Syracuse University – IST 664

Final Project

Detection of SPAM in email

Katie Haugh & Toby Anderson

December 21, 2021

Introduction

Emails are an essential communication medium throughout the business world. However, according to Campaign Monitor, a popular email marketing platform, the average worker receives 120 business emails each day, of which they only respond to about 25%. The low open and response rates result from the rise of spam emails. Every day, the four billion email users worldwide are sent 60 billion spam emails. As a result, each individual's inbox fills up with these unwanted communications, creating a significant pain point for most email users.

For this project, we were provided a set of known spam emails and a set of regular (ham) emails from an energy company known as Enron and challenged to create a spam detection model using various text processing and features. This report details the text processing, features, and experiments that lead us to our final model.

Text Processing

The data includes over 3,600 ham emails from the Enron corporation spanning from 1999 to 2002, and 1500 spam emails collected from other sources spanning from 2003 to 2005. In order to showcase Natural Language Processing techniques, date and time features were not used. Instead, all features were taken from the body of the emails.

Example of ham email:

```
Subject: unify close schedule
the following is the close schedule for this coming month ( year - end . ) please
keep in the mind the following key times . . . .
unify to sitara bridge back 1 : 45 p . m . thursday , dec 30 th ( all errors must be
clear by this time )
mass draft at 6 p . m . thursday evening , dec 30 th .
accrual process begins friday morning , dec 31 st at 6 : 30 a . m . ( if your group
impacts the accrual , please ensure that the necessary people are available
for support if needed , as this is an enron holiday . )
please feel free to contact me should you have any questions .
thank you , melissa x 35615
```

Example of spam email:

```
Subject: re [ 8 ] : dear friend -
size = 1 > order confirmation . your order should be shipped by january , via fedex .
your federal express tracking number is 45954036 .
thank you for registering . your userid is : 56075519
learn to make a fortune with ebay !
complete turnkey system software - videos - turorials
clk here for information
clilings .
```

The emails and features were structured as pandas dataframes using the NLTK PlaintextCorpusReader, as opposed to nested list structures used in the standard NLTK Bayesian classifier. Not only do dataframes facilitate easier introspection and combination of features, it also more easily transfers to other modern classification models and data processing techniques, such as the use of sklearn pipelines, vectorizers, and advanced models.
The email column contains the file's name, the body has the file's contents, and the spam column indicates whether the email was spam or not, which we used to test the model's performance.

Example of nested list structure:

```
[({'last_letter': 'a', 'suffix(2)': 'i', 'vowels': 5, 'first': 'g'}, 'female'),
 ({'last_letter': 'a', 'suffix(2)': 'y', 'vowels': 3, 'first': 'k'}, 'female'),
 ({'last_letter': 'h', 'suffix(2)': 't', 'vowels': 2, 'first': 'g'}, 'female'),
 ({'last_letter': 'a', 'suffix(2)': 'l', 'vowels': 2, 'first': 'z'}, 'female'),
 ({'last_letter': 'a', 'suffix(2)': 'n', 'vowels': 3, 'first': 'r'}, 'female'),
 ({'last_letter': 'a', 'suffix(2)': 'n', 'vowels': 4, 'first': 'b'}, 'female'),
 ({'last_letter': 'k', 'suffix(2)': 'c', 'vowels': 1, 'first': 'n'}, 'male'),
 ({'last_letter': 'a', 'suffix(2)': 'l', 'vowels': 3, 'first': 'm'}, 'female'),
 ({'last_letter': 'd', 'suffix(2)': 'a', 'vowels': 1, 'first': 'c'}, 'male'),
```

Example of dataframe structure:

