Classifying “Ham” and “Spam” to create an Email classifier model Comparing Conventional Machine Learning to a Basic Neural Network

Completed by: Mike Willis, Saptarshi Kundu, Paul George Parakkal, Saurav Singh Kundlas, Tanvir Singh Ahuja

Topic:	Worked on by:
Selecting Dataset	All
Data Exploration	All
Preprocessing	Paul, Saptarshi
Feature Selection	Mike, Paul, Saptarshi
Building, Training & Model Hyperparameter Tuning	All
Presentation	All
Report	All

Contents

Introduction

A common problem everybody faces in the digital age: spam. Spam emails, spam texts, even spam phone calls! While a majority of workers who use email for communications don't receive spam, for some, filtering out spam can cost up to 30-60 minutes over the course of a work day (Fallows, 2003). Through the use of machine learning and deep learning models, data scientists are able to use classifier models to detect which emails are spam and which are not, which in turn will save workers time and increase productivity. This report's focus is on: **Which models, machine learning or deep learning, and which classifiers are the most effective at classifying spam correctly?**

Dataset

The dataset we chose is: Enron Spam dataset. This is a real-life dataset consistent of both sent and received emails. It was put together by former employees of Enron, who went through and labelled their work emails as "Ham" or "Spam." The dataset contains 33665 emails in total. Below we will discuss how the data was picked, cleaned, preprocessed, and how features were selected.

Methodology

Picking a Dataset

We spent much time discussing which direction we wanted to go with this report. While we had many complex and interesting ideas, we needed to be practical about the timeframe and what could be accomplished. For that reason, we decided to go with email classification. We were able to find a well-balanced dataset (as picture below, Ham = 0 and made up 16545 (49.14%, Spam = 1 and made up 17120 (50.85%) that used real world emails and this gave us time to delve into which models we wanted to experiment with and tuning the models for optimal results. Lastly, we used several python scripts to combine 6 different folders of emails to create the database we worked on. We chose to drop all columns except: 'Subject', 'Messages', and 'Spam/Ham.'

Figure 1. Dataset Balance with Spam assigned to 1 and Ham assigned to 0.

Preprocessing

1) Label Encoding
    - The first step was to label encode the 'Spam/Ham' column with 'Spam' values mapped to 0 and 'Ham' mapped to 1, followed by renaming the column to 'label'.
2) Cleaning
    - In the cleaning step, we encountered three distinct cases: first, only the 'Subject' field was empty; second, only the 'Message' field was empty; and third where both 'Subject' and 'Message' was empty.
    - The following table summarizes the actions we took against each scenario.

Subject	Message	Action	#Rows (% of Dataset)
EMPTY	EMPTY	DROPPED	51 (0.15%)
EMPTY	DATA AVAILABLE	REPLACED EMPTY SUBJECT WITH "(NO_SUBJECT)"	238 (0.7%)
DATA AVAILABLE	EMPTY	REPLACED EMPTY MESSAGE WITH "(NO_MESSAGE_TEXT)"	320 (0.95%)

3) RegEx Expressions – Applied independently on 'Subject' and 'Message' columns
    - To reduce the complexity and mismatching of regex expressions employed in the following steps, all texts were lowercased.
    - Then we matched and removed all characters except for alphabets (a-z) and full stop (.). This was done to get rid of special characters that we otherwise presumed might interfere during tokenization.
    - This step was followed by removal of extra white spaces and multiple occurrences of full stop (.).
4) Dropping Sentences – Applied independently on 'Subject' and 'Message' columns
    - The first task was to sentence tokenize each text.
    - Any sentence that contained less than 2 words were dropped and the remaining sentences were joined and replaced the original text.
    - However, if dropping such sentences resulted in an empty text, then original text was retained as such.
5) Lemmatization – Applied independently on 'Subject' and 'Message' columns
    - Primarily, each text was sentence tokenized.
    - Then, we realized that lemmatizer output accuracy is improved when, along with the word, its pos-tag is also passed as an additional parameter.
    - Hence, each sentence was then word tokenized and in turn each word was pos-tagged (part-of-speech).

However, the default ‘nltk.pos_tag()’ function outputs a tag corresponding to the input word from a list of 36 unique tags. Meanwhile, the word_net_lemmatizer() accepts only 4 different basic tags.

