IST644 Final Project - Spam Email Classification

Suihin Wong and Chao Zhang

Introduction

Email has become one of the most effective ways of communication. There are around 3.9 billion active email users globally and spam email raises a huge problem. Spam email is unsolicited email which is usually for business and commercial purposes. However, some of these might be scam emails. In this report, it would include different methods on pre-processing and compare Naive Bayes, svm and MNB model for the best performance.

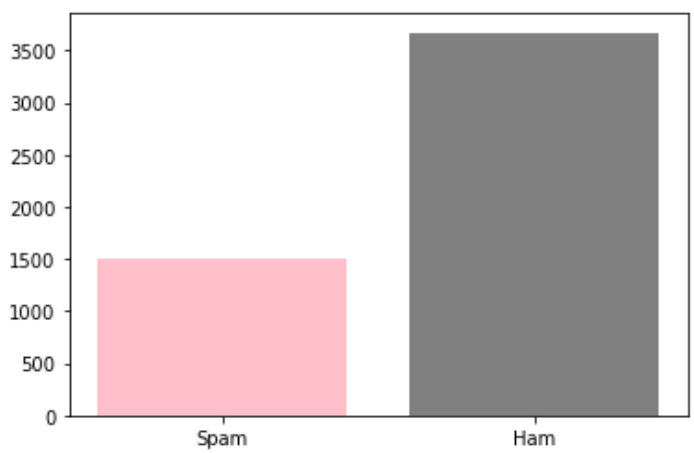
Data

The spam email data set includes 5172 data, 1500 are spam emails and 3672 are ham emails. As shown in the graph, the data set is unbalanced.

```
file_path = 'corpus'
processspamham(file_path)
```

```
Number of spam files:  1500
Number of ham files:  3672
```

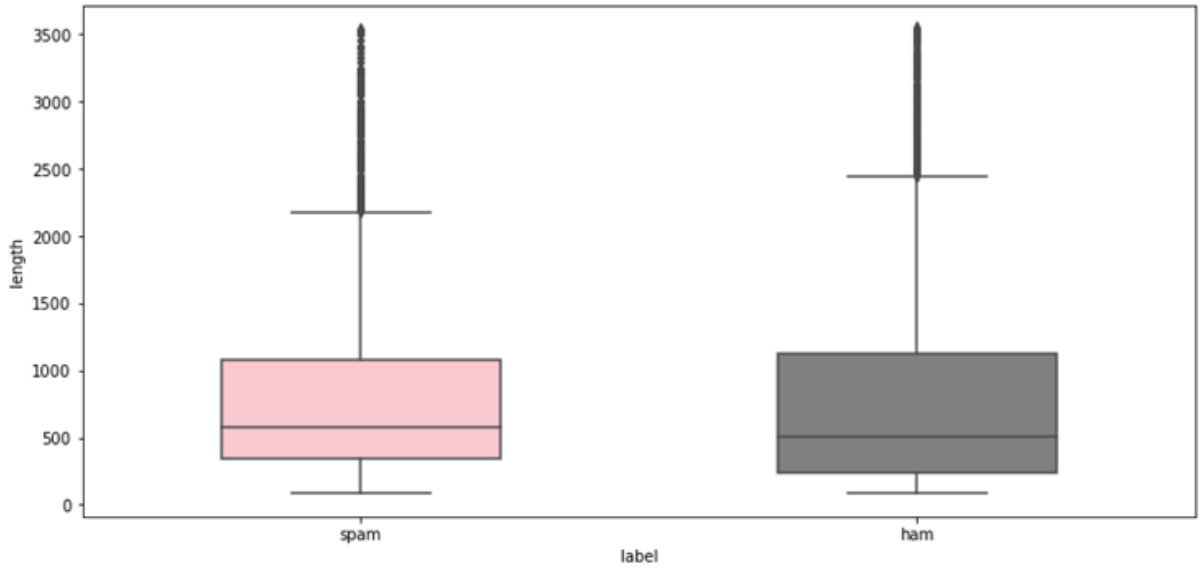
```
[18]: ############# Data exploration
import matplotlib.pyplot as plt
import seaborn as sns

# bar chart to show how many examples in each category
plt.bar('Spam', len(os.listdir('./spam')), color = 'pink')
plt.bar('Ham', len(os.listdir('./ham')), color = 'grey')
plt.show()
```



