Spam Filtering Algorithm Design

The project for CS559 Machine Learning @ Stevens, 2013 Fall. Directed by Prof. Jingrui He

Renjie Weng
Department of Computer Science
Stevens Institute of Technology
rweng@stevens.edu

Abstract

This project will design a Spam Filter based on the date set from Discovery Challenge workshop at ECML/PKDD 2006 in Berlin. I used two basic machine learning technologies learned from the class: Naive Bayes and Logistic Regression. And made some improvement based on practical implementation. Another improvement I have used is Self-Learning which augments the evaluation data on training set.

1 Introduction

This competition was held at German in 2006. I finished the Task A in this challenge. Many teams had already done good job on it. The evaluation criterion is the AUC value which I will explain a bit of it later on. The first ranked team has accessed 0.95 Average AUC during the challenge and up to 0.98 updated after the challenge. I get the Average AUC above 0.95, too. I have used two different classification algorithms, Bayes and Logistic Regression, which belongs to the two Machine Learning technique types respectively.

1.1 Material Provided by Discovery Challenge

The data provided by Discovery Challenge is statistic numbers. Give an example of a mail, this is the first line of the first mail in task_labeled_train.tf:

1 9:3 94:1 109:1 163:1

This line represents a spam email (starting with class label "1") with four words. The word ID of the first token is 9 and this word occurs 3 times within this email, indicated by ":3".

Only training data and tuning data are labeled on the 1st character of each mail, evaluation data are unlabeled with 0 on the 1st character of each mail.

Table 1: Documents Provided by Discovery Challenge

Task	File name	Data set size	Description	Usage
Task A	task_a_labeled_train.tf	4000 emails	Labeled training emails.	**train**
	task_a_u00_eval.tf ...	2500 emails	Unlabeled evaluation data: 3 inboxes from different users.	**test, get the average AUC**

	task_a_u02_eval.tf	each		
	task_a_labeled_tune.tf	4000 emails	Labeled training emails for parameter tuning. Feature representation corresponds only to file task_a_u00_tune.tf.	**build model,** **tune parameter for u01 & u02**
	task_a_u00_tune.tf	2500 emails	Labeled test emails of one user's inbox for parameter tuning. Feature representation corresponds only to file task_a_labeled_tune.tf.	**tune parameter for u00**

I can score my result by giving out the Average AUC value, which can be calculated by the program provided by Discovery Challenge and the true labeled documents comparing to my results.

All these files mentioned above are available on the website of Discovery Challenge 2006: http://www.ecmlpkdd2006.org/challenge.html .

1.2 AUC and ROC

ROC (Receiver Operating Characteristic) is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the total actual positives (TPR = true positive rate) vs. the fraction of false positives out of the total actual negatives (FPR = false positive rate), at various threshold settings.

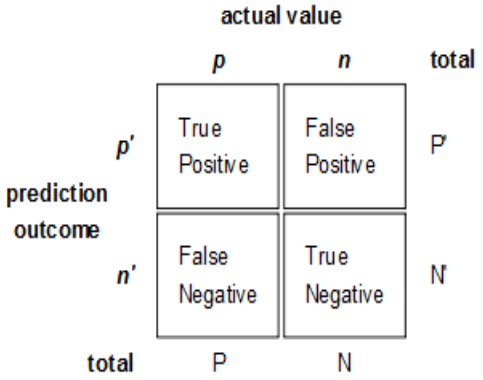

Figure 1: The four possible labels of predict result

Here gives a sample of ROC curves in Figure 2. Where X-axis, called Specificity, is False Positive rate = FP / [FP + TN]; Y-axis, called Sensitivity, is True Positive rate = TP / [TP + FN].

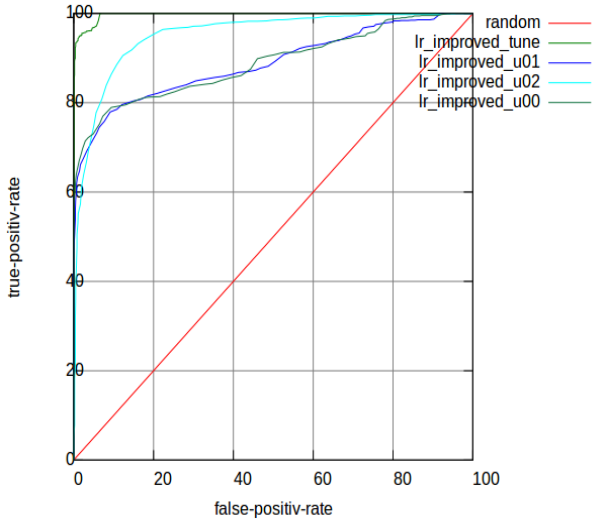

Figure 2: Sample of ROC curves

AUC (Area Under Curve): The AUC value is the area under the ROC curve. The AUC value will between 0 and 1, higher value means a better performance.

