Spam Detection using Classifiers

Team Griffin - Ivan Kutovoi, Dominic Cunningham, Jack Tumulty

Abstract

This project provides an easy and granular way to implement and test spam detection using classifiers.

Current iteration implements and tests Naive Bayes and K-Nearest Neighbors classifiers. The results are the following:

Score / Classifier	Naïve Bayes	K-Nearest Neighbors
Accuracy	98.386%	98.206%
Precision	91.406%	98.214%
Recall	94.355%	85.938%
F1	**92.857%**	**91.667%**

Both classifiers perform well in terms of metrics. However, K-Nearest Neighbors takes significantly longer to run due to the nature of the algorithm. Additionally, KNN performs slightly worse than NB in recall. However, it shows better results in precision.
One classifier isn't necessarily better than the other. The choice depends on organizational priorities. If minimizing false negatives or speed is required, Naive Bayes is the best choice. If speed isn't a concern and the cost of a false positive is high, K-Nearest Neighbors is the better option.

Improvements

Several performance improvements were made to the algorithms over the lifespan of the project.
Firstly, dynamic stop word generation was introduced. Instead of having a predefined list of common and uninformative words, stop words are gathered from the data set. Two parameters are passed: bottom per mille and count. Bottom per mille is responsible for removing all words that are less common than N‰, where N is the parameter. Count is responsible for removing any words that appear less than count times. It turned out that changes in count have little to no impact on performance for both NB and KNN.

Naive Bayes calculation underwent major changes over time. At first, Naive Bayes was calculated based on statistics formulas provided in the assignment. This was changed to use logarithms. Now, samples are initialized as

$$sample = log(WordCount / TotalWordCount)$$

That somewhat improved the classifier. However, the change that more than doubled performance was the change in how probabilities were calculated. The new equation is

$$probability = log(P) \propto log(P(classi)) + \sum log(sample)$$ [Krause, StackOverflow]

This has improved performance significantly, boosting scores for test data and reducing overfitting. The following are two learning curves - for the old model and the new one.

