COMP7103C – Data Mining Assignment 2

Group Information

Name	**UID**
ZHANG JIAYI	3036032798
Zheng Xufeng	3036034019

Diagram 1 – Group Memebers

Dataset Description

After receiving the dataset, we initially start dataset exploration. In this section, we visualized the datasets.

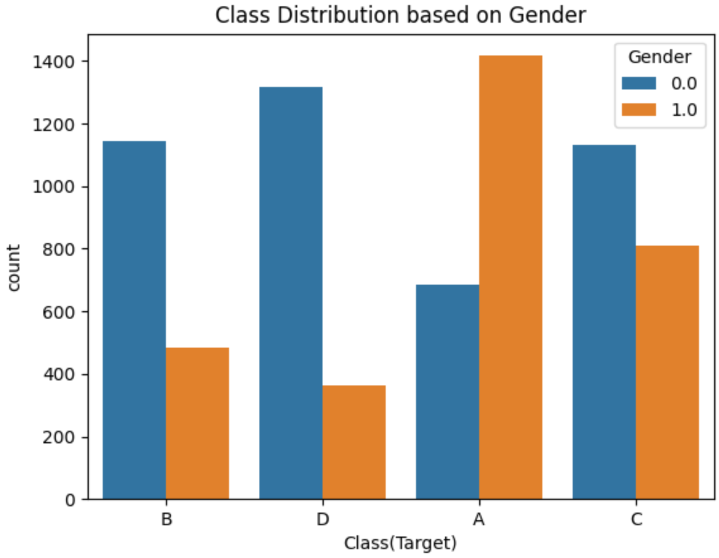

Figure 1 – Class Distribution based on Gender

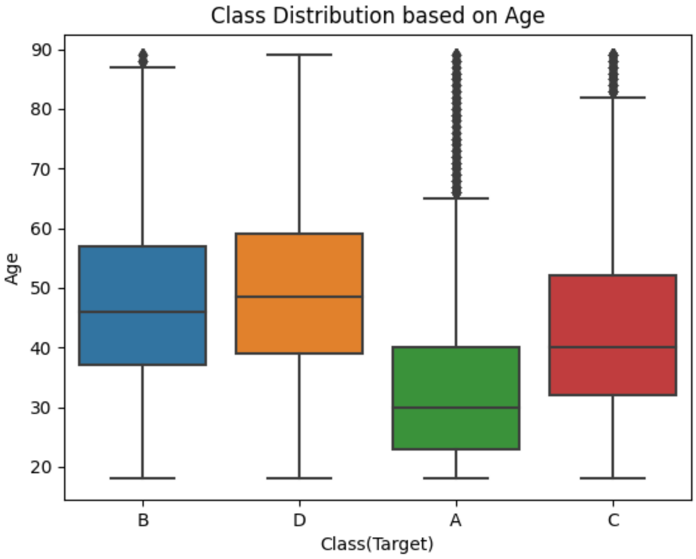

Figure 2 – Class Distribution based on Age

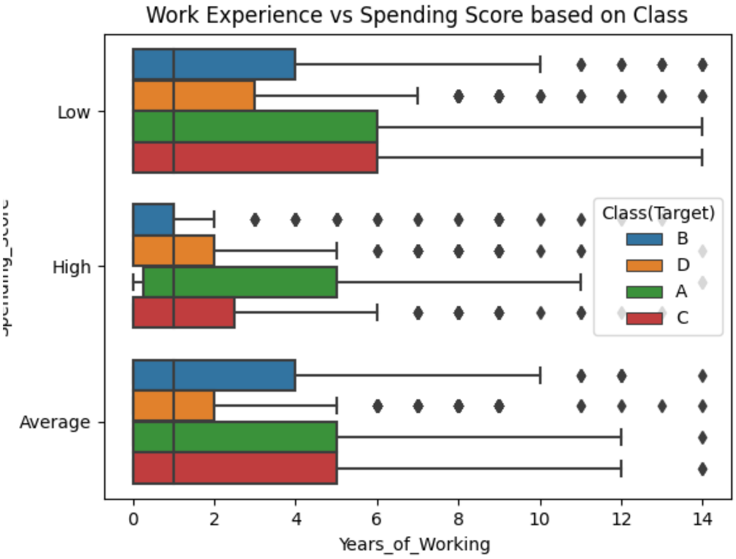

Figure 3 – Work Experience and Spending Score based on Class

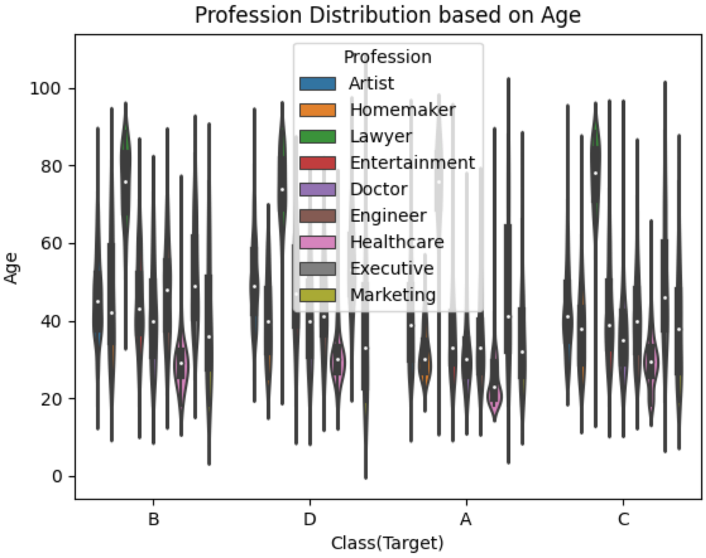

Figure 4 – Profession Distribution based on Age

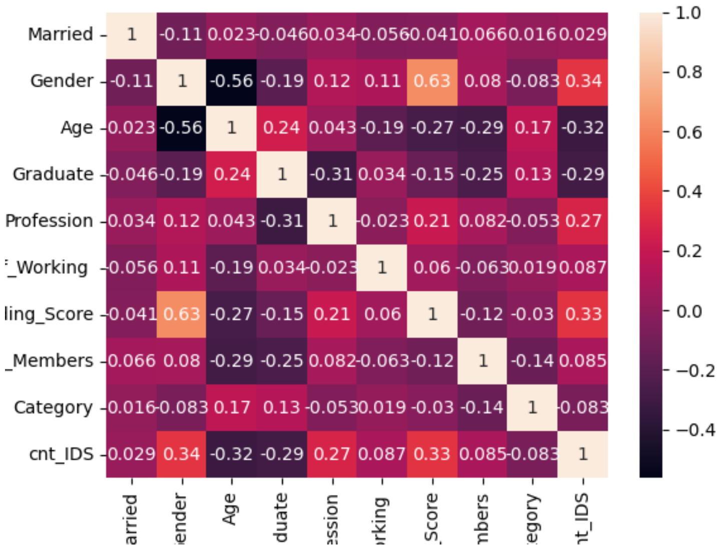

Figure 5 – Correlation Matrix

Dataset Preprocessing

We processed the dataset in the following manners.

1. Married: Fill all NaN with 0 if any.

```
df['Married'].fillna(0, inplace=True)
```

2. Gender: Fill all NaN with 2 if any.

```
df['Gender'].fillna(2, inplace=True)
```

3. Age: Fill all NaN with -1 if any.

```
df['Age'].fillna(-1, inplace=True)
```

4. Graduate: Fill all NaN with 2 if any.

```
df['Graduate'].fillna(2, inplace=True)
```

5. Profession: Replace 9 different professions with their corresponding integers, and fill all NaN with 0.

```
df.loc[df['Profession'].notna(), 'Profession'] =
df.loc[df['Profession'].notna(), 'Profession'].replace(
       {
           'Artist': 1,
           'Doctor': 2,
           'Engineer': 3,
