SMS/Ham Classification for SYDE 522

Hayley Smith, Joash Colaco, Lama Khadra, Sarah Salem
Department of Systems Design Engineering
hj2smith@edu.uwaterloo.ca, j2colaco@edu.uwaterloo.ca, lakhadra@edu.uwaterloo.ca, s4salem@edu.uwaterloo.ca

Abstract *– SMS spam classification and detection is an important problem that is difficult yet critical for businesses. By using labelled data, the problem becomes easier to classify any new dataset of SMS as either ham or spam. Different classifiers were used in this paper to predict the category including: Decision Trees, Naïve Bayes, Support Vector Machine, k-Nearest Neighbor and Neural Networks. Exploration has helped find that neural networks had the highest accuracy and precision scores for classification. Runtime is also important when evaluating different classification methods and neural networks was also found to be the fastest method in comparison to the rest.*

Keywords: Spam Classification, Decision Trees, Naïve Bayes, SVM, k-NN and Neural Networks

1 Introduction

Spam is defined as irrelevant or inappropriate messages sent on the Internet to many recipients. Spam is sent in the form of text messages, emails, and other form of messages. It is used by marketers in the form of unwanted advertisement or by fraudsters to lure users or by attempting to direct people to phishing pages [1].

Spam emails contribute to 40% of all emails which is about 15.4 billion emails per day! This is a huge cost of $355 million for internet users per year [2]. Consequently, spam is a significant issue that costs users time, space, and money. Machine learning has been used to combat this issue. Classification algorithms are used to learn classification rules from a set of training data. These algorithms include Neural Networks, K-Nearest Neighbors (k-NN), Decision trees and Support Vector Machine (SVM). The explanation, implementation and results of using the mentioned algorithm will be discussed in this paper.

2 Background Review

Since 2004, various models have been used to automate the classification of SMS spam messages. As part of a spam detection study by Almeida, Gómez Hidalgo, and Yamakami in 2011, 13 different models were tested on a corpus [1]. The outcome of the tests determined that SVM with alphanumeric tokenization had a better performance with an accuracy and false positive rate of 97.64% and 0.18%. Decision trees was a close second with an accuracy of 97.5% [3].

Furthermore, different variation of algorithms were tested for SMS spam detection. For example, in 2010, Jie, Bei and Wenjing added a cost function to Naïve Bayes that penalized classification of false positives [1]. This technique focused on increasing the precision metric for detecting spam.

The rise of artificial intelligence and technological resources have enabled neural networks to become an appropriate model for spam detection among companies. Google has improved its spam detection algorithm from 99% to 99.9% accuracy through the use of neural networks for its Gmail application in 2017 [4]. The model also has a false positive rate of 0.05% [4]. Google has credited neural networks for its ability to recognize junk e-mails and phishing emails in comparison to other algorithms.

Some of the model constraints include the time it takes to classify SMS. The average time to classify messages ranges from a fraction of a second to 2-4 seconds as shown in a study done by Junaid and Farooq in 2011 [4]. Time is an important feature for the models as users are dissatisfied with a service if messages are not being delivered promptly. Thus, algorithms like k-NN may not feasible depending on the dimensions of the dataset.

The efforts done on the same dataset performed similar comparisons between the different classifiers for example Logistic Regressions showed a 94% accuracy score. Gaussian Naive Bayes showed a score of 88%. MLP Classification showed a score of 98%, Decision Trees showed a score of 97%, SVM showed a score of 94% and Neural Networks showed a score of 98%. These efforts however did not show the precision score of all of the classification models as well as any model improvements that were performed. For this reason, it would be interesting to look into model improvements for all the classification methods that will be discussed and explore the different parameters that can be altered in each of the classification methods [5].

3 Data

3.1 Data Inputs & Cleaning

The dataset consists 5572 rows of SMS messages containing 4825 and 747 ham and spam messages. 13.4% of the dataset are spam messages. The classification labels for

ham and spam were encoded to 0 and 1, respectively. For each SMS messages, several data cleaning steps were implemented. Firstly, all non-alphabetical characters were removed from the messages. Stop words that do not add any additional meaning to the SMS such as "the" or "of" were removed. Furthermore, each word in each message was stemmed to its root word reducing the count of unique words in the dataset from 7107 to 5834. Any word that has less than 2 characters in the dataset was also removed. After cleaning the data, 27 messages were eliminated from the dataset as they consisted of zero characters (blank messages) reducing the number of unique words to 5824. They were deleted as it accounts for less than 0.5% of data.