EDA

For more information about the email text data, below graph shows the distribution of the length for email text. The length of email ranges from 86 to 3547 with 95 percentile. In the box plot, it showed that the average length of spam emails and not spam emails were very similar. It

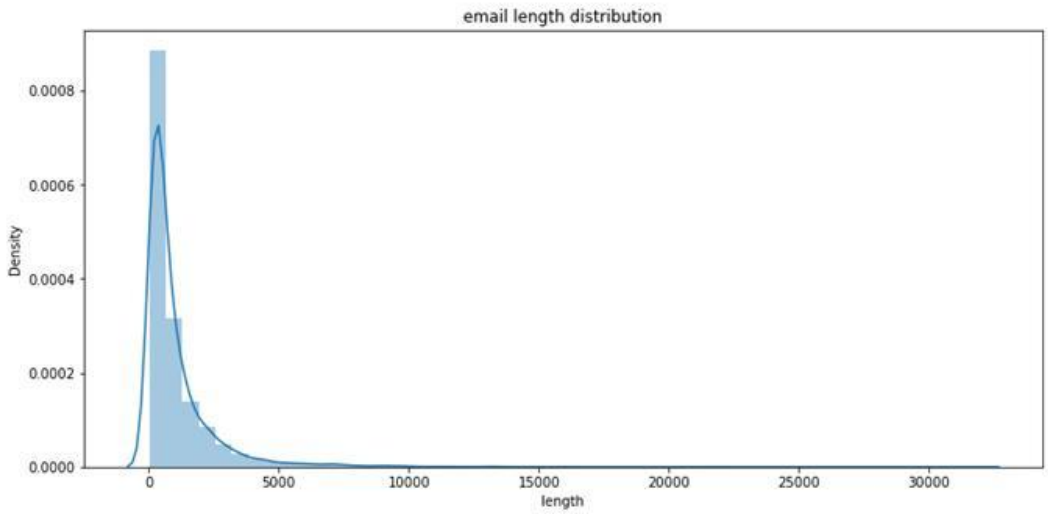
indicated that email length can not distinguish spam or ham email, so we didn't include length of email as a new feature.

