ECS 171 Project Report: Spam Detection

Elisabeth Henkens, Nathan Hoffman, Randal Murphy
Jordan Palmer, Ruvim Lashchuk, Cathy Hsieh
Nico Lupacchino, Joyce Christiansen-Salameh, Hannah Brown
Ethan Chiang, Michael Nguyen

November 2019

https://github.com/nathanah/ECS171Project

Contents

1 Abstract **3**

2 Introduction **3**

3 Pre-processing **3**
- 3.1 Local Outlier Factor . . . 3
- 3.2 Isolation Forest . . . 3

4 Neural Network **4**
- 4.1 Activation Functions . . . 5
  - 4.1.1 Sigmoid . . . 5
  - 4.1.2 Linear . . . 5
  - 4.1.3 Leaky reLU . . . 5
  - 4.1.4 Tanh . . . 5
- 4.2 Loss Functions . . . 5

5 Bayes Classifier **5**
- 5.1 Definition of Prior . . . 6
- 5.2 Definition of Likelihood . . . 6
- 5.3 Data Transformation . . . 6
  - 5.3.1 Transformation - Quantile Buckets . . . 7
  - 5.3.2 Transformation - Mean Buckets . . . 7

6 Results **7**
- 6.1 Artificial Neural Networks . . . 7
- 6.2 Bayes Classifier . . . 9
  - 6.2.1 Effect of Bucket Count on Misclassification . . . 9
  - 6.2.2 Effect of Threshold on Misclassifications . . . 10
  - 6.2.3 Effect of Threshold on Percentage of FP and FN . . . 10
- 6.3 Effect of Outlier Detection . . . 10
  - 6.3.1 Bayesian Model . . . 10
  - 6.3.2 Artificial Neural Networks . . . 11

7 Discussion **12**
- 7.1 Effectiveness of Preprocessing . . . 12
- 7.2 ANN . . . 12
- 7.3 Bayesian - Data transformation: Quantile VS Mean Buckets . . . 12

8 Author Contributions **13**

Bibliography **14**

1 Abstract

Spam emails are not only an inconvenience for email users, but also pose security risks to personal information and take up email server memory space, network bandwidth, and CPU power and time [1]. The 2017 Kaspersky Lab Spam Report estimated that 56% of global email traffic is spam [2]. In this experiment, we applied machine learning methods to classify spam and not spam based on email content in the form of a set of 57 word frequencies from UCI's SPAMBASE dataset [3]. We explored two different approaches of classification: feed forward neural network and a Bayes classifier, while also using a couple different methods of pre-processing. Using these approaches a high spam classification accuracy rate was achieved, with the neural network achieving approximately 94% accuracy. While with the Bayes Classfier with mean buckets preprocessing, we achieved an average of 90% accuracy. The neural network learning method we used to classify spam emails was fairly accurate as well, achieving an accuracy of 94% for the best model.

2 Introduction

We all deal with spam emails constantly flooding our inboxes. Not only does spam take up the time of employees who could otherwise be engaged in more productive work, but it also puts a heavy processing load on the mail server and eats up disk space on server/client machines. In addition spam poses a security threat to e-mail users because attackers can use it to point users to malicious websites or scripts. Spam is notably claimed as one of the major challenges for the technology world. Email providers like Gmail, Yahoo, and Outlook implement machining learning strategies to identify spam with input from the sender's history, engagement metrics, and email content such as phrases, formatting, and external links [1]. These machine learning strategies must be very accurate when filtering out non spam and spam e-mails. Otherwise users will run into the issues of filtering out important messages or too many spam messages in your inbox.

This paper explores two approaches for classifying spam: a naïve Bayes model and a feed forward neural network (FFNN). Both approaches were trained and tested using the UCI SPAMBASE dataset [3]. Previous Bayes neural network models have achieved misclassification errors of 12% and 7% respectively, which we aim to improve upon by using pre-processing algorithms and other strategies to improve the accuracy of our models.

3 Pre-processing

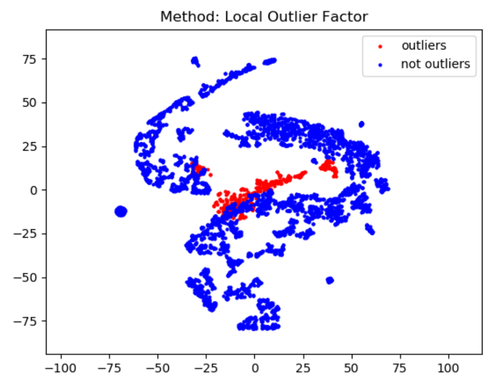
3.1 Local Outlier Factor

There are various forms of outlier detection, one of the more common algorithm being the Local Outlier Factor algorithm. This method, in a sense, compares the distance of each sample's data points with the surrounding data points to determine the density of neighbors for each sample. A low ratio of neighbors compared to the rest of the data set means indicates that it is outside of the ”normal” range of values, classifying it as an outlier.