1.3 The Two Types of Classifiers

There are two types of Machine Learning Techniques [3]: Discriminative Model (Conditional Model) and Generative Model.

Discriminative Model is modeling the conditional probability distribution P(y|x), which can be used for predicting y from x. It includes Logistic Regression, SVM, etc.

Generative Model specifies a joint probability distribution over observation and label sequences. It includes Naive Bayes, etc.

I used Naive Bayes (NB) and Logistic Regression (LR) in my project.

2 Bayesian Filter

The basic algorithm of my Bayesian Filter is studied from Paul Graham's post "A PLAN FOR SPAM" [1]. I also do some improvement based on it to access higher AUC.

2.1 Basic Theory

Given an unlabeled mail, I can calculate the probability of how much it will be a Spam [7].

$$P = \frac{P(S|W1) * P(S|W2) * P(S)}{P(S|W1)P(S|W2)P(S) + (1 - P(S|W1))(1 - P(S|W2))(1 - P(S))}$$

, where P(S|Wi) is the probability of a mail to be Spam when it contains the word Wi.

Assuming P(S) = 0.5, then

$$P = \frac{P(S|W1) * P(S|W2)}{P(S|W1)P(S|W2) + (1 - P(S|W1))(1 - P(S|W2))}$$

In order to calculate P, I just need to know the P(S|W) value for each Word, which I can learn in training process from the labeled data.

$$P(S|W) = \frac{P(W|S)*P(S)}{P(W|S)*P(S)+ P(W|H)*P(H)}$$

, where P(W|S) is the static likelihood of a word existing in all Spam, which I call learned from training data. The same as P(W|H), H is the first Character or Healthy, means Non-spam. I can also assume P(S) = P(H) = 50% here (Actually, the provided data also has half number of Spam and half number of Non-spam.).

2.2 Implement Graham's Method

I implemented Graham's method in four steps.

Step 1. Learn the likelihood of each word existing in Spam and Non-spam from train_data or the 1st-half of tune_data;

e.g. There are 2000 Spam and 2000 Non-spam in train_data, the number of Spam which has word W1 is 1234, the number of Non-spam which has word W1 is 567. So, P(W|S) = 1234/2000, P(W|H) = 567/2000;

Step 2. I can now evaluate each mail from u01_eval, u02_eval, or the 2nd-half of tune_data;

For each word in a mail, I calculate its P(S|W), and then I sort all the P(S|W) of words in this mail by Decreasing order. And I just pick the first some of these words (e.g. 15, called interesting words) to calculate the Combining Probability of this mail. This is the probability of this mail to be a Spam.

e.g. P = P1P2…P15 / P1P2…P15 + (1-P1)(1-P2)…(1-P15), where P1 is the simple symbol of P(S|W1)

I can predict whether a mail is a Spam or not based on this probability P.

Step 3. There are some parameters not decided yet for evaluation. So the tune_data is used to decide those parameters. I cannot do this tuning process directly on u02_eval because if I did so, I would cheat based on test data. And, I cannot add tune_data into train_data because the word IDs of these two files are different.

Step 4. For u00, I have also a sample of u00_tune_data, which could be used to tune parameters only for u00_eval. With this different parameters, I also train on train_data and then evaluate on u00_eval.

The result of this basic implementation is shown in Table 2:

Table 2: AUC of Basic Implementation of NB

	tune	u01	u02	u00_tune	u00
AUC	0.998056	**0.899351**	**0.951194**	0.992355	0.869827
u00_tuned				0.996812	**0.894598**

The second line in Table 2 is results that evaluated with parameters tuned from file task_a_labeled_tune.tf. The third line is results that evaluated with parameters tuned from file task_a_u00_tune.tf. So this make sense that u00 has its individual features which could be learned separately.

And following Figure 3 is the roc curves in one plot. The red line is a random classifier; the green line of u00 is the curve for result of u00 after parameters tuned from task_a_u00_tune.tf.