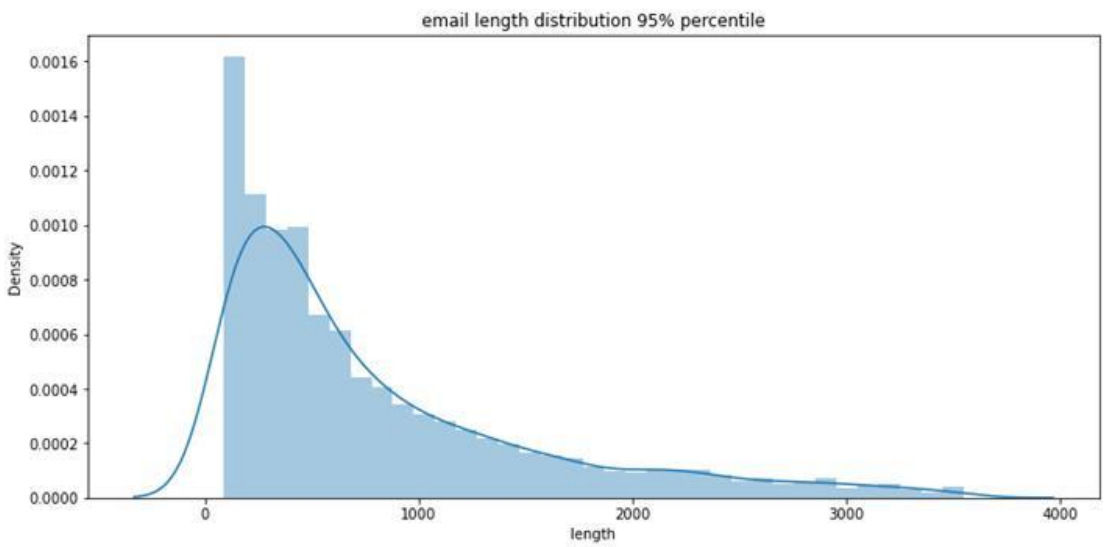


[12]:
```
# email length distribution histogram

# all emails included
plt.figure(figsize=(12.8,6))
sns.distplot(df_very_raw['length']).set_title('email length distribution')

# set up 95% percentile& 5% percentile and remove extreme value to check distribution
quantile_95 = df_very_raw['length'].quantile(0.95)
quantile_05 = df_very_raw['length'].quantile(0.05)
df_very_raw_95 = df_very_raw[df_very_raw['length'] < quantile_95]
df_very_raw_05_95 = df_very_raw_95[df_very_raw_95['length']>quantile_05]

# only the middle 90% of the data
plt.figure(figsize=(12.8,6))
sns.distplot(df_very_raw_05_95['length']).set_title('email length distribution 95% percentile')
```

```
C:\Users\guide\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated funct
ode to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for
  warnings.warn(msg, FutureWarning)
C:\Users\guide\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated funct
ode to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for
  warnings.warn(msg, FutureWarning)
```

[12]:
```
Text(0.5, 1.0, 'email length distribution 95% percentile')
```





Methods

Pre-processing

There were no null values in the dataset. There would be three different pre-processing methods to clean the data. The first one is cleaning the raw data, we removed all words which contain non-alphabetical words, removed default english stopwords and “subject” as custom stopwords, converted to all lower case letters. The second one is cleaning with stemming, and the third one is cleaning with lemmatization. The data with three different pre-processing methods will be used to fit different models to compare the performance to find the best model.

Bag of words

In the NLTK model, for three different methods of preprocessing data, the top 2000 were selected with unigram, unigram and bigram, unigram, bigram and trigram to extract the bag of words features for the model.

Result

NLTK Naive Bayes Model

Accuracy, precision, recall and F1 score would be used to evaluate the models with 10 fold cross validation. For unigram with stemming dataset, the mean accuracy score is around 91.99%, mean precision is around 85.30%, mean recall is 87.38% and mean F1 score is around 86.30%. In the lemmatization dataset, the mean accuracy score is around 93.50%, mean precision is around 86.46%, mean recall is 92.01% and mean F1 score is around 89.13%. In the dataset without lemmatization and stemming, the mean accuracy score is around 80.74%, mean precision is around 60.86%, mean recall is 93.97% and mean F1 score is around 73.81%.

```python
# randome shuffle the featureset before running the function
random.shuffle(featureset_stem)
random.shuffle(featureset_lemma)
random.shuffle(featureset_raw)

# featureset here only includes bag of word feature
cross_validation(10, featureset_stem)
cross_validation(10, featureset_lemma)
cross_validation(10, featureset_raw)
```

```
0 :  0.9032882011605415
1 :  0.9226305609284333
2 :  0.9342359767891683
3 :  0.9168278529980658
4 :  0.9032882011605415
5 :  0.9303675048355899
6 :  0.9284332688588007
7 :  0.9226305609284333
8 :  0.9187620889748549
9 :  0.9187620889748549
Mean accuracy:  0.9199226305609285
Mean precision 0.8529819509227545
Mean recall 0.8737602064360634
Mean F1 score 0.8629790137744507
        Precision       Recall          F1
0           0.802        0.887       0.843
1           0.837        0.850       0.844
2           0.865        0.878       0.871
3           0.850        0.896       0.872
4           0.832        0.832       0.832
5           0.893        0.893       0.893
6           0.856        0.875       0.865
7           0.883        0.861       0.872
8           0.865        0.865       0.865
9           0.846        0.899       0.872
```

```
0 :  0.9458413926499033
1 :  0.9226305609284333
2 :  0.9323017408123792
3 :  0.9303675048355899
4 :  0.9400386847195358
5 :  0.9264990328820116
6 :  0.9516441005802708
7 :  0.9593810444874274
8 :  0.9129593810444874
9 :  0.9284332688588007
Mean accuracy:  0.9350096711798839
Mean precision 0.864578793421454
Mean recall 0.9201376825126187
Mean F1 score 0.8912999022296765
        Precision       Recall          F1
0           0.870        0.934       0.901
1           0.831        0.901       0.865
2           0.868        0.898       0.883
3           0.844        0.925       0.882
4           0.891        0.919       0.905
5           0.860        0.894       0.877
6           0.898        0.949       0.923
7           0.892        0.972       0.931
8           0.850        0.866       0.858
9           0.842        0.943       0.890
0 :  0.8297872340425532
1 :  0.8065764023210832
2 :  0.8065764023210832
3 :  0.7988394584139265
4 :  0.8143133462282398
5 :  0.8143133462282398
6 :  0.7911025145067698
7 :  0.7794970986460348
8 :  0.8278529980657641
9 :  0.804642166344294
Mean accuracy:  0.8073500967117988
Mean precision 0.6085831576304197
Mean recall 0.9397270015412194
Mean F1 score 0.7380684806845098
        Precision       Recall          F1
0           0.649        0.955       0.773
1           0.611        0.941       0.741
2           0.615        0.914       0.735
3           0.594        0.925       0.723
4           0.638        0.932       0.758
5           0.578        0.969       0.724
6           0.597        0.949       0.733
7           0.533        0.946       0.682
8           0.641        0.927       0.757
9           0.630        0.939       0.754
```

