|  |
| --- |
| University of North Carolina, Charlotte |
| Machine Learning: HW2 Write-Up |
| Implementing a Naive Bayesian Learner to classify SMS into SPAM/HAM |
|  |
| **Ankur Huralikoppi** |
| **10/3/2013** |

|  |
| --- |
| This document explains about the design decisions taken for the homework, other experiments done and performance calculations and inferences |

Contents

[1. Design/Implementation Outline 3](#_Toc368917803)

[1.1 Data Reading & Randomizing 3](#_Toc368917804)

[1.1.1 SMSData Class 3](#_Toc368917805)

[1.1.2 AttributeCountsAndProbabilites Class 3](#_Toc368917806)

[1.1.3 Randomizing data 3](#_Toc368917807)

[1.2 Five - Fold Cross Validation 4](#_Toc368917808)

[1.3 Performance Metrics 4](#_Toc368917809)

[1.4 Design Decision Rationales 4](#_Toc368917810)

[1.4.1 Choice of the classes 4](#_Toc368917811)

[1.4.2 Implementation: Language Optimizations 5](#_Toc368917812)

[2. Algorithm Details 5](#_Toc368917813)

[2.1 Learner Construction [trainNaiveBayesian] 5](#_Toc368917814)

[2.2 Validation Construction [testNaiveBayesian] 5](#_Toc368917815)

[3. Extra Credit Experiments 6](#_Toc368917816)

[3.1 Sum of Logs of Probabilities V/S Product of Probabilities. 6](#_Toc368917817)

[3.2 Removal of Punctuation marks & Smileys 6](#_Toc368917818)

[3.2.1 One Small Catch 7](#_Toc368917819)

[3.3 Replacing most common abbreviations with original word 7](#_Toc368917820)

[3.3.1 One more Catch 7](#_Toc368917821)

[4. Analysis 7](#_Toc368917822)

[4.1 Performance Metrics Calculations 7](#_Toc368917823)

[4.2 Impact of various experiments 7](#_Toc368917824)

[4.2.1 Impact of Sum of Logs of probabilities V/s Product of probabilities 7](#_Toc368917825)

[4.2.2. Impact of replacing most common abbreviations: 8](#_Toc368917826)

[4.3 Most important Metrics 8](#_Toc368917827)

[5. Outputs & Verdict 8](#_Toc368917828)

[5.1 Output 8](#_Toc368917829)

[5.1.1 Screenshot showing the various performance improvements/degradations 9](#_Toc368917830)

[5.2 Verdict 9](#_Toc368917831)

[5.2.1 100 Top Words removed 9](#_Toc368917832)

[5.2.2. Removing All Punctuation Marks 10](#_Toc368917833)

[5.2.3. Taking Sum of Logs 10](#_Toc368917834)

# 1. Design/Implementation Outline

This section provides an outline about all the major design decisions taken and how they were implemented.

## 1.1 Data Reading & Randomizing

The input file consists of 5574 lines of SMS'. Each of them has the classification and the SMS body. The input was read by opening the file and creating a list of SMS'. The below are the two major classes those were used in the list to encapsulate the SMS body and any required operation on it.

### 1.1.1 SMSData Class

The SMSData class is just an OO representation of each line of the input file. The SMSData class has two members:

* ***hypClass***: The attribute to represent the classification class of the SMS
* ***attributes***: The SMS body split by spaces. This is an array of each word in the SMS body.

### 1.1.2 AttributeCountsAndProbabilites Class

The AttributeCountsAndProbabilites class is the most important class of the project. This class can be viewed to be a representation of the following data structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Word** | **Spam Occurrences** | **Ham Occurrences** | **Total Occurrences** |
| *Word-1* | 243 | 4834 | 5077 |
| *Word-2* | 2000 | 143 | 2143 |

The class has the following attributes:

* ***Word***: Could be any word that appears in the SMS body. This word can be used as a lookup "Key". Given a word, this class will provide us information about how many times has it occurred in SMS' those were classified as SPAM/HAM.
* ***Spam Occurrences***: Keeps a count of how many times a word has occurred in an SMS that's been classified as SPAM.
* ***Ham Occurrences***: Keeps a count of how many times a word has occurred in an SMS that's been classified as HAM.
* ***Total Occurrences***: Keeps a count of how many times we've seen this word till now. This is required to calculate the probability of the word for the two classes.

This class also provides functionality for adding the word to a SPAM/HAM class, get the total number of times it has occurred etc.

