# CSC 425

# Spam Filter Project- Milestone 2

Instructor: Dr. Yangyang Tao

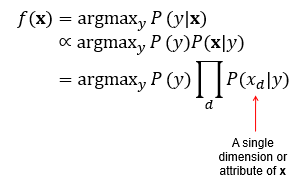
Semester: Fall 2022

Team Members: Trang Do, Ethan Ennis, Maen Marashdeh & Patricia Aguilar (Worldwide Group)

**I. Project Introduction**

In this project, we are trying to implement the process of filtering spam emails. We are going to try to build an expert system to create some set of rules that will help us exceed the accuracy of the classifiers that are generated automatically. The typical spam filter is a program used to detect unwanted, unsolicited and virus-infected emails and prevent those messages from getting to a user’s inbox and move those emails to a user’s spam inbox folder. To know the difference between a spam email and another that is not, they use specific filtering methods to identify the content of the emails or their senders and then mark the email as spam; spam filters base their judgement on certain criteria, as do other types of filtering programs. For instance, one of the simplest and earliest methods can be set to watch for particular words the subject lines of messages and exclude them from the user’s inbox.

We are going to use Multinomial Naïve Bayes Classifier, a probabilistic learning method, for this project. The Naïve Bayes Classifier attempts to classify an email to ham, or spam based on the frequency of words in the email. Multinomial Naïve Bayes Classifier method consists of a sample x, and a class label y that we want to find; normally we know what a category looks like and based on that we can collect a set of unknown samples, so we have P(x|y). ∝ means “proportional to” and is used to indicate change in relation to something else.



The elements of a function’s domain where the function values are maximized are known as the arguments of the maxima in mathematics. Arg max refers to the inputs, or arguments, at which the function outputs are as large as possible. Multinomial Naïve Bayes Classifier classifies the email to the class with higher probability.  
**II. Project Dataset Paragraph (50 points)**

This dataset consists of 962 emails in total, divided into training set and test set. In the original dataset file, there are 702 text files for training and 260 text files for testing. We will split the 962 files into training and test data with different ratios, which will be discussed in the Methodology section. Both training and test dataset have equal number of normal emails and spam emails. Normal emails files are named “X-XXXmsgX”, and spam files are named “spmXXX”.

In this project, we use 𝑑 ∈ 𝐷 represents one email text file, where 𝐷 is all the email files. 𝐶 = {𝑐1, 𝑐2} = {0, 1} represent ham, spam classes, respectively. For the email files in the training set we have the following file and label pair, < 𝑑, 𝑐 >   
**III. Project Methodology (120 points)**   
● Flowchart showing the working of all the steps of your program.