For unigram and bigram with stemming dataset, the mean accuracy score is around 92.15%, mean precision is around 85.56%, mean recall is 87.76% and mean F1 score is around 86.62%. In the lemmatization dataset, the mean accuracy score is around 93.46%, mean precision is around 86.54%, mean recall is 91.63% and mean F1 score is around 88.99%. In the dataset without lemmatization and stemming, the mean accuracy score is around 80.81%, mean precision is around 60.94%, mean recall is 93.94% and mean F1 score is around 73.91%.

```
# run cross validation function with the bigram featureset
# random shuffle first to avoid precision/recall has division by zero error
random.shuffle(featureset_bi_stem)
random.shuffle(featureset_bi_lemma)
random.shuffle(featureset_bi_raw)

cross_validation(10, featureset_bi_stem)
cross_validation(10, featureset_bi_lemma)
cross_validation(10, featureset_bi_raw)
```

```
0 :  0.9187620889748549
1 :  0.9168278529980658
2 :  0.9206963249516441
3 :  0.9245647969052224
4 :  0.9264990328820116
5 :  0.9245647969052224
6 :  0.9245647969052224
7 :  0.9129593810444874
8 :  0.9168278529980658
9 :  0.9284332688588007
Mean accuracy:  0.9214700193423597
Mean precision 0.8556146238181531
Mean recall 0.8775875979673249
Mean F1 score 0.8662200196616429
        Precision       Recall          F1
0           0.845      0.889       0.866
1           0.829      0.871       0.849
2           0.888      0.849       0.868
3           0.851      0.901       0.875
4           0.859      0.883       0.871
5           0.846      0.887       0.866
6           0.876      0.881       0.879
7           0.830      0.880       0.854
8           0.872      0.855       0.863
9           0.861      0.879       0.870
```

```
0 :  0.9361702127659575
1 :  0.9323017408123792
2 :  0.9381044487427466
3 :  0.9168278529980658
4 :  0.9361702127659575
5 :  0.9264990328820116
6 :  0.9361702127659575
7 :  0.9361702127659575
8 :  0.9381044487427466
9 :  0.9497098646034816
Mean accuracy:  0.934622823984526
Mean precision 0.8654421779779466
Mean recall 0.916344721903978
Mean F1 score 0.8899938617479183
        Precision       Recall          F1
0           0.865      0.918       0.890
1           0.861      0.931       0.895
2           0.842      0.932       0.885
3           0.837      0.877       0.856
4           0.871      0.922       0.896
5           0.853      0.926       0.888
6           0.900      0.894       0.897
7           0.865      0.897       0.881
8           0.862      0.923       0.891
9           0.898      0.943       0.920
0 :  0.7949709864603481
1 :  0.8027079303675049
2 :  0.8278529980657641
3 :  0.7949709864603481
4 :  0.8027079303675049
5 :  0.8123791102514507
6 :  0.8007736943907157
7 :  0.8085106382978723
8 :  0.8259187620889749
9 :  0.8104448742746615
Mean accuracy:  0.8081237911025145
Mean precision 0.6093809061370573
Mean recall 0.9394126601107382
Mean F1 score 0.7390579384667564
        Precision       Recall          F1
0           0.591      0.926       0.721
1           0.631      0.940       0.755
2           0.655      0.944       0.774
3           0.587      0.939       0.723
4           0.618      0.969       0.755
5           0.604      0.973       0.745
6           0.604      0.921       0.730
7           0.590      0.905       0.715
8           0.619      0.965       0.754
9           0.594      0.913       0.720
```