Input Sample:

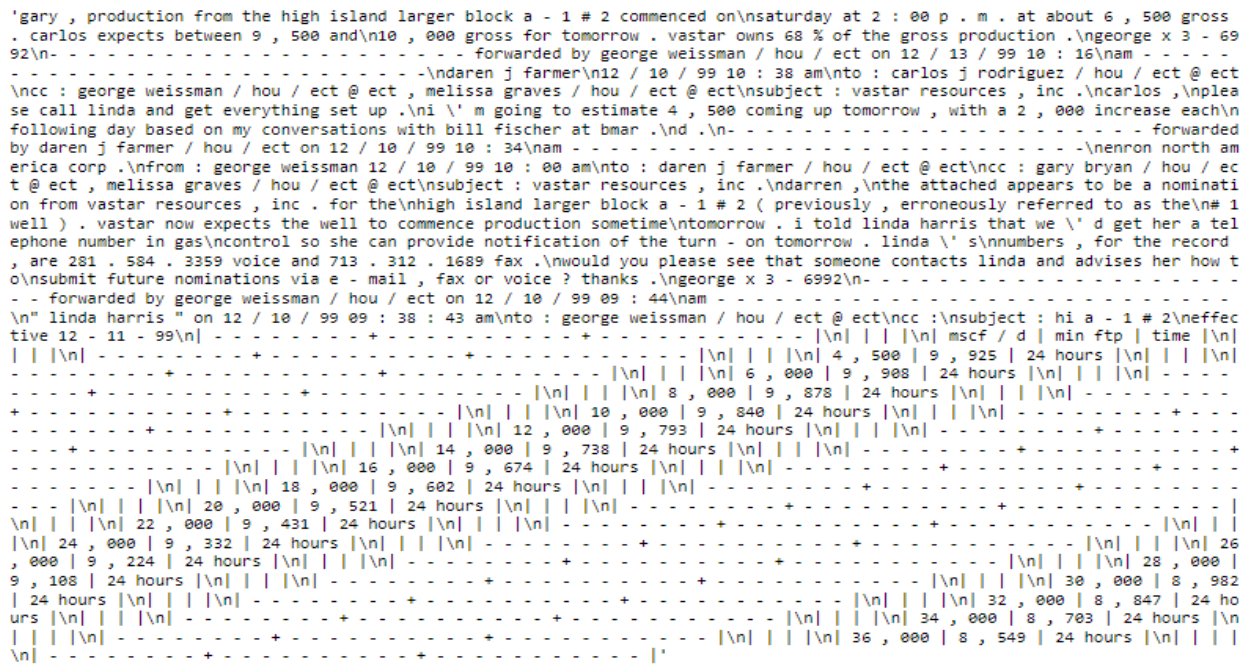

Figure 2. Raw Data Sample.

Processed Input Sample:

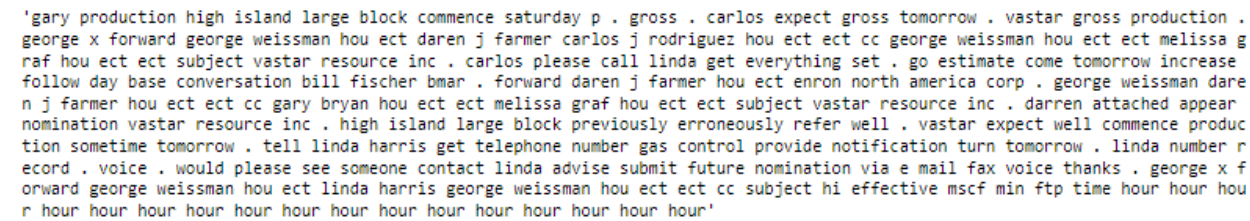

Figure 3. Processed Input Sample.

Despite the extra precautions and steps taken to avoid generation of empty texts after the pre-processing step, some texts did turn up empty (around 60 texts). An analysis of the text revealed that they were composed entirely of either numerals or special characters or a combination of both. Since we wouldn’t have been able to process such text in any other manner, the resultant empty fields were replaced with the text ‘Subject’ and ‘Message’.

Feature Selection

At this point, we had decided to consider the two columns of the dataset, 'Subject' and 'Message', separately, and develop two different independent models. This was due to the presumption that since humans effortlessly recognize spam emails with high accuracy just based on the subject line alone, maybe the accuracy obtained by using only the subject line data might be on par or even higher than the one obtained through using message body.