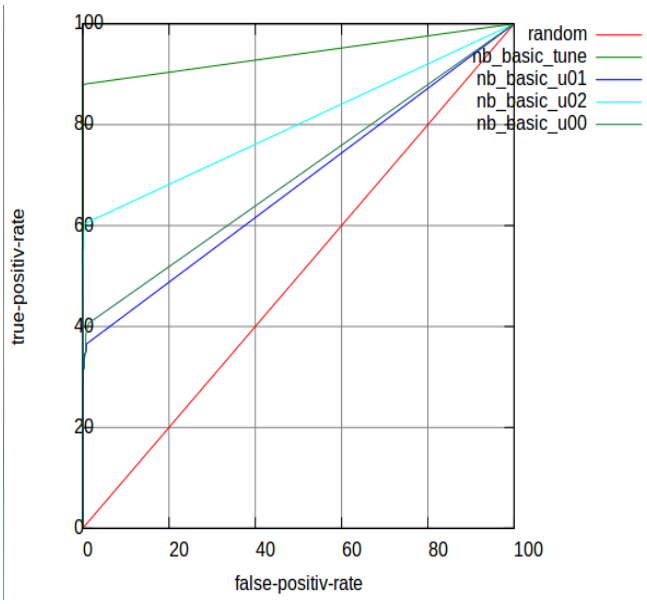

Figure 3: ROC curves of Basic NB

2.3 Improve Naive Bayes: Frequency & Anti-Spoof

When I statistic the likelihood, I only +1 time when a word exists in Spam or Non-spam. I have not taken advantage of the frequency information yet.

But if I simply add the true frequency value, it would also not reflect the real likelihood.

e.g. in task_a_labeled_train.tf there is a mail, -1 9:84 35:324 67:2 74:18 92:6 132:3

I think the useful frequency should distinguish the low value, but should not be too big when the value is high.

So I hope the frequency can be transformed to something like Figure 4.

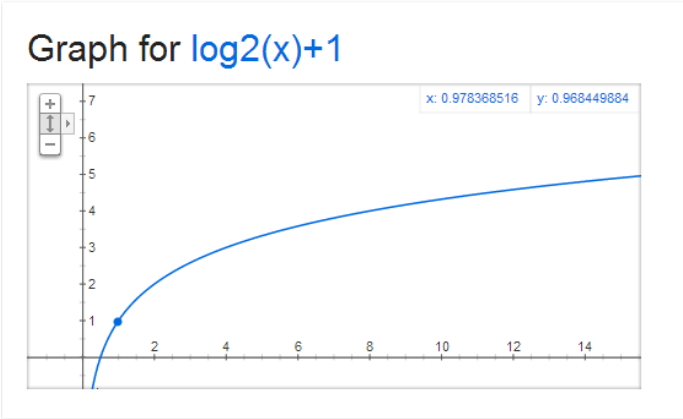

Figure 4: Graph for $y = \log_2 x + 1$

I choose 2 as the base of the logarithm for the reason that the conversion can distinguish 1 and 2. Here is the relation: 1 → 1; 2 → 2; [3,4] → 3; [5..8] → 4; etc.

However, there is a problem pointed out by Graham, "such an algorithm would be easy for spammers to spoof: just add a big chunk of random text to counterbalance the spam terms." [10]

I solve this problem by setting a maximum boundary for transformed frequency. When the transformed value is greater than this boundary, I just omit it for regarding this as a

spoofing.

This improvement works, and increases the AUC a bit.

Table 3: AUC of Improved NB

	tune	u01	u02	u00_tune	u00
AUC	0.998196	**0.912002**	**0.952196**	0.991569	0.874134
u00_tuned				0.991257	**0.901313**

I lost the parameters here, actually I can get 1% higher AUC here.

Figure 5 shows ROC curves for Table 3:

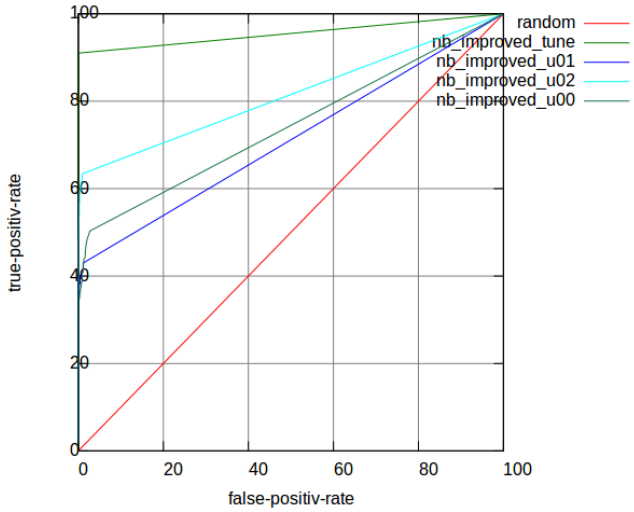

Figure 5: ROC curves of improved NB

So, for this part, I implemented Graham's method for Bayesian Filtering, and made a little improvement for taking frequency into consideration. By the way, I think Graham should have already taken frequency into account after published the first version of his Spam Filter, or he would not have mentioned those words about Spoof in the Better post [10].

