show the relationship between the value of α and test accuracy in Figure 1

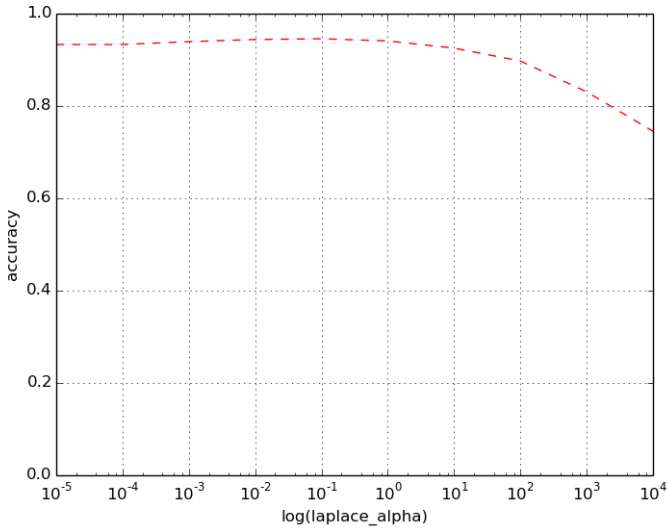

Figure 1: test accuracy with different Laplace smooth α

laplace smooth α	accuracy
10^{-5}	93.32%
10^{-4}	93.32%
10^{-3}	93.94%
10^{-2}	94.41%
0.1	94.57%
1	94.10%
10	92.55%
100	89.75%
1000	83.07%
10000	74.53%

Table 3: test accuracy with different Laplace smooth α

We achieve the best accuracy performance when $\alpha = 1$, the test result shows that the test accuracy becomes larger firstly as Laplace smoothing α increasing (while $\alpha \leq 0.1$). Then it starts to decrease as α keep on increasing (while $\alpha \geq 0.1$). We think the reason might be as following:

- When α is quite small, by increasing α, we increase the consideration of unknown words. Then the multiplied probability would not go to zero, we could achieve a better result of posterior probability, which means better prediction.
- When α reaches to a certain point (which is 0.1 in our experiment), keep increasing it, we introduce a big probability for unknown words, it would cause distraction on the probability distribution over features. Thus, the prediction might go to wrong.

2. Identifying important features

(1) Reduce common words with high probabilities, $\log P(\mathbf{x}|y)$

Expectation: As we observed the high probability words, the high probability words do not really have discriminative power for classifying documents, and those words are more likely trivial words. Therefore, we expected to have a higher accuracy result with reduced size of words.

Strategy: We pick the different number of top words with high probability from each model. For example, by looking at the probability estimated from Bernoulli, we picked words commonly appearing over the documents. On the other hand, we consider the frequency of a word in each class. Once we picked the top words, we learn our model with reduced words in two different types of models; Bernoulli and Multinomial. In our experimentation, we choose the number of top words as [10, 100, 1000, 5000].

Selected words w_i, based on two different probabilities:

$$\text{argmax}\, P_b(\mathbf{x}|y) = \prod_i^{|V|} (P_{i|y}^{x_i}(1 - p_{i|y})^{1-x_i}), \text{ from Bernoulli model}$$

$$\text{argmax}\, P_m(\mathbf{x}|y) = \prod_i^{|V|} (P_{i|y}^{x_i}), \text{ from Multinomial model}$$

Result: Unlikely with our expectation, however, the accuracy decreases as we removed top words more. It shows that although the common words seem play a trivial role to discriminate documents, it could be better to have all words as we can view the documents in a more broden way. On the other hand, learning time with both Bernoulli and Multinomial models are significantly reduced due to the lower number of features. Therefore, reducing features can make contribute to time performance, but it would considerably deteriorate the prediction accuracy. (See Fig. 2-4)