We used the tf-idf method of vectorization for feature extraction. The feature space of the dataset based on the 'Subject' column was 33665 x 13923 (13,923 unique word vocabulary) and based on the 'Message' column was 33665 x 128539 (128,539 unique word vocabulary).

```
[8] #Applying Tfidf Vectorization separately on subject data and message data

    from sklearn.feature_extraction.text import TfidfVectorizer

    sub_vectorizer=TfidfVectorizer()
    subject=sub_vectorizer.fit_transform(db['Subject'])
    nof_elements_sub=int(str(subject.__str__).split(' ')[str(subject.__str__).split(' ').index('stored')-1])
    print(f'Sparse matrix of size {subject.shape} has {nof_elements_sub} elements, {nof_elements_sub/(subject.shape[0]*subject.shape[1])*100:.4f}% dense')

    msg_vectorizer=TfidfVectorizer()
    message=msg_vectorizer.fit_transform(db['Message'])
    nof_elements_msg=int(str(message.__str__).split(' ')[str(message.__str__).split(' ').index('stored')-1])
    print(f'Sparse matrix of size {message.shape} has {nof_elements_msg} elements, {nof_elements_msg/(message.shape[0]*message.shape[1])*100:.4f}% dense')

    Sparse matrix of size (33665, 13923) has 121410 elements, 0.0259% dense
    Sparse matrix of size (33665, 128539) has 2744163 elements, 0.0634% dense
```

Figure 4. Code showing how we applied TFIDF Vectorizer on Subject and Message Data.

Model Selection

Note: When running the RandomizedSeachCV tuning, the parameter dictionaries for each model remained the same for both 'Subject' and 'Messages.' The optimal parameters were different and will be highlighted in the Hyperparameters Tuning section.

Random Forest Classifier

Random Forest is an ensemble learning method where the collective knowledge of multiple individual decision trees is extracted to arrive at a classification.
The random forest implementation through scikit-learn library has multiple parameters that can be tweaked to produce better results.

Parameter	Description	Parameter Values for RandomizedSearchCV
'n_estimators'	the number of independent decision trees that are used to populate the 'forest'	10, 30
'max_depth'	the maximum depth the trees can go before their growth is terminated	5, 10, None
'min_samples_split'	the minimum number of samples that is required to proceed with splitting the node	2, 5
'max_features'	the maximum number of features to consider while searching for best split	'log2' – where log2(num_features) are considered None – where all the features are considered
'bootstrap'	sample features with or without replacements	True / False

Logistic Regression

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Parameter	Description	Parameter Values for RandomizedSearchCV
solver	Algorithm to use in the optimization problem	saga
'max_iter'	Maximum number of iterations taken for the solvers to converge.	1000
'C'	C is the penalty parameter of error term	0.1,1,10
'penalty'	Specify the norm of the penalty:	elasticnet
'l1_ratio'	The Elastic-Net mixing parameter	0.3
'random_state'	Used when solver == 'sag', 'saga' or 'liblinear' to shuffle the data	42
'n_jobs'	Number of CPU cores used when parallelizing over classes if multi_class='ovr'	-1

Support Vector Machine Classifier

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it's common to have access to a dataset of at most a couple of thousands of tagged samples.

Parameter	Description	Parameter values for RandomizedSearchCV
'gamma'	gamma is a parameter for non linear hyperplanes.	scale
'class_weight'	The class_weight is a dictionary that defines each class label and the weighting to apply to the C value in the calculation of the soft margin.	balanced
'random_state'	random_state is the seed used by the random number generator	42
'C'	C is the penalty parameter of the error term	0.01,1,100
'kernel'	Kernel Function is a method used to take data as input and transform into the required form of processing data	Linear, rbf, sigmoid

Neural Network Model

We decided to go with a simple artificial neural network with an input layer followed by 2 hidden layers terminating in an activation/output layer. The number of neurons in the input layer scales according to the available feature space. 13923 neurons for the 'Subject' column-based model and 128539 neurons for the 'Message' column-based model.

The following layers are identical for both models. The first hidden layer has 20 units with 'tanh' activation and the second layer has 8 units with 'sigmoid' activation. The 'ADAM' optimizer was used for training and loss was measured using 'BINARY_CROSSENTROPY' while using 'ACCURACY' as the metric for scoring. With using a validation split of 20% and a batch size of 10, the training sessions ran for a total of 50 epochs.