           'Entertainment': 4,
           'Executive': 5,
           'Healthcare': 6,
           'Homemaker': 7,
           'Lawyer': 8,
           'Marketing': 9
       })
df['Profession'].fillna(0, inplace=True)
```

6. Years_of_Working : Fill all NaN with -1 if any.

```
df['Years_of_Working '].fillna(-1, inplace=True)
```

7. Spending_Score: Replace 3 different spending scores with their corresponding integers, and fill all NaN with 0.

```
df.loc[df['Spending_Score'].notna(), 'Spending_Score'] =
df.loc[df['Spending_Score'].notna(), 'Spending_Score'].replace(
        {
            'Low': 1,
            'Average': 2,
            'High': 3
        })
df['Spending_Score'].fillna(0, inplace=True)
```

8. Family_Members : Fill all NaN with 0 if any.
9. Category: Replace 7 different spending scores with their corresponding integers, and fill all NaN with 0.

```
df.loc[df['Category'].notna(), 'Category'] = df.loc[df['Category'].notna(),
'Category'].replace(
        {
            'Cat_1': 1,
            'Cat_2': 2,
            'Cat_3': 3,
            'Cat_4': 4,
            'Cat_5': 5,
            'Cat_6': 6,
            'Cat_7': 7
        })
df['Category'].fillna(0, inplace=True)
```

Model Construction and Evaluation

In this section, we build 7 different models independently and also use 4 different ensemble learning methods to integrate the output of 7 different models' output. Hence there are totally 11 evaluated models.

