Data Mining Final Project

Name: Shang-Yung Hsu

ID: sh555

Tool

Sklearn

Matplotlib

panda

Check and Modify the Dataset

The car evaluation dataset is from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation.

I give the column name to each column.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data'
column_names = ['buying_price', 'maint_price', 'num_doors',
                'people_cap', 'luggage_boot_size', 'safety', 'class']
data = pd.read_csv(url, names=column_names)
print(data.head())
```

```
  buying_price maint_price num_doors people_cap luggage_boot_size safety  class
0        vhigh       vhigh         2          2             small    low  unacc
1        vhigh       vhigh         2          2             small    med  unacc
2        vhigh       vhigh         2          2             small   high  unacc
3        vhigh       vhigh         2          2               med    low  unacc
4        vhigh       vhigh         2          2               med    med  unacc
[Finished in 1.8s]
```

Check the dataset if there is any null value.

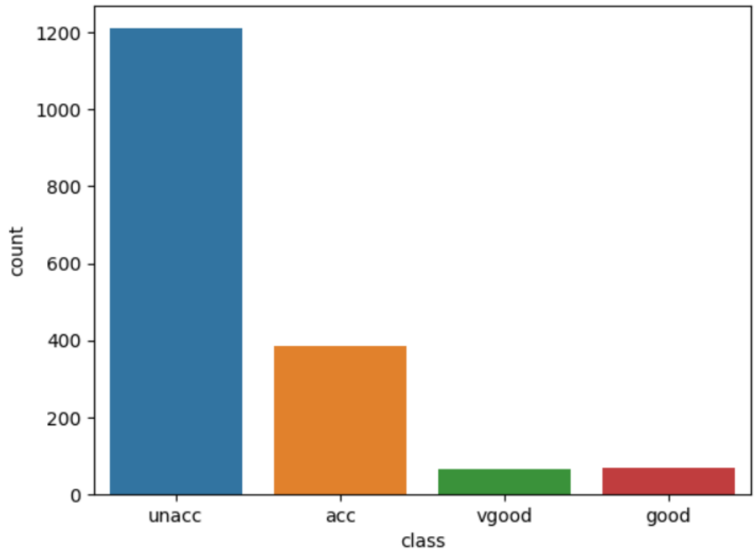
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying_price         1728 non-null object
maint_price          1728 non-null object
num_doors            1728 non-null object
people_cap           1728 non-null object
luggage_boot_size    1728 non-null object
safety               1728 non-null object
class                1728 non-null object
dtypes: object(7)
memory usage: 47.3+ KB
None
[Finished in 2.8s]
```

Seeing the description of the car evaluation dataset, you will find that the 'class ' distribution is really unbalance.

```
 9. Class Distribution (number of instances per class)

    class      N          N[%]
    -----------------------------
    unacc     1210     (70.023 %)
    acc        384     (22.222 %)
    good        69     ( 3.993 %)
    v-good      65     ( 3.762 %)
```

```
sns.countplot(data['class'])
plt.show()
```



In this case, it will have an imbalanced multiclass classification problem. So that, the accuracy may not be the best stander to check this model.

