DATA MINING ASSIGNMENT 2

# ASSIGNMENT CLASSIFICATION: SPAM PREDICTION

October 31, 2018

Student ID: 20174797
Student Name: Baturay Ofluoglu
University of Antwerp
Computer Science: Data Science

# Contents

## INTRODUCTION

In this project, the purpose is to classify emails of private company employees as ham, spam or suspicious. Even though the company uses regular spam filtering, their employees still receive spam e-mails. These spam e-mails increase their cost because it steals time of their employees.

The managers of the company know that there is a trade-off between reducing costs of the potential spam emails with a classification algorithm and misclassified ham emails due to missing business opportunities. Due to this trade-off, they indicate the penalty for each email misclassification which is shown below. Thanks to this penalty table, they will be able to make a cost analysis to decide whether or not to integrate this classification algorithm.

For this project, the company provides spam, ham and mixed emails from its employees. Each email has subject and body field. These emails will be used as a train data for the spam prediction Classifier algorithm.

## THE GENERAL OVERVIEW OF THE PROJECT

1. Understanding the Data
2. Preparing Data
3. Extract Extra Feature
4. Data Preprocessing and Cleaning
5. Selecting a Classification Model
6. Evaluating The Performance of the Model

# 1 UNDERSTANDING THE DATA

The training data includes 9690 spam emails, 3048 filtered ham emails from 10 employees and 3055 mixed emails (approximately 95% ham and 5% spam) from another ten employees. Each email is stored in a text file. The first row of emails consists of the mail subject, and the rest constitutes mail body field. These emails will be used as a train data for the spam detection algorithm. Also, we have the unclassified email stream of all the users.

# 2 PREPARING DATA

As mentioned, the emails are stored in a separate text file. I prefer to collect all emails under a pandas data frame called *emails*. In this table, each rows represents one email. It also has three columns which are *email_subject*, *email_body* and *label*. As the name suggests, the first column stores the subject of the email and the second column has the body of the emails. *"Label"* is a binary column. If *Label* is 1 and if it is 1, it means that it is a spam email else ham email.

While transferring all emails from text files to *emails* data frame, I remove *\r\n* characters symbolizing a new line which is unnecessary for the classification model. Also, all the punctuations are deleted.

## 2.1 Extract Extra Feature

Based on my observations, all forwarded emails are classified as ham, and none of the spam emails has *"forwarded by"* keyword. Therefore, I plan to create a new binary column called *isForwarded* to store forward status. Directly sent e-mails (*isForwarded* = 0) does not imply whether the mail is spam or not but if *isForwarded* is 1 then we know that it should be classified as ham because none of the spam emails has *"forwarded by"* keyword.

The reason why I specifically add this column and not let the classifier algorithm reveals this rule is that *forwarded by* keyword includes a stop word (*by*). All stop words will be removed at the following section, and for this reason the classifier may miss this rule. To guarantee to capture this rule, I plan to store it in a different column.