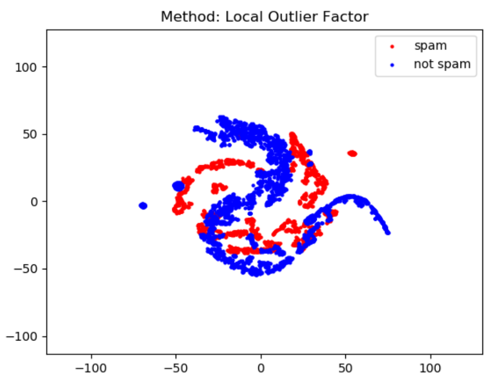
Initially when processing the algorithm, a threshold, k, is given where the k-distance is defined as the distance to the kth closest neighbor. Once the k-distance is determined, the reachability distance between two points, let's use arbitrary points a and b, can be determined. The reachability distance between a and b is the maximum of either the k-distance of point b, or the Euclidian distance of the two points. In other words, if point a is within the set of k closest neighboring points to b, the reachability distance is set to be the k-distance of b. Once the reachability distance is found, the local reachability density for each point can be determined. This is done by taking the inverse of the average reachability distance of the closest k neighbors of a point. This makes sense because the larger the distance to its closest neighbors, the less dense that location of points is. The inverse is taken since the larger the distances are, the less dense that location would be [4].

3.2 Isolation Forest

Another common algorithm for the detection of outliers is the Isolation Forest algorithm. The way that an outlier is determined is by creating trees through various decisions that will eventually isolate all data points.

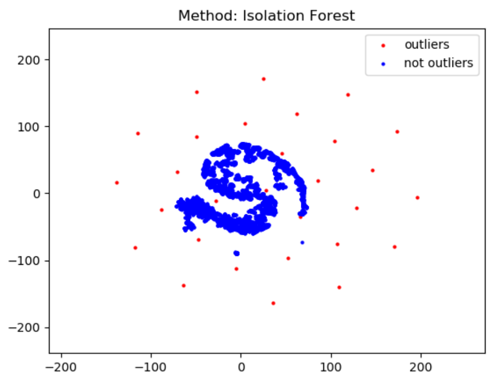


(a) Outliers and non-outliers found by Local Outlier Factor, represented by t-SNE feature reduction.

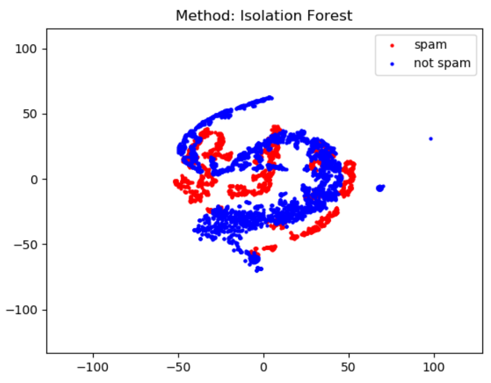


(b) Spam and non-spam emails without outliers found by Local Outlier Factor, represented by t-SNE feature reduction.

To create these trees, one would take a random feature of the data set, take a random value of that feature, and split the data into two sections using these random selections. This new sub-section is then split into a smaller sub-section and this process repeats until all of the values in the data set are isolated. Once one tree is created, this process can be repeated for any number of trees to get a more accurate representation of outliers. The outliers can typically be easily recognized since it would take a significantly less number of splits per tree to reach the isolation of that point. So the less deep the tree goes until it reaches an isolated point can determine whether or not a point is an outlier. This is due to the fact that outliers are typically not very close to other points when considering their values as coordinates, meaning that samples with anomalous feature values are more likely to stray away from the most dense locations and be easier to isolate. Isolation Forest algorithms are able to generate results relatively quickly and accurately for higher-dimensional data sets since it does not have to calculate distances between each point, such as the Local Outlier Factor method. This method is more applicable to real life situations as it is considered to handle large databases extremely well.[5]



(a) Outliers and non-outliers found by Isolation Forest, represented by t-SNE feature reduction.



(b) Spam and non-spam emails without outliers found by Isolation Forest, represented by t-SNE feature reduction.

4 Neural Network

One method to classify the spam data set is to construct and train an artificial neural network. We vary several parameters to optimize both accuracy and efficiency of the model. Since our data is not time dependent, we used a feed forward network in our implementation. To address differences in the ranges of input values, we applied min-max normalization to map input onto the [0,1] interval.