3 Logistic Regression Filter

I studied this method from one of papers posted on Third Conference on Email and Anti-Spam (CEAS 2006) [2]. I implemented the basic Logistic Regression (LR) method followed by the conduct of it and it worked as good as the basic implementation of Graham's Method.

3.1 Concepts

I will learn a set of weights for each word in the body or header of the message. When a new message arrives, I find this list of words, and sum the weights associated with those words.

$$P(Y = \text{spam}|\bar{x}) = \frac{\exp(\bar{w} * \bar{x})}{1 + \exp(\bar{w} * \bar{x})}$$

, where $\bar{w}$ is the vector of weights, and $\bar{x}$ is a vector of 1's or 0's with a 1 in the position corresponding to each word.

The equation results a probability between 0 and 1. If the probability is over some threshold, I predict that the email is spam; otherwise predict the email is ham.

So, this is the equation I will use to evaluate how much is the probability of a new mail to be

Spam. A similar prediction as Naive Bayes Filter. And then I will learn the weights of every word in training process.

The actual training algorithm is very simple. Given as following:

$\bar{w} = 0$ // initialize weights to 0
for each $\bar{x}_i$, y_i
  P = $\exp(\bar{x}_i * \bar{w}) / (1 + \exp(\bar{x}_i * \bar{w}))$
  if (p>0.5)
    predict Spam;
  else
    predict Non-spam;
  if (y_i == 1)
    $\bar{w} = \bar{w} + (1 - p) * x_i * rate$
  else
    $\bar{w} = \bar{w} - p * x_i * rate$

Suppose threshold is 0.5 here; learning rate makes sure that the step taken in the direction of the gradient is not too large

3.2 Implement Basic Logistic Regression Filter

I implemented Basic Logistic Regression method in three steps.

Step 1. Learn $\bar{w}$ on train_data;

Step 2. Evaluate each mail based on $P(Y = spam|\bar{x}) = \frac{\exp(\bar{w}*\bar{x})}{1+\exp(\bar{w}*\bar{x})}$, so that I could get probability of a mail being Spam given words vector $\bar{x}$ of this mail;

Step 3. The parameter tuning and train to evaluate process are similar to that of Bayesian Filtering.

The basic result is shown in Table 4:

Table 4: AUC of Basic LR

	tune	u01	u02	u00_tune	u00
AUC	0.999212	**0.867380**	**0.912238**	0.999411	0.848224
u00_tuned				0.999588	**0.862130**

Figure 6 shows ROC curves for Table 4:

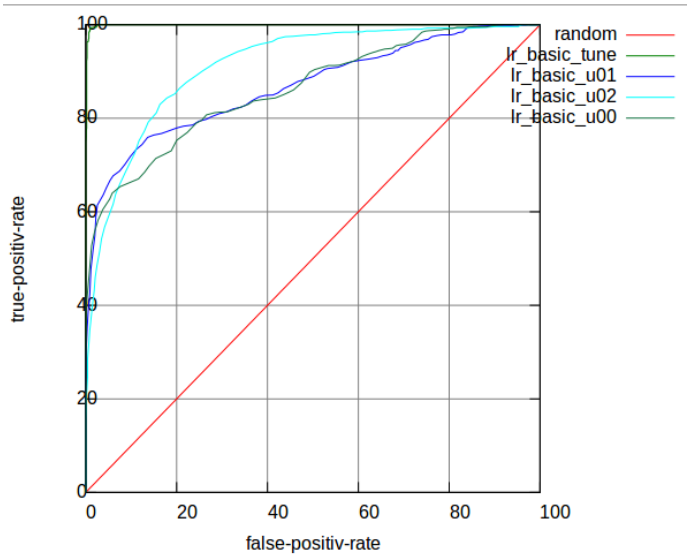

Figure 6: ROC curves of Basic LR

3.3 Improve Basic Logistic Regression: re-modified TF-IDF

Logistic Regression with TF-IDF is a common technique in text classification. TF-IDF is generally used in search engine. It is a numerical statistic which reflects how important a word is to a document in a collection.

3.3.1 First Attempt: Default TF-IDF

I learned how to use TF-IDF [5][6][8] to restructure the weight vector $\overline{w}$ of each word.

$TF_{t,d}$ (Term Frequency) = frequency of a term t in document d / total words count (or most frequent words count) in document d;

IDF_t (Inverse Document Frequency) = $\log(\frac{Corpus\ size}{Document\ count\ contains\ term\ t\ +1})$.