3.2 Feature Extraction

In order to classify messages as spam or ham, features must be extracted from the messages. Using the entire dataset, the messages were transformed to a bag-of-words representation. The dimensions of the messages in this representation is 5545 by 5824. The bag-of-words was then transformed into a term frequency–inverse document frequency (tf-idf) representation as it normalizes the words between 0 and 1 depending on their importance in each specific message. The tf-idf was used as the inputs for the following models; k-NN, Decision Trees, Support Vector Machine, Naïve Bayes and Neural Networks.

4 Model Results and Discussion

4.1 K-Nearest Neighbors

4.1.1 Model Implementation

K-Nearest Neighbors is a supervised learning algorithm that predicts the class of a new data point by using its k nearest data points found via Euclidean distance. The algorithm was implemented with a k value of one using all 5824 features from the tf-idf for various k-values with 10-fold cross-validation. This generated a validation accuracy and precision value of 96.8% and 94.6% respectively.

4.1.2 Model Improvement

The first step for model improvement was to determine the optimal value of k. For larger k-values, the algorithm will likely categorize all messages as ham due to the abundance of messages classified as ham in the dataset. Thus, it was hypothesized that smaller k-values will result in a higher model accuracy. The results displayed in the Figure 1 below display the average validation accuracy and precision across the ten validation sets.

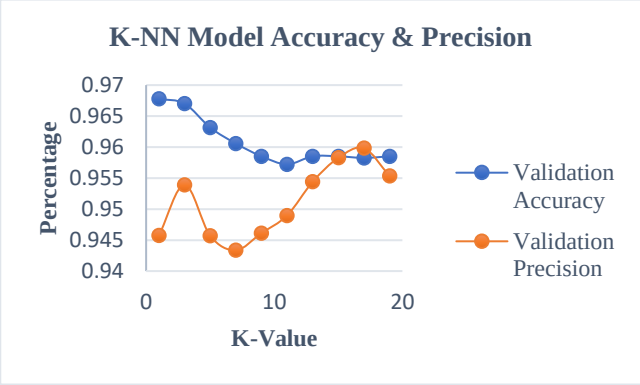

Figure 1: k-NN Model Accuracy and Precision

The highest model accuracy of 96.8% with a precision value of 94.6% for a k-value of 1. As the value of k increases, the model accuracy decreases which confirms the hypothesis mentioned above. Furthermore, it is interesting to note that as the value of k increases, the model precision tends to increase. This is because the count of false positives (predicted as spam but actually ham) reduces as k-value increases, resulting in higher precision but lower model accuracy.

Next step was to check if a selected number of maximum features from the tf-idf can improve the model instead of using all 5824 features for each input. The k-NN model was implemented using several maximum feature values shown in Figure 2 below and a k-value of one. The highest average model validation accuracy and precision combination was 94.7% and 99.4% for a maximum feature value of 4000. Although the precision value has significantly increased, the average ratio of false positives to true positives for 4000 maximum features is only 43:1. Thus, all 5824 features will be used as it has a higher model accuracy.

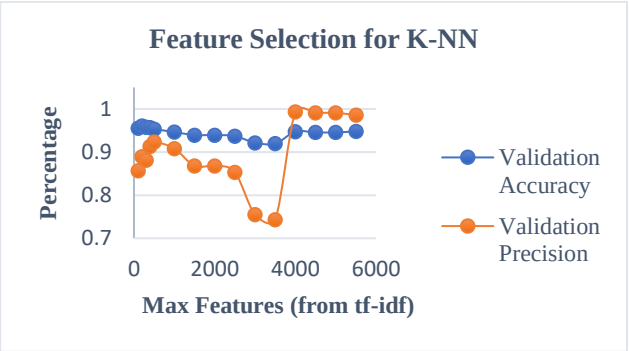

Figure 2: Feature Selection for k-NN

4.1.3 Model Results and Evaluation

A k-value of 1 and feature value of 5824 was selected as parameters of the k-NN algorithm. The dataset

was split into 70% train set and 30% testing set. With these parameters, the model should have a lower precision value as seen in Figure 1. However, focusing on a higher accuracy brings about a better model as the average due to the ratio of true positive to false positive (section 4.1.2). The algorithm was executed on the test dataset and generated an accuracy and precision of 96.8% and 95.9%. Across 10 random test sets, the accuracy and precision ranges from approximately 95-96% and 94-96%. This is consistent with the validation model results (section 4.1.2). This simple algorithm generated high model metrics, however, more sophisticated methods such as Naïve Bayes or SVM may get a better classification performance in less time. k-NN took 1.2 minutes to classify 1664 SMS messages.

