ABSTRACT

Machine learning is a branch of artificial intelligence concerned with the creation and study of systems that can learn from data. A machine learning system could be trained to distinguish between spam and non-spam (ham) emails. We aim to analyze current methods in machine learning to identify the best techniques to use in content-based spam filtering.
# Table of Contents

1. Introduction .......................................................................................................................... 3

2. Machine Learning Basics .................................................................................................... 4

3. Methods ................................................................................................................................. 5
   3.1 Preprocessing .................................................................................................................... 5
      3.1.1 Creating the Feature Matrix ...................................................................................... 5
      3.1.2 Reducing the Dimensionality .................................................................................. 6
   3.2 Classifiers .......................................................................................................................... 7
      3.2.1 $k$-Nearest Neighbors .............................................................................................. 7
      3.2.2 Decision Tree ........................................................................................................... 8
      3.2.3 Naïve Bayesian ......................................................................................................... 10
      3.2.4 Logistic Regression .................................................................................................. 11
   3.3 Implementation .................................................................................................................. 12

4. Results .................................................................................................................................. 13
   4.1 Performance Depending on Hashing .............................................................................. 13
   4.2 Performance of $k$-Nearest Neighbors .......................................................................... 14
   4.3 Performance of Different Classifiers .............................................................................. 14

5. Conclusion ............................................................................................................................ 15

6. Acknowledgements ............................................................................................................... 16

7. References ............................................................................................................................. 17
1. Introduction

Email has become one of the most important forms of communication. In 2014, there are estimated to be 4.1 billion email accounts worldwide, and about 196 billion emails are sent each day worldwide.[1] Spam is one of the major threats posed to email users. In 2013, 69.6% of all email flows were spam.[2] Links in spam emails may lead to users to websites with malware or phishing schemes, which can access and disrupt the receiver’s computer system. These sites can also gather sensitive information. Additionally, spam costs businesses around $2000 per employee per year due to decreased productivity.[3] Therefore, an effective spam filtering technology is a significant contribution to the sustainability of the cyberspace and to our society.

There are currently different approaches to spam detection. These approaches include blacklisting, detecting bulk emails, scanning message headings, greylisting, and content-based filtering:[4]:

- Blacklisting is a technique that identifies IP addresses that send large amounts of spam. These IP addresses are added to a Domain Name System-Based Blackhole List and future email from IP addresses on the list are rejected. However, spammers are circumventing these lists by using larger numbers of IP addresses.

- Detecting bulk emails is another way to filter spam. This method uses the number of recipients to determine if an email is spam or not. However, many legitimate emails can have high traffic volumes.

- Scanning message headings is a fairly reliable way to detect spam. Programs written by spammers generate headings of emails. Sometimes, these headings have errors that cause them to not fit standard heading regulations. When these headings have errors, it is a sign that the email is probably spam. However, spammers are learning from their errors and making these mistakes less often.

- Greylisting is a method that involves rejecting the email and sending an error message back to the sender. Spam programs will ignore this and not resend the email, while humans are more likely to resend the email. However, this process is annoying to humans and is not an ideal solution.

Current spam techniques could be paired with content-based spam filtering methods to increase effectiveness. Content-based methods analyze the content of the email to determine if the email is spam. The goal of our project was to analyze machine learning algorithms and determine their effectiveness as content-based spam filters.
2. Machine Learning Basics

When we choose to approach spam filtering from a machine learning perspective, we view the problem as a classification problem. That is, we aim to classify an email as spam or not spam (ham) depending on its feature values. An example data point might be \((x, y)\) where \(x\) is a \(d\)-dimensional vector \(<x^{(1)}, x^{(2)}, ..., x^{(d)}>\) describing the feature values and \(y\) has a value of 1 or 0, which denotes spam or not spam.

Machine learning systems can be trained to classify emails. As shown in Figure 1, a machine learning system operates in two modes: training and testing.

