Text Classification on Spam Email Filtering

Yukai Yang
New York University
yy2949@nyu.edu

Dianjing Fan
New York University
df1854@nyu.edu

ABSTRACT

Nowadays people's lives are heavily relying on emails. However, at the same time, we often spend a lot of time checking and deleting spam emails, which often makes our inbox crowded. In our project, we aim at classifying spam and non-spam emails in an efficient way, so that people can spend far less time arranging their email inbox. The dataset we use to train the models is the Enron email dataset, which contains approximately 500,000 emails generated by the employees of the Enron Corporation. The models we analyze are logistic regression, naïve bayes, SVM, random forest, ANN, and RNN using LSTM. For the first five machine learning algorithms and the last recurrent neural network model, we utilize different approaches to extract features from the text. During the training process, we modify the hyperparameters to compare the result and obtain an optimized model. In the final stage, we use testing accuracy and f1-score to evaluate the models. The result shows that logistic regression and LSTM outperform the other models.

1 INTRODUCTION

Emails are involved in our daily lives. Students or white collars, we work, study, and communicate with each other through emails. The email system brings us efficiency and convenience. At the same time, it is also used for advertising by companies. The websites and apps force us to sign up for an account using an email address, and then they would continuously send us tons of promotional emails. Our inboxes are often filled with those emails, which cost us a lot of time to recognize and delete, and sometimes we may even miss those important emails. Our project focuses on classifying spam and non-spam emails efficiently using five traditional machine learning algorithms and one LSTM algorithm. In this paper, we first introduce our dataset and how we preprocess the emails. Then we discuss the methods for the transformation from texts to the feature vectors, namely, bag of words and the GloVe pre-trained Word2Vec model. We analyze six models, compare their performance, and discuss the advantages and disadvantages of the models. In the last section, we consider the limitations and future improvements for our project.

2 DATASET DESCRIPTION AND PREPROCESSING

The original Enron email dataset collected by Carnegie Mellon University contains approximately 500,000 emails generated by the employees of the Enron Corporation. Such a comprehensive dataset would be sufficient for us to use various algorithms to train an optimized model. For our project, we use the processed version of this dataset specifically for the purpose of spam and non-spam classification. This processed dataset is a file folder consisting of 30207 text documents, each text document containing an email message. The text documents are labeled as "spam" or "ham" in the text file name. Within these 30207 messages, 13662 are spam emails and 16545 are non-spam emails.

For data preprocessing, we first check whether each text document only contains one email message. If not, we remove any unnecessary information such as subject, date, forwarded messages, etc. in order to have clean and plain text messages. We also make all the words lowercase and remove non-alphabetic words, and also remove stopping words that appear frequently but do not provide us useful information. Then we extract the ground truth label from the text file name. The spam emails are labeled as one and ham emails are labeled as zero. We store all the email messages and their corresponding labels in two lists. For cross validation, we split the whole dataset equally into six parts: four training datasets, one testing dataset, and one validation dataset.

3 METHODS

3.1 Feature Extraction

3.1.1 Bag of Words. The first step before we train the model is to convert the texts to feature vectors. For the first five machine learning models, we use the most straight-forward bag of words approach. We tokenize the messages and construct a dictionary of all the words and their frequency. Then we extract the most common 3000 words as the features for our classification problem. We iterate through each message and record the occurrences of each feature. In this way, the text is converted into a N (total number of messages) by 3000 feature matrix. Though the bag of words approach is straight-forward and easy to understand, it may result in a sparse feature matrix containing many zero values. Another limitation is that the feature matrix does not contain the relationship information between each word in a message.

3.1.2 GloVe. The feature extraction method we use for the LSTM model is based on GloVe, the Stanford's pre-trained Word2Vec model. Word2Vec produces a vector space by transforming every word into a unique vector. The words that have similar semantic meaning are located close to each other in the space. GloVe is trained on the data of Wikipedia with about 6 billion tokens and a 400,000 word dictionary. The advantage of this word embedding is that GloVe does not depend on our dataset, but global statistics, which makes the feature vector more robust.