Model	Accuracy on Validation Set	Output File Name
Decision Tree	***0.46960***	***Test_DT.csv***
KNN	*0.43966*	***Test_KNN.csv***
Logistic Regression	*0.44060*	***Test_LR.csv***
MLP	*0.45182*	***Test_MLP.csv***
Naive Bayes	*0.41628*	***Test_NB.csv***
Random Forest	*0.41534*	***Test_RF.csv***
SVM	*0.44808*	***Test_SVC.csv***

Diagram 2 – Weak Learning Evaluation Result

Model	Accuracy on Validation Set	Accuracy on Training Set	Output File Name

7-Output + MajorityVote	*0.4584*	*0.5119*	***Test_Integration_MajorityVote.csv***
7-Output + MajorityVoteWithWeight	*0.4153*	*0.9507*	***Test_Integration_MajorityVoteWithWeight.csv***
7-Output + MLP	*0.4153*	*0.9507*	***Test_Integration_MLP.csv***
7-Output + AdaBoost	***0.4153***	***0.9507***	***Test_Integration_Ada.csv***

Diagram 3 – Ensemble Learning Evaluation Result

Decision Tree

Build the model with the following parameters.

```
model = DecisionTreeClassifier(max_depth=4, criterion='entropy').fit(X_train,
y_train)
```

The accuracy for the Validate set is 0.46960. Figure 6 shows in the confusion matrix.

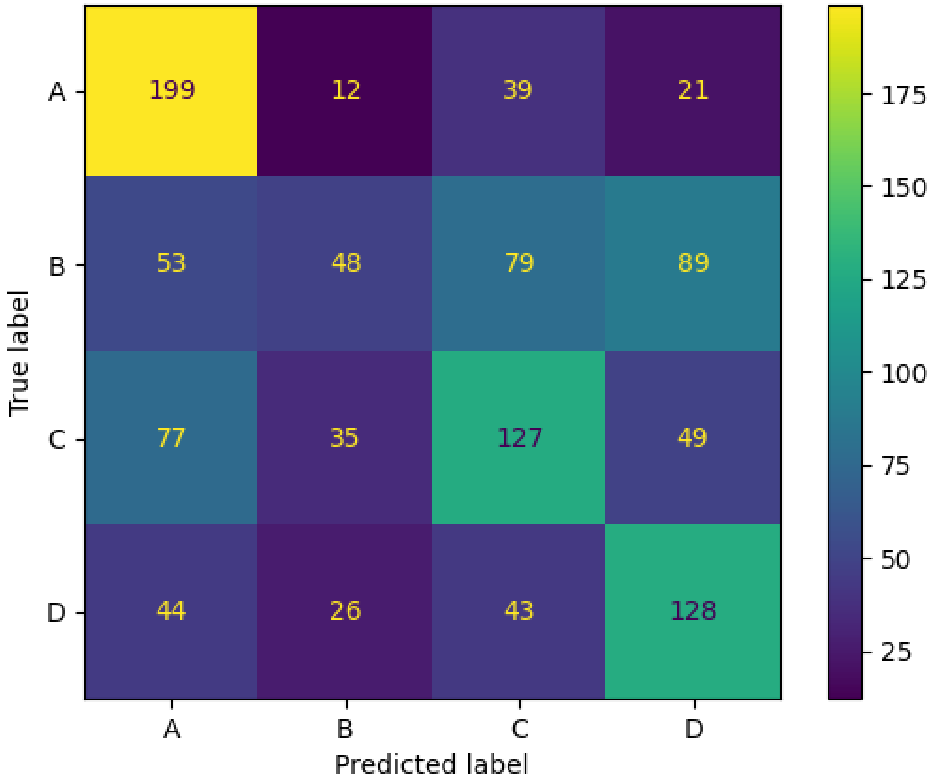

Figure 6 – Confusion Matrix for Decision Tree on Validation Set

KNN

First we find the tuned k among 100 candidates by using the following way.

```
iteration = 100
error_rate = []
acc = []
scores = {}
```

```
for i in range(1, iteration):
   model_knn = KNeighborsClassifier(n_neighbors=i)
   model_knn.fit(x_train, y_train)
   y_pred_knn = model_knn.predict(x_val)
   error_rate.append(np.mean(y_pred_knn != y_val))
   scores[i] = metrics.accuracy_score(y_val, y_pred_knn)
   acc.append(metrics.accuracy_score(y_val, y_pred_knn))
```