Tuning Model Hyperparameters

Subject

Random Forest Classifier

Score: 90.84%

Parameters: {'n_estimators': 10, 'min_samples_split': 5, 'max_features': 'log2', 'max_depth': None, 'bootstrap': True}

Logistic Regression
Score: 91.53%
Parameters: {'C': 10}

Support Vector Machine Classification
Score: 92.36%
Parameters: {'kernel': 'rbf', 'C': 1}

Message

Random Forest Classifier
Score: 94.62%
Parameters: {'n_estimators': 10, 'min_samples_split': 2, 'max_features': 'None', 'max_depth': None, 'bootstrap': False}

Logistic Regression
Score: 98.10%
Parameters: {'C': 10}

Support Vector Machine Classification
Score: 98.20%
Parameters: {'kernel': 'linear', 'C': 100}

Results

We ran all the conventional machine learning models with 100% of the data using a 66/33 train test split. After tuning, SVM was the best model for both 'Subject' and 'Messages' scoring 93.75% and 98.27% respectively. Due to our computer configurations, we weren't able to run the neural network (NN) model at 100%, instead maxing out at 25% of the dataset. For this reason, after finding that SVM was the best conventional machine learning model we ran it again with 25% of the data, maintaining the same train test split, so it could be accurately compared to the NN model.

Subject

Best Conventional Model

Best model for Subject: SVM
Accuracy: 93.75%

Confusion Matrix

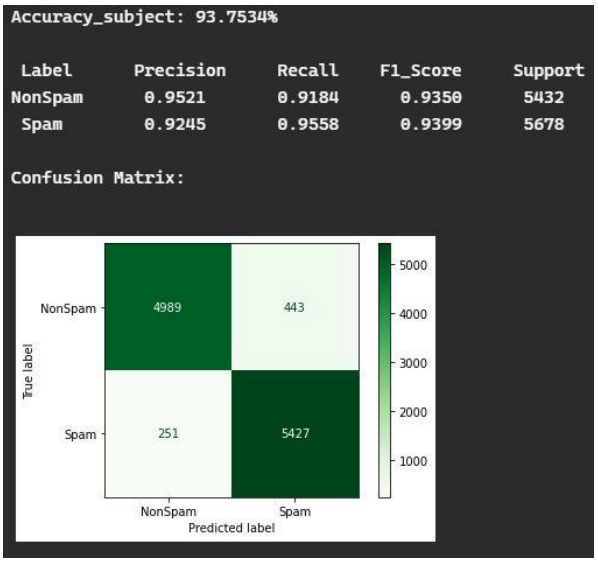

Figure 5. SVM model accuracy and confusion matrix using 100% of the 'Subject' feature set.

Messages

Best Conventional Model

Best Model for Message: SVM

Accuracy: 98.27%

Confusion Matrix for SVM

Figure 6. SVM model accuracy and confusion matrix using 100% of the 'Messages' feature set.

Comparison between Neural Network and SVM with 25% of the data

Subject

Neural Network (25% of data)

Accuracy: 91.3681%

Label	Precision	Recall	F1_Score	Support
NonSpam	0.9156	0.9070	0.9113	5432
Spam	0.9119	0.9200	0.9159	5678

Confusion Matrix:

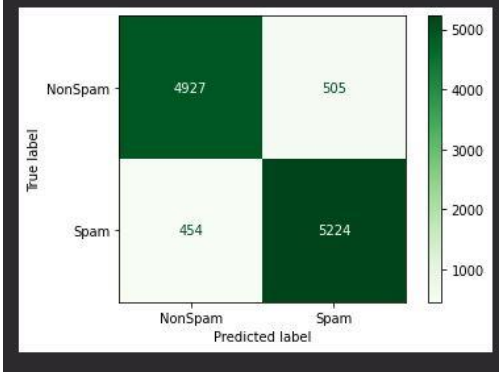

Figure 7. NN model accuracy and confusion matrix using 25% of the 'Subject' feature set.