3.2 Machine Learning Models

3.2.1 Logistic Regression. Logistic regression is a discriminative model that maps directly from the feature vector to the labels 0, 1. It is easy to implement and efficient to train, which is a good choice for our huge dataset. Another advantage is that it does not make assumptions about the distribution of the classes. The limitation is that logistic regression requires that the data are linear separable

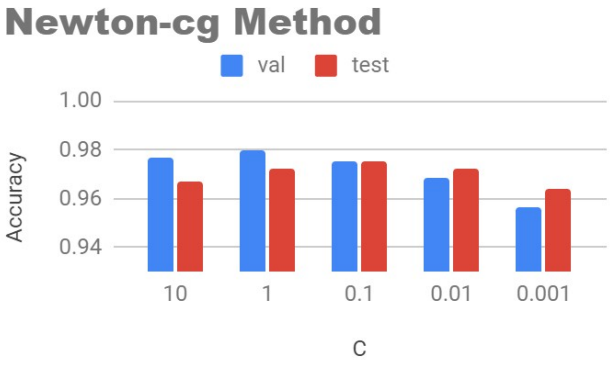

Figure 1: Accuracy Score of Logistic Regression Using Newton-cg Method

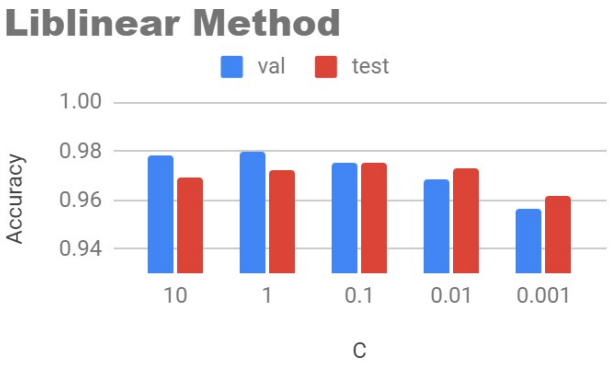

Figure 2: Accuracy Score of Logistic Regression Using Liblinear Method

and have average or no multicollinearity, while our dataset does not guarantee that.

We apply regularized logistic regression to our dataset. We use L2 regularization in order to avoid model overfitting. Here we tune two hyperparameters: C and solver. C represents the inverse of regularization strength, and smaller C values mean stronger constraint. We compare the results for C equals 0.001, 0.01, 0.1, 1, and 10. Solver indicates the method of learning the parameters. We include 3 methods, which are Newton's Metho-d, Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm, and A Library for Large Linear Classification in our model.

The result shows that the model achieves the best accuracy score when C is 0.1 using Newton or lbfgs method. During our training process, we find that lbfgs has a faster training speed than Newton method. The best validation accuracy is 97.8%, and the best testing accuracy is 97.6%. For model evaluation, we compute the f1 score, which is 0.9635. The high f1 score indicates that the logistic regression model has a high precision and recall, meaning that a high portion of the positive cases are correctly predicted.