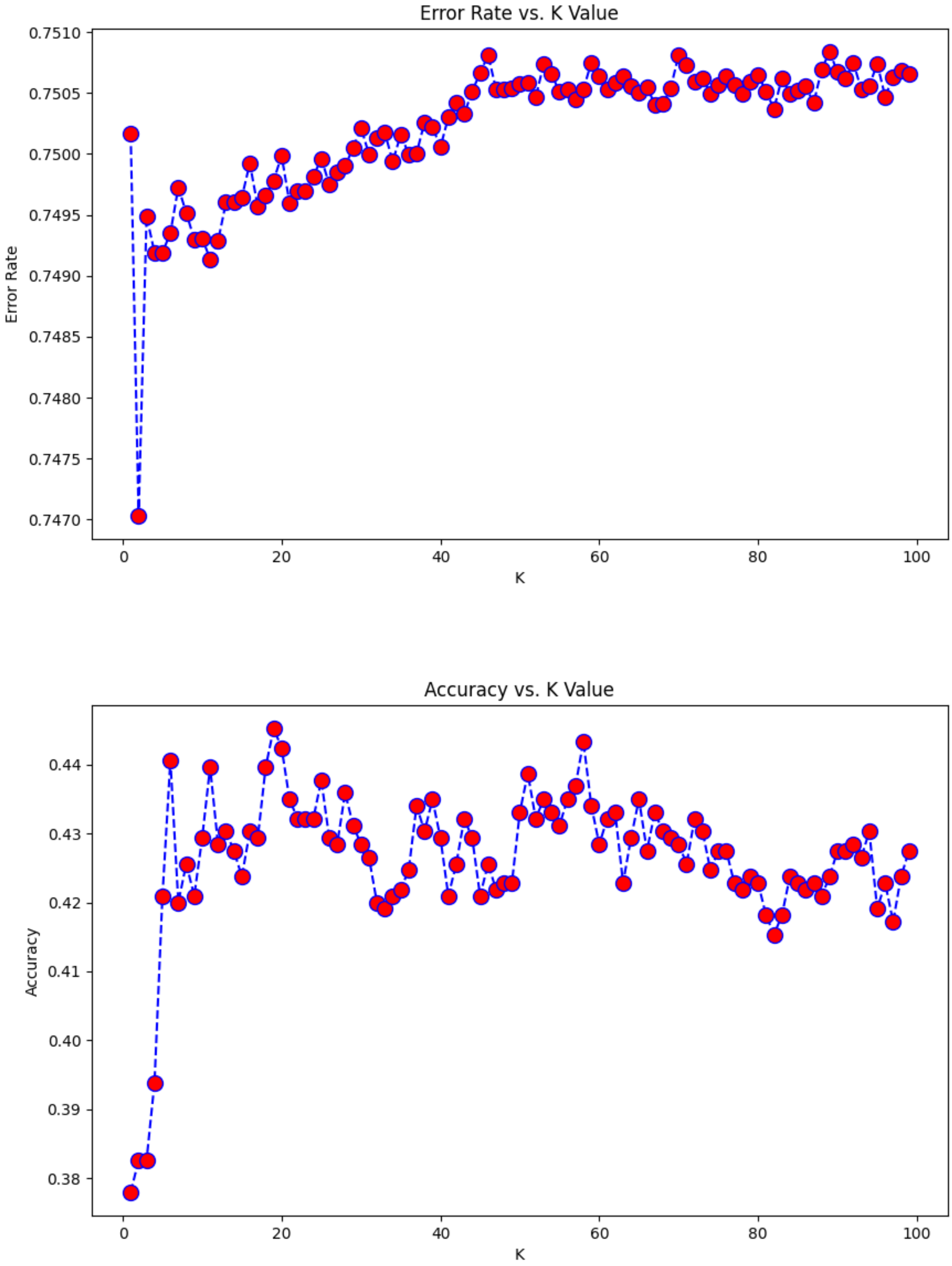

Figure 7 – Different K Loss and Accuracy

Yield that the best parameter for k is 18.