	email	body	spam	date
0	0006.2003-12-18.GP.spam.txt	Subject: dobmeos with hgh my energy level has ...	1	2003-12-18
1	0008.2003-12-18.GP.spam.txt	Subject: your prescription is ready . . oxwq s...	1	2003-12-18
2	0017.2003-12-18.GP.spam.txt	Subject: get that new car 8434\r\npeople nowth...	1	2003-12-18
3	0018.2003-12-18.GP.spam.txt	Subject: await your response\r\ndear partner ,...	1	2003-12-18
4	0026.2003-12-18.GP.spam.txt	Subject: coca cola , mbna america , nascar par...	1	2003-12-18

```
from nltk.corpus import PlaintextCorpusReader
def get_df():
    spam_corpus = PlaintextCorpusReader('/kaggle/input/ham-and-spam-emails/corpus/spam
', '.*', encoding='latin2')
    body = [spam_corpus.raw(fileids=[f]) for f in spam_corpus.fileids()]
    spam = [1 for f in spam_corpus.fileids()]
    name = spam_corpus.fileids()
    spam_df = pd.DataFrame(
        {'email': name,
        'body': body,
        'spam': spam})

    ham_corpus = PlaintextCorpusReader('/kaggle/input/ham-and-spam-emails/corpus/ham',
'.*', encoding='latin2')
    body = [ham_corpus.raw(fileids=[f]) for f in ham_corpus.fileids()]
    spam = [0 for f in ham_corpus.fileids()]
    name = ham_corpus.fileids()
    ham_df = pd.DataFrame(
        {'email': name,
        'body': body,
        'spam': spam})
    df = pd.concat([spam_df,ham_df],ignore_index=True)
    df['date'] = pd.to_datetime(df.email.str.split('.',expand=True)[1])
    return df
```

We created a model that trained on 80% of the corpus, leaving 20% for testing. A Gaussian Naive Bayes classifier was determined to be the best option, as we introduced negative values in our sentiment feature set that a multinomial naive bayes classifier could not handle. Finally, our scoring method evaluated precision, recall, f1 score, and accuracy. We produced these metrics using 10-fold cross-validation in order to improve the reproducibility of the results and give meaningful comparisons between experiments.

```
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor
e, make_scorer
from sklearn.naive_bayes import GaussianNB

class spam_predicter:
    def __init__(self, df, target='spam'):
        self.X = df.select_dtypes(['number']).drop([target], axis=1)
        self.y = df[target]
        X_train, self.X_test, y_train, self.y_test = train_test_split(self.X, self
.y, test_size=0.2, random_state=42, stratify=self.y)
        self.features = X_train.columns

        self.model = GaussianNB()
        self.model.fit(X_train, y_train)

        self.pred = self.model.predict(self.X_test)

    def score(self):
        scoring = {'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score),
            'accuracy' : make_scorer(accuracy_score)}
        results = cross_validate(self.model,self.X,self.y,cv=10,scoring=scoring)
        print('Precision:\t',round(results['test_precision'].mean(),3))
        print('Recall:\t\t',round(results['test_recall'].mean(),3))
        print('F1 Score:\t',round(results['test_f1_score'].mean(),3))
        print('Accuracy:\t',round(results['test_accuracy'].mean(),3))
```

The corpus was vectorized using two different techniques. First, we performed a binary vectorization, which creates a column for each unique word in the corpus and places a one if the word is present in the email or zero if it is not. We used a maximum of 200 features as our baseline, which returned the following scores:

Precision: 0.731
Recall: 0.968
F1 Score: 0.832
Accuracy: 0.887

Sample of Binary Vectorization:

	being	below	best	bob	but
0	0	0	0	0	1
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

We then vectorized the corpus using Term Frequency - Inverse Document Frequency (TFIDF), which is based on the bag-of-words model and evaluates the frequency of a word in the document and its importance, giving more weight to terms that are rare amongst the entire corpus. Once again, using a maximum of 200 features as our baseline, the model returned the following scores :

Precision: 0.815
Recall: 0.961
F1 Score: 0.882
Accuracy: 0.925

Sample of tf-idf vectorization:

	business	but	by	call	can
0	0	0.133619	0.0972173	0	0
1	0	0	0	0	0
2	0	0	0	0	0.232119
3	0.120215	0	0	0.0333286	0
4	0	0	0	0	0.029059

This vectorization alone increased precision by 8.4%, the F1 score by 5.0% and accuracy by 3.8%, with only a slight 0.7% decrease in recall. Therefore, moving forward, we decided to use the TFIDF model with a maximum of 200 features as our baseline.

After establishing our baseline, we evaluated four other filtering options to improve our model:

1. Number of Word Features
2. Stopwords
3. Lemmatization
4. Stemming

Number of Word Features

Our initial maximum number of word features was set to 200. However, we increased that number several times and ultimately decided on a maximum of 1000 features due to the substantial increase in the model's performance.

Precision: 0.908 (9.3% increase)
Recall: 0.956 (0.05% decrease)
F1 Score: 0.931 (4.9% increase)
Accuracy: 0.959 (3.4% increase)

Due to the impressive performance of the model with 1000 word features, we decided to incorporate this feature into all future experiments.