TF-IDF = $TF_{t,d} * IDF_t$, it can evaluate how important a word is in a mail.

Then, the weight of a word can be reconstructed as following formular:

$$w_i = \frac{tf_i * \log(M/f_i)}{\sqrt{\sum_{j=0}^{N}(tf_j * \log(M/f_j))^2}} * w_i$$

, where Wi is the vector of weights, tf_i is term frequency of word i in the e-mail, M is the total number of the email collection, N is the total number of the words in a detailed email.

It also takes the importance of each word into consideration onto weight, which is similar to the method used for picking most interesting words in Bayesian Filter.

However, this attempt failed for even decreasing the AUC results.

3.3.2 Second Attempt: modified TF-IDF

Limitation of traditional TF-IDF: If a word has high term frequency (TF) in both Spam and Non-spam, it does not have good discriminant validity. This also happens in inverse document frequency (IDF). [4]

Improved TF-IDF:

$$W_i = TF_i * IDF_i = log(|Sf_i - Hf_i|) * (|\log(S/n_i) - \log(H/n_j)|)$$

, where Sf_i is the ith word term frequency in Spam, and Hf_i is the ith word term frequency in Healthy. S is total number of Spam, H is total number of Healthy.

By evaluating the difference of word's TF and IDF between Spam and Non-spam, improved TF-IDF can better show the word's discriminate validity, especially the low frequency words.

But these two attempts had not worked well, they even decreased the AUC in u02.

See the result of second attempt in Table 5:

Table 5: AUC of Basic LR

	u00	u01	u02
AUC	0.859492	0.867861	0.847229

3.3.3 Third Attempt: re-modified TF-IDF

The second attempt sound reasonable, it should have taken effect. But it has missed some important concept in the original TF-IDF. It has not taken the evaluate mail itself into consideration.

$$\text{Simple } TF_t = \frac{f_{t,d}}{Max(f_{i,d})}$$

$$\text{Modified } TF_t = log(|Sf_t - Hf_t|)$$

$$\text{Re-modified } TF_t = \frac{\log(|Sf_t - Hf_t|)}{Max(f_{i,d})}$$

, where I added the denominator here, to taken the feature of an evaluate mail into consideration. This increased the AUC decently.

Table 6 shows the result of Re-modified TF-IDF with Logistic Regression Filter:

Table 6: AUC of Re-modified TF-IDF on LR

	tune	u01	u02	u00_tune	u00
AUC	0.999371	**0.887771**	**0.952392**	0.999475	0.881424
u00_tuned				0.999606	**0.888191**

The ROC curves of Re-modified TF-IDF on LR are shown in Figure 7:

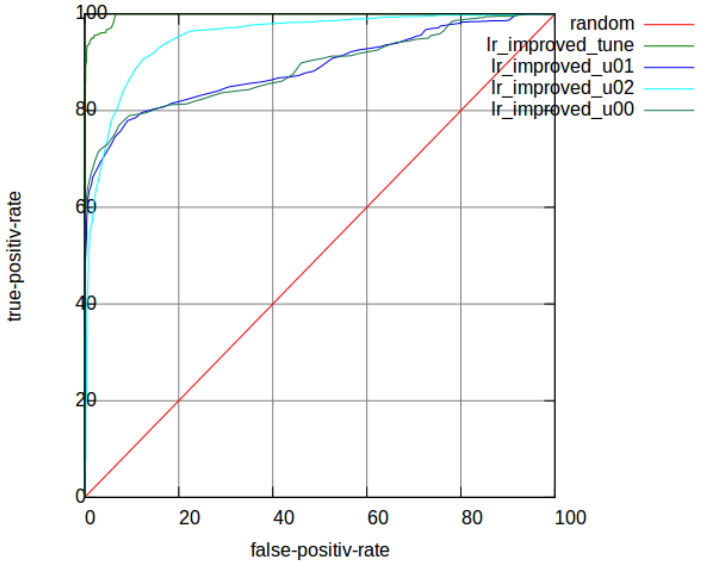

Figure 7: ROC curves of Re-modified TF-IDF on LR

4 Self-Learning

Till then, I have achieved Average AUC of 0.921837 with NB, and 0.909551 with LR. This kind of result would rank at 3rd position in Discovery Challenge 2006.

4.1 Shall We Stop?

No. There are still something confused us a lot.

By observing the result of what I have already got: [tune:0.998106, u00:0.901313, u01:0.912002, u02:0.952196] on Bayesian Filter, [tune:0.999371, u00:0.888191, u01:0.887771, u02:0.952392] on Logistic Regression Filter.

Why AUC on tune_data could be so high? Why u02 has higher AUC than u01, and u00 has lowest AUC but has an individual tuning data?