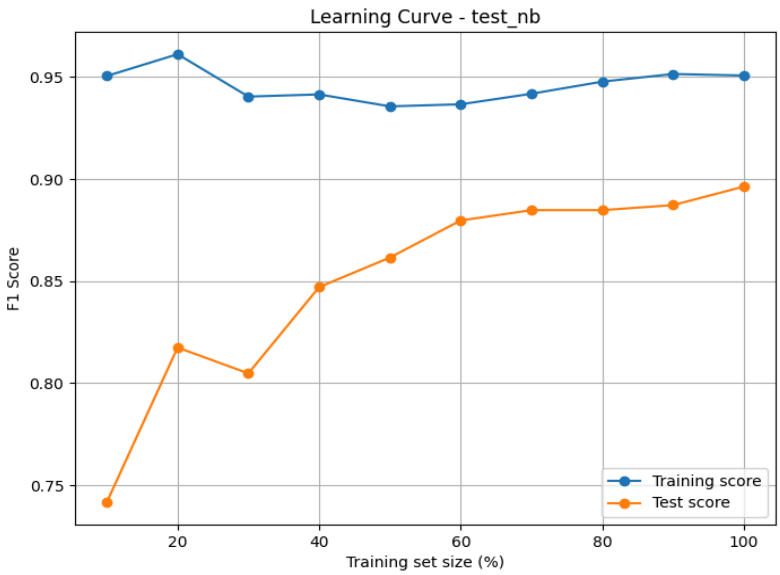

Fig 1. *New NB learning curves with alpha = 4*

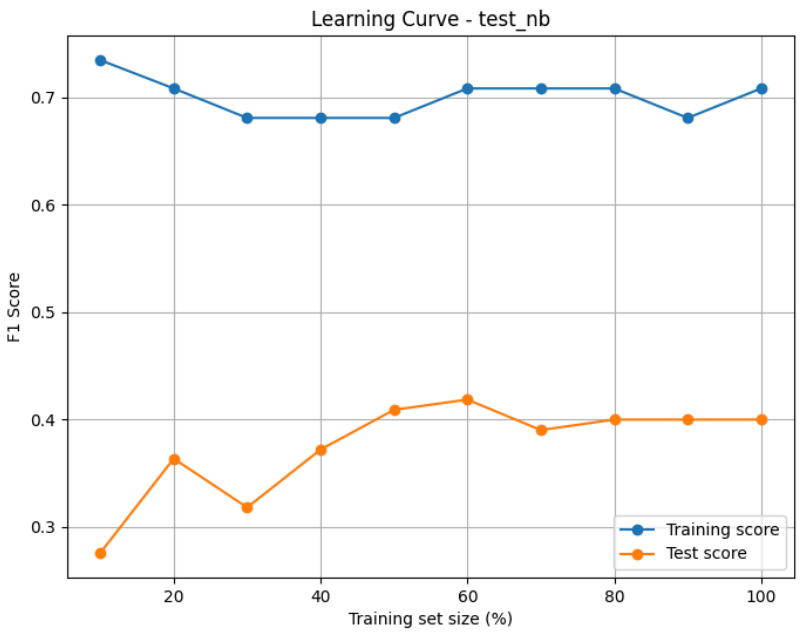

Fig 2. *Old NB learning curves*

Finding Parameters

In order to find the best possible parameter values, an automatic tester was introduced. Parameters were chosen based on F1 score. The following graphs demonstrate the most interesting of those changes.

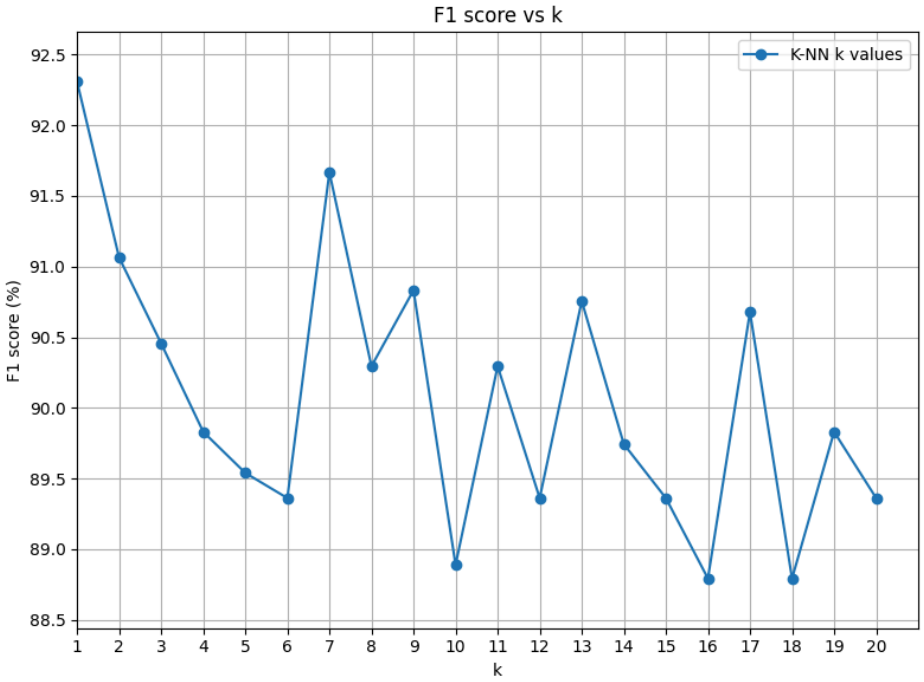

Fig 3. *KNN F1 score vs k*

As seen in Figure 3, performance of K-Nearest Neighbors varies greatly with different *k* values. While performance metrics are the highest with *k* = 1 (see Figure 5), the model does not actually show the best results. It overfits too much, performing perfectly on training data but much worse on test data. In actuality, *k* = 7 is the best choice.