Training
During training, the machine learning system is given labeled data from a training data set. In our project, the labeled training data are a large set of emails that are labeled spam or ham. During the training process, the classifier (the part of the machine learning system that actually predicts labels of future emails) learns from the training data by determining the connections between the features of an email and its label.

Testing (Classification)
During testing, the machine learning system is given unlabeled data. In our case, these data are emails without the spam/ham label. Depending on the features of an email, the classifier predicts whether the email is spam or ham. This classification is compared to the true value of spam/ham to measure performance.

![Figure 1. A model of the machine learning process. Image Credit: “Statistical Pattern Recognition”](image-url)
3. Methods

Figure 1 shows the different steps in the machine learning process. First, in testing and training, the data must be preprocessed so that they able to be used by the classifier. Then, the classifier operates in training or testing mode to either learn from the preprocessed data or classify future data.

3.1 Preprocessing

Our classifiers require a feature matrix that consists of the count of each word in each email. As will we discuss, this feature matrix can grow very large, very quickly. Thus, preprocessing the data involves two main steps: creating the feature matrix and reducing the dimensionality of the feature matrix.

3.1.1 Creating the Feature Matrix

Figure 2 shows an example of an email in our dataset, the TREC 2007 corpus. This corpus is described in more detail in Section 3.3.

![Figure 2. An example of an email in the TREC 2007 dataset.](image)

There are a couple steps involved in creating the feature matrix, as shown in Figure 3. After preliminary preprocessing (removing HTML tags and headers from the email in the dataset), we take the following steps:
1. **Tokenize** - We create "tokens" from each word in the email by removing punctuation.

2. **Remove meaningless words** – Meaningless words, known as stop-words, do not provide meaningful information to the classifier, but they increase dimensionality of feature matrix. In Figure 3, the red boxes outline the stop-words, which should be removed. In addition to many stop-words, we removed words over 12 characters and words less than three characters.

3. **Stem** - Similar words are converted to their “stem” in order to form a better feature matrix. This allows words with similar meanings to be treated the same. For example, history, histories, historic will be considered same word in the feature matrix. Each stem is placed into our "bag of words", which is just a list of every stem used in the dataset. In Figure 3, the tokens in blue circle are converted to their stems.

4. **Create feature matrix** - After creating the "bag of words" from all of the stems, we create a feature matrix. The feature matrix is created such that the entry in row \( i \) and column \( j \) is the number of times that token \( j \) occurs in email \( i \).

These steps are completed using a Python NLTK (Natural Language Toolkit).

![Figure 3. A summary of the preprocessing involved in creating a feature matrix.](image)

### 3.1.2 Reducing the Dimensionality

The bag of word method creates a highly dimensional feature matrix and this matrix grows quickly with respect to the number of emails considered. A highly dimensional feature matrix greatly slows the runtime of our algorithms.

When using 50 emails, there are roughly 8,700 features, but this number quickly grows to over 45,000 features when considering 300 emails (as shown in Figure 4). Thus, it is necessary to reduce the dimensionality of the feature matrix.
To reduce the dimensionality, we implemented a hash table to group features together. Each stem in the bag of words comes with a built-in hash index in Python. We can then decide how many hash buckets (or features) we would like to have. We take the built-in hash index mod the bucket size to find the new hashed index. This process is shown in Figure 5.

![Figure 5: The process of hashing different features to hash buckets.](image)

After this pre-processing is finished, the classifiers are able to use the feature matrix in its training and testing modes.

### 3.2 Classifiers

As mentioned before, a machine learning system can be trained to classify emails as spam or ham. To classify emails, the machine learning system must use some criteria to make its decision. The different algorithms that we describe below are different ways of deciding how to make the spam or ham classification.

#### 3.2.1 $k$-Nearest Neighbors

The basic idea behind the $k$-Nearest Neighbors ($k$NN) algorithm is that similar data points will have similar labels. $k$NN looks at the $k$ closest (and thus, most similar) training data points to a testing data point. The algorithm then combines the labels of those training data to determine the testing data point's label. Figure 6 shows a visual representation of the $k$-Nearest Neighbors algorithm for different values of $k$. 