4.1 Activation Functions

Activation functions enable modeling of independent variables and class labels that have complex non-linear relationships. Any differentiable function can be used as an activation function, however, the choice of function affects the accuracy and computational efficiency of the model. In this study, we consider four activation functions: sigmoid, linear, leaky reLU, and tanh for building a binary classifier neural network. For each function, we perform a grid search testing the number of hidden layers (1, 2, or 3), the number of neurons per layer (3,6,10,15, 30, 50, or 100), and the loss function. The number of hidden layers and number of neurons describe the complexity of the non-linear relationships between input data and class, and are best optimized through trial and error.

4.1.1 Sigmoid

The sigmoid activation function maps interval $[-\infty, \infty]$ onto [0, 1], and reports the probability of belonging to a class. The maximum value of the final output layer designates the class. The sigmoid function may result in the vanishing gradient problem, in which large magnitudes of input yield almost no change in the prediction and the network ceases to learn [6]. We use a reduced learning rate and the aforementioned normalization to avoid the vanishing gradient problem.

4.1.2 Linear

Linear activation functions simply apply a weight to the sum of input into a node. While models with linear hidden layers would typically included a sigmoid output activation function to perform binary classification on the range of 0 to 1. We experimented by using a linear output to see if it was possible and how much it reduces accuracy compared to other activation functions.

4.1.3 Leaky reLU

The reLU function sets all negative values to 0, and maintains all positive values. Leaky reLU overcomes non-differentiability at and below zero and resulting 'reLU dying' by setting negative values to have a small (0.01) slope.

4.1.4 Tanh

Tanh is a similar function to sigmoid but with a steeper gradient that results in a greater update response to small changes. Tanh is also zero-centered, lying on the interval [-1,1], and therefore allows for more efficient back-propagation than sigmoid. Like sigmoid, tanh is also susceptible to the vanishing gradient problem as well as sometimes causing dead neurons [7]. Like sigmoid, these problems were mitigated somewhat by using normalization and carefully choosing the learning rate.

4.2 Loss Functions

In training a neural network, we apply standard gradient descent to iteratively update weights with the objective of minimizing a loss function. We compare two loss functions: mean squared error and binary cross-entropy loss. Mean squared error considers real-valued error between predicted and actual class label, while cross-entropy loss considers the divergence in predicted probability [5].

5 Bayes Classifier

As a point of comparison to see how effective the neural network is at classifying spam emails, we created a Bayes classifier. Following the discussion of Bayes classification in lecture, our section of the project became the implementation of the code necessary to calculate the three probabilities necessary for calculating the posterior of the Bayes rule. That is we needed to calculate:

$$P(Class|X) = P(X|Class) * P(Class)P(X)$$

Following the trick discussed in class of taking the ratio of classification to see if an item is spam lets us remove the evidence from the equation, leaving us with:

$$\frac{P(Class|X)}{P(-Class|X)} = \frac{P(X|Class)P(Class)}{P(X|-Class)(1-P(Class))}$$

Our job is simply to look at the ratio and see if it was greater than or equal to one. This leaves two things to code calculations for: the prior, and the likelihood.

5.1 Definition of Prior

The prior is the simpler calculation for the Bayes classifier. Following the definition discussed in lecture, the prior is simply:

$$P(Class) = \frac{\#ClassSamples}{\#TotalSamples}$$

Since our data is roughly 40% spam, for nearly all divisions of training and testing data, the prior will stay the same at nearly 40%.

5.2 Definition of Likelihood

We followed the definition discussed by Michael Collins' of the naive likelihood of a sample being the product of the probability of having the given value for each of its features for a particular class[8]. That is:

$$P(X|Class) = P(x_1|Class) * P(x_2|Class) * P(x_3|Class)... * P(x_{57}|Class)$$

For calculating each individual probability ($P(x1|Class)$), we first looked at explorations of performing Bayes classification on continuous data. We followed the suggestions outlined by Carlos Guestrin, and assumed the Gaussian Distribution and using that to calculate the likelihood of our continuous valued samples[9]. However, this proved to not be as accurate as hoped (20% misclassification achieved by assuming a Gaussian distribution), so we decided to focus on calculating likelihood using a discrete variable.

Following Michael Collins' same discussion, we then defined the likelihood of assuming a certain value for each feature as being the count of the number of occurrences of that value given its class[8]. That is:

$$P(x_j = v|Class) = \frac{\#Classwherex_j=v}{\#TotalClass}$$

With the definitions of likelihood and prior outlined, we were then able to begin building our classifier.

5.3 Data Transformation

Since we defined our Bayes Network as looking at the likelihood for a certain discrete variable taking a given value, we decided that some transformation had to be done to the data. This UCI data set supplied values on continuous distributions, with many features being a percentage of total words between 0 and 100, and others being counts of a certain feature in the document. Through min-max normalization, these values were all put on a scale between 0 and 1, but their uniqueness remains. In order for the Bayes classifier to get meaningful likelihood values, we needed to come up with a way to take the continuous data and put it to discrete values that can then be counted.