4.2 Decision Trees

4.2.1 Model Implementation

Decision trees are a supervised machine learning classification algorithm that builds a classification model in the form of a tree structure. The result is a tree with decision nodes and leaf nodes. The decision nodes have two or more branches. The leaf node represents the final classification or decision of the input. In the case of a ham or spam problem, the leaf nodes will indicate the final classification of the input text as either spam or ham. The algorithm is trained using a set of training data and a set of rules are built as a result of the training. The rules are then used to classify new input data as either spam or ham.

4.2.2 Model Improvement

To improve the model, two different metrics were compared with the decision tree. These two metrics - Gini and entropy - are for choosing how to split a tree. Gini index is a criterion to minimize the probability of misclassification where entropy is a way to measure impurity. With the Gini metric, the model showed an accuracy of 96.4% and a precision of 90.5% when using a 30% testing set. In comparison with the Gini metric, the entropy metric showed an accuracy of 96% and a precision of 91.8% with a 30% testing set. Hence, by using the entropy metric, the model's accuracy stayed relatively the same while the precision increased by 1%. The difference is very negligible and that is why in most cases, the Gini and entropy metrics are considered equally effective for a decision tree classification problem.

4.2.3 Model Results and Evaluation

The model was evaluated by using a training and testing dataset. The datasets were initially created by using 70% of the original data for the training dataset and 30% for the testing dataset. The accuracy and precision of the Gini decision tree model were 96.4% and 90.5% respectively. The accuracy and precision of the entropy decision tree model were 96% and 91.8% respectively. The models' validation was improved using k-fold cross validation where k=10. The accuracy and precision from the k-fold cross validation were 96.3% and 85.9% for the Gini model. The accuracy and precision from the k-fold cross validation were 96% and 87.9% for the entropy model. In comparison to other models, the precision of the decision tree classification was lower than k-NN.

4.3 SVM

4.3.1 Model Implementation

Support Vector Machine (SVM), falls under the supervised learning group of machine learning. It is considered to be a combination between linear machine learning and kernel function. The procedure that SVM follows is it divides different classes of variables by creating a hyperplane. The goal of SVM is to maximize the separation between the hyperplane and the points that are closets to it by placing weights on each of the feature vectors. The quadratic optimization problem that is aimed to be solved by SVM can be outlined in equation 1 below.

$$max_{\alpha\in\mathbb{R},w\in\mathbb{R}^d,\xi_i\geq 0}\gamma - C\sum_{i=1}^{n}\xi_i \;\; s.t. \;\; \forall_i: y_i(w^T x_i + b) \geq \|w\|\gamma - \xi_i \qquad (1)$$

SVM is advantageous in that it does not consider the relative size of each class when classifying, as it only aims to maximize the margin between the classes in the higher dimension and does not aim to minimize the error rate at any point. In addition, SVM is considered to be a favorable and promising classification technique as it is known to be able to work with big feature spaces.

SVM was implemented on the data and the variables that needed to be considered are the C and Gamma values and the kernel type. The C value decides how much it is desired for the margin to correctly separate as many instances as possible. The higher the c value the more separation between the instances and the lower the c value the less classification of different training examples. The Gamma defines how much influence a single training example has on the overall model. Different kernel types were used for the comparison of the model such as linear, rbf and poly.

4.3.2 Model Improvement

10-fold cross validation was performed on the dataset to assess the predictive performance. The results displayed an accuracy and precision of 97% and 96%, respectively. The poly SVM model had an accuracy of 94% and precision of 98%. Finally, the accuracy and precision using the rbf kernel type and gamma value=0.1 were 91% and 99 %, respectively.