Stopwords

We removed the English stopwords from the corpus to filter out potential low-level information. While the removal of stopwords can often produce positive results, our model did not benefit from this type of filtering. Because all of our metrics decreased slightly and knowing our intention of running sentiment analysis on the corpus, we decided to abandon this filtering.

Precision: 0.906 (0.002% decrease)
Recall: 0.957 (0.001% increase)
F1 Score: 0.930 (0.001% decrease)
Accuracy: 0.958 (0.001% decrease)

Lemmatization

Next, we used NLTK's WordNetLemmatizer to normalize the words in the data set in hopes of an increase in our model's performance. Lemmatization has slightly increased our model's performance.

Precision: 0.915 (0.07% increase)
Recall: 0.957 (0.001% increase)
F1 Score: 0.935 (0.004% increase)
Accuracy: 0.961 (0.002% increase)

Sample of Lemmatized words:

	c	med	requirement	ee	'	title	[	member	statement	channel
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.0673342	0	0	0	0.0811855	0

Stemming

For our last text processing technique, we normalized the corpus this time by stemming using NLTK's PorterStemmer. We wanted to compare the results of our lemmatization approach to that of stemming to decide if we should implement one over the other. Stemming, unfortunately, produced negative results lowering all the metrics in our model.

Precision: 0.907 (0.001% decrease)
Recall: 0.952 (0.004% decrease)
F1 Score: 0.928 (0.003% decrease)
Accuracy: 0.957 (0.002% decrease)

Sample of Stemmed words:

	particip	probabl	remov	ee	morn	member	statement	possibl	intern	deliveri
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0.0297511	0	0
4	0	0	0	0	0	0	0.0805143	0	0.0524404	0

Feature Engineering

To improve our predictive model further, we explored various natural language processing techniques to incorporate. These features range from fundamental document features and n-grams to more complex features, including sentiment analysis and word embeddings. We tested these features independently against our baseline model (TFIDF vectorization, 1000 word features).

Document Features

We considered the following document features:

1. Number of Characters
2. Number of Words
3. Average Word Length
4. Word Diversity

Sample of Document features:

	num_words	word_diversity	num_chars	word_len
0	0.0198623	0.662921	0.0316308	0.422236
1	0.00970545	0.727273	0.0140478	0.371895
2	0.00361133	0.911765	0.00505473	0.355335
3	0.0661325	0.462585	0.0993891	0.391138
4	0.0841891	0.378342	0.12702	0.393172

Each of these features was checked against the model independently and then together. While the number of characters and number of words produced higher recall, average word length and word diversity were superior in precision, F1 scores, and accuracy. With the document features combined, recall was maximized at 98.6%, while all other metrics dropped compared to the baseline. Precision experienced the most significant drop. Overall, the F1 score did not improve with any combination of document features, which means that any gain in Precision or Recall was offset by a loss in another metric. This means that together the features did not provide a meaningful improvement to the model.

	Baseline	Number of Characters	Number of Words	Average Word Length	Word Diversity	All Document Features
Precision	0.908	0.779	0.843	0.912	0.909	0.781
Recall	0.956	0.984	0.969	0.952	0.956	0.986
F1 Score	0.931	0.867	0.901	0.931	0.931	0.869
Accuracy	0.959	0.91	0.937	0.954	0.959	0.911

Bi-Grams and Tri-Grams

We looked further into the statistical analysis of text with the introduction of bigrams and trigrams as features. However, both bigrams and trigrams increased recall, but failed to level up on precision, F1 score, and accuracy. Because of this, we opted not to experiment further with ngrams.

	Baseline	Bigrams	Trigrams
Precision	0.908	0.868	0.861
Recall	0.956	0.962	0.971
F1 Score	0.931	0.912	0.912
Accuracy	0.959	0.946	0.946

Example of Bigram features:

	susan smith	click here	month of	of their	like to
0	0	0	0	0	0
1	0	0.151629	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Part-of-Speech Tags

We analyzed the effectiveness of part of speech tags in our model with two different approaches. The first approach uses NLTK's pos_tag to count each sentence's part-of-speech tags. Without normalizing, these tags produced subpar results. After normalizing, we expected to see the scores rise. While recall increased to 97%, all other metrics dramatically decreased. This means that the POS features allowed the model to become more conservative, flagging more emails as spam, while not improving the overall performance of the model. We decided to take part-of-speech tags one step further and look at ratios amongst different labels. We included noun-to-verb, noun-to-adjective, and proper-noun-to-noun ratios. Our reasoning for this approach stemmed from our assumption that spammers are not likely to use proper nouns, so incorporating them as a feature might increase the model's performance. This experiment produced slightly better results than our previous normalized experiment but did not reach our baseline.