### 1.1.3 Randomizing data

The Randomizing of the data happens in two phases:

* First, the data from the input file is read and a list of SMSData objects is created. Now, we have all the data we need to randomize in the memory and we know the count of the number of SMS' read.
* An array is created with numbers which are later used as indices to access the SMSData objects. The array initially has: 0, 1, 2, 3, 4, 5, -------- , <No. of SMS' read - 1>. Once this array is created, the elements are randomized by shuffling the contents randomly. Then this random array elements are used as index references to the SMSData list.

For example, let's assume the array reads this after shuffling: 3493, 2, 445, 5134, 2234, 4233, 764, 784, 888, 234, 45, 65... This is the order in which the elements are read for construction of the learner.

## 1.2 Five - Fold Cross Validation

Once we have our data read and randomized, five chunks are created from the input data. These chunks have almost equal number of SMS' in them each. With respect to our data, the five chunks created had 1114, 1115, 1115, 1115 SMS' each, each randomly sampled.

The program then runs in five iterations. In each iteration, one of the chunks is chosen to be the "Validation" set and the rest four are used to train the learner.

## 1.3 Performance Metrics

The following performance metrics were used:

* True Positives Rate: Rate of properly classified positives.
* False Positives Rate: Rate of improperly classified positives.
* Sensitivity: Ability of the learner to identify positive results.
* Specifity: Ability of the learner to identify negative results.
* Recall [in our context]: the probability that a relevant class set is retrieved in a search.
* Precision [in our context]: is the probability that a retrieved SMS-set for classification belongs to the SPAM class.
* True Positives: correctly identified
* True Negatives: correctly rejected
* False Positives: incorrectly identified
* False Negatives: incorrectly rejected

The formulae used to calculate each of the metrics were:





http://upload.wikimedia.org/math/a/8/7/a87a5d89797001aa6c8d9a7031caf1ad.png

http://upload.wikimedia.org/math/9/1/b/91b88600b433b3059101d0295735daf5.png

**NOTE:**

**All the formulae were taken from Wikipedia:**

[**Precision & Recall**](http://en.wikipedia.org/wiki/Precision_and_recall)

[**Sensitivity & Specificity**](http://en.wikipedia.org/wiki/Sensitivity_and_specificity)

## 1.4 Design Decision Rationales

### 1.4.1 Choice of the classes

The nature of the input is a two tuple entry per line in the text file. Every SMS class being the key and the SMS body being the value. It was easy to visualize the required probability tables with this data. The two classes mentioned above could be used to hold the data and then create the probability tables.

The SMSData class is the first one to be invoked during the reading of the file. A list of SMSData is created and then used to construct a randomized input to the learners. Once the SMSData list is populated, an array of indexes is created and shuffled based on a random generator that's seeded to ensure we get new shuffled arrays. This array is then used to create five chunks from the SMSData list. The algorithm operates five times and each time, one of the chunks is chosen as a validation chunk and the rest four are used to train the learner.

Then, the class AttributeCountsAndProbabilites is used to create a tabular structure to hold the required parameters to calculate the probability of a word being SPAM/HAM. The table structure is explained above. The creation of the table is done every time an iteration is done in the 5 fold validation.

Each stage learner is constructed with four chunks of the data and one chunk is used as a validation set. Every iteration triggers the calculation of the required performance [of the learner] parameters.

### 1.4.2 Implementation: Language Optimizations

A use of a list data structure was the most optimal. When a table of the probabilities is created, there are two major operations we do on it:

* Search for a word and get its attributes required for probability calculations, and
* Delete top 'N' times occurring words.

If the table is created and stored as a list, both the above operations can be linear and logarithmic respectively. The search for the word will be in logarithmic complexity, by sorting in a logarithmic time, and the deletion is a simple "delete" operation 'N' times.

### 1.4.3 Implementation Constraints

Python provides an inbuilt heap. The heap construction is all packaged in the "heapq" namespace. However, if the elements to be inserted in the heap are not standard comparable elements, then the heapq implementation requires all elements to be a class objects, which is basically a value-object pair and the implementation uses the value as a comparable value.

However, the idea of using a heap would require implementing two heaps, that each kept a count of the SPAM and HAM counts, since a comparable object has to be just one single value. Thus, there were two possibilities:

* Use two heaps with one keeping a count of SPAM and one for HAM. This increases space complexity to quadratic in count of SMS.
* Use a single list of elements, and use it to lookup elements based on the SPAM/HAM counts.

