SVM method(以下以linear演示，後面有標註與RBF之差異)

1. 先將檔案分類的資料讀入後，把所有內容轉存為整數

```
In [1]: import numpy as np
```

```
In [2]: testTemp = [[]]
```

```
In [3]: with open('training_new.txt', mode='r', encoding='utf-8') as f:
            for i in f.readlines():
                line = i.split()
                testTemp.append(line)
```

```
In [4]: test = [[]]
```

```
In [5]: for i in range(13):
            temp = []
            for j in range(15):
                temp.append(int(float(testTemp[i + 1][j + 1])))
            test.append(temp)
        test.pop(0)
```

2. 讀入作業一中所有求出的tfidf檔案，並存入一個總存的二維ndarray中

```
In [6]: tfidfTest = np.empty([1095, 19130])
```

```
In [7]: for i in range(1095):
            file = np.loadtxt('PA1-answer/' + str(i + 1) + '.vec')
            tfidfTest[i] = file
```

3. 將所有tfidf檔案按照是否為訓練檔做分裝，同時以testLabel存取

```
In [8]: x_train = np.empty([195, 19130])
        x_test = np.empty([900, 19130])
        testLabel = []
```

```
In [9]: countTest = 0
        for k in range(1, len(tfidfTest) + 1):
            check = 0
            for i in range(13):
                for j in range(15):
                    if k == test[i][j]:
                        x_train[i * 15 + j] = tfidfTest[k - 1]
                        check = 1
                        break
            if check == 0:
                x_test[countTest] = tfidfTest[k - 1]
                countTest += 1
                testLabel.append(k)
        x_train
```

```
Out[9]: array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                0.        ],
               [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                0.        ],
               [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                0.        ],
               ...,
               [0.        , 0.03412166, 0.        , ..., 0.        , 0.        ,
                0.        ],
               [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                0.        ],
               [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                0.        ]])
```

4. 將訓練資料的結果存入一個一維list

In [10]:
```
y_train = []
for i in range(13):
    for k in range(15):
        y_train.append(i + 1)
```

5. 導入SVM method做資料訓練（以linear演示，若要做RBF則改變kernel係數）

In [11]:
```
from sklearn.svm import SVC
SVM_model = SVC(kernel='linear', C=1.0)
```

In [12]:
```
SVM_model.fit(x_train,y_train)
```

Out[12]:
```
SVC(kernel='linear')
```

6. 將測試資料丟入訓練完的分類器中，再以csv檔將內容輸出

In [13]:
```
predicted_results = []
```

In [14]:
```
predicted_results.extend(SVM_model.predict(x_test))
```

In [25]:
```
import csv
```

In [26]:
```
with open('hw2_b09704078.csv', mode='w', encoding='utf-8') as outcome:
    writer = csv.writer(outcome)
    writer.writerow(["Id","Value"])
    for i in range(len(testLabel)):
        writer.writerow([testLabel[i], predicted_results[i]])
```

下面為有關precision的操作

7. 將訓練資料以1:9的比率做切割(stratify=y_train)

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.1, random_state=3, stratify = y_tr
```

8. 將切割完的訓練資料丟入前面的分類器中，並以原正解設為expected_result

In [16]:
```
from sklearn.metrics import PrecisionRecallDisplay
```

In [17]:
```
predicted_result = []
expected_result = []
```

In [18]:
```
predicted_result = SVM_model.predict(X_test)
predicted_result
```

Out[18]:
```
array([ 3, 12,  9,  7,  2,  9, 12, 10,  6,  8, 13,  8, 11,  4, 10,  5,  4,
        1,  1, 11])
```

In [19]:
```
expected_result = Y_test
expected_result
```

Out[19]:
```
[3, 12, 9, 7, 2, 9, 12, 10, 6, 8, 13, 8, 11, 4, 10, 5, 4, 1, 1, 11]
```

9. 將訓練結果與正解做f1-score, precision，並輸出結果

```
In [20]: from sklearn.metrics import precision_score
         precision = precision_score(predicted_result, expected_result, average='micro')
```

```
In [21]: from sklearn import metrics
         print(metrics.classification_report(expected_result, predicted_result))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         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         1
           7       1.00      1.00      1.00         1
           8       1.00      1.00      1.00         2
           9       1.00      1.00      1.00         2
          10       1.00      1.00      1.00         2
          11       1.00      1.00      1.00         2
          12       1.00      1.00      1.00         2
          13       1.00      1.00      1.00         1