We will outline a couple of approaches that we took to transform our data. However we eventually settled on an approach of bucketing values based on the mean of the feature for each class.

5.3.1 Transformation - Quantile Buckets

The first transformation that was proposed for turning our continuous data into discrete values was by simply splitting the values for each feature into n-buckets. The idea was as follows:

1. Select a column j, and sort data set based on that column
2. Split the data into n equal sized buckets in sorted order (quantiles)
3. For each sample in the bucket label column j of that sample with the bucket number
4. Repeat for each column in the data (except the spam label).

Using this method, our training and testing data were bucketed separately as to avoid making assumptions on the testing data using the training data.

By bucketing, our data into discrete values, we would be able to easily calculate the likelihood for each sample, as now each feature could only take on one of n values. Adding varying amounts of buckets will allow for varying amounts of precision when discerning values for each class, allowing clearer divisions about where ranges for spam values begin and end.

5.3.2 Transformation - Mean Buckets

Our second method for preprocessing the data is similar to our first in that we are splitting our data into two buckets. However the method for calculating these buckets is significantly different.

Whereas in the previous method we just sorted the column and labeled quantiles, in this method we calculate the mean for each column for spam and not spam samples separately, labeling each value as 1 for being closer to the spam average or 0 for the not spam average. The process is as follows:

1. Select a column j, and calculate the mean of that column for spam and not spam separately
2. For each sample i, label the value in the column:
   (a) 1 if $X[i][j] - spam\ mean < X[i][j] - not\ spam\ mean$
   (b) 0 otherwise
3. Repeat for each column

Using this method, our means were calculated using only the training data as to avoid making assumptions about testing data when training our model.

6 Results

6.1 Artificial Neural Networks

The best models from the grid search of model hyperparameters are those with the lowest misclassification rate, or (1 - accuracy), listed below in Tables 1 and 2. To assess the generalizability of each model, we split randomly shuffled data into 10 groups to perform k-fold cross-validation. K = 10 is chosen to align with the common practice of incorporating 90% of data in training. Plots of the generalized error along with ROC and PR curves for the three best models can be seen in Figs. 3 to 5. Initially, we trained each model with 1000 epochs, tracking the generalized training and testing misclassification rates per epoch. An increase in testing misclassification rate indicates overfitting. We identified the number of epochs after which testing misclassification rate begins to increase, and opted to reduce the number of epochs to avoid overfitting and leading to a better performing model.