Since the features of these 3 inboxes are different. If the features between training data and evaluation data are similar, then I guess I can achieve as high accuracy as what I get on divided tune_data, whose 1st-half partition is used for training, and the 2nd-half partition is used for evaluating.

So there is a solution that how about learning from the evaluation data, this is the idea of Self-Learning, which is also inspired by my daily use of Gmail Service. When I put some advertisements into Spam category, those kind of mails never appears again in my inbox.

To avoid cheating, I can learn from the labeled result of evaluation data. That means, I should have confidence that my Spam Filter is accuracy enough. I call this Self-Learning, based on how self-confident I am on my filter.

4.2 Implementation of Self-Learning

Because I don't know the true label when evaluating mails, I should not learn all of them. But I can still learn some part of those evaluating data based on the confidence I have on them.

So, I set a learning boundary when evaluating mails. I believe those mails which have probabilities > 0.99 or < 0.01 are predicted correctly. Then I can treat those mails as labeled to augment my training data.

Another advantage is that these extra training data has more similarities on features to the evaluating data.

Table 7 shows the improved result of Self-Learning on Naive Bayes:

Table 7: AUC of Self-Learning on NB

	tune	u01	u02	u00_tune	u00
AUC	0.997701	**0.927123**	**0.980120**	0.989169	0.889985
u00_tuned				0.992031	**0.958331**

The ROC curves plotted on Figure 8:

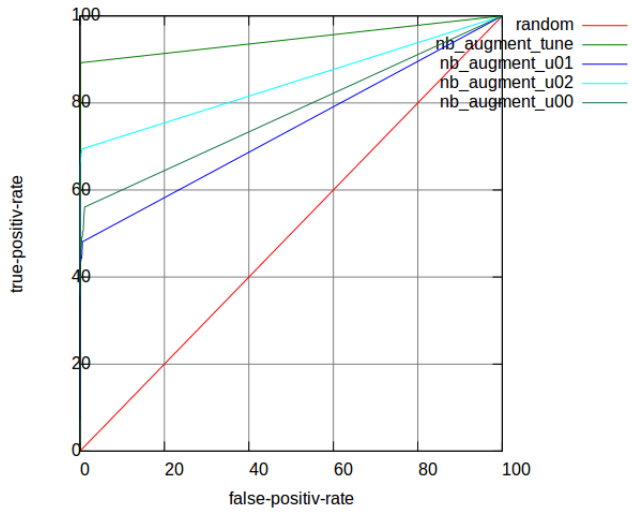

Figure 8: ROC curves of Self-Learning on NB

The Bayesian Filter has been benefited a lot from self-learning, but Logistic Regression Filter didn't. I will analyze the reason afterwards. Hope it is true. The key word is converge speed of LR.

Table 8 shows the improved result of Self-Learning on Logistic Regression:

Table 8: AUC of Self-Learning on LR

	tune	u01	u02	u00_tune	u00
AUC	0.999315	**0.899148**	**0.960306**		
u00_tuned				0.999421	**0.892779**

Following 4.3 might be a further step on Self-Learning.

4.3 Shall We Stop Now?

It depends.

Since the self-learned data has more similarities to those evaluation data, I want to increasing their weight while evaluation. I can import an enlargement coefficient on the self-learning rate. I got this idea also from the concept of LR learning rate.

But this enlargement is obviously not the bigger the better. When I learned from predicted probability, I also unavoidably learned some wrongly classified mails. It's not good for this kind of mistakes to be enlarged.

Table 9 shows the enlargement result on Bayesian Filter:

Table 9: AUC of Enlarged Self-Learning on NB

Enlarged x3

	tune	u01	u02	u00_tune	u00
AUC	0.997126	**0.936075**	**0.985456**		
u00_tuned				0.990459	**0.976759**

Enlarged x20

	tune	u01	u02	u00_tune	u00

AUC	0.994268	**0.953053**	**0.987347**		
u00_tuned				0.979197	**0.988825**

5 Conclusion

5.1 Compare the Two Methods

Through my implementation of these two types of Machine Learning Techniques, I concluded that:

Logistic Regress Filter:

-- Weights will trend to a balanced point, and converges very fast; this explains why that it did not work when augmenting evaluation data on training set.

-- Need limited train data compared to Bayesian Filter;

-- Will not be improved by Self-Training;

-- More true-positive-rate, more false-positive-rate.

Bayesian Filter:

-- More data more precise;

-- Good performance when Self-Learning;

-- Less false-positive-rate, less true-positive-rate.