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

以下為畫Precision-Recall Curve
10. 調用訓練器OneVsRestClassifier將目前的20個訓練資料訓練成multi-class的機率

```
In [28]: from sklearn.preprocessing import label_binarize

         # Use label_binarize to be multi-label like settings
         Y = label_binarize(expected_result, classes=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
         n_classes = Y.shape[1]
```

```
In [33]: from sklearn.multiclass import OneVsRestClassifier
         from sklearn.preprocessing import StandardScaler
         from sklearn.pipeline import make_pipeline
         from sklearn.svm import LinearSVC

         classifier = OneVsRestClassifier(make_pipeline(StandardScaler(), LinearSVC()))
         classifier.fit(X_test, Y_test)
         y_score = classifier.decision_function(X_test)
```

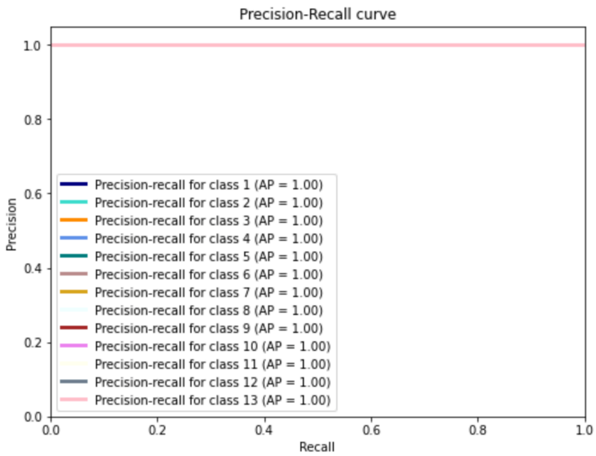
11. 針對每一個class的訓練資料做precision_recall_curve的處理

```
In [30]: y_score = np.array(y_score)
```

```
In [31]: from sklearn.metrics import precision_recall_curve
         from sklearn.metrics import average_precision_score

         # For each class
         precision = dict()
         recall = dict()
         average_precision = dict()
         for i in range(n_classes):
             precision[i], recall[i], _ = precision_recall_curve(Y[:, i], y_score[:, i])
             average_precision[i] = average_precision_score(Y[:, i], y_score[:, i])
```

12. 利用matplotlib做圖

In [32]:
```
import matplotlib.pyplot as plt
from itertools import cycle

# setup plot details
colors = cycle(["navy", "turquoise", "darkorange", "cornflowerblue", "teal", "rosybrown", "goldenrod", "azure", "bro

_, ax = plt.subplots(figsize=(8, 6))

for i, color in zip(range(n_classes), colors):
    display = PrecisionRecallDisplay(recall=recall[i],precision=precision[i],average_precision=average_precision[i])
    display.plot(ax=ax, name=f"Precision-recall for class {i + 1}", color=color, linewidth=3)

handles, labels = display.ax_.get_legend_handles_labels()
ax.axis([0.0, 1.0, 0.0, 1.05])
ax.legend(handles=handles, labels=labels, loc="best")
ax.set_title("Precision-Recall curve")

plt.show()
```



Bernoulli Naïve Bayes method

1. 先將檔案分類的資料讀入後，把所有內容轉存為整數

In [1]:
```
testTemp = [[]]
```

In [2]:
```
with open('training_new.txt', mode='r', encoding='utf-8') as f:
    for i in f.readlines():
        line = i.split()
        testTemp.append(line)
```

In [3]:
```
test = [[]]
```

In [4]:
```
for i in range(13):
    temp = []
    for j in range(15):
        temp.append(int(float(testTemp[i + 1][j + 1])))
    test.append(temp)
test.pop(0)
```

Out[4]: []

2. 重新讀取所有文字檔，並存入files

In [5]:
```
files = []
```

In [6]:
```
for i in range(1, 1096):
    with open('PA1-data/' + str(i) + '.txt', mode='r', encoding='utf-8') as f:
        file = [f.read()]
        files += file
```

3. 將所有檔案做Multi-hot vector，並存入x_total

```
In [7]: from sklearn.preprocessing import MultiLabelBinarizer
        import numpy as np
```

```
In [8]: x_total = np.empty([0])
```

```
In [9]: processTrain = MultiLabelBinarizer()
        x_total = processTrain.fit_transform(files)
```

4. 判讀檔案是否為訓練檔，並將Multi-hot vector分裝於兩個ndarray中

```
In [10]: x_train = np.empty([195, 85])
         x_test = np.empty([900, 85])
         testLabel = []
```

```
In [11]: countTest = 0
         for k in range(1, len(x_total) + 1):
             check = 0
             for i in range(13):
                 for j in range(15):
                     if k == test[i][j]:
                         x_train[i * 15 + j] = x_total[k - 1]
                         check = 1
                         break
             if check == 0:
                 x_test[countTest] = x_total[k - 1]
                 countTest += 1
                 testLabel.append(k)
```