SVM (25% of data)

Accuracy_subject: 93.7534%

Label	Precision	Recall	F1_Score	Support
NonSpam	0.9521	0.9184	0.9350	5432
Spam	0.9245	0.9558	0.9399	5678

Confusion Matrix:

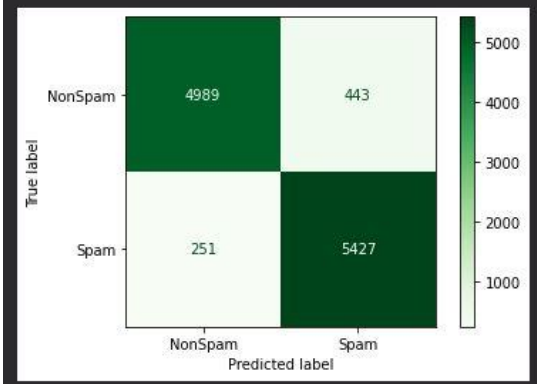

Figure 8. SVM model accuracy and confusion matrix using 25% of the 'Subject' feature set.

Messages

Neural Network (25% of data)

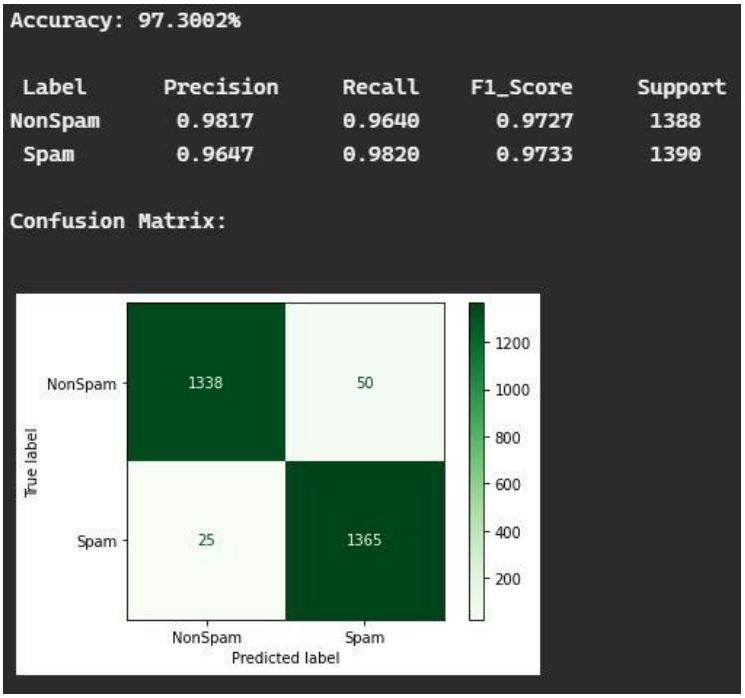

Figure 9. NN model accuracy and confusion matrix using 25% of the 'Messages' feature set.

SVM (25% of data)

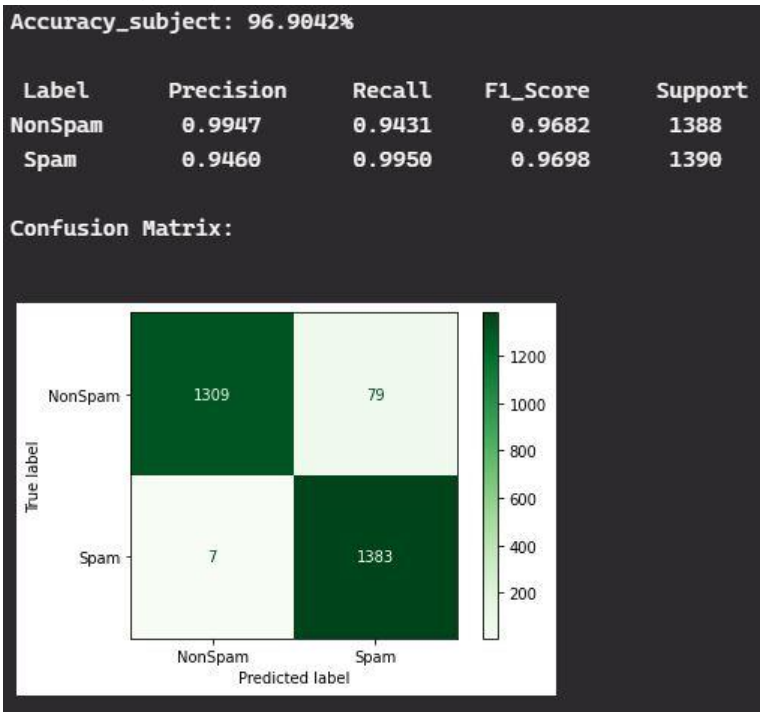

Figure 9. SVM model accuracy and confusion matrix using 25% of the 'Messages' feature set.

