Spam or Ham? an Email Classifier

Fnu Ferry, Hoang Chung

Literature Review

Dada, E. G., et al. [1] reviews and explores approaches of creating a spam filter and various things to take into consideration when building one. This source motivated us to choose precision as an evaluation metric since it's an extremely important and relevant evaluation metric for spam classifiers.

Josh Fuchs [2] outlines the steps needed to create a spam filter for SMS messages with Naive Bayes. We found this helpful to understand how we needed to preprocess text data and represent it to be fit to be used as training data for our model.

Meruvu Likith [3] provides a dataset of 193582 labeled email samples. We used this dataset to train our models.

Jyoti Prakash Maheswari [4] discusses the reasons why dataset size matters as well as how to approach a ML problem when there is only a small amount of data available. It explained why our traditional ML models had a larger performance boost from small to medium dataset sizes, but a much smaller performance increase from medium to large dataset sizes. We used this to justify the difference in the performance increase in the discussion section.

Kavita Nagesan [5] introduces multiple text preprocessing techniques for NLP and ML applications and how they work. We were able to get a clearer picture of what techniques we could use to preprocess our text. We eventually determined that TF-IDF and stopword removal were the most suitable techniques for our use case.

The scikit-learn documentation on Naive Bayes introduces how Naive Bayes works and the assumptions it makes, along with a brief explanation of variants of Naive Bayes [6]. The article mentioned that Naive Bayes has worked well in spam classifiers, so it motivated us to choose the Gaussian and multinomial Naive Bayes as the learning algorithm for our first few models.

This Q&A thread [7] goes over various reasons why a random forest may perform poorly. We used this thread to find out that random forests don't work well with sparse inputs.

The scikit-learn documentation on SVC, a specific implementation of SVM in scikit-learn, introduces how SVM works and its limitations [8]. We learned that SVMs take a long time to train as the size of the training data increases, at a rate of at least quadratic to the dataset size.

Introduction

Email is a major method of communication especially in professional settings in today's age. Unfortunately, unscrupulous bad actors try to exploit this method of communication to take advantage of people. They send unsolicited emails to numerous users for various reasons, ranging from promotional marketing to unethical phishing attempts. This is known as spam.

Certain issues make this problem hard to solve. The English and most foreign language's vocabularies are huge. Also, those who send spam can adapt and change their tactics when they determine that their current strategy is ineffective or getting filtered out. Both of these reasons rule out rule-based systems as a viable solution. Therefore, an ML system that can adapt and capture complex patterns in the email text is required to solve this.

Being able to filter out spam before they reach the user's inbox saves the person's time and protects them from harmful phishing and more malicious spam. Most, if not all, email services today have a spam filter that keeps your inbox clean and safe. That shows how important it is to solve this problem. Before spam filters, a user couldn't be sure if an email that they received was trustworthy or not. At best, they would waste their time by having to go through spam emails. At worst, they could unwittingly give bad actors unfettered access to their accounts, or even their company network if they were on a work computer. The latter may require more than just a spam filter to solve, but a filter does catch cruder social engineering attempts.

We also extended our problem to consider cases where there was limited training data available. It may not always be possible to get large amounts of high-quality, relevant data.

Initial Goals

Originally, we aimed to create a model that could retrieve the title, author, and genre from a book cover with a combination of OCR and Computer vision. However, we could not achieve satisfactory results. Our main challenge involved parsing title and author information from the diverse book cover designs due to varying text placement and irrelevant text like NY bestseller tags. We determined that this would require us to develop an additional NLP model to parse the text.

Achieving our initial goals would require more time than we anticipated. In the end, we decided that it would take too much time, considering our other commitments. As such, we switched directions, deciding to create an email spam classifier.

Solution

To protect email users from spam, we trained and developed an email spam classifier that determines whether an email is spam or ham/legitimate based on the body of the email. Also, we also wanted to explore situations where we had varying amounts of data. We wanted to find which ML algorithm would perform best depending on the amount of training data it received. With this in mind, we trained multiple models, each using a different ML algorithm and dataset size, and determined the ML algorithm that led to the best performing model for each of the dataset sizes we tested.

The major advantage of using ML techniques to develop our email spam classifier is that it's practical to develop. As mentioned in the introduction, the vocabularies of languages are very expansive. Our ML-based email spam classifier is able to quickly determine the correlation between words and the probability of an email being spam for many words in the vocabulary. These learned 'rules' or relations could then be used to determine if an email is spam.