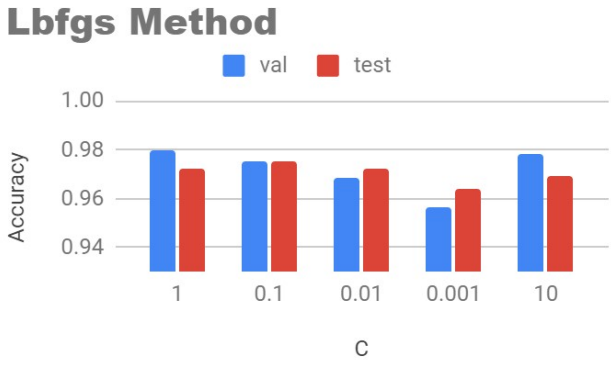

Figure 3: Accuracy Score of Logistic Regression Using Lbfgs Method

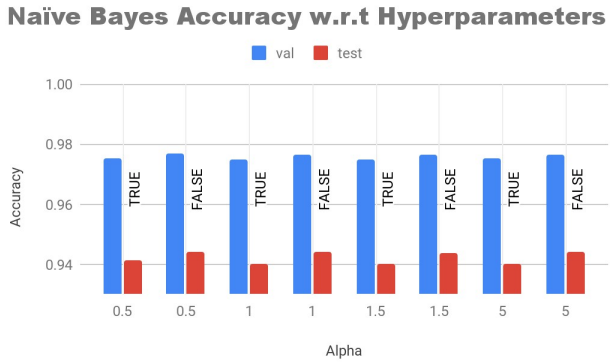

Figure 4: Accuracy Score of Naïve Bayes

3.2.2 Naïve Bayes. Naive Bayes is a generative model which makes predictions based on Bayes Rule. Similarly to logistic regression, Naive Bayes model is easy to understand and fast to train. However, the model assumes that the predictors are independent. The result may not be as accurate as other models in text classification since some words in a sentence are correlated.

During the training process, we tune two hyperparameters: the value of the smoothing parameter alpha and whether to learn the class prior probabilities. If the hyperparameter fit_prior is set to false, a uniform prior distribution would be used. When alpha = 1 or 5 and fit_prior is false, we get the best accuracy score. The validation accuracy is 97.6% and the testing accuracy is 97.4%. F1 score is 0.9262, which is not as high as the logistic regression.

3.2.3 SVC. The third model we choose to use is Support Vector Classifier. The SVC model creates a feature space for the predictors, each dimension of which represents a "feature" of an object. The feature indicates the prevalence of a particular word in text classification. SVC works effectively for high-dimensional features. Also, SVC does not have strict assumptions. The model is suitable for our dataset because the feature vectors transformed from the text messages are high-dimensional. However, it turns out that for

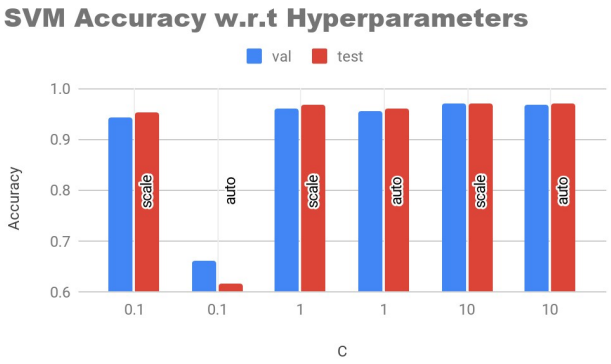

Figure 5: Accuracy Score of SVM

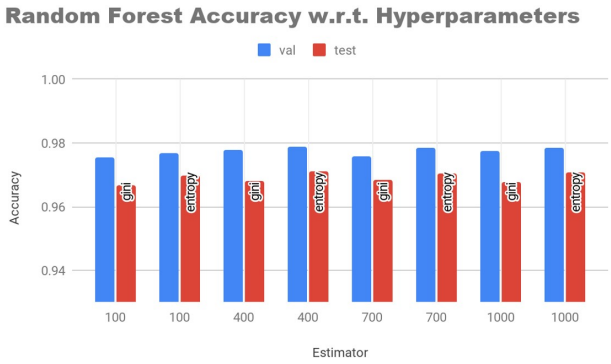

Figure 6: Accuracy Score of Random Forest

our large dataset, SVC takes much longer time compared to other models.

SVC can also handle the cases when the classes are not linear-separable by setting a kernel. For our model, we use the Radial Basis Function kernel. We tune two hyperparameters for the rbs kernel: C and Gamma. C is a regularization parameter inversely representing the strength of the regularization. Gamma defines how far the influence of a single training sample reaches. The best result is achieved when C is 10 and Gamma is set to "scale", meaning that we use a different gamma value instead of the default value. The validation accuracy is 96.9%, the testing accuracy is 97.1%, and the f1 score is 0.9537. We also notice that during the training process, there is an unusual result. The accuracy score is 60%, which is low compared to other results, for C is 0.1 and gamma is the default value.