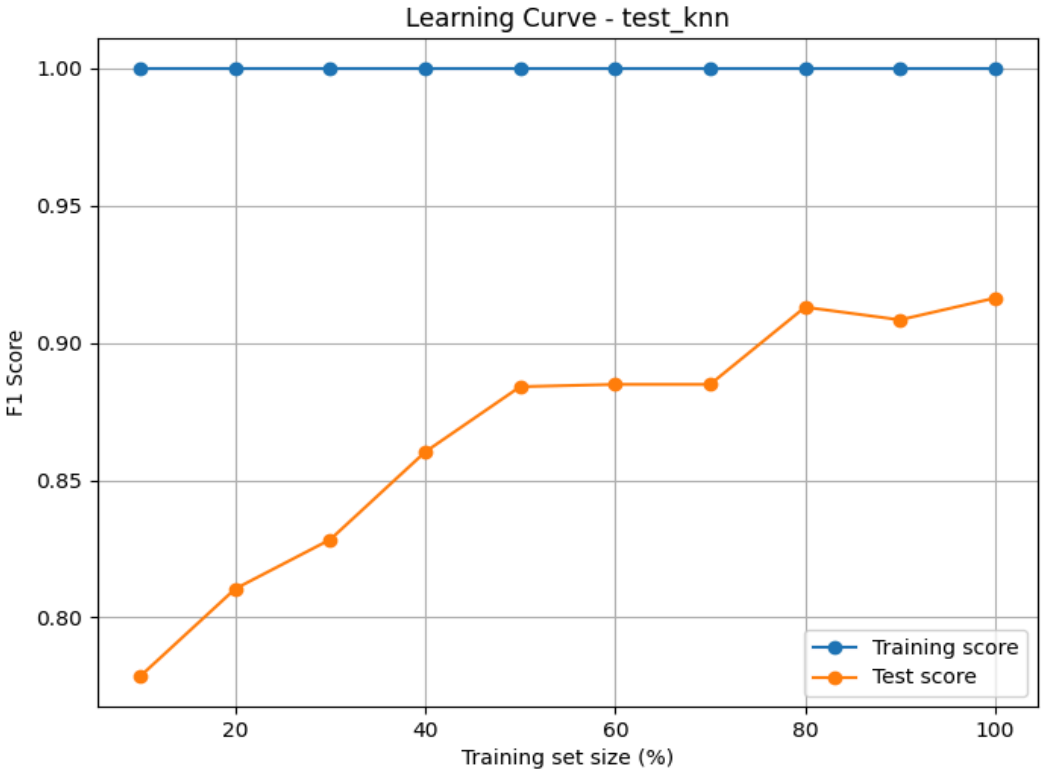

Fig 4. *KNN Learning Curves with k = 1*

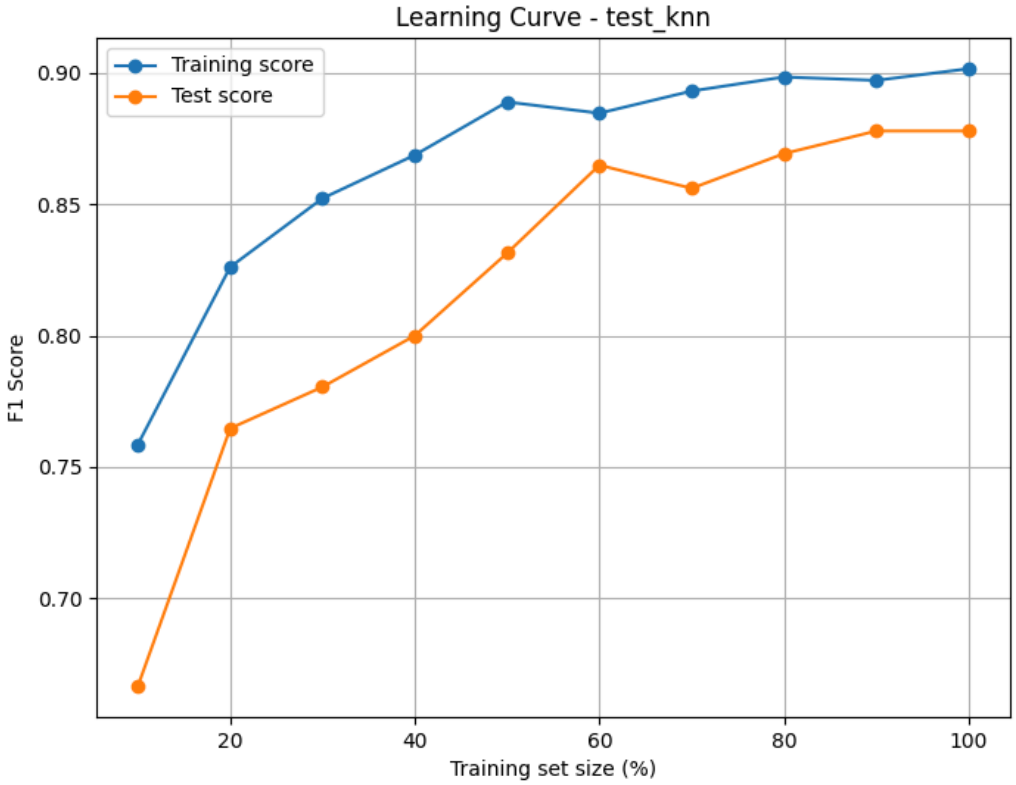

Fig 5. *KNN Learning Curves with k = 7*

Laplace smoothing alpha used in Naive Bayes was another tested parameter. Usually, NB works well with smaller alpha, often even less than 1. Due to the nature of data used in this project's case, a higher alpha is preferrable. Interestingly enough, the test curve doesn't change much when using a low alpha (0.7). More than that, training data score is higher if using a low alpha. However, recall drops from 94.355% to 81.333%.