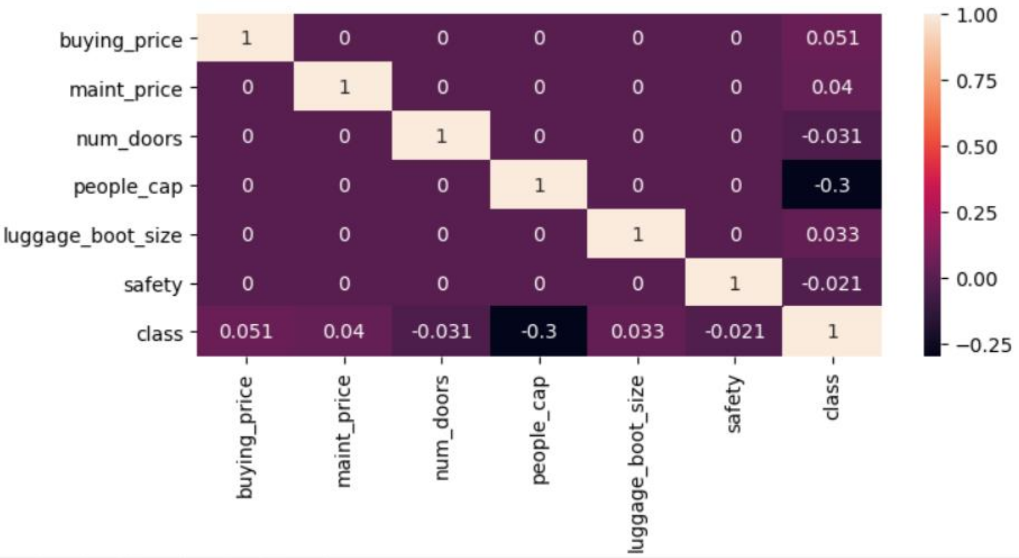
As scikit-learn algorithms do not work with string values, so I converted the string categories to integers.

```
lable=LabelEncoder()
for i in data.columns:
    data[i] = lable.fit_transform(data[i])
print(data.head())
```

```
   buying_price  maint_price  num_doors  ...  luggage_boot_size  safety  class
0             3            3          0  ...                  2       1      2
1             3            3          0  ...                  2       2      2
2             3            3          0  ...                  2       0      2
3             3            3          0  ...                  1       1      2
4             3            3          0  ...                  1       2      2

[5 rows x 7 columns]
[Finished in 2.6s]
```

Looking at the Heatmap of the columns on dataset with each other. Will find that most of the columns shows very weak correlation with 'class'. ('persons' column is the weakest). Other columns except 'class' shows no correlation with each other.

```
fig=plt.figure(figsize=(8,4))
sns.heatmap(data.corr(),annot=True)
plt.show()
```



So, doing any analysis on them might not give any useful output.

Training and Analyze dataset

X is the dataframe containing input data / features

y is the series which has results which are to be predicted.

Random Forest

Divide data in train and test sets and train by Random Forest Algorithm.

```
X=data[data.columns[:-1]]
y=data['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
rfc=RandomForestClassifier(n_jobs=-1,random_state=51)
rfc.fit(X_train,y_train)
predictions = rfc.predict(X_test)
print("Random Forest Accuracy:",rfc.score(X_test,y_test))
print("Random Forest F1-score:",f1_score(y_test,rfc.predict(X_test),average='macro'))
print(classification_report(y_test,predictions))
```

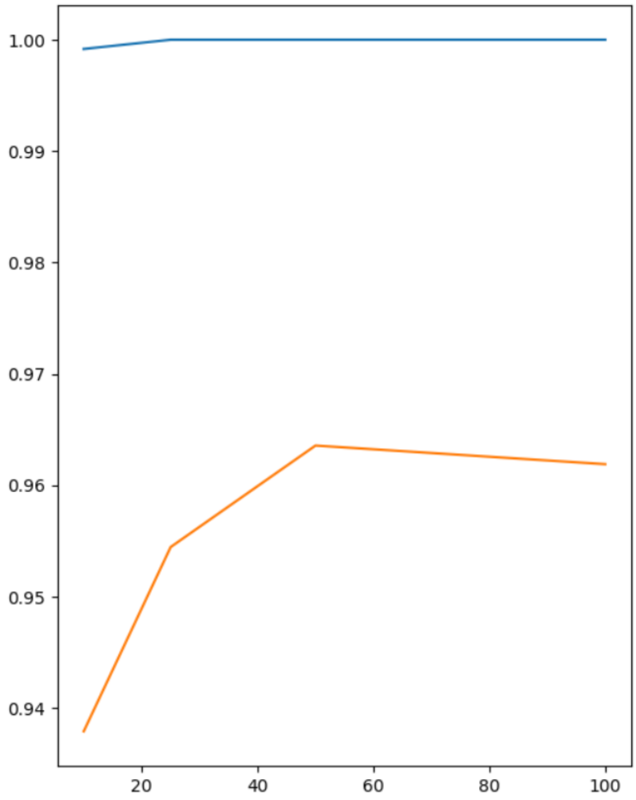
Because this dataset is an imbalanced dataset, so like I mention before, we should also consider the f1 score instead of only see the accuracy.

```
Random Forest Accuracy: 0.9499036608863198
Random Forest F1-score: 0.8843956728712123
              precision    recall  f1-score   support

           0       0.87      0.92      0.89       118
           1       0.83      0.79      0.81        19
           2       0.99      0.98      0.98       358
           3       0.87      0.83      0.85        24

   micro avg       0.95      0.95      0.95       519
   macro avg       0.89      0.88      0.88       519
weighted avg       0.95      0.95      0.95       519
```

The basic model of RFC is giving almost 95% accuracy.