4.3.3 Model Results and Evaluation

The model was run to train the data by splitting it up into training and testing data. The dataset was created initially by

splitting it in 70% training of the original data and 30% testing of the original data. The accuracy and precision of the linear kernel of the SVM model were 97.9% and 97.5%, respectively. The accuracy and precision of the poly kernel of the SVM model were 95% and 99% respectively. Finally, the precision and accuracy of using the rbf kernel type and gamma value=0.1 were 96% and 99% respectively. Upon comparing the accuracy and precision results with the 10-fold cross validation and without it seems that the accuracy scores are slightly lower across the different kernel types. However, the precision numbers seem to remain constant [6].

4.4 Naïve Bayes

4.4.1 Model Implementation

Naïve Bayes as a machine learning algorithm uses probabilities to predict how likely something is to fall into all the possible classes. These probabilities are calculated using the Naïve Bayes formula:

$$P(A|B) = P(A)P(B|A)/P(B) \quad (2)$$

Of the probabilities calculated, the class associated with the largest probability is where the input is classified. These probabilities are found when the training data is used. The probability of a given SMS message being spam or ham is found by looking at the ratio of total spam to the total number of SMS messages and total ham to the total number of SMS messages. Additionally, words and their frequency are saved in dictionaries based on their classification in the training. For example, if the word "hello" is come across in training in an SMS classified a spam then it would be saved the "spam" dictionary and the word would either be added to this dictionary with a count of 1 or, if it is already in there then the count would be incremented by 1. Once the training is complete, an SMS's classification is calculated by calculating the probability it is spam and calculating the probability it is ham and choosing the higher probability and associated class. The spam probability for example, is calculated by multiplying the probability a given message is spam by the probabilities of the text being spam given each word present in the SMS (using the spam word frequency dictionary discussed earlier and dividing that value by the total number of "spam" words). The code used to classify the data was modified from the code provided in *How To Build a Simple Spam-Detecting Machine Learning Classifier* [7].

4.4.2 Model Improvement

A very basic implementation involves training the data so that each word has a frequency which is then used to calculate a probability associated with it. However, it is unlikely that every word in the test data will have shown up in the training data. Those words, therefore will not have any frequencies, and therefore no associated probabilities. In order to correct this, Laplace Smoothing was added so that words that had not been seen in the training data did not break the model or lead to inaccurate probabilities and therefore, misclassification. This was done by adding a smoothing factor with alpha equal to 1.0. The general principles outlined previously remain the same, however the example of calculating spam given above is changed by adding alpha to each word's frequency and dividing by the total number of "spam" words plus alpha multiplied by the number of total unique words. This is necessary because otherwise we would end up with probabilities of zero for previously unseen words and having a probability of zero means something that has no possibility of occurring has occurred, which does not logically make sense.

4.4.3 Model Results and Evaluation

When the data is randomly split into a 30% testing set and a 70% training set, an average accuracy of 97.2% and a precision of 99.1% was achieved from the Naïve Bayes model. However, when the model was enhanced to use k-fold validation, where k=10, the precision increased to 100% consistently, while the accuracy fell to about 86.2%. This could be due to how the k-fold built in library is selecting the data, since there are significantly less spam SMS messages than ham SMS messages.

Overall, precision is one of the most important metrics on measuring an algorithm's effectiveness on this problem. Since this model has a precision between 97% and 100% (depending on if k-fold cross validation is used or not) it is highly effective at not misclassifying ham as spam. This is the most important because this is where the highest potential for damage to the user can occur; user's ham SMS messages being misclassified as spam which could result in them missing something important at the worst case and is generally annoying enough for users to stop using the spam/ham classification at best case. Therefore, overall this model has quite high accuracy and very reliable, and so for data such as the data set used here, it is very appropriate.

4.5 Neural Networks

4.5.1 Model Implementation

A neural network is a computing system of interconnected layers representing the human brain. The initial implementation of the neural network on the tf-idf matrix was two hidden layers with 32 and 64 neurons in each layer. Dropouts of value 0.2 were add for each hidden layer. The average validation accuracy and precision values after 100 epochs across 10-fold cross validation sets were 97.2% and 95.6%.

4.5.2 Model Improvement

There are several parameters that need to be optimized to improve the performance of the neural network such as number of maximum features from tf-idf, epochs, hidden layers, neurons within each hidden layer and dropout value.