The second one was chosen for legibility and simplicity purpose. However, the heapq namespace provides a method that returns the top 'N' occurring elements which has a complexity of O(n lg k) [reference: [Stack Overflow Link](http://stackoverflow.com/questions/471544/worse-is-better-is-there-an-example/472302%23472302)]

# 2. Algorithm Details

## 2.1 Learner Construction [trainNaiveBayesian]

The learner construction is a sequential counting process. Every chunk, except the validation set, is iterated and each word is counted against a classified Hypothesis class [SPAM/HAM]. If a word is not previously seen by the learner, then a new entry is created for the word and a count of 1 is set against the classified class.

|  |
| --- |
| ***trainNaiveBayesian (chunksOfTrainingData[5], validationChunkIndex)***  *for each chunk except validation chunk do*  *for each sms in chunk do*  *read each word and add to the list the following:*  *if word already exists do*  *increment SMS class counter by 1*  *else do*  *create a new entry, initialize the SMS class counter to 1 and add to list.*  *Remove top 'N' occurring words from the list.* |

### 2.2.1 Complexity

The outer most for loop runs four times. The second for loop is linear in time with the number of SMS in the input data. Then the line which splits the line into words and counts it is linear with an average number of words in each SMS. The next if is basically a lookup in the list, which is again linear in time about the size of input SMS data. Overall, the Complexity will come upto: O(n^2 . m), where

n - no. of SMS in the input data

m - avg. number of words in an SMS. Also, the removal of top 'K' elements takes a complexity of O(n lg k).

## 2.2 Validation Construction [testNaiveBayesian]

Once the learner is constructed, the job of classifying an input SMS is a sequential evaluation of all the SMS in the validation set. The standard Smoothed Naive Bayesian formula is used to classify the new SMS'.

|  |
| --- |
| ***testNaiveBayesian(validationSet, probabilityParameters, vocabularyCount, totalTrainingDataCount)***  *For each sms in the validation set do*  *Calculate the probabilities as per Naive Bayesian formula and classify.*  *P(word | class) = argmax(class in SPAM, HAM) P(class) . π P(word | class)*  */\**  *#=======================================================================*  *# The probability that this SMS is HAM/SPAM is evaluated using the Naive*  *# Bayesian estimate:*  *#*  *# P(word | SPAM/HAM) = (nk + 1)/(n + |V|)*  *# nk -> No. of SMS' with word as SPAM/HAM*  *# n -> Total no. of training data.*  *# |V| -> Vocabulary Size*  *#=======================================================================*  *\*/*  *Iteratively add to the evaluation parameters*  *Calculate all the performance parameters and print them.* |

### 2.2.1 Complexity

The complexity of the learner's testing method is quadratic in the number of SMS in the validation set. All the parameters required to calculate the probability of the words can be pre calculated as a preprocessing step and passed as parameters to it. The iteration of elements of the validation set and the lookup of the count of words in the table takes up linear times: O(mn) where

m- size of validation set,

n- size of the training set.

# 3. Extra Credit Experiments

## 3.1 Sum of Logs of Probabilities V/S Product of Probabilities.

The Naive Bayesian formula classifies an input sample by multiplying the probabilities of all the individual attributes. A thing to note here is that, all the probabilities are less than one. And a product of an input with a large number of attributes and probabilities each lesser than 1, will be very low and the hardware of the machine might treat it to be '0' while comparing it with the probability of another hypothesis class.

One solution to solve this problem is by normalizing each small value by taking a lg of it. Also, we could employ the property of logarithms that a lg of a product of two variables is a sum of individual logs.

*log(a.b) = log(a) + log(b),*

*P(word | class) = argmax(class in SPAM, HAM) lg(P(class)) . ∑ lg( P(word | class))*

This will ensure that each number we get as a probability estimate will be an approximate that's not a very small number and the comparison operators in the language will work fine. Also, a very slight accuracy improvement was noticed because of this. The results are discussed more later in the document.

## 3.2 Removal of Punctuation marks & Smileys

During the initial phase of testing the algorithm, it was noticed that there were a huge number of words in the table of parameters which meant nothing individually. For example, all the hyphens, exclamation marks etc. were each an entry in the parameter table with their counts also for each of the classes. In the real world, punctuation marks don't really contribute so much to the classification as the combination of sets of words do.

Experiments were carried out to test the algorithm with the punctuation marks and without the punctuation marks. And the results of both the tests were exactly the same. Not even a single entry was properly/improperly classified with the change. Hence, it was decided that no punctuation marks will be stored in the parameters' counts table.