3.2.4 Random Forest. The Random Forest Classifier generates a set of decision trees. Each tree splits on a predictor from a randomly selected subset of the predictors. The final result is aggregated from the results of each tree. One advantage is that random forest reduces the overfitting problem and increases the accuracy compared to decision trees, but it is computationally-intensive and time-consuming to train the model.

To optimize the model, we tune one hyperparameter: estimator, which represents the number of trees. We use two different loss functions, gini index and cross entropy loss, to compare the results. The best result is achieved when the number of trees is 400 using cross entropy loss. The validation accuracy is 97.9%, the testing accuracy is 97.1%, and f1 score is 0.9581.

3.3 Neural Network Models

3.3.1 ANN. In addition to previous models, we also utilize the algorithms in deep learning. Artificial neural networks (ANNs) generate the model by simulating how human brains process the information. Since an ANN model could contain many hidden layers, it is robust to the noise in training data and could generate a high accuracy score. One disadvantage is that the optimal structure of the neural network must be achieved by trials or previous experience, thus the choice of hyperparameters is important. Another drawback is

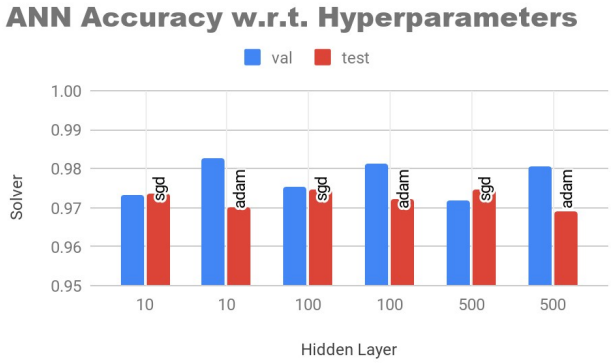

Figure 7: Accuracy Score of ANN

that the result of ANNs is hard to interpret. The model does not explicitly give the clue for its solution.

The hyperparameter we tune is the number of hidden layers. For the choice of other hyperparameters, we use Stochastic Gradient Descent and Adam optimizer to train the parameters. The activation function is Relu activation, with batch size 200, and the loss function is cross entropy loss. We get the best result when the number of hidden layers is 10 using Adam optimizer. The validation accuracy is 98.2%, testing accuracy is 97.0%, and f1 score is 0.9522.

3.3.2 LSTM. LSTM, which stands for Long Short Term Memory networks, is a special kind of Recurrent Neural Network. For a text message, the LSTM model is able to model the long term dependencies by memorizing and utilizing the information from many steps back, which is a great advance that all of our previous models do not have. However, the main drawback is that the LSTM model takes an extremely long time to train. The advantages and disadvantages are both significant and distinct for this model.

Our model structure consists of one word embedding layer using GloVe pre-trained Word2Vec, one LSTM layer, and one fully-connected layer with dropout. There are several hyperparameters we choose to use: the number of features in the hidden state h is 32; the number of recurrent layers is 1. We use cross entropy loss with