# 3 DATA PREPROCESSING AND CLEANING

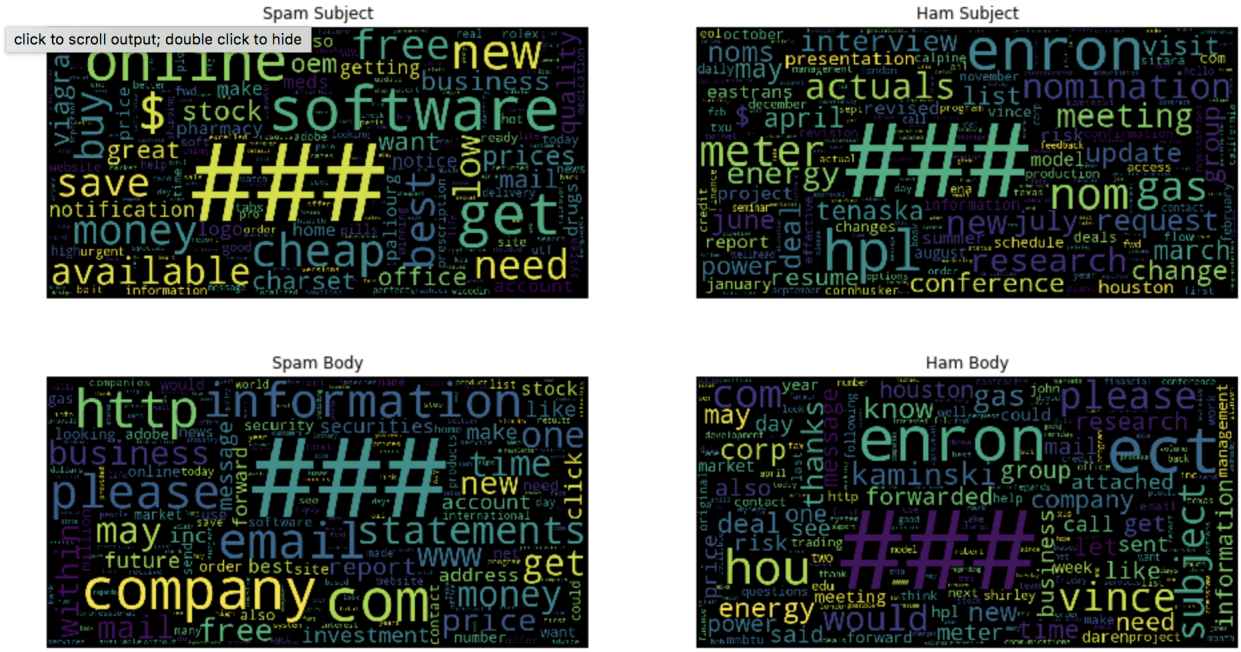
In this section, the stop words are eliminated from all the email bodies and headers. The words that have less than three chars are also removed. If the email has an empty header or an empty message, it may also have a meaning to classify. Therefore, the empty fields are replaced with '$' sign. From the word frequency visualization, it is observed that most of the empty subjects are classified as spam. In addition, I think that the number of numeric values may have extra information. For this reason, instead of removing numeric values, I replace them with '###' character. Because both spam and ham emails have a significant amount of numbers, I first run the classification model with '###' characters and it gives better results compared to the one in which the numbers are deleted. After this modifications, I saved the data frame as *_emails*

Also, I plan to detect misspelling words with a pre-trained spelling library. However, it was very slow and, it did not go over all the terms in the data frame in 4 hours. Note that it is still feasible to add misspelling word detection back if there is no running time constraint. The errors usually occur in the spam emails. Spam mail writers may intentionally write misspelled words to hack spam detection algorithms. Therefore, I plan to detect misspelling words.

### 3.1 Most Common Terms Visualization After Preprocessing

The most frequent terms are visualized. There is two class which are ham and spam emails. Also, each class has two fields called email body and subject. At this four figures, you can analyze which words are seen most common for each classes and fields.

**Figure 1:** Word Frequency For Spam and Ham Emails




## 4 Selecting a Classification Model

To classify texts, before implementing any machine learning algorithms we need to indicate our cost function to evaluate our model. The problem that we work is slightly different than other classification problems because we need to customize our scoring function.

After building scoring function, I create a frequency table for each term to analyze it with the classifiers. Lastly, I start implementing machine learning algorithms to classify emails by using frequency data.

### 4.1 Generating Cost Function

The main objective of this project is to classify ham emails as accurate as possible and minimize the total cost of mistakenly classified emails. The manager of the company warns

that classifying a ham email as spam requires 0.5h of work. Because average hourly cost of employees is 40 €, this classification costs 40 € * 0.5h = 20 €. If the analytics team classifies an ham email as suspicious, then it costs 5 €. On the other hand, classifying a spam email as suspicious or ham costs 0.4 €. Below there is a visualization of the costs to better understand regarding modeling perspective. After summing up all the costs, I divide it to the total number of emails to find the average cost of classification algorithm per e-mail.