	Baseline	Part of Speech Tags	Normalized Part-of-Speech Tags	Part-of-Speech Tag Ratios
Precision	0.908	0.89	0.843	0.844
Recall	0.956	0.953	0.97	0.971
F1 Score	0.931	0.92	0.901	0.902
Accuracy	0.959	0.951	0.937	0.938

Example of POS features:

	pos_uh	pos_rp	pos_ex	pos_jjr	pos_vbp
0	0	0.0397411	0	0.127296	0.0842024
1	0	0.0889311	0	0.0949529	0.0942124
2	0	0	0	0	0.322135
3	0	0.0661127	0	0.0282358	0.0770428
4	0	0.00873068	0	0	0.0508704

Sentiment Analysis

Sentiment analysis involves the detection of attitudes in the text. It can often be an essential indicator for spam detectors. We evaluated two different types of sentiment analysis. The first approach involves NLTK's vader_lexicon, which looks at the positive, negative, and neutral scores, along with a compound metric. The compound is a score from -1 to 1 that is calculated by normalizing the other three lexicon ratings. We discovered that the compound metric had the highest impact through our analysis. The other sentiment analysis approach uses Textblob, which uses subjectivity and polarity to classify the emails. The Textblob method has zero influence on our model's performance. NLTK 's Vader method increased our baseline model's precision and F1 score while slightly sacrificing recall and accuracy remaining stable.

	Baseline	Sentiment - Vader	Sentiment - Textblob
Precision	0.908	0.912	0.908
Recall	0.956	0.954	0.956
F1 Score	0.931	0.932	0.931
Accuracy	0.959	0.959	0.931

Sample of Sentiment features:

	pos	neg	compound	neu
0	0.1448	0.0194	0.33922	0.8358
1	0.116167	0.0216667	0.108633	0.362167
2	0	0	0	1
3	0.153789	0.009	0.474495	0.784632
4	0.0620645	0.0323548	0.0941968	0.905548

Word Embeddings

Word embeddings are a numerical vector representation of words. These vectors are created by comparing word contexts using large corpuses and are intended to capture the relative contextual meaning and function of the word. Words that have similar contexts have similar embeddings. By including these embeddings in our data, we can potentially capture patterns of similar meaning, regardless of specific word usage.

We used the spaCy pretrained "en_core_web_sm" model to extract word embeddings from each email. Each word provides a 96 length vector. We then averaged these vectors together to produce the document embedding.

Example of Word Embedding Features:

	vec_49	vec_94	vec_95	vec_74	vec_5
0	0.205374	0.741384	0.526765	0.452555	0.46697
1	0.234339	0.710045	0.501758	0.486786	0.501362
2	0.352095	0.570671	0.299355	0.439936	0.639876
3	0.238217	0.678228	0.38863	0.440115	0.57232
4	0.218231	0.666257	0.564642	0.525092	0.435365

Experiments

We ran a total of nine experiments. In the first three experiments, we combined the most impactful features demonstrated in the sections above. In the last six experiments, we combined advanced features, including word embeddings, parts-of-speech, and using an XGBoost model.

Summary of experiment results:

Experiment	# words	doc_features	lemmatization	sentiment	embeddings	POS	Model	Accuracy	Precision	Recall	F1 Score
Baseline	200	No	No	No	No	No	GaussianNB	0.921	0.815	0.961	0.882
Basic 1	1000	No	Yes	No	No	No	GaussianNB	0.961	0.915	0.957	0.935
Basic 2	200	Yes	No	Yes	No	No	GaussianNB	0.925	0.815	0.961	0.882
Basic 3	1000	Yes	Yes	Yes	No	No	GaussianNB	0.961	0.915	0.956	0.935
Adv 1	200	No	No	No	Yes	Yes	GaussianNB	0.924	0.813	0.96	0.88
Adv 2	200	No	No	No	No	No	XGBoost	0.956	0.901	0.953	0.926
Adv 3	1000	Yes	Yes	Yes	No	No	XGBoost	0.974	0.942	0.969	0.955
Adv 4	200	Yes	Yes	Yes	Yes	Yes	XGBoost	0.971	0.939	0.964	0.951
Adv 5	1000	Yes	Yes	Yes	Yes	Yes	GaussianNB	0.961	0.915	0.957	0.935
Adv 6	1000	Yes	Yes	Yes	Yes	Yes	XGBoost	0.973	0.949	0.959	0.954