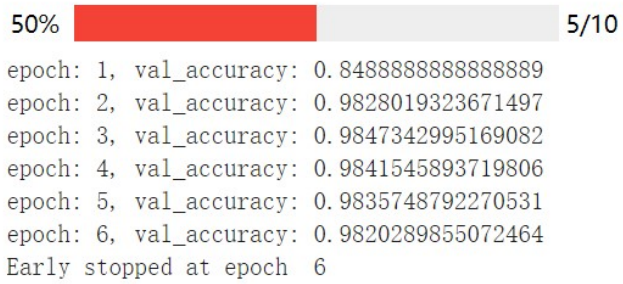

Figure 8: Early Stopping in Epoch 6

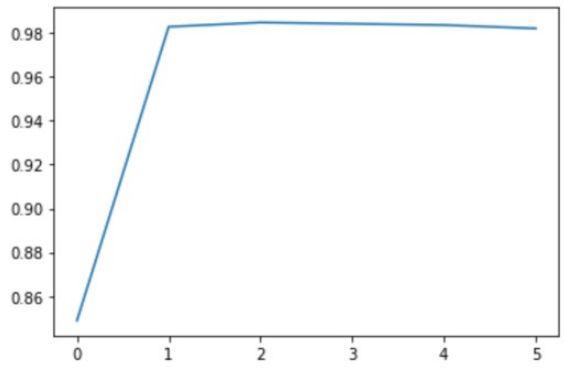

Figure 9: Validation Accuracy of LSTM

Adam optimizer and the final dropout rate is 0.3. We also use early stopping in order to avoid overfitting. We set the number of total epochs to 10, and the validation accuracy does not increase after epoch 6. The final validation accuracy is 98.2% and testing accuracy at epoch 6 is 97.8%.

4 RESULTS SUMMARY

Table 1: Accuracy Score for Different Models

Accuracy	LR	NB	SVM	RFC	ANN	LSTM
Validation	97.8%	97.6%	96.9%	97.9%	98.2%	98.2%
Testing	97.6%	97.4%	97.1%	97.1%	97.0%	97.8%

4.1 Model Performance Comparison

After multiple series of hyperparameters tuning, we find that logistic regression and LSTM neural network performs the best out of the six. Though the differences, in general, are not very extensive. As for the reasons, spam filtering is a rather straightforward text classification task, as both the models and our human intuition would focus on searching for frequent signs of spam emails, also known as signal words. This makes the boundary rather clear, and thus leads to a desirable output from logistic regression. Other models such as SVM and random forest might be good at dealing with more nasty datasets(less clear boundary line), but may not perform that well on our project here.

ANN was expected to perform at least equally well as logistic regression, but its accuracy turns out to be a little bit lower. More importantly, when we maintain other hyperparameters as default, and only look at the number of hidden layers, we get the best result when the number of hidden layers is 10, instead of 100 or 500. This should be due to the overfitting problem, meaning that a very deep neural network may understand the training set much better, but won't be familiar with a new testing dataset. LSTM, by applying early stop and an appropriate drop out rate, can solve this problem.

As for the training time, logistic, not surprisingly, is the fastest to train with (about a few seconds on a Macbook). SVM turns out to be the most time-consuming traditional machine learning model. Indeed, by tuning the hyperparameter gamma to a certain integer, the model spends about 5 hours training itself on a Macbook. Other models' training time varies from less than a minute to a few minutes, and are in general acceptable. Different from all the others, LSTM requires a tremendously longer time to train on a normal computer, so we use the GPU of Google Colab to do the training work. It only takes a few seconds to do that, but LSTM certainly had a huge tradeoff between its performance and training time.

Nevertheless, all the models here have reached a quite good result. Despite the difference in testing accuracy and training time or epoches, we can be confident to use our models to classify real-life business emails.

4.2 Misclassification Cases Analysis

In this section we will look at several typical cases that all the models classified incorrectly. This may help us understand why there are error cases in an already quite mature model.

Firstly, when the email is very short, it can confuse the machine more often. We have found several misclassified emails that only have a subject(topic) line, but have no further context. Our models tend to misclassify these normal emails as spam, and it is reasonable to say that it's because the data is too insufficient for the model to make a good decision, when there are less than five words to be filtered into the model after preprocessing. A shortage of data can easily confuse the model, as is confusing humans.

Another case is that we will find some of our samples have anti-intuitive ground truth labels. An email asking users to purchase for some discount events is considered spam in a normal pattern. However, some of them in our dataset are labeled as non-spam. Thus, this should be taken as an exception case, instead of a typical error we must avoid.