Neural Network Model Training & Loss

When looking at the model loss graph of the model based on 'Subject' column, the fact that there is a continual increase in the validation losses with increase in epochs, all the while observing a substantial decrease in training losses, leads us to the conclusion the model might be overfitting. This belief is further strengthened by the lack of improvement in the accuracy of model as observed from the model accuracy graph.

However, in the case of the model based on 'Message' column, the training losses decreases in alignment with the validation loss and thereby leads us to the conclusion that the model indeed is training as expected. This belief is further strengthened by the observation that the model accuracy is constantly maintained over the epochs while the losses are reducing.

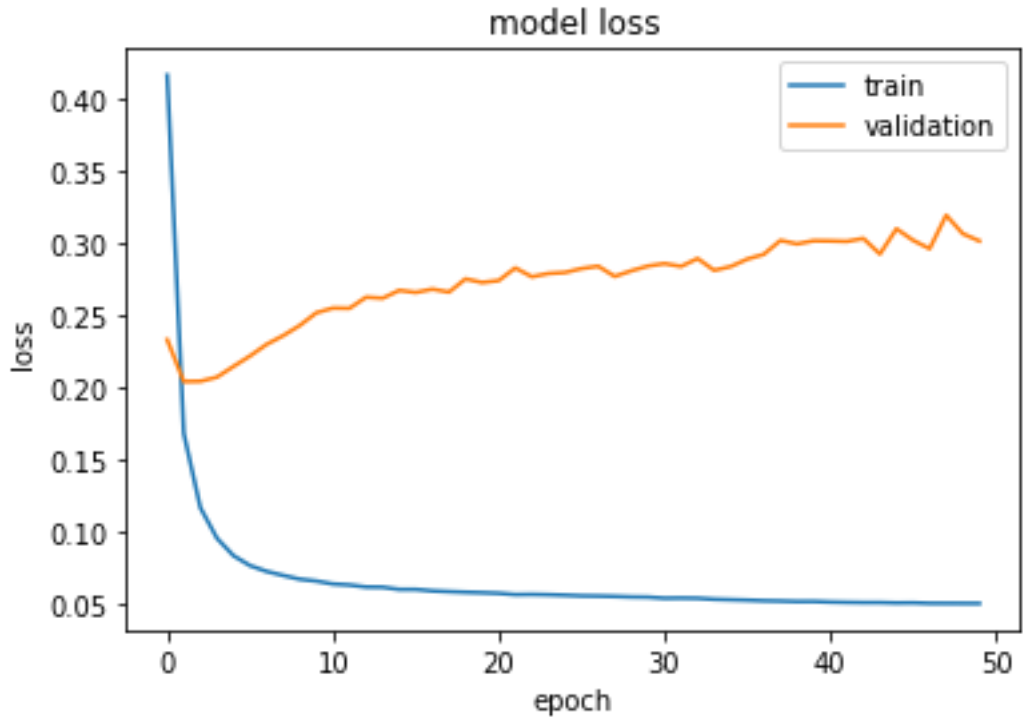

Figure 10. NN training accuracy using 25% of the 'Subject' feature set.