Overall, we note that the experiments provided only marginal gains or losses to recall, however, with the large improvements to Precision, the F1 scores improved. When comparing basic experiments, we find that the majority of the improvement occurs from increasing vocabulary size from 200 to 1000. When using any combination of features with a small vocab size of 200, F1 score does not change.

Additionally, when comparing advanced experiments, we find that the majority of the improvement involves switching to the XGBoost model. For instance, there are large improvements to precision and accuracy between the “Baseline” and “Adv 2”, but the only change is in the model. The word embeddings and POS features provide no benefit when combined with all other features, as demonstrated between experiment “Adv 3” and “Adv 6”. Combining these observations together, using 1000 words and XGBoost provides the best model results, as shown in “Adv 3” and “Adv 6”.

The reason we chose to experiment with XGBoost on this dataset, is that a large number of features can be prone to overfitting, which prevents the generalization of the model and the transfer of accuracy between training and test data. XGBoost employs several forms of regularization, which penalizes model complexity and thus mitigates overfitting. This regularization clearly improved model performance on the test results during cross validation, and we would expect the model to perform better in production, on unseen data.

Conclusion

With spam emails rising in popularity, the market for high-performing spam filters is becoming more and more competitive. So, we start to ask ourselves what determines a good filter. Is it those with higher recall, which would reduce the number of false-negative emails, or higher precision, decreasing the number of false-positive emails? Of course, we always want the end-user to receive all of their important emails, making precision the most significant metric and what we focused on when selecting our final model.

We narrowed down our spam filter to two similar models from our experiments. The first model is from our third advanced experiment, which utilizes 1,000-word vectors, document features, lemmatization, and sentiment analysis using XGBoost as our classifier. The performance of that model is:

Precision: 0.942
Recall: 0.969
F1 Score: 0.955
Accuracy: 0.974

The second model is from our sixth advanced experiment, which utilizes 1,000-word vectors, document features, lemmatization, word-sentiment, word-embeddings, and part-of-speech tags, also using XGBoost as our classifier. The performance of that model is:

Precision: 0.949
Recall: 0.959
F1 Score: 0.954
Accuracy: 0.973

The addition of word-embeddings and part-of-speech tags increased precision by 0.7%, while recall took a 1% dip, which dropped our F-score and accuracy slightly. We decided that the extra time cost and the dip in recall, F1 score, and accuracy offset the precision advantage. Therefore, our final model is made up of

1,000 word vectors
Document Features
Lemmatization
Sentiment Analysis
XGBoost Classifier

With this model, we would expect 5.1% of ham emails to be incorrectly identified as spam and 4.1% of spam emails incorrectly identified as ham.

While comparing those numbers to Gmail's spam filter, we struggled to find comparative metrics. For example, Gmail claims their spam filter is 99.9% accurate but doesn't record precision and recall. Because they use billions of emails as training data and rely on users to "Mark as Spam" for those spam emails that passed through, they have extensive training data, which would explain their spectacular results.

Consequently, Gmail's filter would result in only a single Ham email being filtered out of 1000 Ham emails. With our model's Precision performance, over 1 out of 20 Ham emails would be incorrectly filtered into a Spam folder. This may be decent precision for the scope of this report, having a very limited set of training data (around 3000 points), however, in practice, losing 1 out of 20 emails would not be acceptable.

Our experimental findings indicate that a production-worthy spam filter could be created if we could train on data from an entire email server, use a large vocabulary, and utilize a method to prevent over-training, such as regularization.

Task Attribution

Task	Contributor
Preprocessing	Toby Anderson
Ngrams	Toby Anderson
Stop Words	Toby Anderson
Lemmatization	Toby Anderson
Stemming	Toby Anderson
Word Embeddings	Toby Anderson
Doc Features	Katie Haugh
POS Features	Katie Haugh
Sentiment	Katie Haugh
Experiments	Toby Anderson
Text Processing analysis	Katie Haugh
Feature analysis	Katie Haugh
Experiment analysis	Toby Anderson
Conclusion	Katie Haugh