And to choose one of them, I would choose Bayesian Filter as a reliable Spam Filter Algorithm, according to Paul Graham's assertion [10]:

Spam filtering is not just classification, because false positives are so much worse than false negatives that you should treat them as a different kind of error. And the source of error is not just random variation, but a live human spammer working actively to defeat your filter.

5.2 Our Results

Table 10: Improved Bayesian Filter

	u00	u01	u02	Average AUC
AUC	0.901313	0.912002	0.952196	0.921837

Table 11: Improved Logistic Regression Filter:

	u00	u01	u02	Average AUC
AUC	0.888191	0.887771	0.952392	0.909551

Table 12: Bayesian Filter with Self-Learning

	u00	u01	u02	Average AUC
AUC	0.958331	0.927123	0.980120	0.955191

Table 13: Enlarge X3 for Bayesian Filter with Self-Learning

	u00	u01	u02	Average AUC
AUC	0.976759	0.936075	0.985456	0.966097

I felt quite exciting when compared to Ranking in Discovery Challenge 2006, where the 1st Rank was 0.950667. Admittedly, I probably have had more time to do to this challenge. And the updated results of the challenge showed 4 teams have achieved 0.95 even one achieved 0.98 Average AUC after the challenge. I guess they should have used the same improvement as Self-Learning, which will break the limitation of similarity in training data.