Then, we should check the effect of n_estimators on the model.

```
param_range=[10,25,50,100]
curve=validation_curve(rfc,X_train,y_train,cv=5,param_name='n_estimators',param_range=param_range,n_jobs=-1)

train_score=[curve[0][i].mean() for i in range (0,len(param_range))]
test_score=[curve[1][i].mean() for i in range (0,len(param_range))]
fig=plt.figure(figsize=(6,8))
plt.plot(param_range,train_score)
plt.plot(param_range,test_score)
plt.xticks=param_range
plt.show()
```



So, with the increasing n_estimators, the accuracy is also increasing. The best evaluating value is at n_estimators=50. After n_estimators=50, model starts overfitting. Therefore, we reached the accuracy to almost 96%.

Also, we can use the GridSearch to get the combination of best parameters. It is a simpler way to get all the parameters, but it takes more time to complete.

```
param_grid={'criterion':['gini','entropy'],
            'max_depth':[2,5,10,20],
            'max_features':[2,4,6,'auto'],
            'max_leaf_nodes':[2,3,None],}

grid=GridSearchCV(estimator=RandomForestClassifier(n_estimators=50,n_jobs=-1,random_state=51),
                  param_grid=param_grid,cv=10,n_jobs=-1)
grid.fit(X_train,y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 10, 'max_features': 6, 'max_leaf_nodes': None}
0.9842845326716294
```

So, with those parameters. We can reach 98.4% accuracy.

Support Vector Machine - SVM

Divide data in train and test sets and train by SVM Algorithm.

```
svm = SVC(kernel = 'rbf', probability = True, gamma='auto')
svm.fit(X_train, y_train)
y_predictions = svm.predict(X_test)
print("SVM Accuracy:",svm.score(X_test, y_test))
print("SVM F1-score:",f1_score(y_test,svm.predict(X_test),average='macro'))
```

like I mention before, we should also consider the f1 score instead of only see the accuracy.

```
SVM Accuracy: 0.8998073217726397
SVM F1-score: 0.735024338854984
              precision    recall  f1-score   support

           0       0.83      0.73      0.77       118
           1       0.80      0.21      0.33        19
           2       0.92      0.99      0.96       358
           3       0.88      0.88      0.88        24

   micro avg       0.90      0.90      0.90       519
   macro avg       0.86      0.70      0.74       519
weighted avg       0.89      0.90      0.89       519
```

We can also use the GridSearch to find the best parameters. (Check the kernel separate)

```
params_model = {'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001]}

model = GridSearchCV(SVC(probability=True),
                     params_model,
                     refit=True,
                     return_train_score=True,
                     cv = 5)
model.fit(X_train, y_train)
print(model.best_params_)
print(model.best_score_)
```

```
{'C': 1000, 'gamma': 0.1}
0.989247311827957
```