Build the model with the following parameters.

```
# TODO HERE
model_knn = KNeighborsClassifier(n_neighbors=18)
model_knn.fit(x_train, y_train)
```

The accuracy for the Validate set is 0.439663. Figure 8 shows in the confusion matrix.

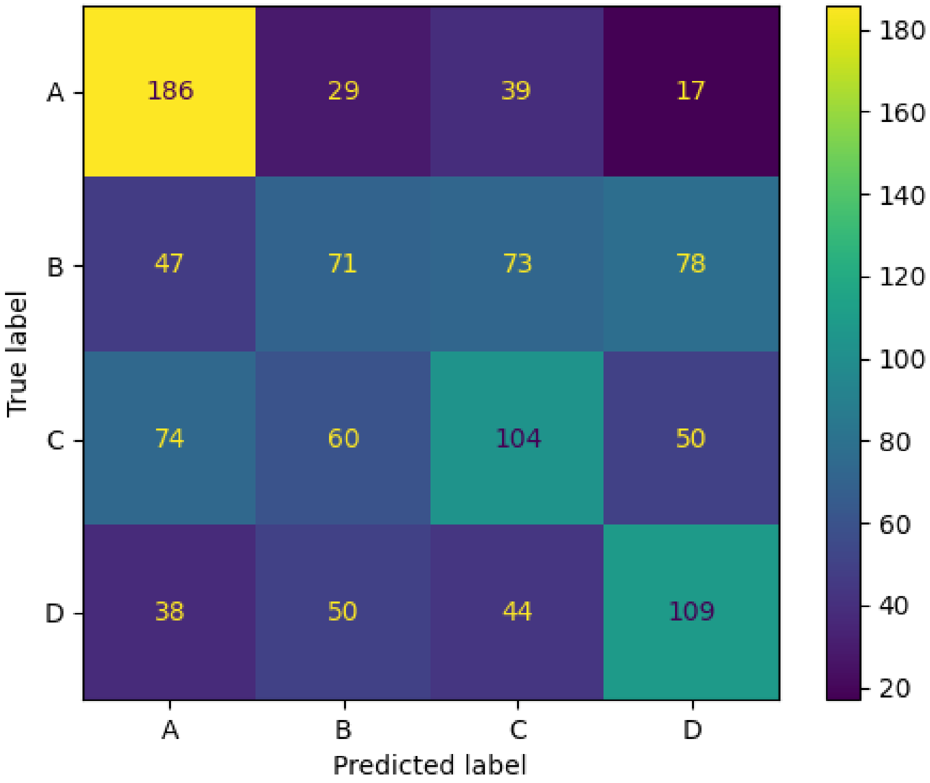

Figure 8 – Confusion Matrix for KNN on Validation Set

Logistic Regression

Build the model with the following parameters.

```
model = LogisticRegression(max_iter=600)
model.fit(x_train, y_train)
```

The accuracy for the Validate set is 0.440599. Figure 9 shows in the confusion matrix.