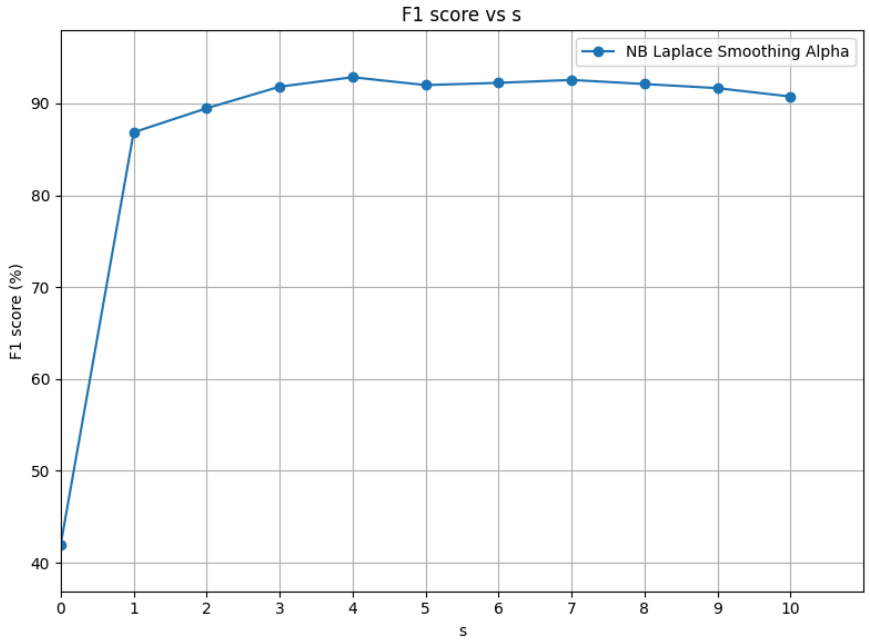

Fig 6. *NB F1 score vs Laplace Smoothing Alpha*

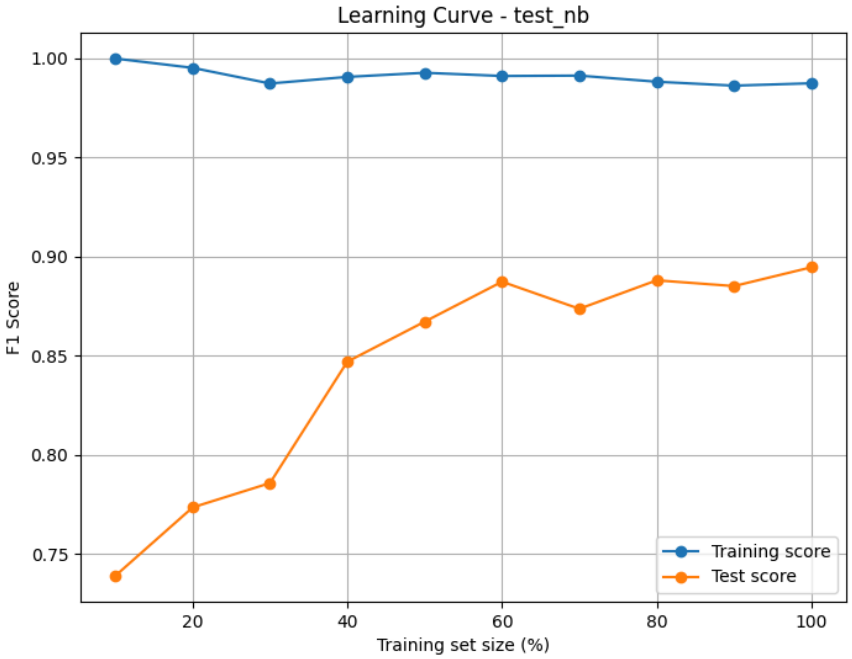

Fig 7. *NB learning curves. Alpha = 0.7*

Comparing Models

In terms of performance metrics, both classifiers perform well, with NB having higher recall and KNN having higher precision. These differences are due to the nature of the algorithms and, possibly, biased data set.
Naive Bayes tends to be more inclusive in predicting positive instances, capturing a broader range of actual positives, thus increasing recall. However, this can sometimes lead to more false positives
K-Nearest Neighbor offers high precision because it tends to predict positively only when the nearest neighbors are very similar, thus reducing false positives. However, this can come at the cost of lower recall if a sample is not sufficiently represented among the nearest neighbors.

As seen in Figures 1 and 5, K-Nearest Neighbors overfits much less than Naive Bayes.

What the graphs don't show is how the algorithms work. Naive Bayes has a learning phase. It goes over the training data samples and calculates the probabilities of different classes and the likelihood of the features given each class. All of the values are added to the model's memory after the training process. Classification is done separately - testing samples are evaluated based on the values that were calculated during training. K-Nearest Neighbors, on the other hand, is not trained. Instead, KNN stores all the training samples and processes them directly during runtime. When a new data point needs to be classified, KNN calculates the distance between the new point and all stored samples - the only thing the model remembers are these stored samples, so there's no need for training.