**Table 1:** Cost Table

| | **Actual Ham** | **Actual Spam** |
|---|---|---|
| **Predicted Ham** | 0 | 0.4 € |
| **Predicted Spam** | 20 € | 0 |
| **Predicted Suspicious** | 5 € | 0.4 € |

## 4.2 Generating Sparse Frequency Matrix

To train classification algorithm, my strategy is to generate a frequency matrix for both subject and body columns of *_emails* data frame where there are terms at the columns, document ids at the rows and term frequencies at the matrix values. The idea is to train classifiers with this frequency matrix.

This dense matrix, however, has some problems regarding memory because it is highly sparse. Suppose there are 3000 different terms in total at the *email_body* column and the first email ID has *Hello Baturay* body message. In this case, only the *Hello* and *Baturay* columns will have values other than 0 and all the other 2998 columns will be 0. This situation is highly undesirable and it occupies a significant amount of memory. We only need to keep the values more than 0 for each row. Numpy library offers a good solution for this scenario and it is possible to create a very efficient sparse matrix with this library. The encouraging part is many classification algorithms also accept this sparse matrix as a train data. Therefore, we do not need to keep dense matrix. Thanks to sparse matrix implementation, both memory consumption significantly decrease and the machine learning algorithms run faster.

For the values of the matrix, I tried different scenarios such as Term Frequency (TF), term frequency-inverse document frequency (TF-IDF) and Term Counts. Within three different ways, TF performs better.

Also, we know that the length of the e-mails varies a lot. There may be some e-mails with only one sentence or 500 sentences. TF can handle this high variance. It counts each terms occurring in an email field (subject or body) and, it divides the counted value to the total length of email field. It helps to handle changing length of the emails. Suppose *Hello*

term occurs twice in document *A* and it has 4 terms in total. Also, document *B* has three *Hello* terms and it has 9 terms. In this case, document *A* will have 0.5 scores for *Hello* term while document *B* has 0.3. So, the machine learning algorithm will give more importance to document *A*.

## 4.3 Creating Naive Bayes Classification Model

Naive Bayes is a probabilistic classifier. It is a simple, easy to understand and fast algorithm commonly used in text classification. Another big advantage is that python scikit-learn implementation supports to work with sparse data that we create and it is very efficient compared to a dense matrix.

In our data, the values are "discrete" and not binary. For this reason, I prefer to use Multinomial Naive Bayes implementation. In addition, we know that the distribution of target variable is not homogeneous because the number of spam emails is higher than ham emails. Therefore, I used stratified k-fold cross-validation to validate the model and to prevent over-fitting. Stratified k-fold keeps the percentage of samples for spam and ham stable. At the beginning of the model, I hold the default values and, I do not play with the Naive Bayes parameters.

## 4.4 Optimizing K for Stratified K-Fold Cross Validation

Finding more accurate results requires having a near optimal K value for the stratified k-fold cross-validation. We know that if k increases bias decrease but variation increase and if k decreases variation decrease but bias decrease. An ideal K should try to keep both variation and bias lower.

To find an ideal K parameter, I run Naive Bayes algorithm for both subjects and body attributes separately with different K values from 2 to 20. Note that random states are set to a static number for reproducibility purpose. For each k value, the algorithm runs 6 times with different random states. This process repeats for each k value. The errors of each fold for all random states are calculated with our custom cost function and saved into an array. After calculating error for each K values, the error arrays are plotted with a box plot to observe the variance. As we see from the k-fold graphs for both subject and body attributes, when k fold increases the variance also increases. Error for test set is higher for the very low k values. After 9 fold it is minimized and the train error does not decrease significantly. Therefore, it is reasonable to use 9 fold cross validation for both email and subject attributes.