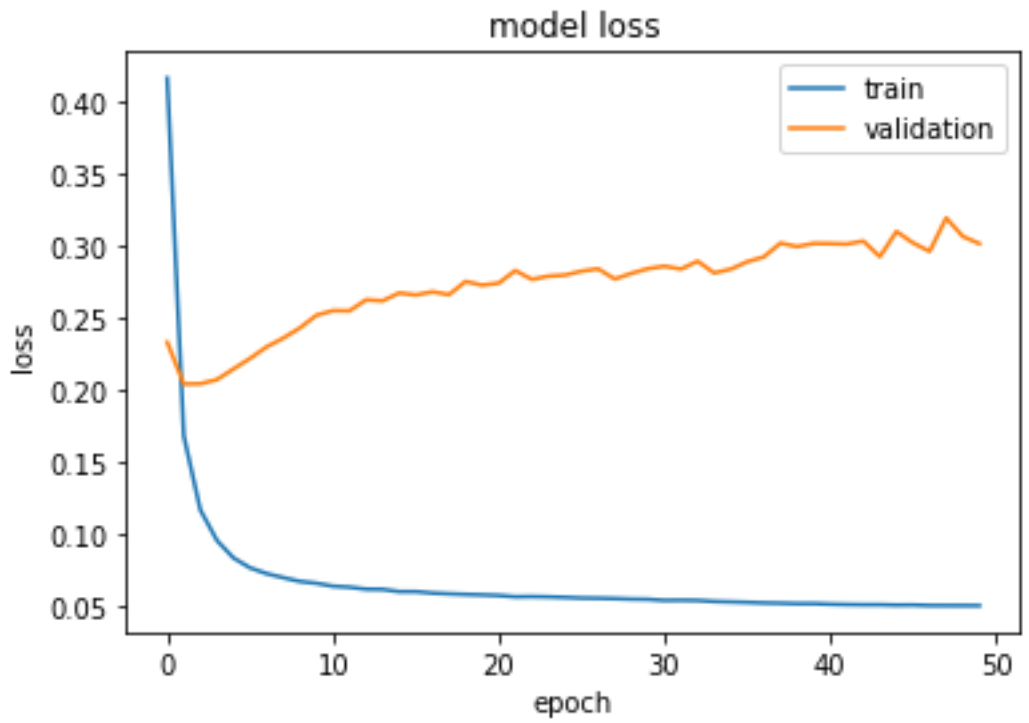

Figure 11. NN training loss using 25% of the 'Subject' feature set.

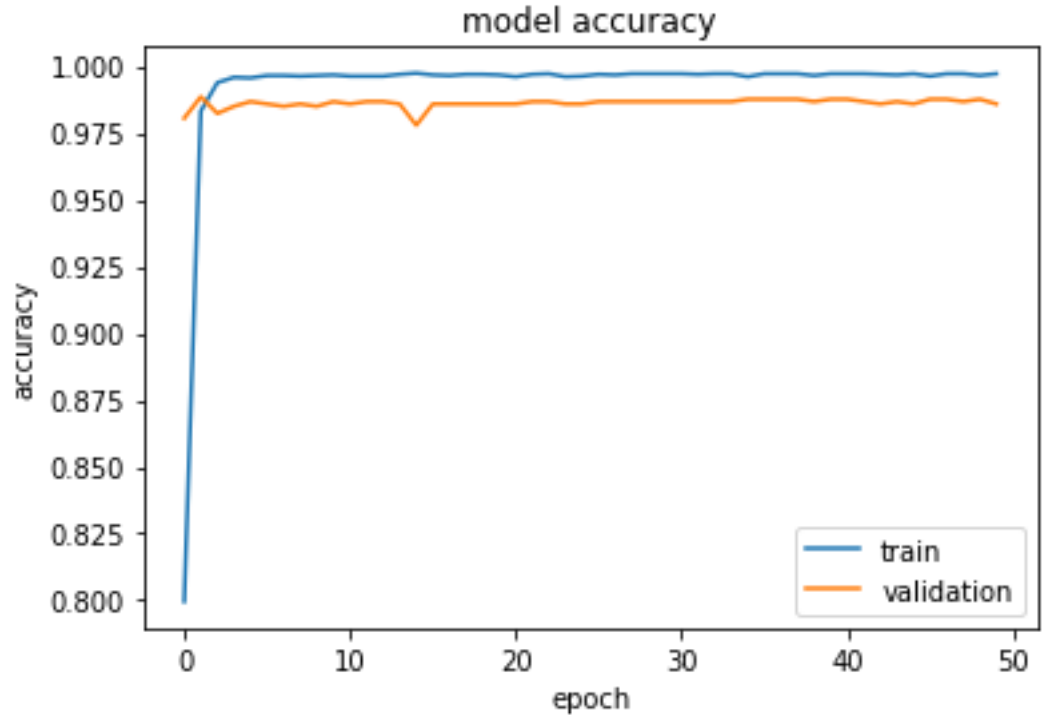

Figure 12. NN training accuracy using 25% of the 'Message' feature set.