5. 將將訓練資料的結果存入一個一維list後，導入BernoulliNB做資料訓練

```
In [12]: from sklearn. naive_bayes import BernoulliNB
```

```
In [13]: y_train = []
         for i in range(13):
             for k in range(15):
                 y_train.append(i + 1)
```

```
In [14]: train = BernoulliNB()
```

```
In [15]: train.fit(x_train, y_train)
Out[15]: BernoulliNB()
```

6. 將測試資料丟入分類器中，並以list存取

```
In [16]: predicted_results = []
```

```
In [17]: predicted_results.extend(train.predict(x_test))
         predicted_results
Out[17]: [2,
          13,
          8,
          8,
          8,
          10,
          13,
          9,
          8,
          2,
          8,
          10,
          8,
          5,
          13,
          13,
          8,
          8,
          5,
```

7. 將測試資料丟入訓練完的分類器中，再以csv檔將內容輸出

In [18]:
```
import csv
```

In [19]:
```
with open('hw2_b09704078.csv', mode='w', encoding='utf-8') as outcome:
    writer = csv.writer(outcome)
    writer.writerow(["Id","Value"])
    for i in range(len(testLabel)):
        writer.writerow([testLabel[i], predicted_results[i]])
```

下面為有關precision的操作

8. 將訓練資料以1:9的比率做切割(stratify=y_train)

In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.1, random_
```

9. 將切割完的訓練資料丟入前面的分類器中，並以原正解設為expected_result

In [21]:
```
from sklearn.metrics import PrecisionRecallDisplay
```

In [22]:
```
predicted_result = []
expected_result = []
```

In [24]:
```
predicted_result = train.predict(X_test)
predicted_result
```

Out[24]:
```
array([13, 12,  9,  1,  5, 12, 13,  5,  6,  8,  8,  8,  8, 13,  8,  5, 13,
        1,  1, 13])
```

In [25]:
```
expected_result = Y_test
expected_result
```

Out[25]:
```
[3, 12, 9, 7, 2, 9, 12, 10, 6, 8, 13, 8, 11, 4, 10, 5, 4, 1, 1, 11]
```

10. 將訓練結果與正解做f1-score, precision，並輸出結果

In [26]:
```
from sklearn.metrics import precision_score
precision = precision_score(predicted_result, expected_result, average='micro')
```

In [27]:
```
from sklearn import metrics
print(metrics.classification_report(expected_result, predicted_result))
```

```
              precision    recall  f1-score   support

           1       0.67      1.00      0.80         2
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         2
           5       0.33      1.00      0.50         1
           6       1.00      1.00      1.00         1
           7       0.00      0.00      0.00         1
           8       0.40      1.00      0.57         2
           9       1.00      0.50      0.67         2
          10       0.00      0.00      0.00         2
          11       0.00      0.00      0.00         2
          12       0.50      0.50      0.50         2
          13       0.00      0.00      0.00         1

    accuracy                           0.40        20
   macro avg       0.30      0.38      0.31        20
weighted avg       0.32      0.40      0.33        20
```

以下為畫Precision–Recall Curve

11. 調用訓練器OneVsRestClassifier將目前的20個訓練資料訓練成multi–class的機率

In [27]:
```
from sklearn.preprocessing import label_binarize

# Use label_binarize to be multi-label like settings
Y = label_binarize(expected_result, classes=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
n_classes = Y.shape[1]
```

In [36]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC

classifier = OneVsRestClassifier(make_pipeline(StandardScaler(), LinearSVC()))
classifier.fit(X_test, Y_test)
y_score = classifier.decision_function(X_test)
```

12. 針對每一個class的訓練資料做precision_recall_curve的處理

In [37]:
```
y_score = np.array(y_score)
```

In [38]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(Y[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(Y[:, i], y_score[:, i])
```

13. 利用matplotlib做圖

In [31]:
```
import matplotlib.pyplot as plt
from itertools import cycle

# setup plot details
colors = cycle(["navy", "turquoise", "darkorange", "cornflowerblue", "teal", "rosybrown", "goldenrod", "azure", "bro

_, ax = plt.subplots(figsize=(8, 6))

for i, color in zip(range(n_classes), colors):
    display = PrecisionRecallDisplay(recall=recall[i],precision=precision[i],average_precision=average_precision[i])
    display.plot(ax=ax, name=f"Precision-recall for class {i + 1}", color=color, linewidth=3)

handles, labels = display.ax_.get_legend_handles_labels()
ax.axis([0.0, 1.0, 0.0, 1.05])
ax.legend(handles=handles, labels=labels, loc="best")
ax.set_title("Precision-Recall curve")

plt.show()
```