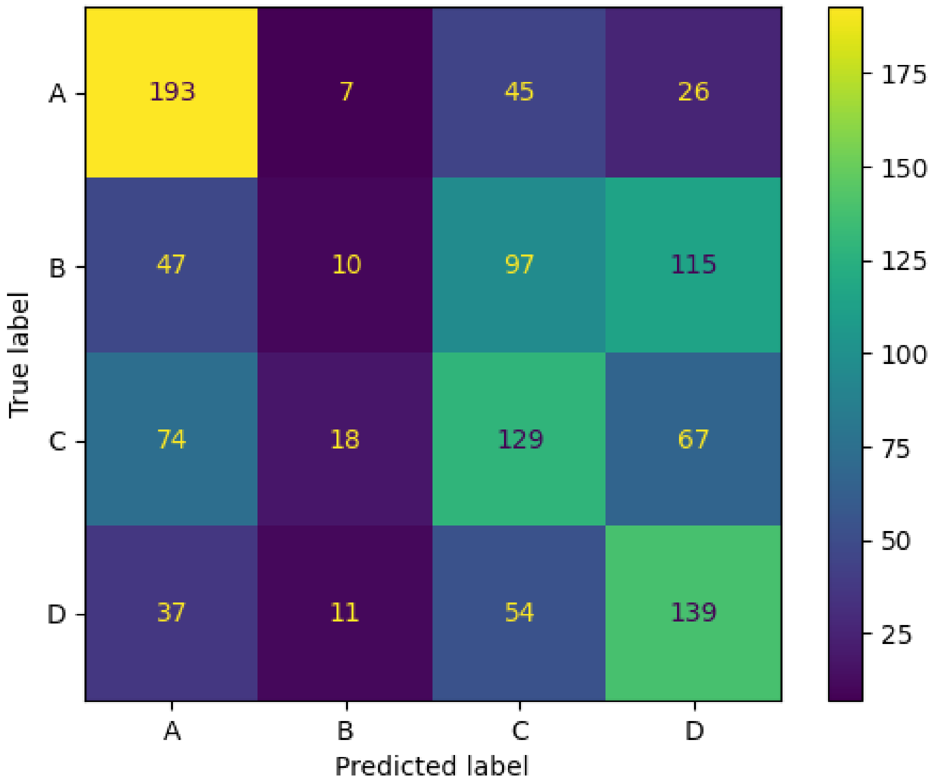

Figure 9 – Confusion Matrix for Logistic Regression on Validation Set

MLP

Build the model with the following parameters.

```
model = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(100,), random_state=1).fit(X_train, y_train)
```

The accuracy for the Validate set is 0.45182. Figure 10 shows in the confusion matrix.

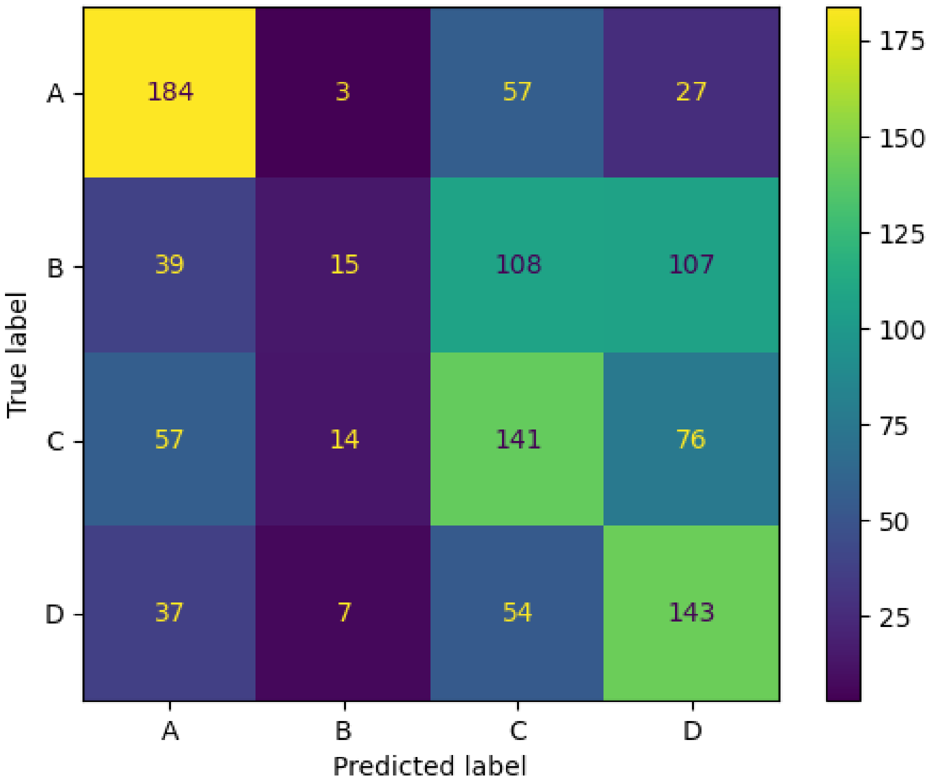

Figure 10 – Confusion Matrix for MLP on Validation Set

Naive Bayes

Build the model with the following parameters.

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

The accuracy for the Validate set is 0.41628. Figure 11 shows in the confusion matrix.