![Figure 4: The growth of number of features with respect to the number of emails considered.](image)
For our $k$-Nearest Neighbors classifier, we used Euclidean distance as the distance measure. Euclidean distance is calculated as shown in Equation 1. When implementing the $k$-Nearest Neighbors algorithm, the designer can choose which distance metric she wants to use. Some other possibilities include Scaled Euclidean and L1-norm.

$$
\left[ (x_j^{(1)} - x_{new}^{(1)})^2 + (x_j^{(2)} - x_{new}^{(2)})^2 + \ldots + (x_j^{(d)} - x_{new}^{(d)})^2 \right]^{1/2}
$$

Equation 1. Calculation of Euclidean distance from a training data point $x_j$ to a testing data point $x_{new}$

After determining the $k$ nearest neighbors, the algorithm combines those neighbors’ labels to determine the label of the testing data point. For our implementation of $k$-Nearest Neighbors, we combined the labels used a simple majority vote. That is, if the neighbor is one of the $k$ nearest neighbors, we consider its label equally likely for the new data point. Other possibilities include a distance-weighted vote, in which labels of data points that are closest to the testing point have more of a vote for the testing point's label.

### 3.2.2 Decision Tree

The decision tree algorithm aims to build a tree structure according to a set of rules from the training dataset, which can be used to classify unlabeled data. In our implementation, we used the well-known iterative Dichotomiser 3 (ID3) algorithm invented by Ross Quinlan to generate the decision tree.

#### ID3 Tree Before Pruning

This algorithm builds the decision tree based on entropy and information gain. Entropy measures the impurity of an arbitrary collection of samples while the information gain calculates the reduction in entropy by partitioning the sample according to a certain attribute. $[9]$ Given a dataset $D$ with categories $c_j$, entropy are calculated by the following equation:

$$
entropy(D) = - \sum_{j=1}^{c} Pr(c_j) \log_2 Pr(c_j)
$$
Entropy and information gain are related by the following equation, where \( \text{entropy}_{Ai}(D) \) is the expected entropy when using attribute \( Ai \) to partition the data.

\[
\text{gain}(D, Ai) = \text{entropy}(D) - \text{entropy}_{Ai}(D)
\]

The algorithm was implemented according to the following steps:
1. Entropy of every feature at every value in the training dataset was calculated.
2. The feature \( F \) and value \( V \) with the minimum entropy was chosen. If more than one pair have the minimum entropy, arbitrarily chose one.
3. The dataset was split into left and right subtrees at the \( \{F, V\} \) pair.
4. Repeat steps 1-3 on the subtrees until the resulting dataset is pure, i.e., only contains one category. The pure data are contained in the leaf node.

By splitting the dataset according to minimum entropy, the resulting dataset has the maximum information gain and thus impurity of the dataset is minimized. After the tree is built from the training dataset, testing data points can be fed into the tree. The testing data will go through the tree according to the predefined rules until reaching a leaf node. The label in the leaf node is then assigned to the testing data point.

![Diagram of a tree](image)

Figure 7. A example of a tree before pruning. \( F \) refers the features, or words in the case of spam filter. \( V \) refers the values, or word frequencies in this case. \( C \) represents the labels, which are spam/ham for spam filter.

**Pruned ID3 Tree**
The ID3 tree algorithm requires all leaf nodes to contain pure data. Thus, the resulting tree can overfit training data. A pruned tree can solve this problem and improve the classification accuracy and computation efficiency.