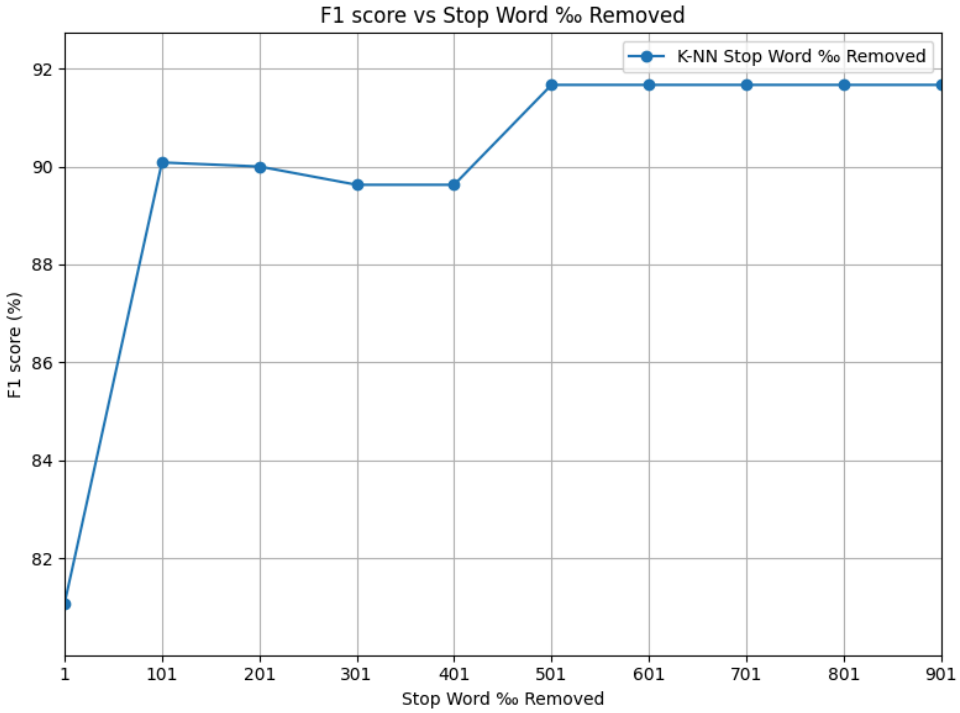

Fig 8. *KNN F1 score vs Bottom ‰ of Words removed*

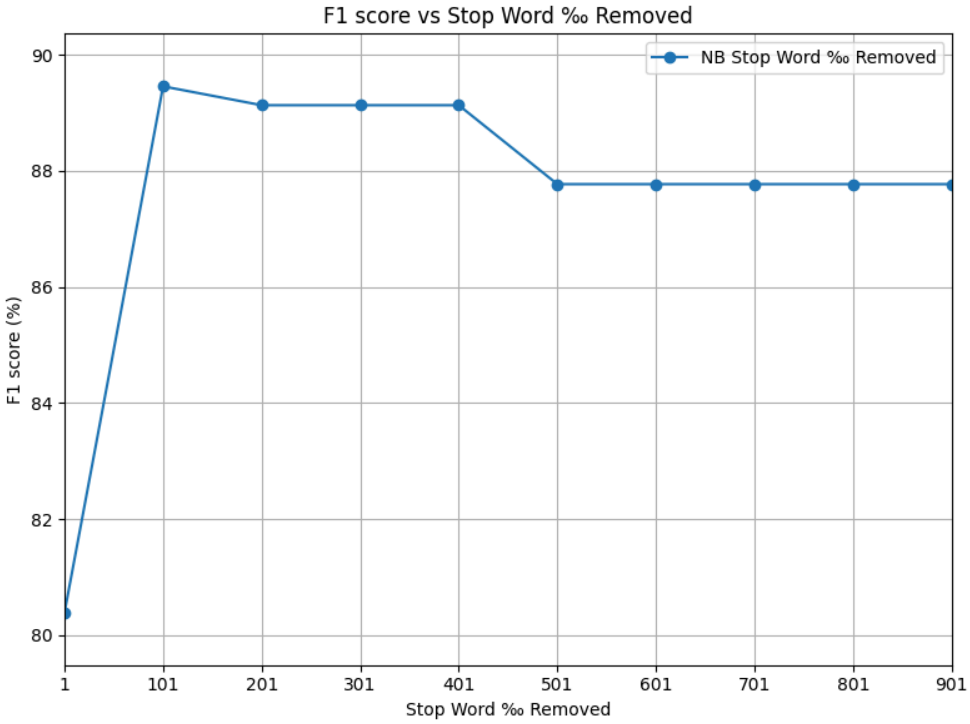

Fig 9. *NB F1 score vs Bottom ‰ of Words removed*

Graphs on Figures 8 and 9 are very interesting - they show F1 scores based on the bottom ‰ of words removed. Both classifiers perform poorly when almost none of the words are removed. However, they are complete opposites of each other when per mille is between 100 to 400, and 400 to 900.
NB does a better job when less words are removed, while KNN classifies more correctly when more words are removed. This happens because of how these algorithms work. Naive Bayes assumes that all features are independent, making it less affected by the curse of dimensionality. When more words are present, the algorithm has more information to correctly classify the new data point.
K-Nearest Neighbors, on the other hand, is negatively affected by a high amount of dimensions. As more dimensions are introduced, the distances between neighbors become increasingly similar, making it harder to classify the new data point.