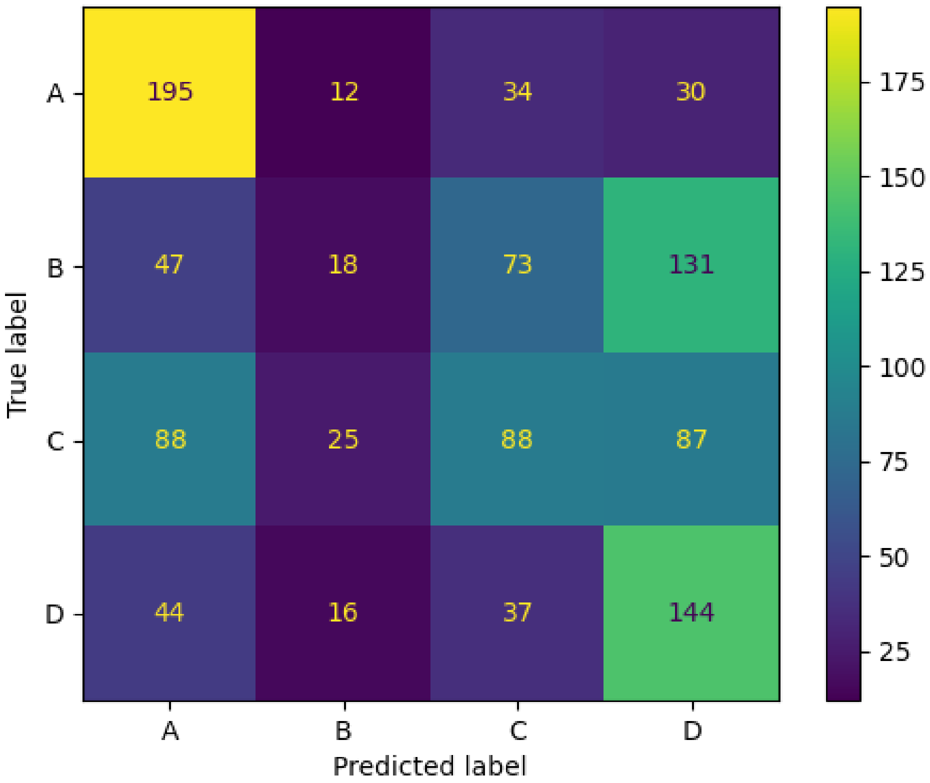

Figure 11 – Confusion Matrix for MLP on Validation Set

Random Forest

Build the model with the following parameters.

```
model_rf = RandomForestClassifier(n_estimators=100, criterion='gini',
random_state=0)
model_rf.fit(x_train, y_train)
```

The accuracy for the Validate set is 0.41534. Figure 12 shows in the confusion matrix.

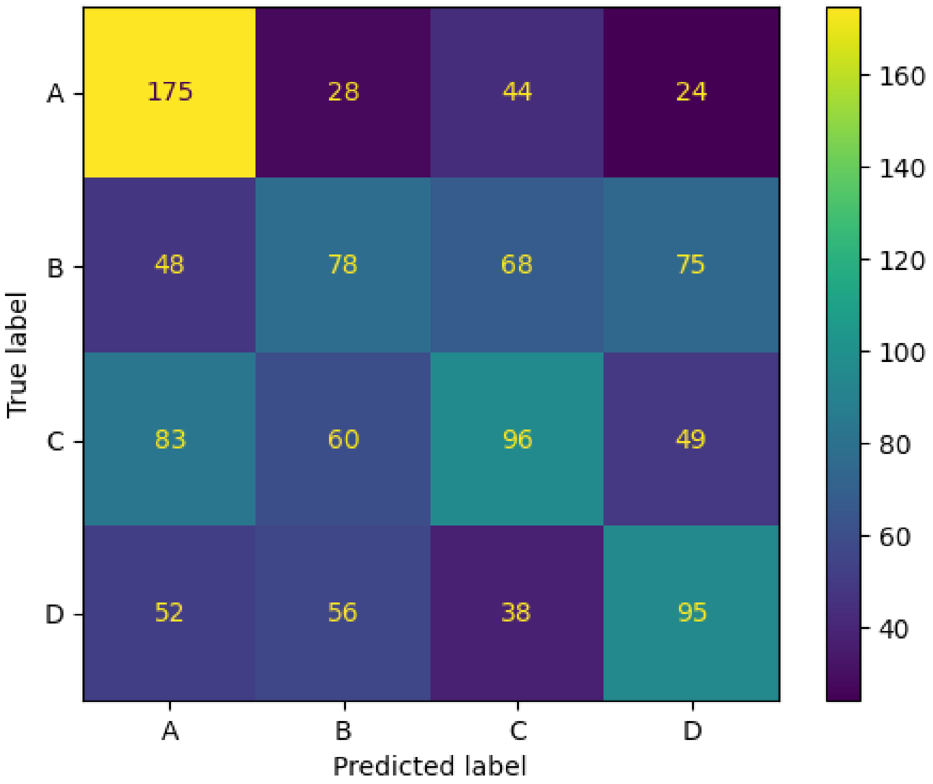

Figure 12 – Confusion Matrix for Random Forest on Validation Set

SVM

Build the model with the following parameters.

```
model = SVC(kernel='linear', C=1).fit(X_train, y_train)
```

The accuracy for the Validate set is 0.44808. Figure 13 shows in the confusion matrix.