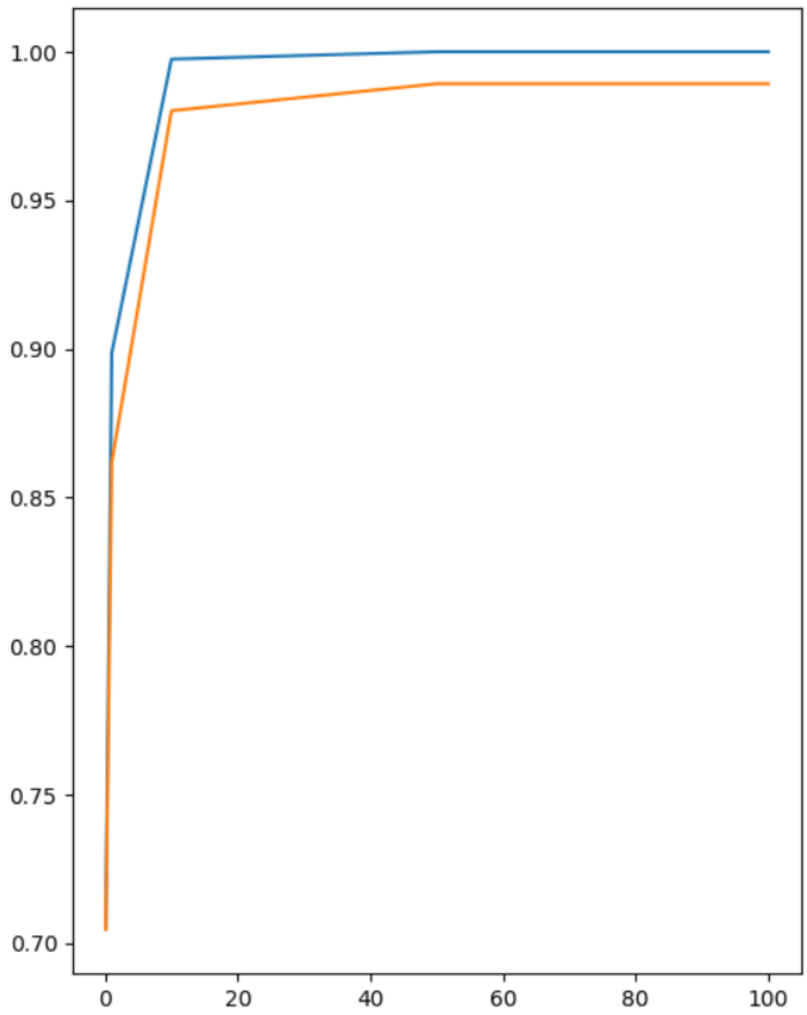
It looks high accuracy, but the value of C is too large. When C tends to infinity, the problem is that samples with classification errors are not allowed. That is a hard-margin SVM problem (overfitting). So, it might not the result we want.

The reason why it causes the overfitting is because of the dataset is imbalance.

Then, we should check the effect value of C in this model.

```
param_range=[0.1,1,10,50,100]
curve=validation_curve( SVC(kernel = 'rbf', probability = True, gamma = 'auto'),X_train,y_train,cv=5,param_name='C',param_range=param_range,n_jobs=-1)

train_score=[curve[0][i].mean() for i in range (0,len(param_range))]
test_score=[curve[1][i].mean() for i in range (0,len(param_range))]
fig=plt.figure(figsize=(6,8))
plt.plot(param_range,train_score)
plt.plot(param_range,test_score)
plt.xticks=param_range
plt.show()
```

This is for kernel rbf.

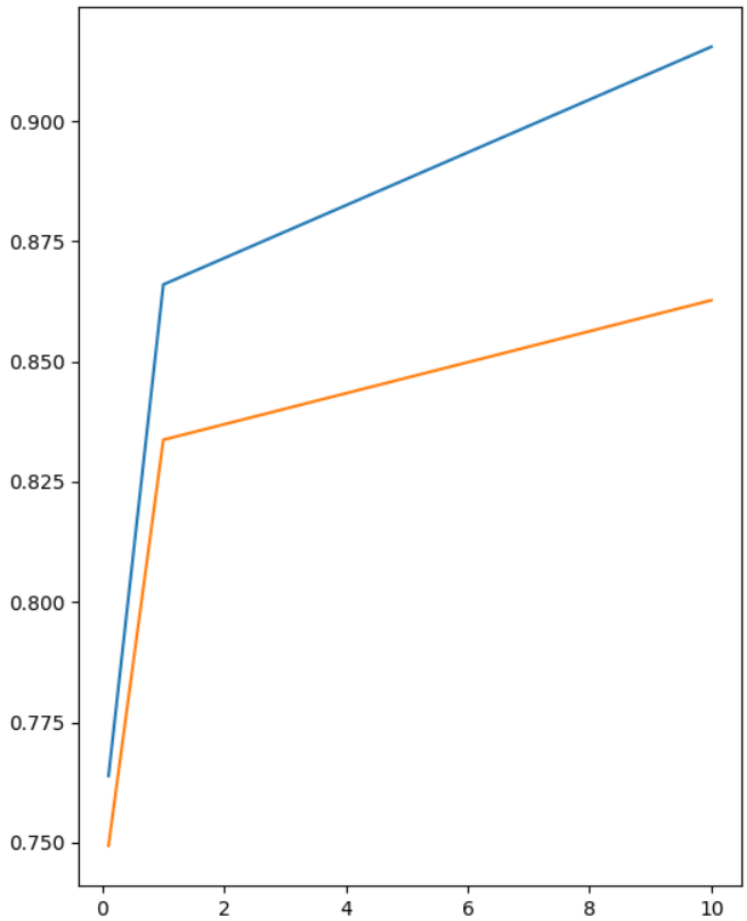


For the kernel rbf. When C is bigger than 15, it will overfitting. The best value of C will be 15 for kernel rbf.

```
{'C': 15, 'gamma': 0.1}
0.9760132340777502
```

This is for kernel poly.