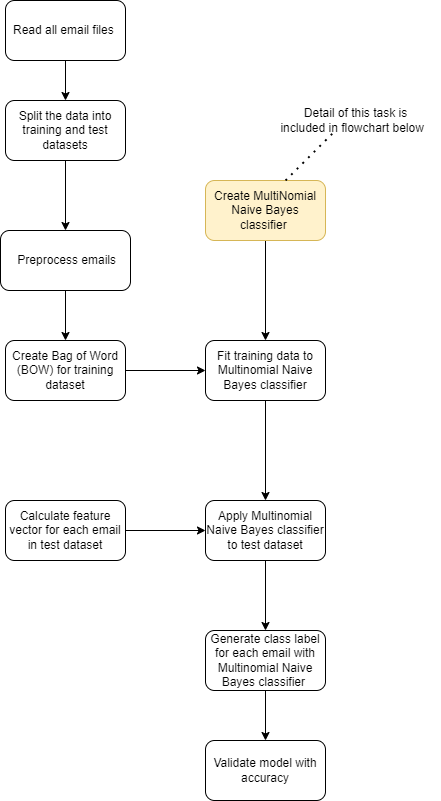


Fig 1: Flowchart of implementing Spam Filter

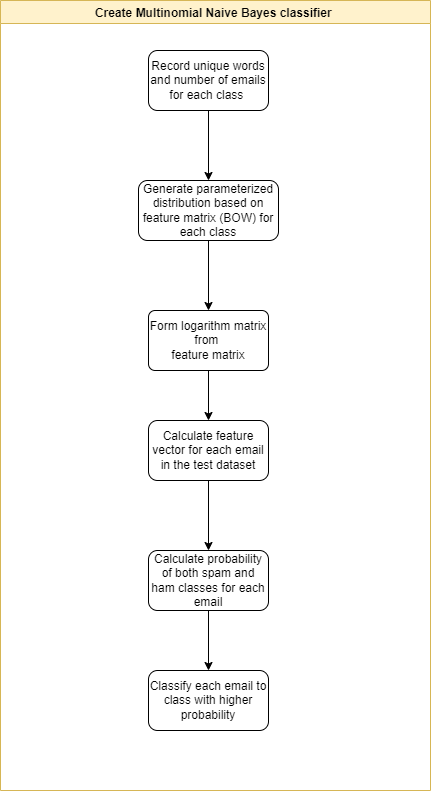


Fig 2: Flowchart of implementing Multinomial Naïve Bayes

Step-by-step explanation of the flowchart:

Fig 1: Flowchart of implementing Spam Filter

(1) Read all email files

Use the read\_file() function to extract content in each email file and store it.

(2) Split the data into training and test datasets

In this step, we split the original dataset into training and test datasets with specified ratios:

(i) training data: 60% and test data: 40%

(ii) training data: 75% and test data: 25%

(iii) training data: 80% and test data: 20%

(iv) training data: 85% and test data: 15%.

(3) Preprocess emails

 Preprocess emails by removing stop words, tokenizing and lemmatizing from email.

* Stop words are a set of commonly used words in any language such as “a,” “the,” “an,” etc. These words do not add much meaning to the text. Therefore, we remove those words.
* Tokenization refers to separating the text into a specified unit (one word in our case)
* After the emails are tokenized, we apply lemmatization on each word. Lemmatization refers to transforming words to their root form

(4) a - Create Bag of Words (BOW)

The BOW counts occurrences of a particular word/token in our training dataset. The higher the occurrence, the more important the word in the text. For each token or word, we will have a feature column, this is called text vectorization. We take all unique words from the training set to form a dictionary, then we use text vectorization to represent each of the email files. However, the dictionary may contain too many unique words, and some of these words may not convey useful meanings. Therefore, we sort the words in the dictionary based on their frequency and choose the 3000 most frequent words in the original dictionary to form a new dictionary for the BOW.

(4) b - Create Multinomial Naive Bayes classifier (Fig 2)

(i) Record unique words and number of emails for each class (spam and ham specifically)

(ii) Generate parameterized distribution based on feature matrix (BOW) for each class

(iii) Form a logarithm matrix from the feature matrix. The logarithm matrix is shown below

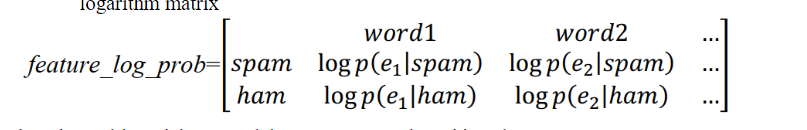
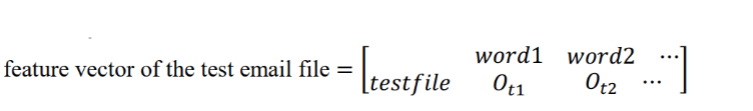


Fig 3: Logarithm matrix

(iv) For each email file in the test set, we calculate its related feature vector based on the word in the dictionary



(v) Calculate probability of both spam and ham classes for each email

The probability of each class is calculated by doing elementwise product for both each class and adding log of corresponding prior probability.

For ham class probability:

ℎ0\_𝑡𝑒𝑠𝑡= ∑1≤𝑘≤𝑛𝐸 log 𝑃(𝑒𝑘|ℎ0) ∗ 𝑂𝑡𝑒𝑘 + log 𝑃(ℎ0)

For spam class probability:

ℎ1\_𝑡𝑒𝑠𝑡= ∑1≤𝑘≤𝑛𝐸 log 𝑃(𝑒𝑘|ℎ1) ∗ 𝑂𝑡𝑒𝑘 + log 𝑃(ℎ1)

vi - Classify each email to class with higher probability

If ℎ0\_𝑡𝑒𝑠𝑡 > ℎ1\_𝑡𝑒𝑠𝑡, then the test email file belongs to the ham class. Otherwise, the test email file

belongs to the spam class.

(5) Fit training data to Multinomial Naive Bayes classifier

By creating a MultinomialNB\_class() object with training data, we generate a Multinomial Naive Bayes classifier with feature matrix and log of prior probability for each class.