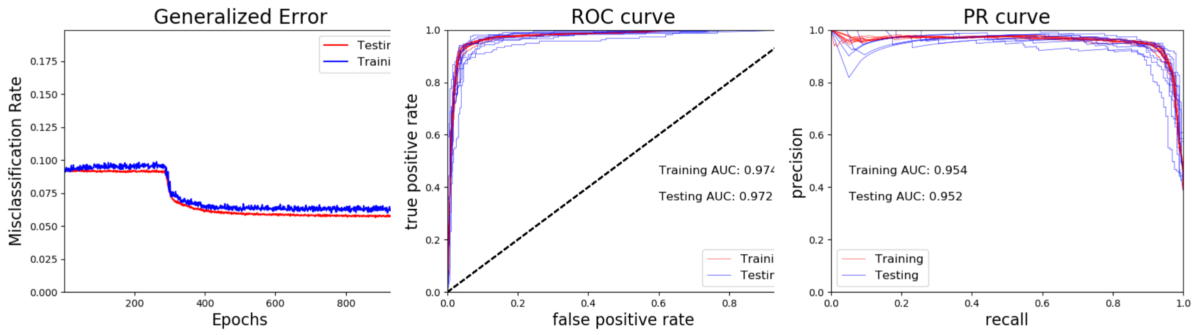


Figure 3: Evaluation plots for optimal sigmoid model using MSE

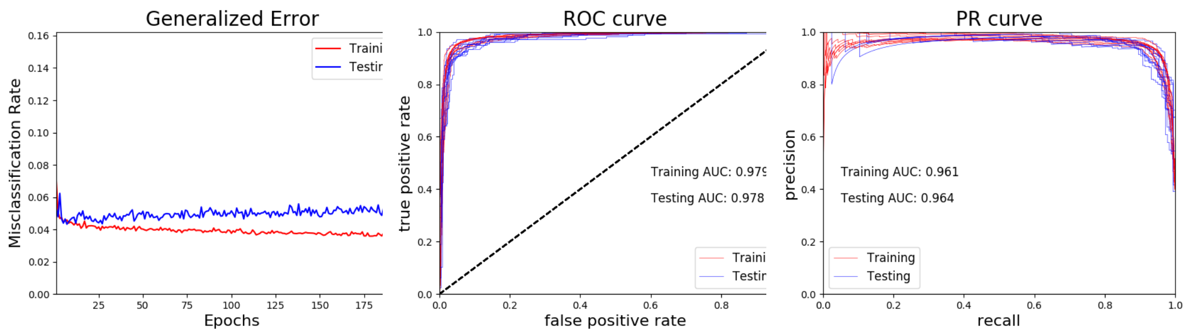


Figure 4: Evaluation plots for optimal tanh model using MSE

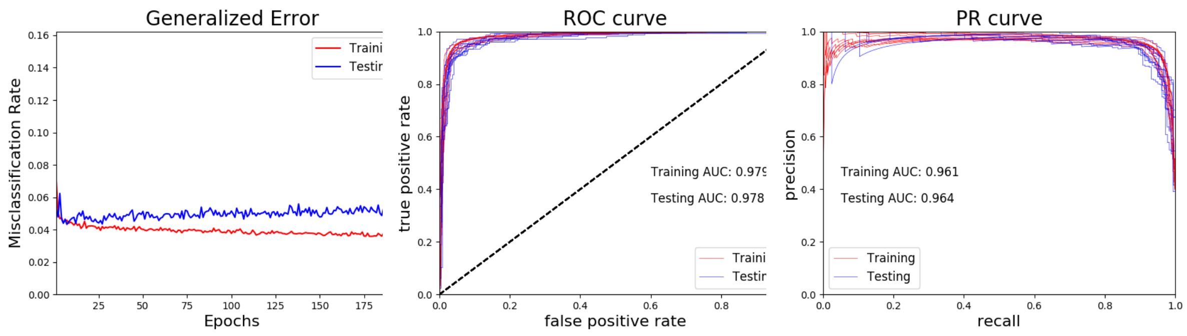


Figure 5: Evaluation plots for optimal leaky ReLU model using MSE

Activation Function (hidden, output)	Optimal Parameters	Generalized Training Misclassification Rate	Generalized Testing Misclassification Rate
Sigmoid, Sigmoid	3 layers, 30 nodes	6.17%	6.83%
Tanh, Sigmoid	3 layers, 10 nodes	4.98%	6.22%
Linear, Linear	1 layer, 50 nodes	11.87%	11.33%
Leaky ReLU, Sigmoid	2 layers, 10 nodes	4.40%	5.19%

Table 1: Best models and their generalized training and testing misclassification errors found using grid search with mean squared error as the loss function.

Activation Function (hidden, output)	Optimal Parameters	Generalized Training Misclassification Rate	Generalized Testing Misclassification Rate
Sigmoid, Sigmoid	2 layers, 10 nodes	6.70%	7.04%
Tanh, Sigmoid	2 layers, 30 nodes	5.08%	6.39%
Linear, Linear	1 layer, 50 nodes	29.53%	29.15%
Leaky ReLU, Sigmoid	2 layers, 50 nodes	4.86%	5.61%

Table 2: Best models and their generalized training and testing misclassification errors found using grid search with binary cross entropy as the loss function.

6.2 Bayes Classifier

Below are the average training and testing misclassification rates as well as the min testing misclassification for some of the different configurations of our Bayes classifier over 10-fold cross validation with a threshold of 1. We tested no data transformation by assuming both a Gaussian continuous distribution for likelihood, and discrete counting of values. All bucketing tests were done using the discrete method for calculating likelihood.

Misclassification rates	No Bucketing (Gaussian)	No Bucketing (Using Discrete)	Mean Buckets	2-Buckets	5-Buckets	20-Buckets
Training Average	22.31%	0.582%	9.17%	18.80%	11.33%	10.21%
Testing Average	22.71%	31.45%	9.35%	19.37%	11.98%	11.17%
Testing Min	14.78%	26.74%	7.61%	15.43%	7.61%	8.26%

6.2.1 Effect of Bucket Count on Misclassification

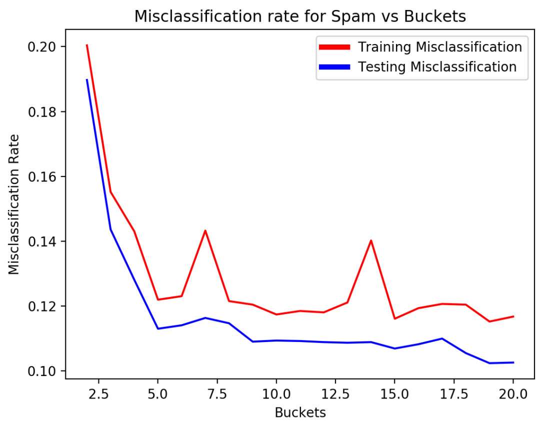


Figure 6: Change in average misclassification over 10-fold cross validation based on the number of buckets used for n-quantile bucketing. Quickly plateaus around 5-10 buckets. Never reaches the misclassification provided by meaned-buckets.