Holding everything else constant, several maximum feature values of the tfidf matrix were tested as inputs for neural networks. The validation metrics peaked for a maximum feature value of 1500 with accuracy and precision values of 97.4% (Figure 3). It is important to note that neural networks consistently performed better with more features. For a low number of features, the validation accuracy of the model drops below 96%. For high maximum number of features, the validation accuracy ranges from 96.5-97.5%.

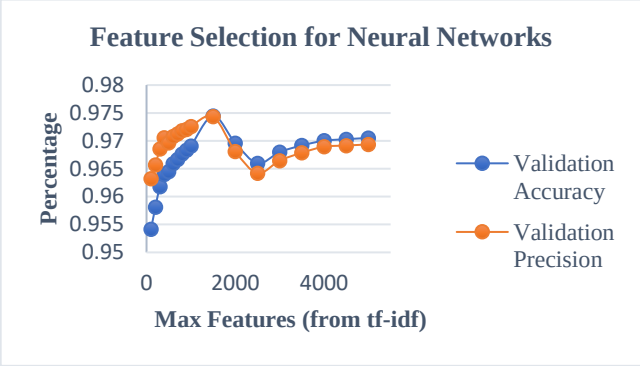

Figure 3: Feature Selection for Neural Networks

Furthermore, several neural networks with three or more hidden layers were implemented. It was concluded that the higher the complexity of the network architecture, the lower the model metrics as the validation accuracy and precision values both dropped below 96.0%. Therefore, a simple architecture is needed for better model performance. Various combinations of neurons within two hidden layers were implemented (Figure 4). Based on the contour plot, the first and second hidden layer should have 60 and 5 neurons as it generates the highest validation accuracy and precision combination of 97.6% and 97.5%.

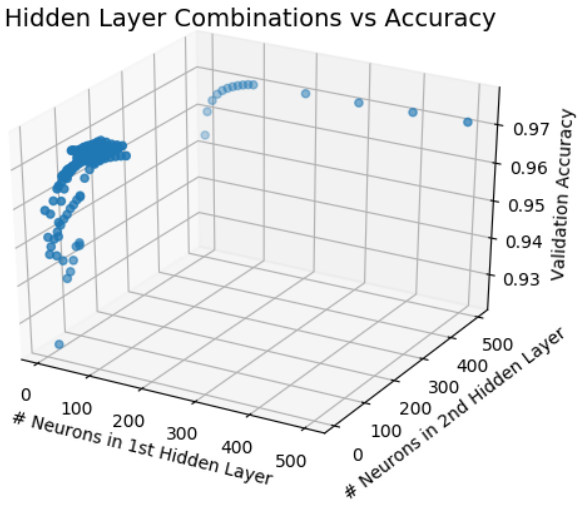

Figure 4: Hidden Layer Combinations vs Accuracy

The validation metrics were further improved by optimizing the number of epochs used to train the network. The highest validation accuracy and precision generated was 98.0% and 98.5% when the model was trained with 10 epochs in comparison to 100 epochs (Figure 5). Training the network with more epochs over fits the training data and does not generalize well for the validation set.

Finally, the last aspect of the neural network that has to be optimized is the count and value of the dropout layers. Dropout values ranging from 0.05-0.8 were tested on different layers. The addition of dropout values to input, hidden and/or output layers resulted in similar validation accuracy and precision as networks without dropouts. Since the neural network is already using only 1500 maximum features, a network architecture without dropouts was chosen for this dataset.

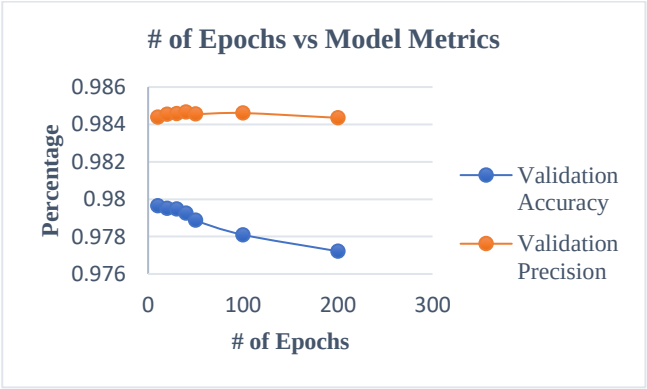

Figure 5. Number of Epochs vs Model Metrics

4.5.3 Model Results and Evaluation

The final neural network model consists of two hidden layers with 60 and 5 neurons, 1400 maximum features from tf-idf as the input, and 10 epochs to train the network. This model generated a test accuracy and precision of 98.0% and 97.0%. Across 10 random test sets, the accuracy and precision ranges from approximately 96-98% and 94-97%. The test metrics are similar to the validation metrics in section 4.5.2.