This will lead to three advantages:

* **Improvement in time**: Since each punctuation is stripped off, no time is wasted calculating it's counts for the classes.
* **Improvement in storage**: Each punctuation mark appeared as one new entry in the table. All the space occupied by them will be available now.
* No change to the algorithm behavior.

### 3.2.1 One Small Catch

It has to be noted that most of the punctuation marks are appended to words. And when the SMS is split into words, each of the punctuation mark will appear as part of the word and doesn't actually count as an entry in the table. However, there are cases when there are several dots typed in with space in between them, there are smileys which are put intermittently with spaces from words. Such sets of punctuation marks cause an entry each in the parameter table.

## 3.3 Replacing most common abbreviations with original word

Most SMS senders type in words which are short forms of longer words. Also, some shorter words like 'You', 'okay' etc are replaced by 'u', 'k' etc. The algorithm was modified to maintain a list of such replacements and then the algorithm changes such short forms.

Experiments were run in two forms:

* changing smaller abbreviations to the longer words and
* changing the longer words to the smaller abbreviations.

Both the experiments yielded the same results. But however, there's a general problem that was faced. For words like 'for', 'to' etc. most SMS user type in the numbers "4", "2". We cannot use the replacements because the use of the numbers might many time not mean the user is implying the longer words. For example, " I saw the number 4 written on the car", has the number '4' and is not a replacement for "for".

The performance of this change is discussed in detail in the analysis.

### 3.3.1 One more Catch

One more thing to be noted is that, we're doing one more step of optimization: removing top "N" occurring words. Most of them are all 1-4 lettered words those appear very frequently, and all the words and replacements are all abbreviations of such small words. Hence, the overall abbreviation replacement might not be a legitimate task to do in case of runs where we are removing words which appear more than 10 times.

# 4. Analysis

## 4.1 Performance Metrics Calculations

The formulae mentioned above were used to calculate the values of each of the metrics. Counters were added to keep a track of all the required values for the calculations and then used at the end of the algorithm to calculate the values.

## 4.2 Impact of various experiments

### 4.2.1 Impact of Sum of Logs of probabilities V/s Product of probabilities

Like mentioned earlier, there was a slight improvement in the accuracy of the algorithm when logs were applied and the sum of them was used to calculate the classification probability. The reason might be as mentioned earlier. The products of numbers smaller than one result in a number that is very small and the programming language operators treat them as 0's and the comparisons return wrong results.

Although the improvement was not very significant, but atleast, the errors due to the limitations of the hardware can be mitigated with this change.

### 4.2.2. Impact of replacing most common abbreviations:

Another experiment conducted was to replace the most common abbreviations that users use in messages. The algorithm was modified to accomplish this and it helped gain more Specifity.

This change marked a remarkable improvement in "Specifity" of the algorithm. The reason might be, a uniform translation. For example, consider the words "okay", "k", "ok". Let us assume there are 100 SMS' which each have these three words. Without the replacement algorithm, there would be three entries in the parameter table with each of them having their classification counts. Let us say the work "okay" has 90% probability that its HAM. And the rest two of them have 90% probabilities each that the SMS is SPAM. Now when the word 'okay' comes in a SPAM message in the validation set, it'll be classified to be HAM! Ideally, an overall picture would tell us that the word "ok" should have a SPAM probability higher than HAM. But the algorithm misses it because it doesn't know all the three words mean the same!

However, it was also noticed on different randomized runs that the accuracy fell down upto 7-10 wrong classifications. The reason might be again the conflicting meanings. For example, somebody really meant 'u' and it was replaced by "you" etc.

## 4.3 Most important Metrics

The most important metrics for the performance of the algorithms are:

Recall: In our context, a high recall rate means that the algorithm classified most SPAM messages as SPAM.

Precision: In our context, a high precision rate means that, the algorithm classified more SPAM messages as SPAM than classifying SPAM as HAM.

Specifity: In our context, a high Specifity rate means that a SPAM classified SMS is indeed SPAM and a HAM classified result is indeed HAM.

Sensitivity: Same as Recall.

The rest of the parameters are all just counters which are used to calculate these above mentioned metrics.

Why are they important?

All the above mentioned metrics are all indicators of how "reliable" the algorithms is. It's very important because a wrongly classified SMS might be configured to delete by some user and he might never read it if an important HAM message was classified to SPAM.

Hence a desirable algorithm will have mostly high values for each of the mentioned metrics.