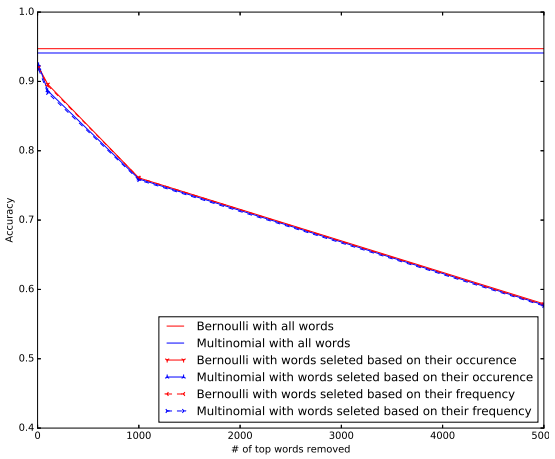

Figure 2: the test accuracy with reduced words

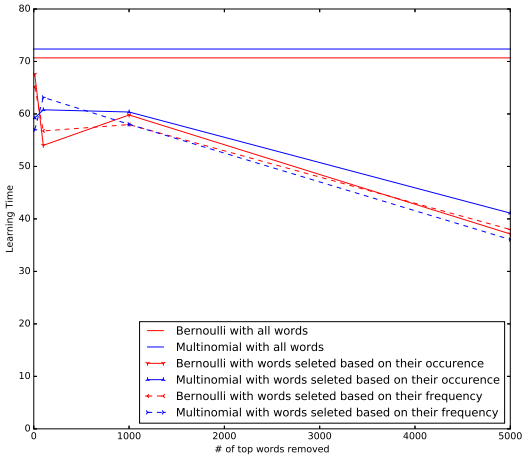

Figure 3: the learning time with reduced words

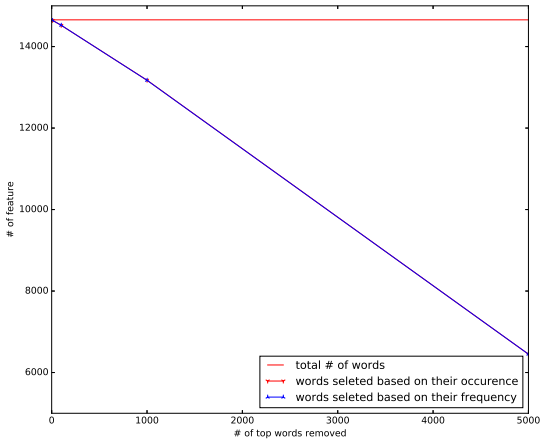

Figure 4: the feature number with reduced words

(2) Reduce balanced words with standard deviations close to 0 — PCA

Expectation: In our vocabulary list, we also have unused words and not common words which appear only once or few times from documents. Since those words cannot explain clearly what the document is about, we tried to eliminate those words appearing equally on both classes.

Strategy: Borrowing an idea from Principal Component Analysis (PCA), which transforms variables into principal components showing extreme variation, we also wanted to draw out the most variable words in the classes. To do that, we calculate STD (in our example, simply calculate the difference of probabilities of a word given two classes)

for all words and removed balanced words with some threshold. In our experiments, we set our threshold as [0.1, 0.3, 0.5, 0.7, 0.9].

Selected words x_i, based on two different variances from Bernoulli and Multinomial probabilities for two classes:

$$Var_b(x_i) = \parallel P_b(x_i|y_0) - P_b(x_i|y_1) \parallel$$
$$Var_m(x_i) = \parallel P_m(x_i|y_0) - P_m(x_i|y_1) \parallel$$

Results: Interestingly, in a multinational model, removing balanced words generates slightly better results comparing to prediction accuracy from multinational model with whole words. For Bernoulli model, reducing words makes a little lower accuracy, but the amount of change is small. The learning time is also largely shortened with smaller number of features. This experiments show the advantage of STD that it removes not only not occurred words but also isolated words, which appear once or twice, and also eliminate all balanced words like common words. Therefore, reducing words with STD aspects performs better than reducing words with high probabilities. (See Fig. 5-7)

(3) Reduce balanced words with lower TF/DIF

Expectation: Lastly, we borrowed an idea from Information Retrieval, TF/IDF, to capture important words discriminating documents well. TF measures term frequency, and IDF means a reverse way of document frequency. From previous experiments, we learned that high probability contains the common insignificant words as well as important words, and also that balanced words can deteriorate prediction accuracy. Therefore, TF/IDF is a good way to measure the importance of words by combining those two aspects; word frequency and variation on document frequency. Therefore, we expected that it would give better prediction results comparing to other approaches.

Strategy: In our experiments, we modified IDF in a slightly differnt way from the classic TF/IDF metric. Instead of using the total number of documents a word occurrs over all claases, we use the min number of documents the word appears from either of the classes. It is because high value of DF (before inversing) does not alreays mean the commonly occurence of a word. A special word beloing to a typical class can frequntly appear over all documents from one class, but not from another class. To prevent this difference, we took a min value of document numbers for the classes. We experimented with different TF/IDF threshold as [1, 2, 3, 4, 5].