**Figure 2:** Test and Train Set Errors for E-Mail Subject Attribute

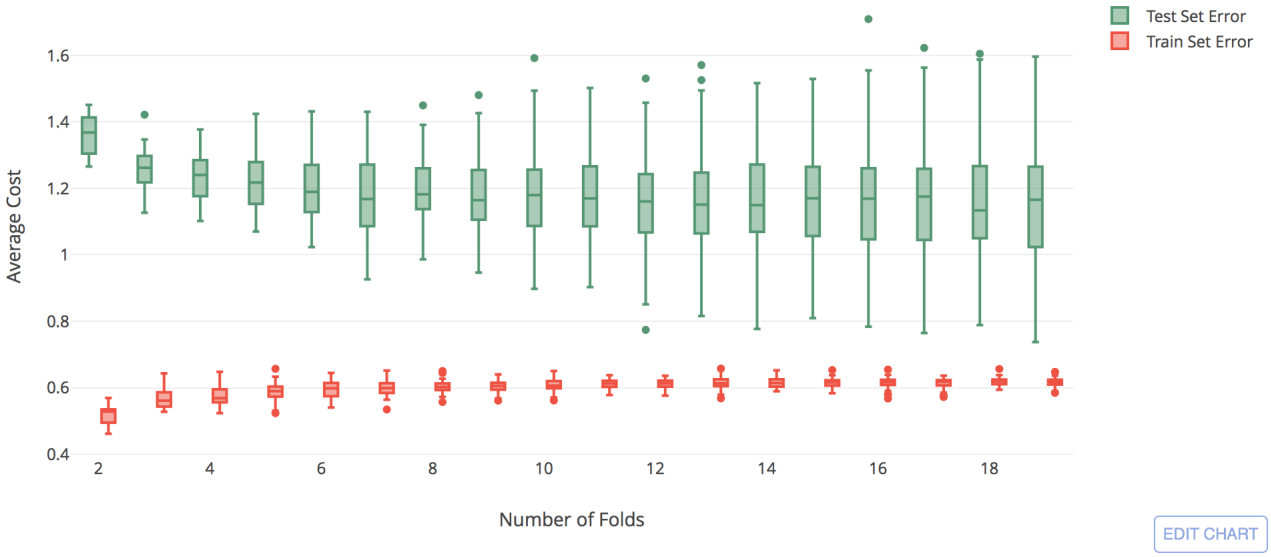
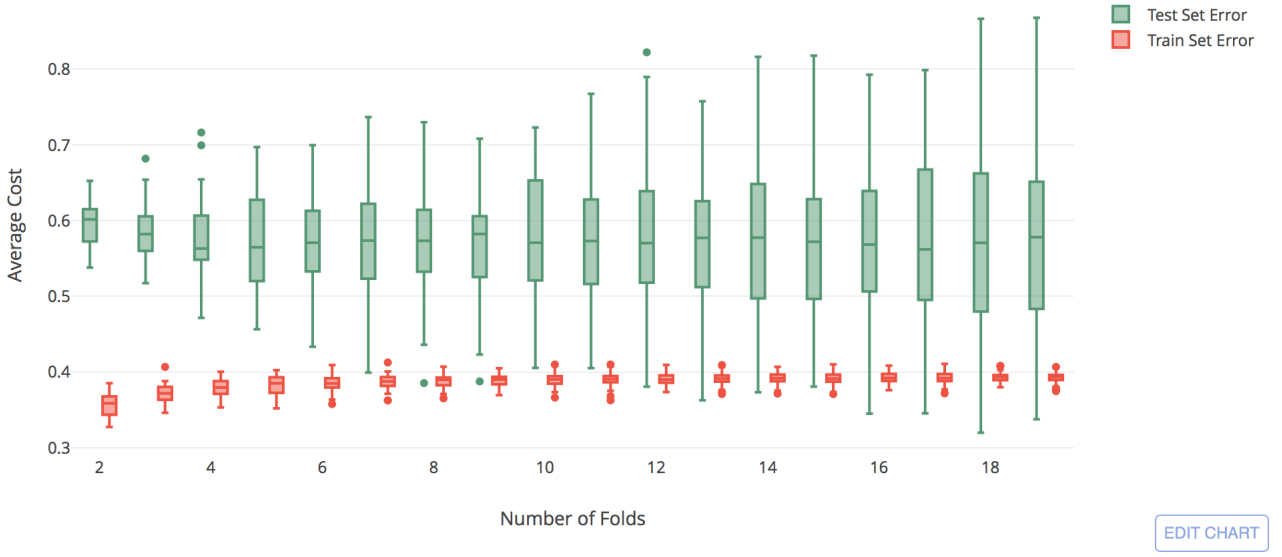