Suppose the ID3 tree without pruning is \( T \) and the classification accuracy by \( T \) is \( T_{\text{Accuracy}} \). Pruning iteratively deletes subtrees of \( T \) at the bottom to form a new tree and reclassify the testing data using the new tree. If the accuracy increases from \( T_{\text{Accuracy}} \), then the pruned tree will replace \( T \). The resulting pruned tree has less nodes, smaller depth, and higher classification accuracy.
3.2.3 Naïve Bayesian

The Naïve Bayesian classifier takes its roots in the famous Bayes Theorem:

\[ P(H|e) = \frac{P(e|H)P(H)}{P(e)} \]

Bayes Theorem essentially describes how much we should adjust the probability that our hypothesis \( H \) will occur, given some new evidence \( e \).

For our project, we want to determine the probability that an email is spam, given the evidence of the email's features \( F_1, F_2, ..., F_n \). These features \( F_1, F_2, ..., F_n \) are just a boolean value (0 or 1) describing whether or not the stem corresponding to \( F_1 \) through \( F_n \) appears in the email. Then, we compare \( P(\text{Spam}| F_1, F_2, ..., F_n) \) to \( P(\text{Ham}| F_1, F_2, ..., F_n) \) and determine which is more likely. Spam and ham are considered the classes, which are represented in the equations below as "C". We calculate these probabilities using the following equation:

\[ p(C \mid F_1, ..., F_n) = \frac{p(C)p(F_1, ..., F_n \mid C)}{p(F_1, ..., F_n)} \]

Equation 2. Equation rooted in Bayes Theorem for determining whether an email is spam or ham given its features.

Note that when we are comparing \( P(\text{Spam}| F_1, F_2, ..., F_n) \) to \( P(\text{Ham}| F_1, F_2, ..., F_n) \), the denominators are the same. Thus, we can simplify the comparison by noting that:

\[ p(C \mid F_1, ..., F_n) \propto p(C)p(F_1, ..., F_n \mid C) \]

Calculating \( P(F_1, F_2, ..., F_n | C) \) can become tedious. Thus, the Naïve Bayesian classifier makes an assumption that drastically simplifies the calculations. The assumption is that the probabilities of \( F_1, F_2, ..., F_n \) occurring are all independent of each other. While this assumption is not true (as some words are more likely to appear together), the classifier still performs very well given this
assumption. Now, to determine whether an email is spam or ham, we just select the class \( C = \text{spam} \) or \( C = \text{ham} \) that maximizes the following equation:

\[
\arg \max_c \ p(C = c) \prod_{i=1}^{n} p(F_i = f_i \mid C = c) 
\]

Equation 3. Decision criterion for the Naive Bayes classifier.

3.2.4 Logistic Regression

Logistic Regression is another way to determine a class label, depending on the features. Logistic regression takes features that can be continuous (for example, the count of words in an email) and translate them to discrete values (spam or not spam). A logistic regression classifier works in the following way:

1. Fit a linear model to the feature space determined by the training data. This requires finding the best parameters to fit the training data. For example, the red line in Figure 9 is described by \( z = \theta_0 + \sum_{i=1}^{n} \theta_i X_i \).

![Figure 9. An example of a linear model fit to the feature space. Image Credit: http://en.wikipedia.org/wiki/File:Linear_regression.svg](http://en.wikipedia.org/wiki/File:Linear_regression.svg)

2. Using the parameters found in step 1, determine the \( z \) value for a testing point.
3. Map this \( z \) value of the testing point to the range 0 to 1 using the logistic function (shown in Figure 10). This value is one way of determining the probability that these features are associated with a spam email.
As shown above, the logistic function helps us translate a continuous input (the word counts) to a discrete value (0 or 1, or equivalently, ham or spam). The graph above is described by the logistic function, which is the decision criterion for the machine learning system. We compare the probability that the email is spam to the probability that the email is ham.

\[
p(\text{Spam} \mid z) = \frac{1}{1 + e^{-z}} \quad p(\text{Ham} \mid z) = 1 - p(\text{Spam} \mid z) = \frac{e^{-z}}{1 + e^{-z}}
\]

Equation 4. Probability an email is spam given its z value. Equation 5. Probability an email is ham given its z value.