For unigram, bigram and trigram with stemming dataset, the mean accuracy score is around 80.85%, mean precision is around 61.02%, mean recall is 94.27% and mean F1 score is around 74.08%. In the lemmatization dataset, the mean accuracy score is around 92.05%, mean precision is around 85.37%, mean recall is 87.58% and mean F1 score is around 86.40%. In the dataset without lemmatization and stemming, the mean accuracy score is around 93.56%, mean precision is around 86.61%, mean recall is 91.96% and mean F1 score is around 89.19%.

```
# run cross validation function with the bi/trigram featureset
# random shuffle before runt function to avoid recall/precision have divison by zero error

random.shuffle(featureset_bi_tri_raw)
random.shuffle(featureset_bi_tri_stem)
random.shuffle(featureset_bi_tri_lemma)


cross_validation(10, featureset_bi_tri_raw)
cross_validation(10, featureset_bi_tri_stem)
cross_validation(10, featureset_bi_tri_lemma)
```

```
0 :  0.7891682785299806
1 :  0.7949709864603481
2 :  0.8239845261121856
3 :  0.8297872340425532
4 :  0.7988394584139265
5 :  0.8181818181818182
6 :  0.8104448742746615
7 :  0.793036750483559
8 :  0.8297872340425532
9 :  0.7969052224371374
Mean accuracy:  0.8085106382978722
Mean precision 0.6102455814064465
Mean recall 0.9427155744090452
Mean F1 score 0.7407584186108964
        Precision       Recall          F1
0           0.596        0.949       0.732
1           0.603        0.929       0.731
2           0.635        0.961       0.765
3           0.626        0.944       0.753
4           0.597        0.912       0.722
5           0.598        0.941       0.731
6           0.615        0.974       0.754
7           0.582        0.945       0.721
8           0.632        0.952       0.760
9           0.618        0.920       0.739
```

```
0 :  0.9226305609284333
1 :  0.9168278529980658
2 :  0.9110251450676983
3 :  0.9226305609284333
4 :  0.9110251450676983
5 :  0.9245647969052224
6 :  0.9110251450676983
7 :  0.9264990328820116
8 :  0.9226305609284333
9 :  0.9361702127659575
Mean accuracy:  0.9205029013539654
Mean precision 0.8536871329669603
Mean recall 0.8758361604083591
Mean F1 score 0.8639782878075657
        Precision       Recall          F1
0           0.862        0.873       0.868
1           0.835        0.895       0.864
2           0.815        0.892       0.852
3           0.872        0.848       0.860
4           0.891        0.814       0.851
5           0.850        0.890       0.870
6           0.826        0.871       0.848
7           0.830        0.914       0.870
8           0.852        0.852       0.852
9           0.904        0.909       0.907
0 :  0.9593810444874274
1 :  0.9323017408123792
2 :  0.9284332688588007
3 :  0.9400386847195358
4 :  0.9284332688588007
5 :  0.9284332688588007
6 :  0.9303675048355899
7 :  0.9439071566731141
8 :  0.9361702127659575
9 :  0.9284332688588007
Mean accuracy:  0.9355899419729206
Mean precision 0.8661362767439542
Mean recall 0.9195888712301233
Mean F1 score 0.8918816741619453
        Precision       Recall          F1
0           0.884        0.986       0.932
1           0.837        0.928       0.880
2           0.888        0.893       0.890
3           0.886        0.925       0.905
4           0.845        0.889       0.866
5           0.859        0.909       0.883
6           0.866        0.910       0.888
7           0.872        0.928       0.899
8           0.873        0.913       0.893
9           0.853        0.914       0.883
```