**Figure 3:** Test and Train Set Errors for E-Mail Body Attribute




## 4.5 Confusion Matrix of Naive Bayes For Subject and Body Attributes

After finding near-optimal K parameter for K-Fold cross-validation, I plot the confusion matrix to observe the accuracy of classifications for both subject and body attributes. The confusion matrix is seen below. Clearly, we see that classifying only with body outperforms classification with subjects attribute. For the simplicity, I did not include the suspicious class

at the early stages. For now, only spam or ham classification is performed.

For this case, accuracy is not that important because there is a significant cost difference between predicting ham email as spam and predicting spam email as ham. The former is 50 times more costly than the latter. Therefore, the purpose is to minimize the amount of misclassified ham email as spam.

By default, if $P(Spam) > P(Ham)$ then Naive Bayes algorithm classify that email as Spam or vice versa. However, we should classify an email as *Spam* only if it is most likely to be a spam. To define "most likely" term, I add a variable that creates a bias to compare $P(Spam)$ and $P(Ham)$. New equation turns out to be $P(Spam) > P(Ham) - i * 0.1$ where i is an integer changing from 0 to 10.

The Naive Bayes algorithm calculates the total cost for each i value and it returns the best i parameter that creates the lowest score. For the estimations using body attribute, the best i is 0.4. It means that if P(Spam) = 0.8 and P(Ham) = 0.2 then it will be classified as Spam because $P(Spam) - P(Ham) > 0.4$. On the other hand for the subject attribute, the best i is 0.8.

**Figure 4:** Naive Bayes Confusion Matrix For Email Body Attribute

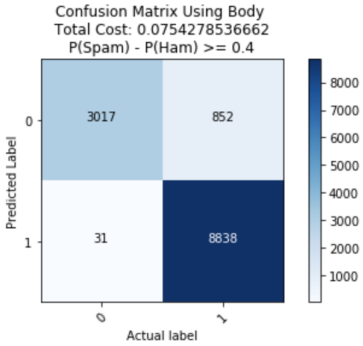
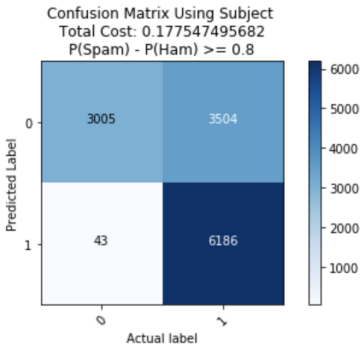



**Figure 5:** Naive Bayes Confusion Matrix For Email Subject Attribute

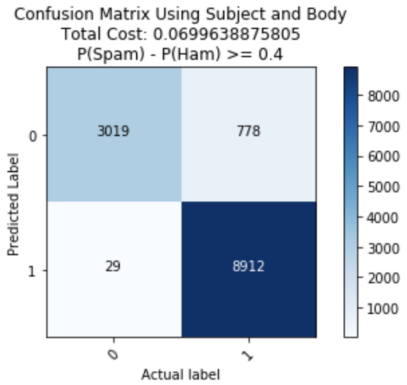


## 4.6 Combining Subject and Body Attributes

Only using subject attribute costs us 0.177 € and using body generates 0.075 € in average per e-mail. In this case, we have two different classifiers and two different $P(Ham)$ or $P(Spam)$ probabilities for each emails. To decrease total cost, I plan to combine these two subject and body attributes.