3.3 Implementation

The dataset used in this report was obtained from 2007 Text REtrieval Conference (TREC) public spam corpus, which contains messages delivered to a particular server between April 8, 2007 and July 6, 2007. The results were based on a training sample of 800 emails and a testing sample of 200 emails. This means our classifiers learned from the data in the training set of 800 emails. Then, we determined the performance of the classifiers by observing how the 200 emails in the testing set were classified.

All programming was implemented in Python and Matlab.
4. Results

We used the following performance metrics to evaluate our classifiers:
- **Accuracy**: % of predictions that were correct
- **Recall**: % of spam emails that were predicted correctly
- **Precision**: % of emails classified as spam that were actually spam
- **F-Score**: a weighted average of precision and recall

4.1 Performance Depending on Hashing

As described in the preprocessing section, we can combine features by using hashing. This reduces the dimensionality of our feature matrix, which reduces run time. However, by combining features, the performance of the classifiers could potentially become worse.

We discovered that reducing the dimensionality did not have a large effect on most of our classifiers. As seen in Figure 11, the One Nearest Neighbor and Logistic Regression classifiers were unaffected by the number of hash buckets used. The Naive Bayesian classifier performed better with a largest number of hash buckets. The precision of the Naive Bayesian classifier increased from 73.5% when using 512 hash buckets to 87.4% when using 2048 hash buckets. The ID3 Decision Tree performed best with 1024 hash buckets.

For future performance analysis, we used 1024 hash buckets.

![Figure 11. The performance of different algorithms depending on the number of hash buckets used.](image-url)
4.2 Performance of $k$-Nearest Neighbors

The performance of the $k$-Nearest Neighbors classifier depends on the number of neighbors, $k$, that we consider in our majority vote. The figure below shows the performance of the $k$-Nearest Neighbor classifier as the value of $k$ is increased. Interestingly, the One-Nearest Neighbor classifier ($k$-Nearest Neighbor with $k = 1$) performed the best (as seen in Figure 12).

![Performance of k-Nearest Neighbors Classifier](image)

Figure 12. The performance of the $k$-Nearest Neighbors classifier depending on the number of neighbors, $k$, considered.

4.3 Performance of Different Classifiers

Using a 1024 hash buckets, we compared the performance of the different classifiers.

![Performance of Different Classifiers](image)

Figure 13. Performance of different classifiers.
As seen in Figure 13, the One-Nearest Neighbor algorithm performed the best with the following performance metrics:

- Accuracy 99.00%
- Precision 98.58%
- Recall 100%
- F Score 99.29%

The ID3 Decision Tree also performed well. The ID3 Decision Tree without pruning achieved the following performance:

- Accuracy 97.00%
- Precision 96.50%
- Recall 99.28%
- F Score 97.87%

After pruning, the ID3 Decision Tree achieved the following performance:

- Accuracy 98.50%
- Precision 97.89%
- Recall 100%
- F Score 98.93%

Overall, these performances are very good, but could be improved using more tuning of parameters and methods.

5. Conclusion

We found that the One-Nearest Neighbor algorithm outperformed the other classifiers. This simple algorithm achieved great performance and was easy to implement. However, we believe that the performance of this algorithm could still be improved. We encourage those who wish to further this research to investigate the effects of a weighted majority vote, enhanced feature selection, and different distance measures.

Another key finding was that we discovered that the recall for all of the algorithms was very high, while the precision was lower. This suggests that our algorithms are very liberal in labeling an email as spam. To combat this, perhaps mapping the features to a higher dimension, as is done in Support Vector Machine algorithms, would be a solution to this problem.

After analysis, we believe that a machine learning approach to spam filtering is a viable and effective method to supplement current spam detection techniques.
6. Acknowledgements

Thank you to our research mentors Xiaoxiao Xu and Arye Nehorai for their guidance and support throughout this project. We would also like to thank Ben Fu, Yu Wu, and Killian Weinberger for their help in implementing the preprocessing, decision tree, and hash tables, respectively.
7. References