Selected words x_i, based on TF/IDF values:

$$\text{TF/IDF} = \frac{\text{TF}}{\text{IDF}} = \frac{P_b(\mathbf{x}|y)}{min(P_m(x_i|y_0), P_m(x_i|y_1))}$$

Result: Unlikely our exception, TF/IDF generates somewhat lower prediction accuracy than accuracy with STD metrics. It maybe because we still have common words with high values of TF, and there might be unique words with low IDF, which not frequently appears. Those factors, being able to be filtered out by PCA, would

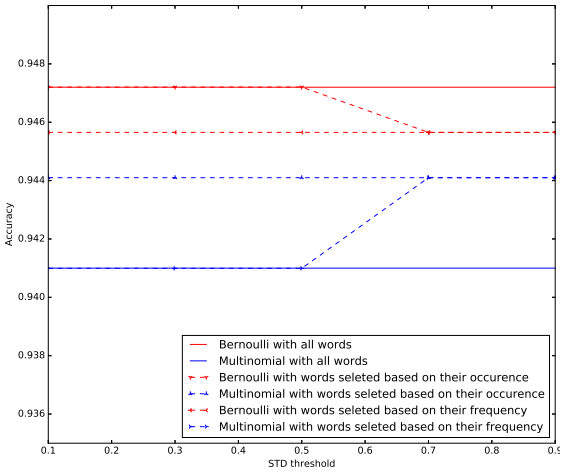

Figure 5: test accuracy with STD threshold

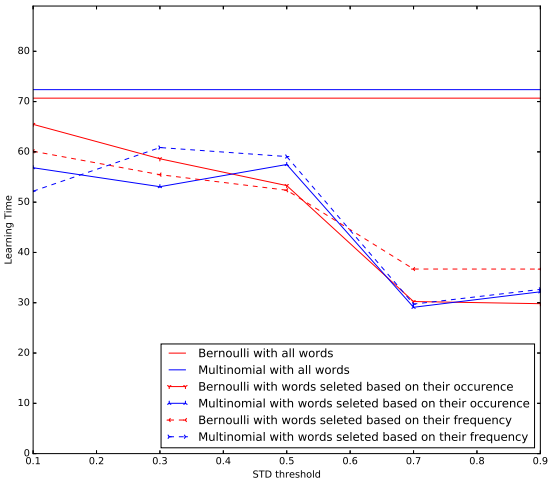

Figure 6: learning time with STD threshold

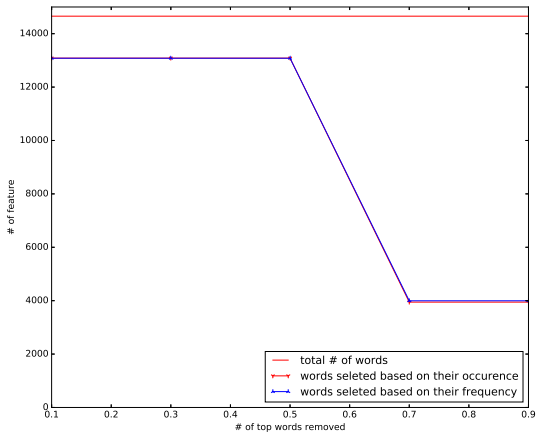

Figure 7: feature number with STD threshold

affect on our results, Nevertheless, this metric enable to capture significant words with high frequencies, but not common, so that it generates sufficiently reasonable prediction accuracy even with small size of features. Therefore, by using TF/IDF metric, we can capture the small number of important features with reasonable time performance and prediction accuracy. (See Fig. 8-10)