# 5. Outputs & Verdict

## 5.1 Output

Five important outputs were created:

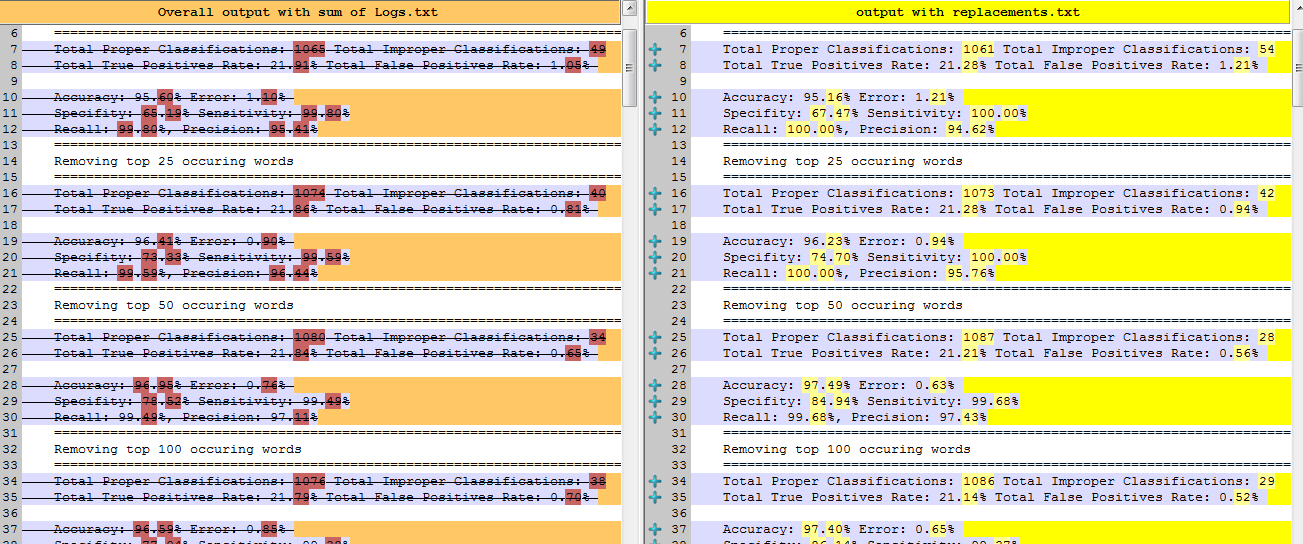
1. The core naive Bayesian algorithm.
2. Product of probabilities replaced with Sum of Logs of Probabilities.
3. Removal of "N" highest occurring words.
4. Punctuation marks replacements.
5. Abbreviations replacements.

The below table explains the outputs of the various experiments and the impact on [various colors used for legibility]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Recall | Precision | Specifity | Sensitivity | Accuracy | Error |
| 1 | Used as a measuring standard | Used as a measuring standard | Used as a measuring standard | Used as a measuring standard | Used as a measuring standard | Used as a measuring standard |
| 2 | Increased | Increased | Increased | Increased | Increased | Decreased |
| 3 | Increased upto 100 words, decreased for 500 | Increased for 100 words, decreased for 500 | Increased for 100 words, decreased for 500 | Increased for 100 words, decreased for 500 | Increased for 100 words, decreased for 500 | Decreased for 100 words, increased for 500 |
| 4 | No Change | No Change | No Change | No Change | No change | No change |
| 5 | Very small Increase | Decreased | increase | increase | Decreased | Increased |

### 5.1.1 Screenshot showing the various performance improvements/degradations

The below screenshot is a comparison of the outputs of one run of sum of logs and one run with abbreviations replaced.



## 5.2 Verdict

After all the experiments and several runs of randomized training and testing, the best performance was found out to be the learner with the following modifications:

* 100 top occurring words removed.
* Removing all the punctuation marks.
* Taking sum of logs of probabilities instead of product of probabilities.

### 5.2.1 100 Top Words removed

Experiments showed that the growth of the parameters was linear for the runs where the words removed were 10, 25, 50 and 100. Then there was a drop. Hence, 100 is the most optimal one and had the peak of all the performance curves.

The below graph shows the curves of the various parameters with respect to removing top 'N' occurring words for one of the runs of the 5 fold validation. It was also seen that the curves were more or less the same for every fold of cross validation.

This parameter has a remarkable improvement in Specifity and minor improvements in others.

### 5.2.2. Removing All Punctuation Marks

Most punctuation marks are suffixes of words. But however, a large number of punctuation marks are written with spaces and smileys are very frequent. These impact space and also the lookup time in the parameter table.

### 5.2.3. Taking Sum of Logs

Like explained earlier, the performance of the metrics overall was better with sum of logs than the product of small valued probabilities.