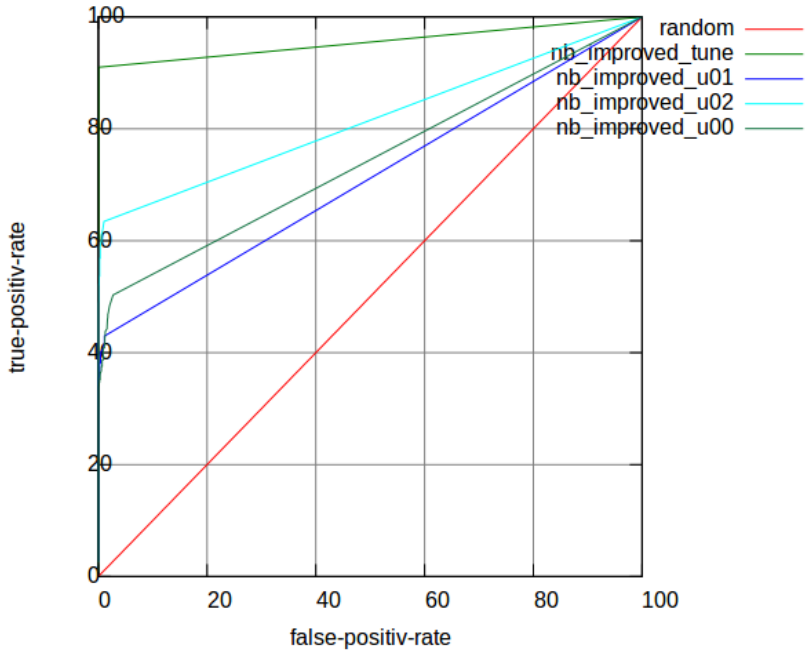

Figure 9: ROC curves of Improved NB

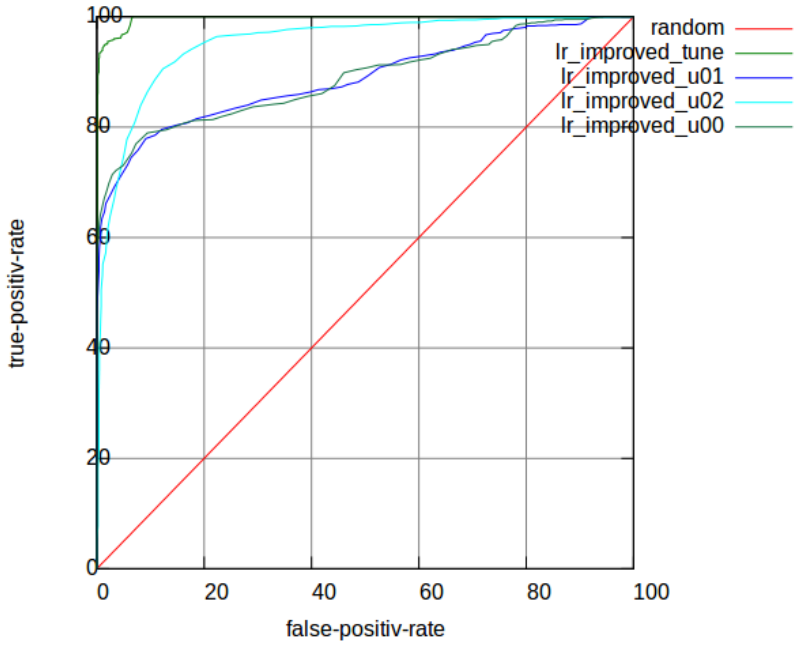

Figure 10: ROC curves of Improved LR

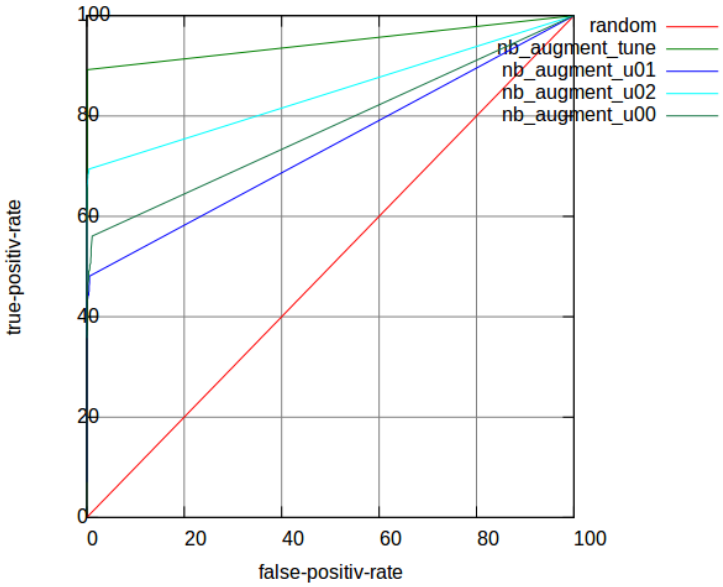

Figure 11: ROC curves of Improved NB with Self-Learning

6 Further Improvements Being Considerable

6.1 Combination of NB and LR

From the ROC curve, I can see the LR has relatively high FPR, as well as high TPR; while NB has little FPR, but lower TPR.

I guess there maybe exist a way to combine those two methods, which will take advantage of both of them on different features.

e.g. I have tried to static the most false-positive/false-negative area in LR, which found out than the false-positive-error most occurred in probability ranges from [60, 80]. So, there is an idea that I can use LR as the 1st Filter, if its decision probability of a new mail is in range [60, 80], then I will use NB as the 2nd Filter there. After my attempt to do so, the AUC really increases, but too little to be caught in eyes.

I'm sure Ada-boost will have effect on this kind of combination.

6.2 More Powerful Self-Learning Will Not Be Considered

During the presentation day of Machine Learning course on December 4th 2013, I learned from other team that they have used more powerful Self-Learning algorithm, which is also applied on a project of text classification having twenty classes, to optimize the accuracy from about 17% to 60%. The powerful algorithm is using the Self-Learning process by recursion. And then at least make the prediction to unlabeled data.

This is definitely a better idea than do self-learning just once and only for new coming mails. But in Spam Filter case, I would not use this method. Cause in most situations, a user doesn't like the idea that his/her will change from Spam to Inbox, or even Inbox to Spam after categorized at the first time. But this powerful recursion will definitely improve the Average AUC in the challenge.

Thus, I prefer to have time and opportunity to implement combination of NB and LR.

References

[1] A Plan For Spam (2002) Paul Graham

[2] Online discriminative spam filter training (2006) Joshua Goodman, Wen-tau Yih. The Conference on Email and Anti-Spam 2006

[3] An empirical study of three machine learning methods for spam filtering (2006) Chih-Chin Lai

[4] Improved feature selection algorithm in spam filtering based on TF*IDF 基于TF*IDF的垃圾邮件过滤特征选择改进算法 (2009) Chen Qi, Wu Zhao-hui, Yao Fang, Song Xiu-rong, Zhang Fu-zhi. Yanshan University, China

[5] Study on applications of spam filtering based on logistic regression 基于Logistic回归的垃圾邮件过滤应用研究 (2008) Wang Qing-xing. Zhejiang University, China

[6] Filtering Chinese Spam Email Using Logistic Regression 基于Logistic回归的中文垃圾邮件过滤方法 (2008) Wang Qin-xing, Xu Cong-fu, He Jun. Zhejiang University, China

[7] Bayesian Inference and Web Application: Introduction to the Theory & Spam Filtering 贝叶斯推断及其互联网应用: 定理简介 和 过滤垃圾邮件 (2011) Ruan Yi-feng

[8] Application of TF-IDF and Cosine Similarity (2013) Ruan Yi-feng

[9] ruby-plot, an open source program to draw curves written by Ruby (2010) Vorgrimmler. University of Freiburg, Germany

[10] Better Bayesian Filtering (2003) Paul Graham