Exercises

4. Suppose that the Accuracy of a malware classifier was 95%, with equal rates of false positives and false negatives (FPR = FNR = 5%). For a test dataset with 10,000 samples and 99.9% benign software, compute:

a. *The expected number of benign and malicious programs in the test set.*
   Benign count: 10,000 * 0.999 = 9,990
   Malicious count: 10,000 * 0.001 = 10
b. *The expected number of TPs, TNs, FPs, and FNs.*
   TP: 10 - 0.5 = 9.5 ~ 10
   TN: 9,900 - 499.5 = 9400.5 ~ 9401
   FP: 9,990 * 0.05 = 499.5 ~ 500
   FN: 10 * 0.05 = 0.5 ~ 1
c. *The Precision and Recall values.*
   Precision: 9.5/(9.5 + 499.5) = 0.019
   Recall: 9.5/(9.5+0.5) = 0.95

5. Suppose that the Accuracy of a classifier on this dataset was 93%, with equal rates of false positives and false negatives (FPR=FNR=7%). Compute:

a. *The expected number of TPs, TNs, FPs, and FNs.*
   Benign count: 10,000 * 0.999 = 9,990
   Malicious count: 10,000 * 0.001 = 10
   TP: 10 - 0.7 = 9.3 ~ 9
   TN: 9,900 * 0.93 = 9270.7 ~ 9271
   FP: 9,900 * 0.07 = 699.3 ~ 699
   FN: 0.07 * 10 = 0.7 ~ 1
b. *The precision and recall values.*

Precision: 9.3/(9.3+699.3) = 0.0131
Recall: 9.3/(9.3+.7) = 0.93

c. *Suppose that you are an intern in the RIT ITS spam division, and you have been tasked with manually checking all the messages detected as SMS spam using this classifier to see if they are spam or not. It takes you one hour to check 100 messages. How much of your time will be spent looking at actual spam? How much will be spent looking at non-spam?*

9/708 * (780/100) = 0.09 hours
699/708 * (708/100) = 6.99 hours

You will spend 0.09 hours or 5.4 minutes looking at actual spam. You will spend 6.99 hours looking at non-spam.

6. *Provide the detailed calculations and results in your report for a toy example.*

P(w|Spam):
- P(click|spam) = 2/13
- P(dude|spam) = 1/13
- P(prize|spam) = 3/13
- P(for|spam) = 3/13
- P(look|spam) = 1/13
- P(winner|spam) = 3/13

P(w|not Spam):
- P(babe|not spam) = 1/6
- P(dude|not spam) = 1/2
- P(look|not spam) = 1/3

“dude! Dude! look!”
P(Spam|W) = 1/10 * 1/13 * 1/13 * 1/13 = 0.0000455166
P(not Spam|W) = 9/10 * 1/2 * 1/2 * 1/3 = 0.075

“Winner babe! Click for prize”
P(Spam|W) = 1/10 * 3/13 * 0/13 * 2/13 * 3/13 * 3/13 = 0
P(not Spam|W) = 0/13 * 1/6 * 0/13 * 0/13 * 0/13 * 0/13 = 0

The classifier does not produce a result for this message as a result of multiple words not having a probability for both spam and not spam. This issue can be solved by using laplace smoothing, adding 1 to each P(w_i|spam) and P(w_i|ham).

References

How to use log probabilities for Gaussian Naive Bayes? Matt Krause. Stack Overflow. https://stats.stackexchange.com/questions/163088/how-to-use-log-probabilities-for-gaussian-naive-bayes

classification - what is the “learning” that takes place in Naive Bayes? - Cross Validated

AI Usage Statement

[o1-preview]

Usage: General advice on Naive Bayes improvements, without providing code or mathematical equations.

Verification: Trying out some suggestions leading to performance improvements.

Prohibited Use Compliance: I confirm this work adheres to course AI policies, with no unauthorized use in assessments/quizzes/exams. All AI-assisted components meet required substantial modification standards.