The weights in the neural network model that are adjusted in the hidden layers are a black-box and difficult to track. Neural networks only took 0.08 seconds to predict 1664 classes once the model was trained; approximately 99.9% faster than the k-NN algorithm to classify

5 Discussion

K-NN model is a time-consuming model that took 72 seconds to predict over 1500 SMSs (Table 1). Even though the algorithm calculations are simple and retractable, better model metrics can be derived from other algorithms.

Decision trees model took about 16 seconds for classification which was less than SVM and KNN. The accuracy was also approximately 96-96.4%, however, the precision of the model's results was lower compared to all the other models. In addition, SVM was even more time consuming taking up to 160s to fit and predict the model. SVM did however contribute to one of the highest accuracy and precision scores in comparison to the other classification models. Naïve Bayes has a comparatively fast run time at about 4.76 seconds to train model and then classify the SMS messages. It also had extremely high precision.

In association to other models, neural networks perform marginally better than all other classification algorithms in terms of test accuracy. However, the precision value of the neural networks model is less than that of Naïve Bayes. Precision is an important model metric as mobile users do not want SMS's frequently classified as spam instead of ham. However, the ratio of true positives to false positives is significantly one sided (approximately 43 true positives for one false positive) making neural networks the best algorithm to classify messages as spam or ham. Neural networks model has a few limitations. Firstly, the calculations that occurs in the hidden layers are a black box and difficult to track. The model calculations to generate classification variables are more transparent for other algorithms. Neural networks are time consuming to train with bigger datasets. However, after the model is trained, the adjusted weights are able to predict classification labels within fraction of a second.

	Accuracy (lower-upper Bound)	**Precision (lower-upper Bound)**	**Run time (s)**
KNN	95-97%	94-96%	72
Decision Trees	96-96.4%	87.9-91.8%	16
SVM	91%-97.9%	96%-99%	160
Naive Bayes	86.2%.	100%	4.76
Neural Networks	96-98%	94-97%.	0.08

Table 1: Comparison of Classification Algorithms

6 Conclusion

This report looked at different machine intelligence models to identify and classify spam from non-spam SMS messages, also called ham. Spam is an unsolicited mass message, this report looked at the medium SMS text message specifically, but spam can be sent across all forms of communication media. The models that were built and used to classify spam were Decision Trees, Naïve Bayes, SVM, KNN and Neural Networks. These models were run on a spam and ham data set from Kaggle after the data was cleaned, including stemming and removing stop words. All methods performed well, with an average of high 90 percent in both accuracy and precision. Precision is the most important measure because of its potential damage to the users. Therefore, since all models performed comparatively well in regards to accuracy and precision, the overall best model was found to be neural networks. This is because it consistently has the highest precision and accuracy combination and has a faster run time than all other models.

7 References

[1] A. Lupher, C. Engle and R. Xin, "Feature Selection and Classification of Spam on Social Networking Sites," [Online]. Available: http://bid.berkeley.edu/cs294-1-spring12/images/archive/6/6a/20120515031244!Spam-lupher-engle-xin.pdf

[2] W.A. Awad, S.M. ELseuofi, "Machine Learning Methods for Spam E-mail Classification," Feb 2011. [Online]. Available: http://airccse.org/journal/jcsit/0211ijcsit12.pdf

[3] S. J. Delany, M. Buckley and D. Greene, "SMS spam filtering: Methods and data," *Elsevier*, pp. 2-3, 2012.

[4] C. Metz, "Google Says Its AI Catches 99.9 Percent of Gmail Spam," Wired, 07 09 2015. [Online]. Available: https://www.wired.com/2015/07/google-says-ai-catches-99-9-percent-gmail-spam/. [Accessed 14 04 2018].