Furthermore, in a very few cases our models consider a well written advertisement email as non-spam: a false negative case. The very structured sentences in those emails and the lack of typical signal words such as URL link and purchase-related phrases may contribute to this misclassification. However, nowadays spam emails are making themselves harder to be detected through similar tricks. To make sure our model can not only find out spam emails that are obvious, but also those that are less apparent, we will need

to come up with some improvements in our works, from word preprocessing to model architecture, in order to not miss these cases even though there are only a few of them.

5 CONCLUSION AND FUTURE WORK

Spam email filtering is a rather simple text classification task. Thanks to the usually obvious signal words in spam emails, all of our models have performed approvingly. However, some slight differences still exist between each model. Logistic regression performs the best in all the traditional models, and it is very fast to be trained. LSTM, on the other hand, aims for even higher accuracy, especially a better understanding of those emails that tricked the previous models. It does perform better than all the others, but since we trained this one on Google Colab's GPU, actually LSTM will take a training time much longer than all the others as well. Others are either performing not that good as these two(Naive Bayes), or take too long training time on a normal CPU (SVM).

As for the future work to improve, we plan to try with a larger and more cleaned-up dataset. As mentioned earlier, the Enron dataset we used was from a bankrupt business company twenty years ago. Today, the way people write emails and the ways spam emails are designed might be a lot different from two decades ago. To make sure our model could fit in the modern environment, it would be a good direction to further train it with more recent data. The problem here is, unlike other types of data machine learning researchers often use such as encyclopedias and images, emails can be hard to access. They are usually encrypted by the email companies, and even scientists who work for those companies cannot have full access to all the emails. A privacy issue remains unresolved here if we want to develop more advanced email-helper technologies.

Also, we want to apply more complicated training models to seek for better performance. We didn't use tf-idf here because we can already reach a satisfactory output with bag of words, and tf-idf cannot improve it too much. More recent models, however, can probably do the job. Transformers, an attention-based modern algorithm, usually considered as a more advanced alternative of LSTM, is on the top of our list. Meanwhile, Bidirectional Encoder Representations from Transformers, or BERT, will be a good choice to refresh our understanding of this task, since bidirectional encoding can usually help to make the model more robust.

Lastly, we may try to reproduce the Smart Reply system in our Gmail accounts. More specifically, what we were trying to do was to develop a "triggering" system, to decide whether a newly received email needs to be suggested with some commonly used responses according to the context. We have learned how to let the computer understand English context in its binary way, and the next challenge here would be: how to let the computer train itself, without having any ground truth labels (i.e. triggered a Smart Reply system or not)? There is literature discussing non-label classification machine learning, and semi-supervised learning is also considered a way to help. Nevertheless, no one has combined them with email classification tasks (the Google team tried a different approach, and they had a dataset tens of thousands of times larger than ours), so this will be a very new and exciting topic.

We look forward to digging deeper into this more integrated non-traditional machine learning problem.

6 GITHUB

You can find related codes and websites on the project's Github page: https://github.com/yk803/ML_Final_Project

7 REFERENCES

[1]Enron Email Dataset, original version. https://www.cs.cmu.edu/enron/
[2]Enron Email Dataset, preprocessed version. http://www2.aueb.gr/users/ion/data/enron-spam/
[3]Olah, Christopher, Understanding LSTM, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[4]Anjuli Kannan et al. Smart Reply: Automated Response Suggestion for Email, 2016. https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/45189.pdf
[5]Klimt, Bryan et al, The Enron Corpus:A New Dataset for Email Classification Research. http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.61.1645rep=rep1type=pdf
[6]Empirical Analysis on Email Classification Using the Enron Dataset, 2018. https://towardsdatascience.com/empirical-analysis-on-email-classification-using-the-enron-dataset-19054d558697
[7]Fundamentals of Bag Of Words and TF-IDF, 2019. https://medium.com/analytics-vidhya/fundamentals-of-bag-of-words-and-tf-idf-9846d301ff22