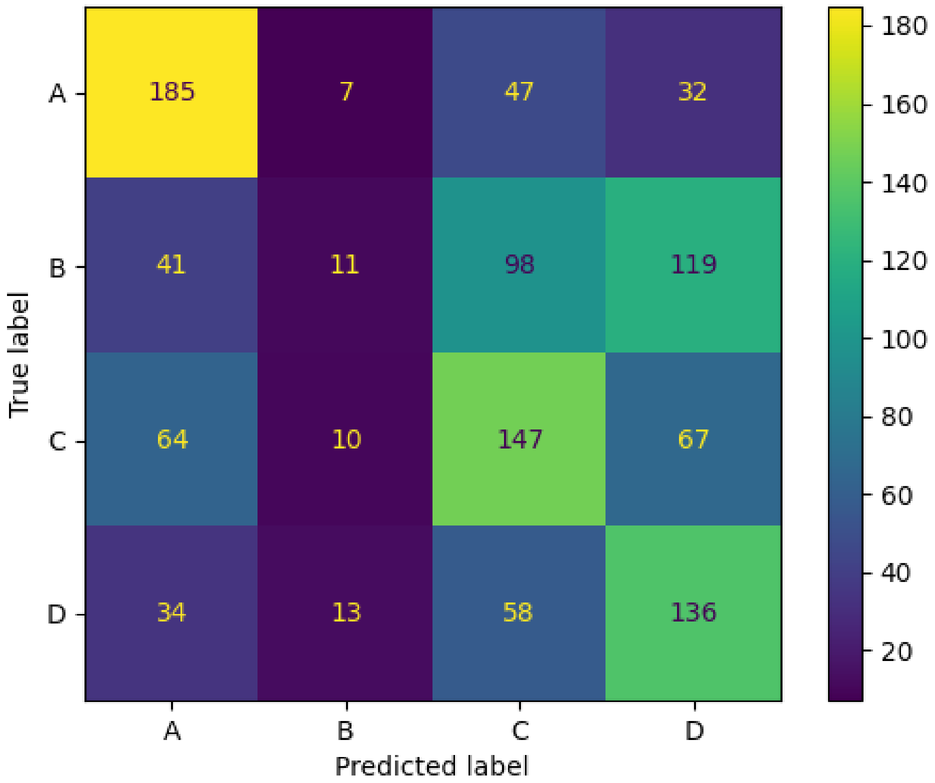

Figure 13 – Confusion Matrix for SVM on Validation Set

7-Output + MajorityVote

We use the above 7 outputs together, and yield a majority results from those 7 outputs. The accuracy on the training set is 0.511954053693068, on the validating set is 0.4583723105706268.

```
df['majority_vote'] = df_train.apply(lambda x: x.mode()[0], axis=1)
df_s['majority_vote'] = df_val.apply(lambda x: x.mode()[0], axis=1)
```

7-Output + MajorityVoteWithWeight

Similar to above, the difference is we add 7-times weights on LR output. The accuracy on the training set is 0.9507145719246695, on the validating set is 0.4153414405986904.

7-Output + MLP

We took 7 output predictions as input for the MLP. The model is built as follows. We convert the labels into integers for the training and testing.

```
# Define the MLP model
model = Sequential()
model.add(Dense(64, input_dim=7, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(4, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

The accuracy for the training set is 0.9507, on the validation set is 0.41534143686294556.

7-Output + AdaBoost

We took 7 output predictions as input for the AdaBoost. The model is built as follows. We convert the labels into integers for the training and testing.

```
model = AdaBoostClassifier(n_estimators=100, learning_rate=0.1,
algorithm='SAMME')
```

The accuracy for the training set is 0.9507145719246695, on the validation set is 0.4153414405986904.

We eventually choose to use **AdaBoost** result as final results.

GitHub

https://github.com/AXNTROYUANXD/Simple_Classification