6.2.2 Effect of Threshold on Misclassifications

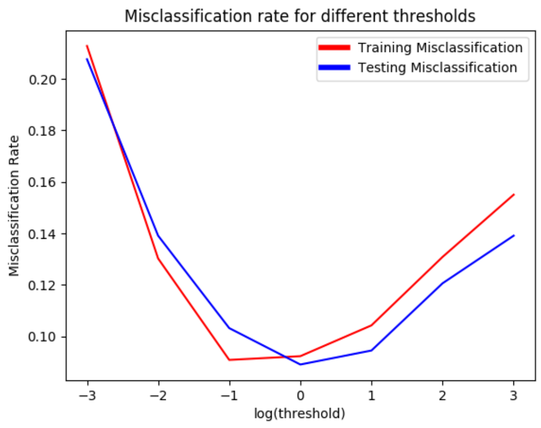


Figure 7: We tested several different thresholds for which our Bayes Classifier will classify an email as spam or otherwise. As seen by the overall misclassification rate for different thresholds, minimizing the rate of false positives will sacrifice some accuracy. We achieve a minimum misclassification rate at a log threshold of around 0.

6.2.3 Effect of Threshold on Percentage of FP and FN

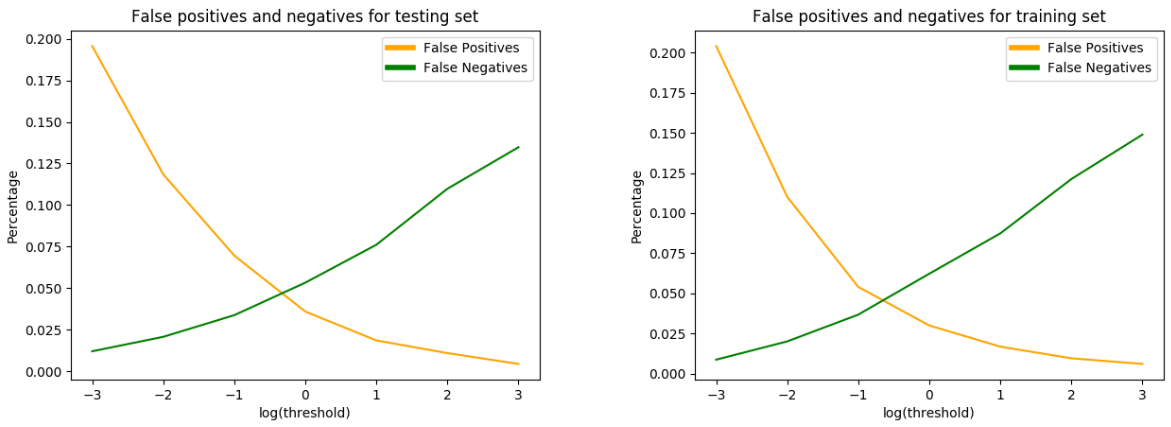


Figure 8: As can be seen, FP and FN reach a sort of equilibrium at a log threshold of 0 in the testing set, which corresponds to a minimum misclassification at that same point.

6.3 Effect of Outlier Detection

6.3.1 Bayesian Model

In order to measure the effect of outlier detection on the Bayesian model, we ran the tests multiple times. We compared the average differences between the models trained on the complete data set versus the models trained on the sets stripped of outliers. Some of the individual results struck us as unusual. In our second run, using zero buckets, the model produced an accuracy of 91% using the Local Outlier Factor data set when we had come to expect around 70% for this field. This result only occurred in Mean Buckets and it only once in the 12 runs we conducted, however the average case didn't show much improvement.

The table below displays the averaged model accuracy differences. We recorded the average across all runs for the three data sets, and with these averages we calculated the difference to compare the model accuracy.

