Homework 2

Yu-Chieh Kuo B07611039

1 Overview

This assignment asks us to evaluate the classification performance of the following classification methods: Bernoulli Naïve Bayes method, SVM method with linear and RBF kernel, then show their precision, recall, and F1 scores and plot their precision recall curves on report.

2 Precision, Recall, and F1 scores

The precision, recall and F1 scores for these three method are represented as below.

```
Precision, Recall, and F1 scores are as below.
              precision    recall  f1-score   support

           1       0.40      1.00      0.57         2
           2       0.00      0.00      0.00         1
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00         2
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         1
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         2
          10       1.00      0.33      0.50         3
          11       1.00      1.00      1.00         1
          12       1.00      1.00      1.00         1

    accuracy                           0.85        20
   macro avg       0.87      0.86      0.84        20
weighted avg       0.89      0.85      0.83        20

F1 scores = 0.8321428571428571
Precision = 0.89
recall = 0.85
```

Figure 1: Precision, Recall, and F1 scores for Bernoulli Naïve Bayes method

```
Precision, Recall, and F1 scores are as below.
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         1
           5       1.00      1.00      1.00         2
           6       1.00      1.00      1.00         3
           8       1.00      1.00      1.00         1
           9       1.00      1.00      1.00         1
          11       1.00      1.00      1.00         1
          13       1.00      1.00      1.00         2

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

F1 scores : 1.0
Precision : 1.0
recall : 1.0
```

Figure 2: Precision, Recall, and F1 scores for SVM method with linear kernel

```
Precision, Recall, and F1 scores are as below.
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1
           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         1
           7       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         4
          11       1.00      1.00      1.00         1
          12       1.00      1.00      1.00         2
          13       1.00      1.00      1.00         3

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

F1 scores : 1.0
Precision : 1.0
recall : 1.0
```

Figure 3: Precision, Recall, and F1 scores for SVM method with RBF kernel

3 Precision Recall Curves

The precision recall curves for these three method are represented as below.