When the value of C equal to 10.



For the kernel poly. When C is bigger than 1. It will be overfitting. Also the accuracy is lower than the kernel rbf.

So, the best kernel to choose is rbf.

Conclusion

Random Forest Classifier is the better model for this data with following parameters:
n_estimators: 50, criterion: entropy, max_depth: 10, max_features: 6, max_leaf_nodes: None
Achieve 98.43% accuracy with this model

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import validation_curve
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from matplotlib.colors import ListedColormap

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data'
column_names = ['buying_price', 'maint_price', 'num_doors',
                'people_cap', 'luggage_boot_size', 'safety', 'class']
data = pd.read_csv(url, names=column_names)
#print(data.head())
#print(data.info())
#sns.countplot(data['class'])
#plt.show()
lable=LabelEncoder()
for i in data.columns:
    data[i] = lable.fit_transform(data[i])
#print(data.head())
#fig=plt.figure(figsize=(8,4))
#sns.heatmap(data.corr(),annot=True)
#plt.show()
X=data[data.columns[:-1]]
y=data['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
rfc=RandomForestClassifier(n_jobs=-1,random_state=51)
rfc.fit(X_train,y_train)
```

```
predictions = rfc.predict(X_test)
#print("Random Forest Accuracy:",rfc.score(X_test,y_test))
#print("Random Forest F1-
score:",f1_score(y_test,rfc.predict(X_test),average='macro'))
#print(classification_report(y_test,predictions))

param_range=[10,25,50,100]
curve=validation_curve(rfc,X_train,y_train,cv=5,param_name='n_estimators',param_
range=param_range,n_jobs=-1)

train_score=[curve[0][i].mean() for i in range (0,len(param_range))]
test_score=[curve[1][i].mean() for i in range (0,len(param_range))]
fig=plt.figure(figsize=(6,8))
plt.plot(param_range,train_score)
plt.plot(param_range,test_score)
plt.xticks=param_range
#plt.show()

param_grid={'criterion':['gini','entropy'],
            'max_depth':[2,5,10,20],
            'max_features':[2,4,6,'auto'],
            'max_leaf_nodes':[2,3,None],}

grid=GridSearchCV(estimator=RandomForestClassifier(n_estimators=50,n_jobs=-
1,random_state=51),
                  #param_grid=param_grid,cv=10,n_jobs=-1)
grid.fit(X_train,y_train)
print(grid.best_params_)
print(grid.best_score_)

svm = SVC(kernel = 'poly', probability = True, gamma = 'auto' , C = 1)
svm.fit(X_train, y_train)
y_predictions = svm.predict(X_test)
print("SVM Accuracy:",svm.score(X_test, y_test))
print("SVM F1-score:",f1_score(y_test,svm.predict(X_test),average='macro'))
#print(classification_report(y_test,y_predictions))
c_matrix = confusion_matrix(y_test,y_predictions)
print ("confusion matrix:")
```

```
print (c_matrix)
plt.matshow(c_matrix)

params_model = {'C':[0.1,10],'gamma':[1,0.1,0.01,0.001,0.0001]}

model = GridSearchCV(SVC(probability=True),
                    params_model,
                    refit=True,
                    return_train_score=True,
                    cv = 5)
model.fit(X_train, y_train)
print(model.best_params_)
print(model.best_score_)

param_range=[0.1,1,10]
curve=validation_curve( SVC(kernel = 'poly', probability = True, gamma =
'auto'),X_train,y_train,cv=5,param_name='C',param_range=param_range,n_jobs=-1)

train_score=[curve[0][i].mean() for i in range (0,len(param_range))]
test_score=[curve[1][i].mean() for i in range (0,len(param_range))]
fig=plt.figure(figsize=(6,8))
plt.plot(param_range,train_score)
plt.plot(param_range,test_score)
plt.xticks=param_range
plt.show()
```