[5] "SMS Spam Classification", Kaggle, 2018. [Online]. Available: https://www.kaggle.com/shivam04/sms-spam-classification. [Accessed: 14- Apr- 2018].

[6] "Cross Validation and Model Selection", Python For Engineers, 2018. [Online]. Available: http://pythonforengineers.com/cross-validation-and-model-selection/. [Accessed: 17- Apr- 2018].

[7] A. Buzdar, "How To Build a Simple Spam-Detecting Machine Learning Classifier," 01 April 2017. [Online]. Available: https://hackernoon.com/how-to-build-a-simple-spam-detecting-machine-learning-classifier-4471fe6b816e.

Appendix A

A-1: Neural Networks Final Model Code

```
import pandas as pd
import csv
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from sklearn.feature_extraction.text import
CountVectorizer
import csv
import numpy as np
from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score

ps = PorterStemmer()
stop_words = set(stopwords.words('english'))

# Stems words to their root words and removes all
characters that are not alphabets
def stem_str(str):
    ret_str = ""
    for w in word_tokenize(str.lower()):
        if w not in stop_words and w.isalpha() and len(w) > 1:
            ret_str = ret_str + " " + ps.stem(w)
    ret_str = re.sub("[^a-zA-Z]", " ", ret_str)
    return ret_str.strip()

def clean():
    #Reading in the file via csv library
    filepath = 'C:\\Users\\Joash\\Desktop\\University
Stuff\\4B uni stuff\\SYDE 522\\522
Project\\SMS_spam_or_ham\\spam'
    csvfile = open(filepath + '.csv', "rt")
    reader = csv.reader(csvfile)
    count = 0
    data =[]
    for row in reader:
        data.append(row)
        count+=1
    data = data[1:]
    df = pd.DataFrame(data, columns=['class', 'sms'])

    #stemming and the removal of stop words via stem_str()
function
    df['stemmed_sms'] = df.loc[:,'sms'].apply(lambda x:
stem_str(str(x)))

    # Printing out the stemmed words to csv
    df.to_csv(filepath + '_result.csv', index=False)


if __name__ == '__main__':
    clean()
    #Reading in the file via csv library
    filepath = 'C:\\Users\\Joash\\Desktop\\University
Stuff\\4B uni stuff\\SYDE 522\\522
Project\\SMS_spam_or_ham' \
               '\\spam_result'
    csvfile = open(filepath + '.csv', "rt", encoding="utf8")
    reader = csv.reader(csvfile)
    sms_stemmed = []
    classification = []
    sms = []
    for row in reader:
        if len(row[2]) != 0:
            sms_stemmed.append(row[2])
            sms.append(row[1])
            if row[0] == "spam":
                classification.append(1)
            elif row[0] == "ham":
                classification.append(0)
    sms_stemmed = sms_stemmed[1:]
    sms = sms[1:]

    random_state = 2
    pre_score = []
    acc_score = []
    scores = []
    predicted_classes = []
    test_pred = []

    X_tr, X_te, y_tr, y_te = train_test_split(sms_stemmed,
classification, test_size=0.30, random_state=random_state)

    max_features = 1500
    tfidf = TfidfVectorizer(max_features=max_features)
    x_tfidf = tfidf.fit_transform(sms_stemmed).toarray()
    classification = np.asarray(classification)
    print(type(x_tfidf), x_tfidf.shape, classification.shape)

    # split into train and test
    X_train, X_test, y_train, y_test = train_test_split(x_tfidf,
classification, test_size=0.30, random_state=random_state)

    model = Sequential()
    model.add(Dense(60, input_shape=(max_features,)))
    model.add(Activation('relu'))
    # model.add(Dropout(0.2))
    model.add(Dense(5))
    model.add(Activation('relu'))
    # model.add(Dropout(0.2))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.summary()
    model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['acc'])
```

```
    model.fit(X_train, y_train, batch_size=64, epochs=10,
verbose=1)

    test_pred = model.predict(X_test)
    predicted_classes = np.around(test_pred, decimals=0)

    accuracy = accuracy_score(y_test, predicted_classes)
    # precision = cm[0, 0] / (cm[0, 0] + cm[1, 0])
    precision = precision_score(y_test, predicted_classes)
    print('Test accuracy is', accuracy)
    print('Test precision is', precision)
```