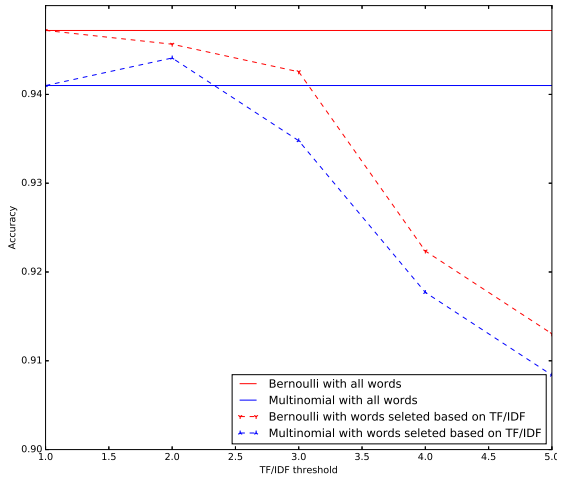

Figure 8: test accuracy with TF/IDF

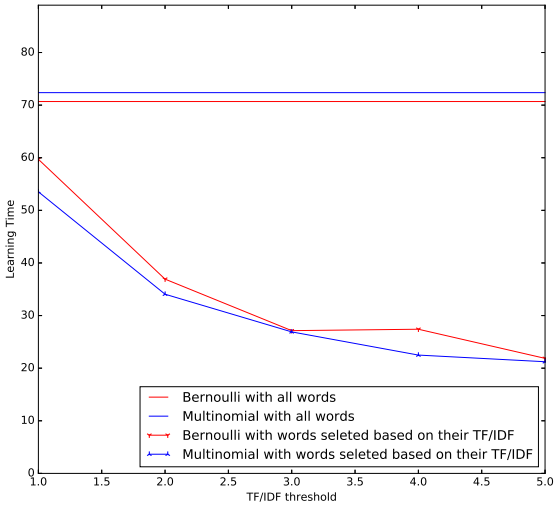

Figure 9: learning time with TF/IDF

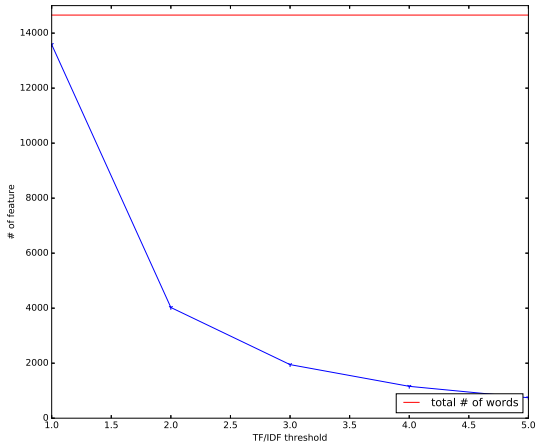

Figure 10: feature number with TF/IDF

Bonus

What else could you do to improve the performance of your classier?

For the case that basic implementation in Bernoulli Model with Laplace Smoothing, we get the confusion matrix, shown again as below.

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	298	8
Clinton	26	312

When looking into the misclassified tweets, we find something interesting. For the cases that class1(Label=HillaryClinton) is predict as class0(Label=realDonaldTrump), we extracted the 8 tweets. Some of them can be predicted better but some should be treated as noise because no bias of emotion or keywords in the tweet points to the neither name, even learning from their history. Below are the examples:

- line 162: Ready for the opening ceremony . U . S . A ! U . S . A ! https://t.co/...
- line 312: (It's only Wednesday .) https://t.co/IPkbU8xdaw
- line 349: Thank you , Connecticut . https://t.co/dEF36fIGyu
- line 592: A great day in East Harlem ,in photos . https://t.co/...

So we can't predict them correctly and can't improve our accuracy from these examples, for now. On the other hand, it seems that class0(Label=realDonaldTrump) is much more likely to be predict as class1(Label=HillaryClinton), maybe it's from Trump's straight character and wide use of words, leading to a confusion. From Trump's particular words and the hobby of creating topics with tags, we can refine our prediction for a better accuracy. For example, he likes to use "#NeverHillary", "#VoteTrump", or like "#VoteTrump2016", "#VoteTrumpNH", "#VoteTrumpNV", "#VoteTrumpSC", "#VoteTrumpMS", "#Trump2016", "#TrumpTrain", "#Trump4President", but **NOT** "#TrumpYourself". So we can add marks to these obvious tags to make direct prediction. Below are the result added to the basic implementation.

(1). Confusion Matrix in Bernoulli with tag marked

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	298	8
Clinton	26	312

$\Rightarrow$

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	301	8
Clinton	23	312

(2). Confusion Matrix in Multinomial with tag marked

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	296	10
Clinton	28	310

$\Rightarrow$

predicted label $\hat{y}$	true label y	
	Trump	Clinton
Trump	299	10
Clinton	25	310

And we get comparison of accuracy in Tab.4

Model	Basic Impl.	Tag Marked
Bernoulli	94.72%	95.19%
Multinomial	94.10%	94.57%

Table 4: Accuracy in basic implementation and tag marked optimization