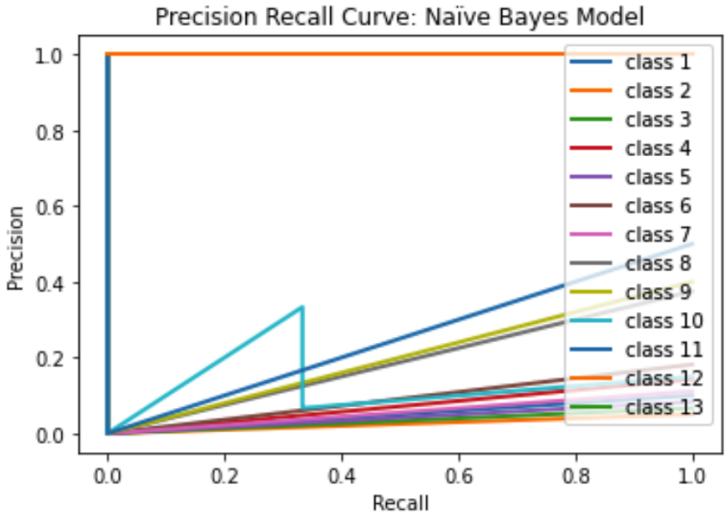

Figure 4: Precision Recall curve for Bernoulli Naïve Bayes method

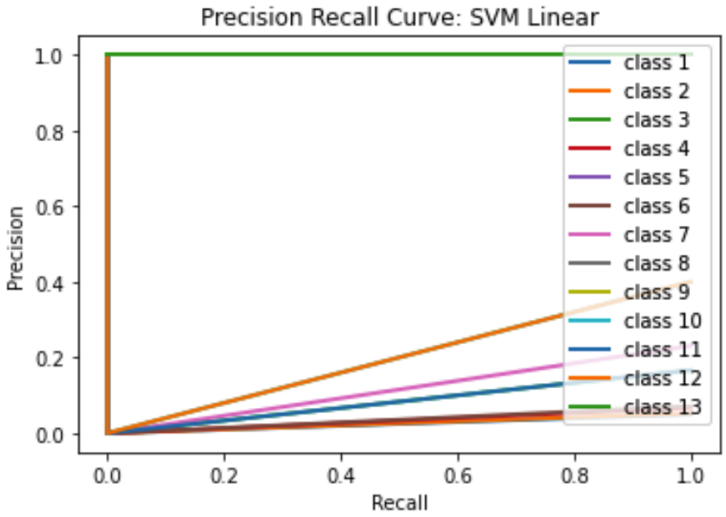

Figure 5: Precision Recall curve for SVM method with linear kernel

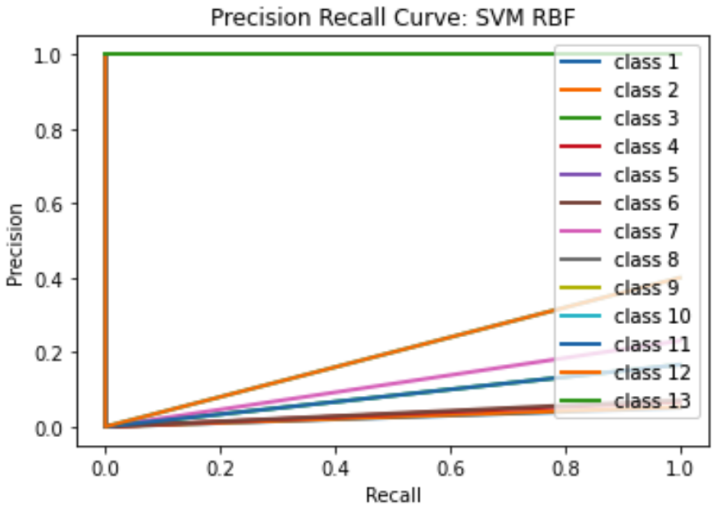

Figure 6: Precision Recall curve for SVM method with RBF kernel

4 Score on Kaggle

The submission score on Kaggle is shown as below. I get 0.98222 on this competition, ranking at 14th before the deadline.

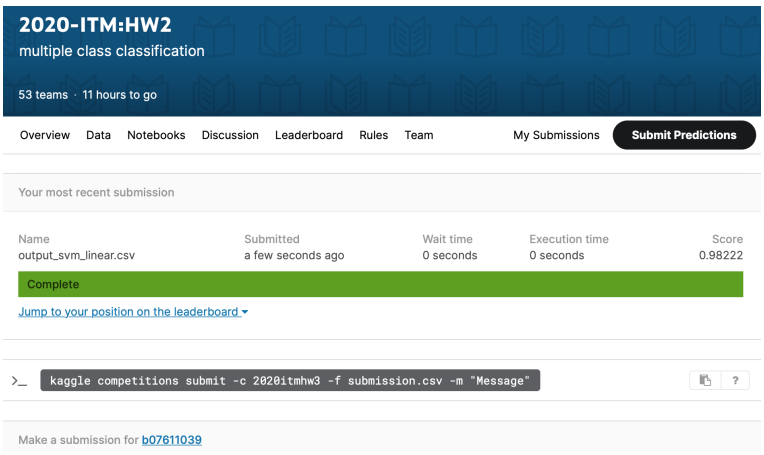

Figure 7: Score on Kaggle

5 Comparison for Classification Method

In these three method, **Linear Kernel SVM** has the best score on the Kaggle competition. For Bernoulli Naïve Bayes, this method uses multi-hot vectors to represent the documents, and the model ignores a term's frequency of existence, which makes mistakes when classifying big documents.

SVM uses a boundary to classify documents into classes. The boundary is obtained by optimizing a max/min problem where the boundary between classes is maximized to be far from each point. For the SVM method with linear kernel, the boundary is a hyperplane splitting the space into two parts, which performs well when data can be

separated linearly. But, if the data is hard to linearly separate, the data will be mapped to a higher dimensional space, which makes a linear classifier more easier to separate the data. Although costing more computation than the original linear way, a kernel function can be used to compute the dot product without mapping the data to a higher dimensional space in advance, and RBF is a well adopted kernel function.