A disadvantage of our email spam classifier has more to do with how we designed it. We trained our models with offline learning. This means our solution is not adaptable. Spammers can adapt and change the words they use in their spam emails to bypass our classifier. The problem then is that our model's performance will continue to deteriorate as the true relationships between words and the probability of an email to be spam deviates from that of the dataset that we trained the model on. Because our implementation doesn't automatically train our models with these new spam emails, we have to train new models on an updated dataset of emails frequently to sustain performance.

A limitation with our solution, like virtually all ML systems, is the possibility of misclassifications. All the models we trained don't perfectly classify all emails. This means that our model can sometimes flag legitimate emails as spam (false positives) or mistakenly let some spam through (false negative).

Implementation

To demonstrate our models, we created a web app where the user can select a specific model and its variant, which refers to what dataset size it was trained on, before inputting some text as the email body. The user can then click the 'predict' button to receive the model's prediction of the email as either spam or ham.

Methodology

Our project uses the email dataset from Meruvu Likith [3], which includes 193582 labeled email (ham/spam) samples. We decided on 4 datasets of different sizes, which were sampled from the dataset from Meruvu Likith [3]. The naming of these sizes is relative to the email dataset. For each dataset size, we trained several binary classifier models.

The Sizes:
Small: ~500 training samples
Medium: ~5000 training samples
Large: ~50000 training samples
Extreme: ~155000 training samples

The Models:
Logistic Regression
Multinomial & Bernoulli Naive Bayes
Random Forest
SVM (Trained on only small & medium)

We preprocessed the data with pandas, setting “spam” as 1 (positive) and “ham” as 0 (negative). For more advanced preprocessing, training, and evaluation, we used scikit-learn. We used their TfidfVectorizer to remove stop words and vectorize the email text. For training, we did hyperparameter tuning with scikit-learn’s GridSearchCV to train and choose good hyperparameters for our models based on five-fold cross-validation accuracy.

Finally, we evaluated the models based on accuracy and precision. Precision is especially important as Dada, E. G., et al. [1] claims that flagging a legitimate email as spam (false positive) is much more costly than letting some spam through (false negative).

Results

	Small Sample		Medium Sample		Big Sample		Extreme	
Model	*Accuracy*	*Precision*	*Accuracy*	*Precision*	*Accuracy*	*Precision*	*Accuracy*	*Precision*
Logistic Regression	88.89%	86.79%	96.18%	96.17%	98.19%	97.74%	99.20%	99.16%
Multinomial Naive Bayes	92.86%	100.00%	94.91%	98.04%	97.79%	99.18%	98.44%	99.39%
Bernoulli Naive Bayes	81.75%	72.06%	89.18%	84.49%	91.19%	85.08%	92.60%	87.08%
Random Forest	84.13%	78.95%	95.07%	95.01%	94.95%	94.92%	94.19%	94.17%
SVM	89.68%	86.00%	92.00%	90.00%				

Figure 1. Accuracy and Precision values for all models at all 4 training sample sizes. Green cells are the models with the highest precision for each sample.

As can be seen in Figure 1, Multinomial Naive Bayes performed extremely well with the highest precision when given the small, medium, large, and extreme datasets. Its accuracy steadily rose from 92.86% to 98.44% as the training size increased from small to extreme. This rising trend of accuracy was consistent in all other models except random forest.

For random forest, the medium-variation accuracy and precision were 95.07% and 95.01%, respectively. These are 10.94% and 16.06% higher than the small variations accuracy (84.13%) and precision (78.95%), respectively. However, the large size and extreme size random forest models performed slightly worse in accuracy and precision (~1%) than their medium variants.

For all models except SVM, they saw a huge improvement (>10%) in accuracy and precision when they received a medium dataset rather than a small dataset. However, when trained on a large or extreme dataset, the models only performed marginally better than their medium variations.

Discussion

We determined that the multinomial Naive Bayes was the best model for all 4 sizes. The multinomial Naive Bayes had the highest precision for all 4 sizes. When compared to the Bernoulli Naive Bayes, the major difference in performance is likely due to their different assumptions.

The scikit-learn documentation on Naive Bayes [6] states that Bernoulli Naive Bayes assumes the data has a multivariate Bernoulli distribution, while a multinomial Naive Bayes assumes the data has a multinomial distribution. Our TF-IDF vectorized data has a more multinomial distribution than a multivariate Bernoulli distribution, so the multinomial Naive Bayes likely performed better than its Bernoulli counterpart since the data fit the former's assumptions more.