My strategy was to take the weighted average of $P(Spam)$ for both attributes. Because using *Email Body* attribute performs much better than *Email Subject*, giving higher weight to *Email Body* may seem reasonable. I test the weights with grid search and I create the following equation. $P(SpamAverage) = P(SpamBody) * i * 0.1 + P(SpamSubject) * (1-i) * 0.1$ where i is an integer between 0 and 10. After running this equation 11 times for each i values, the best weights were 0.8 for *Email Body* and 0.2 for *Email Subject*. The results are expected because *Email Body* performs better than *Email Subject* attribute. The promising part is the average cost per email prediction decreases from 0.075 € to 0.069 € thanks to this combination.

**Figure 6:** Naive Bayes Confusion Matrix For Email Subject and Body Attributes




## 4.7 Approximate Nearest Neighbor Search with Cos Similarity

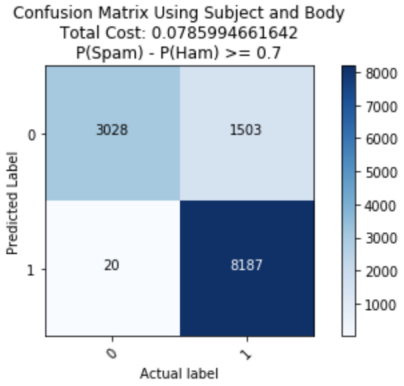
After obtaining predictions from Naive Bayes algorithm, I also want to observe the similarities between emails to increase prediction quality further and decrease total average cost. Because the lengths of emails vary a lot, I prefer to use cosine similarity as a distance measure to check the similarities. The idea is to find nearest 13 email body and subject based on cosine distance, ranking them with Borda Count Algorithm and predict $P(Spam)$ based on this ranking.

To implement the idea, I prefer to use one of the fastest Approximate Nearest Neighbor Search implementation called PySparNN. It finds the specified number of neighbors for a

given vector. In this case, the algorithm tries to find 13 nearest neighbors at maximum. If the cosine of neighbor vector and original vector is more than 0.92, then it will not count as a neighbor because the similarity is not that significant. Note that the neighbor vector that has the lowest score is the nearest neighbor.

In spam detection algorithm, I make an approximate nearest neighbor search for both subject and body fields separately. It means that I find at most 13 neighbors for the subject and 13 neighbors for body field. After, I combine all neighbors to calculate the probability of being a spam. I prefer to use Borda count to be able to give more ranking to the closest neighbors. Note that for each neighbor, we know its cosine similarity with the original vector and its target value (1 for spam and 0 for ham). After we combine subject and body neighbors for each email, I sort the neighbors cosine similarity in increasing order. Then, I give N points to the closest neighbor, N-1 to the second closest, N-2 to the third and by applying the same rule the farthest neighbor has 1 point. N is a total number of neighbors excluding the neighbors that has more than 0.92 cosine distance. After calculating points, I sum the points of spam emails and divide it to the total sum of all points to find the $P(Spam)$. Refer to Table 2 to better understand Borda Count ranking. At that table, all the neighbors are sorted based on cosine distance and the ranking points are given based on this sort. For this case $P(Spam)$ is calculated as $(5+4)/(7+6+5+4+3+2+1) = 0.32$

**Figure 7:** Approximate Nearest Neighbor Search Confusion Matrix For Subject and Body




.

**Table 2:** Example of Borda Count

| Email Index | Neighbor Type | Cos Distance | Target | Ranking Point |
|---|---|---|---|---|
| 1 | Body | 0.09 | 0 | 7 |
| 5 | Subject | 0.12 | 0 | 6 |
| 3 | Body | 0.125 | 1 | 5 |
| 5 | Body | 0.14 | 1 | 4 |
| 2 | Body | 0.28 | 0 | 3 |
| 8 | Subject | 0.6 | 0 | 2 |
| 7 | Subject | 0.9 | 0 | 1 |

## 4.8 Combining Naive Bayes and Approximate Nearest Neighbor Search Results

In this step, we have two prediction results coming from Naive Bayes and Approximate Nearest Neighbor Search algorithms. We need to combine the predictions of this two algorithms to give final classification decision.