6 Implement

6.1 Preprocess

1. Import all necessary libraries, for example, sklearn, numpy, pandas, etc.
2. Lowercase all words in docs, eliminate the stop word in English, replace EOL.
3. Separate the testing data set and the training data set.

```
# classes = the given testing data set
labels = []
for q in range(0, len(docs)):
    for i in range(0, 13):
        for j in range(0, 15):
            if q + 1 == classes[i][j]:
                labels.append([classes[i][j], i+1])
labels = pd.DataFrame(sorted(labels, key = lambda l:l[0]), columns
    = ['training_id', 'classes'])
training_docs = docs[docs['id'].isin(labels['training_id'])]
testing_docs = docs[~docs['id'].isin(labels['training_id'])]
```

Listing 1: Preprocess

6.2 Bernoulli Naïve Bayes

```
binary_vectorizer = CountVectorizer(binary = True)
binary_vectors = binary_vectorizer.fit_transform(training_docs['text'])
binary_vectors_test = binary_vectorizer.transform(testing_docs['text'])
x_train, x_test, y_train, y_test = train_test_split(binary_vectors,
    labels['classes'], test_size = 0.1) # amazing method taught by my
    classmates
model = BernoulliNB()
model.fit(x_train, y_train)
prediction = []
expectation = []
prediction.extend(model.predict(x_test))
expectation.extend(y_test)
```

Listing 2: Bernoulli Naïve Bayes

6.3 Linear Kernel SVM

```
TFIDF_vectorizer = TfidfVectorizer(stop_words = 'english')
TFIDF_vectors_training = TFIDF_vectorizer.fit_transform(training_docs['
    text'])
TFIDF_vectors_testing  = TFIDF_vectorizer.transform(testing_docs['text'
    ])
```

```
4
5 x_train, x_test, y_train, y_test = train_test_split(
    TFIDF_vectors_training, labels['classes'], test_size = 0.1)
6 SVC_Linear_model = SVC(kernel='linear', C = 1.0)
7 SVC_Linear_model.fit(x_train, y_train)
8
9 prediction = []
10 expectation = []
11
12 prediction.extend(SVC_Linear_model.predict(x_test))
13 expectation.extend(y_test)
```

Listing 3: Linear Kernel SVM

6.4 RBF Kernel SVM

```
1 TFIDF_vectorizer = TfidfVectorizer(stop_words = 'english')
2 TFIDF_vectors_training = TFIDF_vectorizer.fit_transform(training_docs['
    text'])
3 TFIDF_vectors_testing  = TFIDF_vectorizer.transform(testing_docs['text'
    ])
4
5 x_train, x_test, y_train, y_test = train_test_split(
    TFIDF_vectors_training, labels['classes'], test_size = 0.1)
6 SVC_RBF_model = SVC(kernel='RBF', C = 1.0)
7 SVC_RBF_model.fit(x_train, y_train)
8
9 prediction = []
10 expectation = []
11
12 prediction.extend(SVC_RBF_model.predict(x_test))
13 expectation.extend(y_test)
```

Listing 4: RBF Kernel SVM

6.5 Representation of scores

```
1 print("Precision, Recall, and F1 scores are as below.")
2 print(metrics.classification_report(expectation, prediction))
3 print("F1 scores :", metrics.f1_score(expectation, prediction, average='
    weighted'))
4 print("Precision :", metrics.precision_score(expectation, prediction,
    average='weighted'))
5 print("recall :", metrics.recall_score(expectation, prediction, average
    ='weighted'))
```

Listing 5: Representation of scores

6.6 Plotting

```
1 precision = dict()
2 recall = dict()
3 for i in range(13):
4     precision[i], recall[i], thresholds = metrics.precision_recall_curve
    (expectation, prediction, pos_label = (i + 1))
```

```
    plt.plot(recall[i], precision[i], lw = 2, label = 'class {}'.format(
    i + 1))

plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend(loc = "upper right")
plt.title("Precision Recall Curve")
plt.show()
```

Listing 6: Plotting