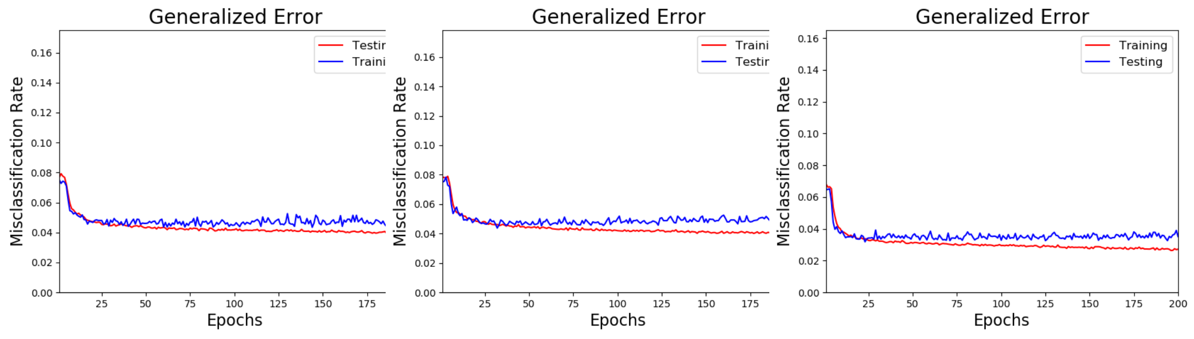
	No Bucketing	Mean Buckets	2-Buckets	5-Buckets	20-Buckets	Sum
Isolation Forest	-0.23140%	0.03297%	-0.11734%	-0.06744%	-0.06394%	-0.44718%
Local Outlier Factor	-0.68217%	0.467608%	0.69373%	-0.17535%	-0.22991%	0.07390%

The Isolation Forest data set performed worse on average and although the Local Outlier Factor data set performed better on average, an increased accuracy of 0.07390% is too slim to deem significant. We would need to run more than just 12 tests for a more accurate representation of which data set produces models of higher quality. However, the margin of improvement is so slim that pruning the data is difficult to justify through this method of classification.

6.3.2 Artificial Neural Networks

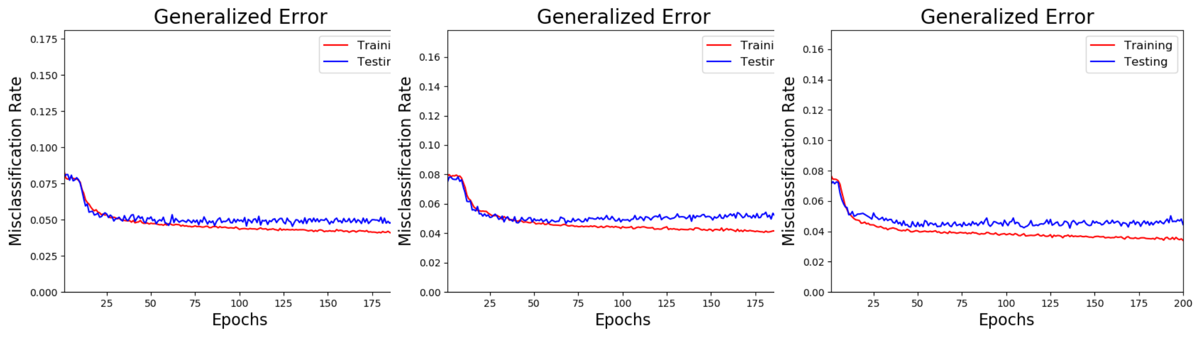
Unlike with our Bayesian model results, the Local Outlier Factor results proved substantial when training ANN models. On the other hand, the Isolation Forest outlier pruning only served to decrease model accuracy. To demonstrate this, we showcase the error results from training our best ReLU models in Figs. 9 and 10, trained on the three data sets.

Specifically, with MSE as our loss function, from the original data set to the LOF data set, the error dropped from 4.79% to 3.58%, an improvement of 34%. With binary cross entropy as our loss function, the improvement was less dramatic, but still substantial at 10%, going from an error rate of 5.14% to 4.69%.



(a) Complete Data Set (b) Isolation Forest Data Set (c) Local Outlier Factor Data Set

Figure 9: Errors using Mean Square Error, Leaky ReLU, Sigmoid, 2 layers, and 50 nodes



(a) Complete Data Set (b) Isolation Forest Data Set (c) Local Outlier Factor Data Set

Figure 10: Errors using Binary Cross Entropy, Leaky ReLU, Sigmoid, 2 layers, and 50 nodes

7 Discussion

7.1 Effectiveness of Preprocessing

Isolation Forest outlier detection proved ineffective with this spam data set. Not enough outliers were able to be detected accurately and there was no significant overall impact. The Bayesian model with outliers removed through the isolation forest algorithm was not useful, and the results with the ANN model were insignificantly improved. Local Outlier Factor, however, identified four hundred outliers, nearly ten percent of the data set. The accuracy of the ANN with outliers removed through the Local Outlier Factor algorithm was greatly increased.

Isolation Forest algorithms are usually considered optimal for larger data sets, where this email set contained less than 5,000 samples which is relatively small. This could explain the lack of significance that the algorithm had in decreasing the error. Although isolation forest algorithms are quicker, local outlier factor seems to be a much better fit for our purposes. Since it performs a greater number of calculations rather than random splits, as isolation forest does, it more accurately determined the outliers and therefore was able to produce a significant increase in accuracy.