Vectorizer

Based on the pre-processing data from the NLTK model, we applied sklearn package and converted the data into a pandas data frame and used term frequency inverse document frequency (TFIDF) vectorizer to vectorize the data with unigram, bigram and trigram. Other parameters included min_df equals 5, use_idf equals true.

```
[38]: # create pandas data frame for word before lemmatized or stemmed
df_raw = pd.DataFrame(email_doc_word_lower_stopped, columns=["text","label"])
# convert list to str and join each word with space
df_raw["text"] = df_raw["text"].apply(lambda x: ' '.join(map(str, x)))


# create pandas data frame for stemmed word.
df_stemmed = pd.DataFrame(email_doc_word_lower_stopped_stemmed, columns=["text","label"])
# convert list to str and join each word with space
df_stemmed["text"] = df_stemmed["text"].apply(lambda x: ' '.join(map(str, x)))


# create pandas data frame for lemmatized word.
df_lemmatized = pd.DataFrame(email_doc_word_lower_stopped_lemmatized, columns=["text","label"])
# convert list to str and join each word with space
df_lemmatized["text"] = df_lemmatized["text"].apply(lambda x: ' '.join(map(str, x)))
```

```
[39]: # extract values from pandas data fram for label column and text column for data before lemmatized and stemmed
y_raw=df_raw['label'].values
X_raw=df_raw['text'].values

# extract values from pandas data fram for label column and text column for lemmatized data
y_lemma=df_lemmatized['label'].values
X_lemma=df_lemmatized['text'].values

# extract values from pandas data fram for label column and text column for stemmed data
y_stem=df_stemmed['label'].values
X_stem=df_stemmed['text'].values
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# several commonly used vectorizer setting
bitrigram_tfidf_vectorizer = TfidfVectorizer(encoding='latin-1', use_idf=True, ngram_range=(1,3), min_df=5, stop_words='english')
```

```
.]: # vectorizer fit the X training data and generate tranfored dataset together
# fit_transform calculate the μ (population avg ) and σ (population SD) and normalize data by z transformation
# transform use the previously calculated μ and σ from fit_transform and normalize a new patch of data

# TFIDF featuersets for data before stemming/lemmatization
tfidf_X_raw_vec = bitrigram_tfidf_vectorizer.fit_transform(X_raw)
print(tfidf_X_raw_vec.shape)

# TFIDF featuresets for data after lemmatization
tfidf_X_lemma_vec = bitrigram_tfidf_vectorizer.fit_transform(X_lemma)
print(tfidf_X_lemma_vec.shape)

# TFIDF featuresets for data after stemming
tfidf_X_stem_vec = bitrigram_tfidf_vectorizer.transform(X_stem)
print(tfidf_X_stem_vec.shape)

(5172, 22807)
(5172, 22565)
(5172, 22565)
```