What we found surprising was how random forest didn't outperform all the other models when given the large and extreme datasets. It had a respectable accuracy of 94.19% and a precision of 94.17% with the extreme dataset, but these metrics were overshadowed by both logistic regression and multinomial Naive Bayes. Random forest's lackluster performance could be due to the sparsity of the TF-IDF vectorized data. Sycorax's answer in [7] notes that random forests struggle when the data is very sparse since the limited non-zero values make it difficult to find any productive split for its decision tree children to make.

SVM performed well with a small amount of data, but not much better when given a medium amount of data. We attempted to train SVM with the large and even extreme datasets, but we found that they took too long to train. The training of the large variant of the SVM wasn't completed by 14 hours, which is when we decided to abort it. The scikit-learn documentation on SVC, the SVM implementation we used, does warn that it becomes impractical when training tens of thousands of samples (The large dataset includes 50000 training samples) [8]. We determined that it was unnecessary to train SVM with large and extreme datasets since it was impractical and was outperformed by other models when trained on small and medium datasets.

It was surprising to see how the performance of all our models increased a lot less when we increased training data from small to medium, but not as much when we

increased it from medium to large or even extreme. However, this can be explained by how traditional ML algorithms, which all our models fall under, have a logarithmic relationship between model performance and dataset size [4]. This means as we kept providing the model more training data, the rate of improvements in performance decreased.

Reflections

We considered what we could've done better for our project. A list of what we found could have improved our results are:

- Wider Grid Search for random forest. Bad hyperparameters might've limited it.
- Bag of words instead of TF-IDF vectorizer for Bernoulli NB. Data should be more like a multivariate Bernoulli distribution.
- Train a deep learning model (i.e. transformer). It would've been interesting to see how its performance changes depending on the size of the dataset.
- Confidence intervals for our metrics. This would give us more information about how reliable our numbers are, especially for the 100% precision we got for Multinomial Naive Bayes - small.

Conclusion

All in all, while we were unable to achieve our initial project idea, we were able to solve what we could consider a more serious problem. We found multinomial Naive Bayes to perform the best for all 4 dataset sizes, achieving >90% accuracy and precision in all cases and even a near 100% precision rate and accuracy rate in the extreme case.

Individual Contributions

Fnu Ferry: I wrote the code for training, testing, and saving the random forest and SVM models. I created the web app demo. I wrote this report and the text on the poster.

Hoang Chung: I wrote the code for training, testing, and saving the Logistic Regression, Multinomial Naive Bayes, and Bernoulli Naive Bayes models. I also wrote the code that loads and preprocesses the data. I trained all the models and recorded their accuracy and precision metrics. I designed and organized the physical poster.

Bibliography

1. Dada, E. G., Bassi, J. S., Chiroma, H., Abdulhamid, S. M., Adetunmbi, A. O., & Ajibuwa, O. E. (2019). Machine learning for email spam filtering: Review, approaches and open research problems. Heliyon, 5(6). https://doi.org/10.1016/j.heliyon.2019.e01802
2. Fuchs, J. (2022, August 15). *Building a spam filter with naive Bayes*. Josh Fuchs. https://joshfuchs.github.io/2022/08/15/naive-bayes-spam-filter.html
3. Likith, M. (2024, March 14). 190K+ SPAM: Ham Email dataset for Classification. Kaggle. https://www.kaggle.com/datasets/meruvulikith/190k-spam-ham-email-dataset-for-classification
4. Maheswari, J. P. (2019, April 23). *Breaking the curse of small datasets in Machine Learning: Part 1*. Medium. https://towardsdatascience.com/breaking-the-curse-of-small-datasets-in-machine-learning-part-1-36f28b0c044d?gi=cdfb26d08b6a
5. Nagesan, K. (n.d.). *All you need to know about text preprocessing for NLP and Machine Learning*. KDnuggets. https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html
6. Naive Bayes. scikit-learn. (n.d.-a). https://scikit-learn.org/stable/modules/naive_bayes.html
7. When to avoid random forest?. Cross Validated. (2014, August 16). https://stats.stackexchange.com/questions/112148/when-to-avoid-random-forest
8. Sklearn.svm.SVC. scikit-learn. (n.d.-b). https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html