(6) Calculate feature vector for each email in test dataset

Similar to (4)-a, we create a feature vector for each email in the test dataset. The feature columns are feature columns created from (4)-a for training dataset.

(7) Apply Multinomial Naive Bayes classifier to test dataset

(8) Generate class label for each email with Multinomial Naive Bayes classifier

By applying MultinomialNB\_predict() function in Multinomial Naive Bayes classifier, we can calculate the probability of an email being spam or ham. The function to calculate probability of each class is implemented in (4)-b-(v).

(9) Validate model with accuracy

We validate the Multinomial Naive Bayes classifier by calculating the proportion of correctly predicted emails to the total number of emails in test dataset.

* **Pseudocode**

**(1)*. SpamEmailFilter.py***

Step 1: Read all emails files

For each file in train and test folder

move to all\_mail folder

End for

Step 2: Split data into training and testing set based on given training proportion

train\_count = total\_emails \* train\_proportion

count\_ham =0

count\_spam = 0

For each file in all\_mail folder

if count\_ham <= train\_count//2 and file\_name contains ‘msg’

move to train folder

count\_ham +=1

end if

if count\_spam <= train\_count//2 and file\_name contains ‘spm’

move to train folder

count\_spam +=1

end if

End for

For each file in all\_mail folder

move to test folder

End for

Step 3: Preprocess data

Remove stopwords and punctuation and lemmatize data with nltk package

Step 4a: Create Bag of Words (BOW) for training dataset

wordMap = {}

commonMap = []

For each file in train folder

for each word in file

if word not in wordMap.keys()

wordMap[word] = 1

else

wordMap[word] += 1

End for

End for

Sort wordMap by value in descending order

Add 3000 first keys of wordMap to commonMap

Create feature vectors for each emails based on 3000 most common words

Step 4b: Create Multinomial Naïve Bayes classifier (Implemented in next slide)

Step 5: Fit training data to Multinomial Naïve Bayes classifier

MultinomialNB = MultinomialNB\_class()

#generate a Multinomial Naive Bayes classifier with feature matrix and log of prior probability for each class

MultinomialNB.MultinomialNB(train\_features, train\_labels)

Step 6: Calculate feature vector for each email in test dataset

Do the same as Step 4a, but with test dataset

Step 7 and 8: to apply Multinomial Naïve Bayes predictor and generate class label for each email

classes = MultinomialNB.MultinomialNB\_predict(test\_features)

Step 9: Validate model with accuracy

Acc = 0

For each file in test folder

if email type is the same as predicted

acc += 1

Accuracy = (float(acc) / number of file in test folder)\*100

**(2)*. MultinomialNB.py***

Step 1: Record unique words and number of emails for each class (spam and ham specifically)

label\_count = np.zeros(2)

for i in labels :

label\_count[int(i)] += 1

Step 2: Generate parameterized distribution based on feature matrix (BOW) for each class

class\_log\_prior = [0.0, 0.0]

class\_log\_prior[0] = math.log(float(label\_count[0])/float(len(labels)))#ham

class\_log\_prior[1] = math.log(float(label\_count[1])/float(len(labels)))#spam

Step 3: Form a logarithm matrix from the feature matrix

smooth\_alpha = 1

for j in range(len(features)) :

for k in range(len(features[j])) :

if label is 0

ham[k] += features[j][k]

sum\_ham += features[j][k]

else

spam[k] += features[j][k]

sum\_spam += features[j][k]

for l in range(most\_common\_word) :

ham[l] += smooth\_alpha

spam[l] += smooth\_alpha

sum\_ham += smooth\_alpha\*most\_common\_word

sum\_spam += smooth\_alpha\*most\_common\_word

for h in range(most\_common\_word) :

feature\_log\_prob[0][h] = math.log(float(ham[h])/float(sum\_ham))

feature\_log\_prob[1][h] = math.log(float(spam[h])/float(sum\_spam))

Step 4: Calculate its feature vector for each email in the test dataset

This has been done in SpamEmailFilter.py, no further actions required

Step 5: Calculate probability of both spam and ham classes for each email

classes = np.zeros(len(features))

ham\_prob = 0.0

spam\_prob = 0.0

for i in range(len(features))

ham\_prob = 0.0

spam\_prob = 0.0

for j in range(len(features[i]))

ham\_prob += feature\_log\_prob[0][j]\*float(features[i][j])

spam\_prob += feature\_log\_prob[1][j]\*float(features[i][j])

ham\_prob += class\_log\_prior[0]

spam\_prob += class\_log\_prior[1]