Sk-learn multinomial Naive Bayes Model

In the Sk-learn MNB model, precision, recall and F1 score would be used to evaluate the models with 10 fold cross validation. The purpose of the model is to detect spam email, so evaluation score would be for spam. For data without stemming and lemmatization, the average accuracy is 96.31%, mean of precision is 92.53%, mean of recall is 94.93% and average F1 score is around 93.72%. For data with stemming, the average accuracy is around 95.63%, mean of precision is around 95.31%, mean of recall is 89.33% and average F1 score is around 92.22%. For data with lemmatization, the average accuracy is 96.44%, mean of precision is 92.95%, mean of recall is 94.93% and average F1 score is around 93.93%.

```
# data before stemming/lemmatization and the model CV results with TFIDF features (uni, bi, trigrams)
scores_raw = cross_val_predict(tfidf_nb_clf_1, tfidf_X_raw_vec, y_raw, cv=10)

scores_raw_report = classification_report(y_raw, scores_raw, target_names = name, digits= 4)
print(scores_raw_report)
```

```
              precision    recall  f1-score   support

         ham     0.9791    0.9687    0.9739      3672
        spam     0.9253    0.9493    0.9372      1500

    accuracy                         0.9631      5172
   macro avg     0.9522    0.9590    0.9555      5172
weighted avg     0.9635    0.9631    0.9632      5172
```

```
# data after stemming and the model CV results with TFIDF features (uni, bi, trigrams)
scores_stem = cross_val_predict(tfidf_nb_clf_1, tfidf_X_stem_vec, y_stem, cv=10)

scores_stem_report = classification_report(y_stem, scores_stem, target_names = name, digits= 4)
print(scores_stem_report)
```

```
              precision    recall  f1-score   support

         ham     0.9575    0.9820    0.9696      3672
        spam     0.9531    0.8933    0.9222      1500

    accuracy                         0.9563      5172
   macro avg     0.9553    0.9377    0.9459      5172
weighted avg     0.9562    0.9563    0.9559      5172
```

```
# data after lemmatization and the model CV results with TFIDF features (uni, bi, trigrams)
scores_lemma = cross_val_predict(tfidf_nb_clf_1, tfidf_X_lemma_vec, y_lemma, cv=10)

scores_lemma_report = classification_report(y_lemma, scores_lemma, target_names = name, digits= 4)
print(scores_lemma_report)
```

```
              precision    recall  f1-score   support

         ham     0.9791    0.9706    0.9748      3672
        spam     0.9295    0.9493    0.9393      1500

    accuracy                         0.9644      5172
   macro avg     0.9543    0.9600    0.9571      5172
weighted avg     0.9647    0.9644    0.9645      5172
```

Sk-learn SVM model

In this step, we applied the SVM model to the datasets with three different pre-processing methods as mentioned above. Besides that, we used gridsearch technique to tune the hyper-parameters of kernels and C values to find the best mode without overfitting. In the Gridsearch parameter part, we chose 1, 2, 5, 10, 25, 50, 100 for the C parameter and poly, rbf, sigmoid and linear for the Kernel with 10 fold cross validation. To determine the best SVM from Gridsearch, we used F1 score in the Gridsearch tuning because F1 score would provide a score by both precision and recall.

In three different pre-processing methods with SVM model, for the data after lowercase process and stopwords removal without stemming or lemmatization, the best parameter for C is 1, and for the kernel is linear, leading to a model with 98.84% accuracy, 97.75% precision, 98.70% recall and 98.22% F1 score to predict spam email.

```
{'C': 1, 'kernel': 'linear'}
              precision    recall  f1-score   support

         ham     0.9945    0.9904    0.9924       727
        spam     0.9775    0.9870    0.9822       308

    accuracy                         0.9894      1035
   macro avg     0.9860    0.9887    0.9873      1035
weighted avg     0.9894    0.9894    0.9894      1035
```

For the data with stemming, the best model recommended by gridsearch is with parameters for C as 1, and for the kernel as rbf. The model is with 98.26% accuracy, 95.31% precision, 99.03% recall and 97.13% F1 score to predict spam email.

```
{'C': 1, 'kernel': 'rbf'}
              precision    recall  f1-score   support

         ham     0.9958    0.9794    0.9875       727
        spam     0.9531    0.9903    0.9713       308

    accuracy                         0.9826      1035
   macro avg     0.9745    0.9848    0.9794      1035
weighted avg     0.9831    0.9826    0.9827      1035
```

For the data with lemmatization, the best parameter for C is 1, and the kernel is linear, leading to a model with 98.65% accuracy, 97.12% precision, 98.38% recall and 97.74% F1 score to predict spam email.

```
{'C': 1, 'kernel': 'linear'}
              precision    recall  f1-score   support

         ham     0.9931    0.9876    0.9903       727
        spam     0.9712    0.9838    0.9774       308

    accuracy                         0.9865      1035
   macro avg     0.9821    0.9857    0.9839      1035
weighted avg     0.9866    0.9865    0.9865      1035
```

```python
# set up tuning parameters for SVM
param_tuning = {
    "C": [1, 2, 5, 10, 25, 50, 100],
    "kernel": ["poly", "rbf", "sigmoid", "linear"]
}

# set up scoring rules
# need to specify 'spam' as postiive value in confusion matrix
scoring_rule = {"accuracy": make_scorer(accuracy_score),
                 "precision": make_scorer(precision_score, pos_label = 'spam'),
                "recall": make_scorer(recall_score, pos_label = 'spam'),
                "f1_score": make_scorer(f1_score, pos_label="spam")}
```