7.2 ANN

Models using a linear activation function performed the worst, with the highest generalized misclassification rate. A previous study reported a misclassification rate of 7% with the constraint of zero false positives [10]. Sigmoid, Tanh, and leaky reLU functions achieved a generalized misclassification rate below 7% with Mean Squared Error loss. In selecting the optimal model of these three, we compared training efficiency, Precision-Recall (PR) curves, and Receiver Operating Characteristic (ROC) curves, shown in Figs. 3 to 5. The number of epochs required for the model to reach minimum error was lowest for tanh, requiring 50 epochs. The model with the highest and closest to ideal AUCPR score was the tanh model with a score of 0.964. The highest AUC score was 0.979 for the leaky reLU model. Though the results of these three models are comparable, we selected the leaky reLU model as our optimal model on the basis that it achieved the lowest generalized testing misclassification rate of 5.19% and the best AUC score.

7.3 Bayesian - Data transformation: Quantile VS Mean Buckets

Looking at the averages and min misclassification values on the testing set, the mean bucketing method provided the strongest model for predicting spam, and is the model of choice were we to deploy a Bayes classifier. The mean bucketing method was able to achieve higher accuracy because it responds to the real distribution of the dataset. By computing means for classification, it can transform continuous values into binary values with respect to its actual distribution.

There was one interesting case, when we classified the training set using no data transformation and discrete counting. The misclassification rate was nearly 0% for the training data in that case due to the uniqueness of each value. The uniqueness of each value lead to extremely low misclassification on training data where each value was associated with a label, but extremely high misclassification rate on the testing set since many of the values in the testing set were not encountered using the training set.

Our Bayes model seems to outperform other similar models trained on the same data set. Previous research done by Rusland, Wahid, Kasim, and Hafit in 2017 resulted in a Bayes classifier using the same spambase data set that achieved an average accuracy rate of about 82.5% and high of 88%[11]. Our implementation of mean buckets was able to achieve a testing average of 90.6% and a high of 92.4% (we get accuracy by doing 1 - misclassification rate). This shows that performing a continuous to discrete transformation on the data set provides a significant boost to performance over using continuous data.

8 Author Contributions

Preprocessing

Ruvim Lashchuk

Outlier pruning, training Bayes without outliers, Outlier/Not Outlier t-SNE visualization

Randy Murphy

Outlier pruning, training ANN without outliers, Spam/Not Spam t-SNE visualization

ANN

Nathan Hoffman

Grid search code, Training, Report Writing, Slides

Hannah Brown

Grid Search/K-Fold code, Training, Report Writing, Slides

Joyce Christiansen-Salameh

Evaluation code, Training, Report Writing, Slides

Bayes

Elisabeth Henkens

N-Quantile Bucketing code/results, creation of general Bayes model, Bayes Discussion

Cathy Hsieh

Bayes Discussion and Slides

Nico Lupacchino

Bayes Data Collection and Reporting

Ethan Yi Chiang

Mean Buckets code/results

Michael Nguyen

Slides and Reporting

Jordan Palmer

Testing of Alternative Bayes models, Bayes Classifier training, Bayes Classifier Code, Slides

Bibliography

[1] E. G. D. et. al., "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, 06 2019. [Online]. Available: https://doi.org/10.1016/j.heliyon.2019.e01802

[2] D. Gudkova, "Spam and phishing in 2017." [Online]. Available: http://securelist.com/spam-and-phishing-in-2017/83833/

[3] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[4] M. M. B. et. al. (2000) Lof: Identifying density-based local outliers. [Online]. Available: https://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf

[5] F. T. L. et. al. (2008) Isolation forest. [Online]. Available: https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf

[6] R. Pascu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *JMLR: WCP*, vol. 28, 2013. [Online]. Available: http://proceedings.mlr.press/v28/pascanu13.pdf

[7] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018.

[8] M. Collins, "The naive bayes model, maximum-likelihood estimation, and the em algorithm." [Online]. Available: http://www.cs.columbia.edu/~mcollins/em.pdf

[9] C. Guestrin, "Naïve bayes (continued) naïve bayes with continuous (variables) logistic regression." [Online]. Available: http://www.cs.cmu.edu/~guestrin/Class/10701-S06/Slides/naivebayes-logisticregression.pdf

[10] U. Mahdiyah, M. I. Irawan, and E. M. Imah, "Integrating data selection and extreme learning machine for imbalanced data," *Procedia Computer Science*, vol. 59, 2015. [Online]. Available: https://doi.org/10.1016/j.procs.2015.07.561

[11] N. F. Rusland, N. Wahid, S. Kasim, and H. Hafit, "Analysis of naïve bayes algorithm for email spam filtering across multiple datasets," *IOP Conference Series: Materials Science and Engineering, 226, 12091*, 2017. [Online]. Available: https://doi.org/10.1088/1757-899x/226/1/012091