Step 6: Classify each email to class with higher probability

if ham\_prob > spam\_prob :

classes[i] = HAM

else :

classes[i] = SPAM

v. Project Result presentation (120 points)

* Split the data into 60.0 % training and 40.0 % testing

The maximum of most\_common can be: 14729

Accuracy of Multinomial Naive Bayes: 96.88%

* Split the data into 75.0 % training and 25.0 % testing

The maximum of most\_common can be: 17267

Accuracy of Multinomial Naive Bayes: 95.83%

* Split the data into 80.0 % training and 20.0 % testing

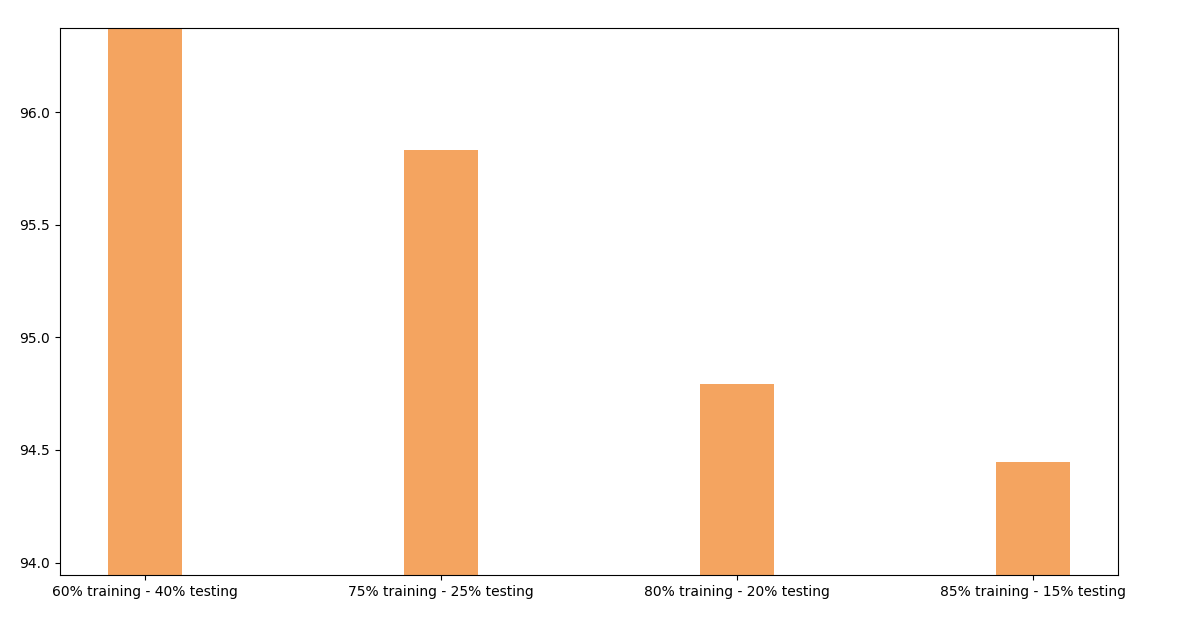
The maximum of most\_common can be: 18524

Accuracy of Multinomial Naive Bayes: 94.79%

* Split the data into 85.0 % training and 15.0 % testing

The maximum of most\_common can be: 19202

Accuracy of Multinomial Naive Bayes: 94.44%



c. Conclusion or summary paragraph. (40 points)

In this project, we cleaned the data and implemented the process of filtering spam emails. We found with 60% training and 40% testing that the filter was at its most accurate state...96.88%. Our project first reads all the email files, splits them into two buckets (training & test), where we then create a bag of words. This bag of words was used to train the program on what words tend to appear in spam emails vs. Ham emails by using a multinomial naïve bayes classifier when testing with results. After that was complete, we analyzed the data.

3. Turn in your source code as one zip file to corresponding Canvas drobox. – 100 points   
  
4. You’re encouraged to participate in the NKU Celebration with your project poster. Turn in your project   
poster if you attended the NKU Celebration activity. – Extra 100 bonus points.