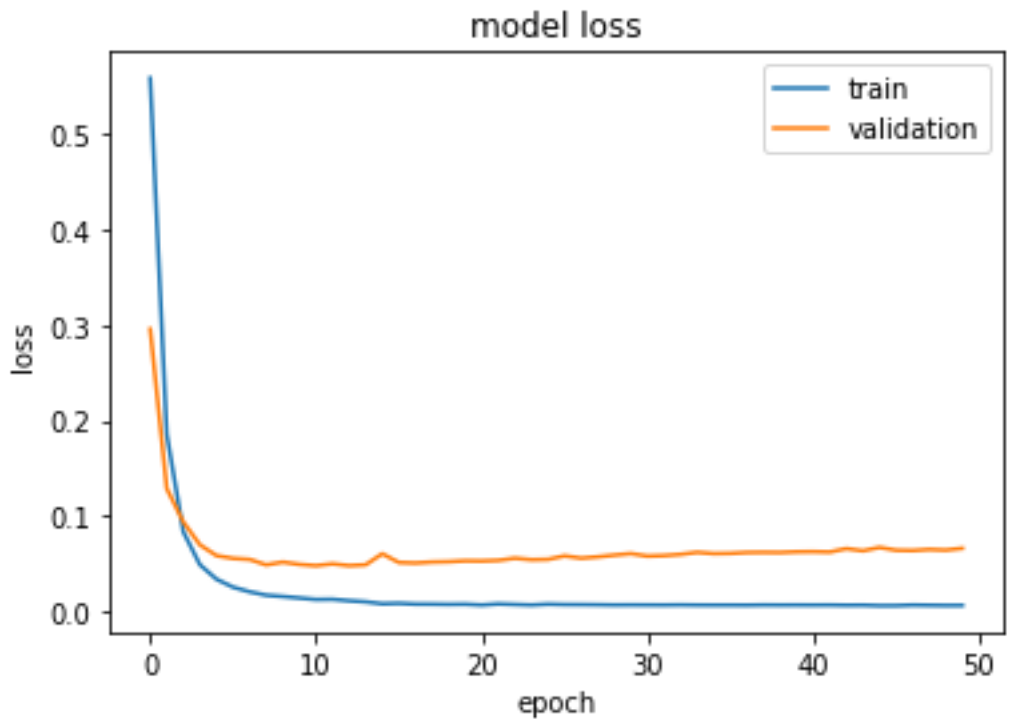

Figure 13. NN training loss using 25% of the 'Message' feature set.

Discussion

All models were evaluated using 3 measures: precision, recall, and F1_score. We decided when we started to split the data into 'Subjects' and 'Messages' and run all models and the NN on each set of data to see differences in prediction capabilities. We were curious how effective classifying emails could be when only the subject line of the email was used. It turns out it was not as effective as using the whole email (what we've called: messages). Having said that, we never got to try running the dataset through both (classifying based on subject, then taking those results and classifying on messages). We'd believe this combination method to have an accuracy somewhere in between the subject and messages accuracies but it would've been interesting to try.

When evaluating our conventional models, we ran them through a series of hyperparameters on 100% of the data to pick the most accurate, then again to actually classify the 2 feature sets. From here we picked the most accurate conventional model and ran it against the NN model. Since we were only able to run the NN model at 25% of the messages we then ran the top conventional model (SVM) at 25% of the data as well. Interestingly, we found that SVM was more accurate when classifying by subject only whereas the NN model was more accurate when classifying the messages. We are not quite sure why this is the case, but further study is necessary to understand the differences.

While completing this project we came across several challenges. The largest was the inability to run 100% of the data through our NN model. This comes down to computer configurations but it would be interesting to try in the future. Additionally, time was a constraint as we were unable to try word2vec instead of TFIDF as a cleaning tool to be used to train the models.

Interestingly, in terms of all our models' inaccuracies, you can see in the confusion matrix the model was less accurate at correctly classifying ham as spam than the other way around. This is not ideal as it meant if this model were deployed, people would need to check their spam folders more to ensure they get all the correct emails. Conversely, only a few spam emails were classified as ham.

In the future it would be interesting to try more hyperparameters and different NN models and see the results. As we learn more about NN's and how they function we plan to circle back and make optimizations in the future. It'd also be interesting if we can find a dataset that properly breaks down classifying even further, into work, promotion, social emails, etc.

References

Fallows, D. (2003, October 22). *Part 3. The Volume and Burdens of Spam.* Pew Research. https://www.pewresearch.org/internet/2003/10/22/part-3-the-volume-and-burdens-of-spam/

Dataset: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html