Conclusion

In the result, the best model is SVM with parameters of C as 1 and kernel as linear, and the email data after lowercase process and stopwords removal which provided the best result of 98.22% in F1 score, 98.70% in recall and 97.75% in precision. In evaluation, F1 score was selected with precision to evaluate the model performance. With the same feature set, Multinomial NB and SVM Model with TF-IDF vectorizer performed better than the Naive Bayes with boolean count vectorizer. TF-IDF is normalized by using term frequency divided by inverse document frequency so the TF-IDF model performed better. Moreover, SVM performs better at full length content and MNB performs better with short text which matches with our result.

With the result, different features like bag of words, unigram + bigram or unigram + bigram + trigrams didn't improve the performance of the model when using Naive Bayes algorithm.

Training F1-score and Precision with 10-folds cross validation and testing round

Model	PP V1 + F V1	PP V1 + F V2	PP V1 + F V3	PP V1 + F V4	PP V2 + F V1	PP V2 + F V2	PP V2 + F V3	PP V2 + F V4	PP V3 + F V1	PP V3 + F V2	PP V3 + F V3	PP V3 + F V4
Naïve Bayes	73.81% 60.86%	73.91% 60.94%	74.08% 61.02%		86.30% 85.30%	86.62% 85.56%	86.40% 85.39%		89.13% 86.46%	89.00% 86.54%	89.19% 86.61%	
Multi-nomial NB				93.72% 92.53%				92.22% 95.31%				93.93% 92.95%
SVM				98.22% 97.75%				97.13% 95.31%				97.74% 97.12%

However, different preprocessing methods like stopword moval, lowercase conversion, stemming or lemmatization provided a different performance.

Training F1-score and Precision with 10-folds cross validation and testing round

Model	PP V1 + F V1	PP V1 + F V2	PP V1 + F V3	PP V1 + F V4	PP V2 + F V1	PP V2 + F V2	PP V2 + F V3	PP V2 + F V4	PP V3 + F V1	PP V3 + F V2	PP V3 + F V3	PP V3 + F V4
Naïve Bayes	73.81% 60.86%	73.91% 60.94%	74.08% 61.02%		86.30% 85.30%	86.62% 85.56%	86.40% 85.39%		89.13% 86.46%	89.00% 86.54%	89.19% 86.61%	
Multi-nomial NB				93.72% 92.53%				92.22% 95.31%				93.93% 92.95%
SVM				98.22% 97.75%				97.13% 95.31%				97.74% 97.12%

Overall, lemmatization performs the best with Naive Bayes and Multinomial Naive Bayes, and stopwords removal + lowercase conversion performs the best in the SVM model.

Training F1-score and Precision with 10-folds cross validation and testing round

Model	PP V1 + F V1	PP V1 + F V2	PP V1 + F V3	PP V1 + F V4	PP V2 + F V1	PP V2 + F V2	PP V2 + F V3	PP V2 + F V4	PP V3 + F V1	PP V3 + F V2	PP V3 + F V3	PP V3 + F V4
Naïve Bayes	73.81% 60.86%	73.91% 60.94%	74.08% 61.02%		86.30% 85.30%	86.62% 85.56%	86.40% 85.39%		89.13% 86.46%	89.00% 86.54%	89.19% 86.61%	
Multi-nomial NB				93.72% 92.53%				92.22% 95.31%				93.93% 92.95%
SVM				98.22% 97.75%				97.13% 95.31%				97.74% 97.12%

In conclusion, besides f1 score, precision is used to evaluate the model over recall because we don't want the model to consider every email as spam email to maximize the precision and lower the recall. In reality, users don't want their useful email classified as spam email, because they might miss some important information. With spam email, even if it is classified as regular email into the inbox, by looking at the subject of the email, users can still be able to move spam email into junk email with a little effort.