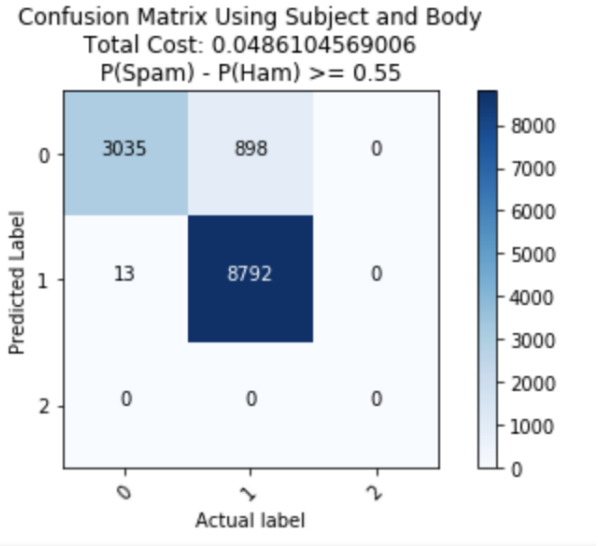
To give the final decision, we have three important scenarios that indicate our classification strategy

1. Email is forwarded: P(Spam) = 0
2. Approximate Nearest Neighbor Search finds less than 3 neighbors that satisfy our distance constraint: P(Spam) = Naive Bayes Prediction
3. Else: P(Spam) = i * 0.05 Naive Bayes Prediction + (1-i) * 0.05 * Approximate Nearest Neighbor Search Prediction where i is an integer changes from 0 to 20

The first two criteria are straightforward. For the third one, I apply grid search for all possible i values. Total prediction cost is calculated for each i value and the least cost determines the best i value. Thanks to this strategy, we can combine the predictions of two different algorithms. In this case, the best i is found as 0.55. It means that if the weighted average of Naive Bayes model is 0.55 and the approximate nearest neighbor search algorithm weight is 0.45, we find the least average score per e-mail.

Before this step, we did not include the suspicious class for the simplicity. Our data does not have a suspicious class. We, therefore, cannot train our algorithms to detect suspicious e-mails. What we can do is that we can classify emails as suspicious if $P(Spam) > P1$ and $P(Spam) \leq P2$. In this case, we need to optimize two parameters P1 and P2 to minimize the average total cost per email. The constraints for P1 and P2 are $0 \leq P1 \leq P2 \leq 1$. By using mixed integer programming, we can find the optimal exact values of P1 and P2. For simplicity,

**Figure 8:** Approximate Nearest Neighbor Search and Naive Bayes Combination




I prefer to use grid search. By satisfying the constraints, I increment P1 and/or P2 by 0.025 until reaching 1. For this settings, the best cost is generated when $P1 = P2 = 0.725$. It means that classifying as suspicious increase our costs. The model does not choose to classify mails as suspicious.

## 5 Evaluating The Performance of the Model

To apply this algorithm in real life, we need to know the ratio of spam emails and all the emails in an email stream. After improving the spam detection algorithm, the best average cost per email is found as 0.048 €. If we do not use this algorithm and suppose all the emails in an email stream is spam, then the average cost per e-mail would be 0.4 €. If we use this algorithm, it saves money if the incoming spam emails ratio is more than 0.048/0.4 = 12%. If it is less than this amount the algorithm should not be used.

This model also has some weaknesses. Users from 11 to 20 included into train data as ham emails. However, approximately 5% of this emails are spam emails but, these emails are given to the train data as ham emails. At the very beginning, I do not plan to include Users 11 to 20. My plan was to train machine learning algorithms with the first 10 users and then try to predict the rest of the users to classify their inbox as spam or ham to find the hidden 5% spam emails. After finding spam and ham emails, I would expand my train data with the "classified" Users 11 to 20 data. The problem was the error of the model was not even close to 5%. Therefore instead of trying to classify those inbox, I directly included into training data

as ham email.