

Int 247

Roll no 4

Registration number 11701145

Pulkit matta

Project Report

Question: Predict the grade of student using different classification techniques
Bachelor of Science (Forensic Science)...

Dataset: Basic Insight

Dataset provided by faculty was a real life data of marks of different branches of students including many other useful information (for my question) and other less useful information. Dataset also included missing values. And required preprocessing (I have included dataset with my project submission).

Compiler used: Spyder

Step Wise Strategy Opted For Solution

1) Extracting (Forensic Science) data from complete dataset

- a) Getting Basic Insight
 - i) Type of features
 - ii) Shape
 - iii) Description
 - iv) Number of missing values

2) Preprocessing -:

- a) Null value handing
- b) Finding categorical data(nunique)
- c) Label encoding (only categorical data)
- d) Standard Scaling (only Remaining features)

- e) Finding Correlation between all features and grades(for pruning non-essential features)
- f) Splitting set into target and data

3) Classification

- a) Splitting data into train and test
- b) Trying different classification algorithm while documenting effects of change in hyper parameters of each
- c) Keeping record of accuracy of all the algorithms
- d) Graphical representation of the best algorithm with best hyper parameter settings according to the question.

Library + Classes Used For Solution

- **Pandas** –: for handling dataset/ analysis/ deletion
- **Numpy**:- for in built features like sum (), array (), arrange () ...etc.
- **Sklearn** :- for classes like SimpleImputer, LabelEncoder, StandardScaler, train_test_split(metrics), naive_bayes, metrics(confusion_matrix, accuracy_score), tree, neighbors(KNeighborsClassifier), ensemble(BaggingClassifier, AdaBoostClassifier, RandomForestClassifier), linear_model(LogisticRegression), svm(SVC)
- **Matplotlib**:- for representing data graphically for analysis.
- **Seaborn**:- for making complex graphical plot for decision regarding data

Code explanation with reasons / output screenshot

And

Result visualization

Key: hyper parameter(value)->(accuracy) ,

hyperparameterSetting1>> hyperparameterSetting2: means 1 performed better with much higher accuracy)

- **Basic features of data**

Dataset consisted of different branches from which I extracted useful data Bachelor of Science (Forensic Science)

Extracting data basic features

```
1 print(".....")
2 print("Extracting Data (Bachelor Of Science(Forensic) )") #data extraction
3 import pandas as pd
4 datasetC=pd.read_csv("G:/1/project.csv")
5 options = ['Bachelor of Science (Forensic Science)']
6 dataset = datasetC[datasetC['MHRDName'].isin(options)]
7 print("Size of extracted dataset ",dataset.shape)# getting basic information from dataset
8 print("\n\n");
9
10
11 #####preprocessing#####
12
13
14 print("\n\n Preprocessing.....")
15 print(dataset.describe())
16 print(dataset.iloc[:10,:])
17 print("Number Of Missing Data In Every Feature ")
18 import numpy as np
19 print(pd.isnull(dataset).sum()) # getting null values
20
21 # removing rows with less than 30 % information (features)
22 print("\n\n As Dataset is Comparatively small dropping only those tuples which has less than 30% information")
23 dataset.dropna(thresh=dataset.shape[1]*.30,inplace=True,axis=0)
24
25 # imputing null values
26 print("\n\n Imputing values in others ")
27 from sklearn.impute import SimpleImputer
28 si=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
29 dataset=pd.DataFrame(si.fit_transform(dataset))
30
31 # data is now complete
32 print("\n\n Number Of Missing Data In Every Feature ")
33 import numpy as np
34 print(pd.isnull(dataset).sum())
35
36 # Label encoding data with unique values <=30 (Categorical)
37 print("\n\n Label Encoding Non Continuous Data :")
```

IPython console output:

```
[10 rows x 22 columns]
Number Of Missing Data In Every Feature
Termid      0
Regd No     0
Course      0
Grade       0
CA_100      1
MTT_50      109
ETT_100     109
ETP_100     257
Course_Att  21
MHRDName    0
CA_1         1
CA_2         1
CA_3         1
CA_4         1
Height       0
Weight       0
ScholarType  0
Direction    0
Gender       0
Medium       0
CourseType   0
ProgramType  0
dtype: int64
```

In [3]:

- **Preprocessing data**

removing rows with less than 30 % information (features)

imputing null values

Label encoding data with nunique values <=30 (Categorical)

dropping data with only 1 data like every student is from forensic science

scaling remain values (Non Categorical Features)

The screenshot shows the Spyder Python IDE with a script in the editor and the IPython console output.

Editor - G:\1\pro.py

```

26 print("\n Imputing values in others ")
27 from sklearn.impute import SimpleImputer
28 si=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
29 dataset=pd.DataFrame(si.fit_transform(dataset))
30
31 # data is now complete
32 print("\n\n Number Of Missing Data In Every Feature ")
33 import numpy as np
34 print(pd.isnull(dataset).sum())
35
36 # Label encoding data with nunique values <=30 (Categorical)
37 print('\n\n Label Encoding Non Continuous Data :')
38 from sklearn.preprocessing import LabelEncoder
39 lb=LabelEncoder()
40 for i in range(dataset.shape[1]):
41     if(dataset.iloc[:,i].nunique())<=30:
42         print(i)
43         dataset.iloc[:,i]=lb.fit_transform( dataset.iloc[:,i])
44
45
46 # dropping data with only 1 data like every student is from forensic science
47 print('\n\n Removing all single valued columns.....')
48 column=[]
49 for i in range(dataset.shape[1]):
50     if(dataset.iloc[:,i].nunique())==1:
51         print(i)
52         column.append(i)
53
54 dataset.drop(columns=column,inplace=True,axis=1)
55
56
57 # scaling remain values
58 print("\n\n Standard Scalling Dataset..... Except Categorical Values")
59 from sklearn.preprocessing import StandardScaler
60 sc=StandardScaler()
61 for i in range(dataset.shape[1]):
62     if(dataset.iloc[:,i].nunique())>30:

```

IPython console

```

Number Of Missing Data In Every Feature
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
20 0
21 0
dtype: int64

Label Encoding Non Continuous Data :
0
2
3
9
14
16
17
18
19

```

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 49 Column: 34 Memory: 52%

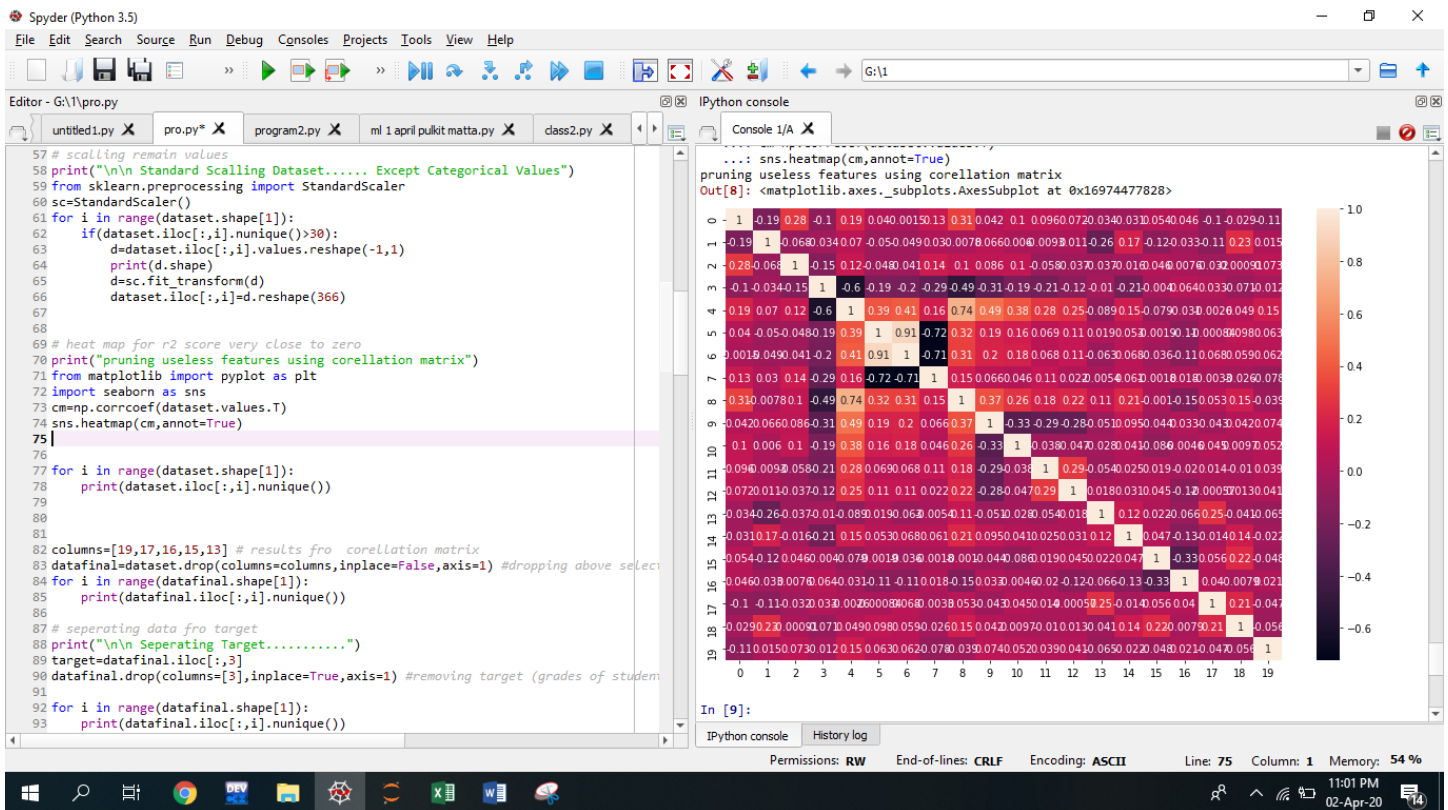
```

[8 rows x 20 columns]
   0    1    2    3    4    5    ...    15    16    17    18    19    20
0  1 -1.976381    4    2 -0.253121  0.458265  ... -0.374231    1    2    0    0    1
1  1 -1.976381    8    0  0.167216  1.198773  ... -0.374231    1    2    0    0    1
2  1 -1.976381   12    0  1.049925  1.075355  ... -0.374231    1    2    0    0    1
3  1 -1.976381   13    0  0.419419 -1.207877  ... -0.374231    1    2    0    0    1
4  1 -1.976381   14    0 -0.127020 -1.207877  ... -0.374231    1    2    0    0    1
5  1 -1.976381   15    0  0.545520  1.137064  ... -0.374231    1    2    0    0    1
6  1 -1.976381   16    0  0.377385  0.951937  ... -0.374231    1    2    0    0    1
7  1 -1.976381   17    0  0.755689  0.766810  ... -0.374231    1    2    0    0    1
8  1 -1.976381   18    1  0.755689  1.383900  ... -0.374231    1    2    0    0    1
9  2 -1.937173   20    0  0.503486 -1.207877  ... -1.640019    1    1    0    0    1

```

removing redundant features

Heat map for correlation coefficient to find redundant features



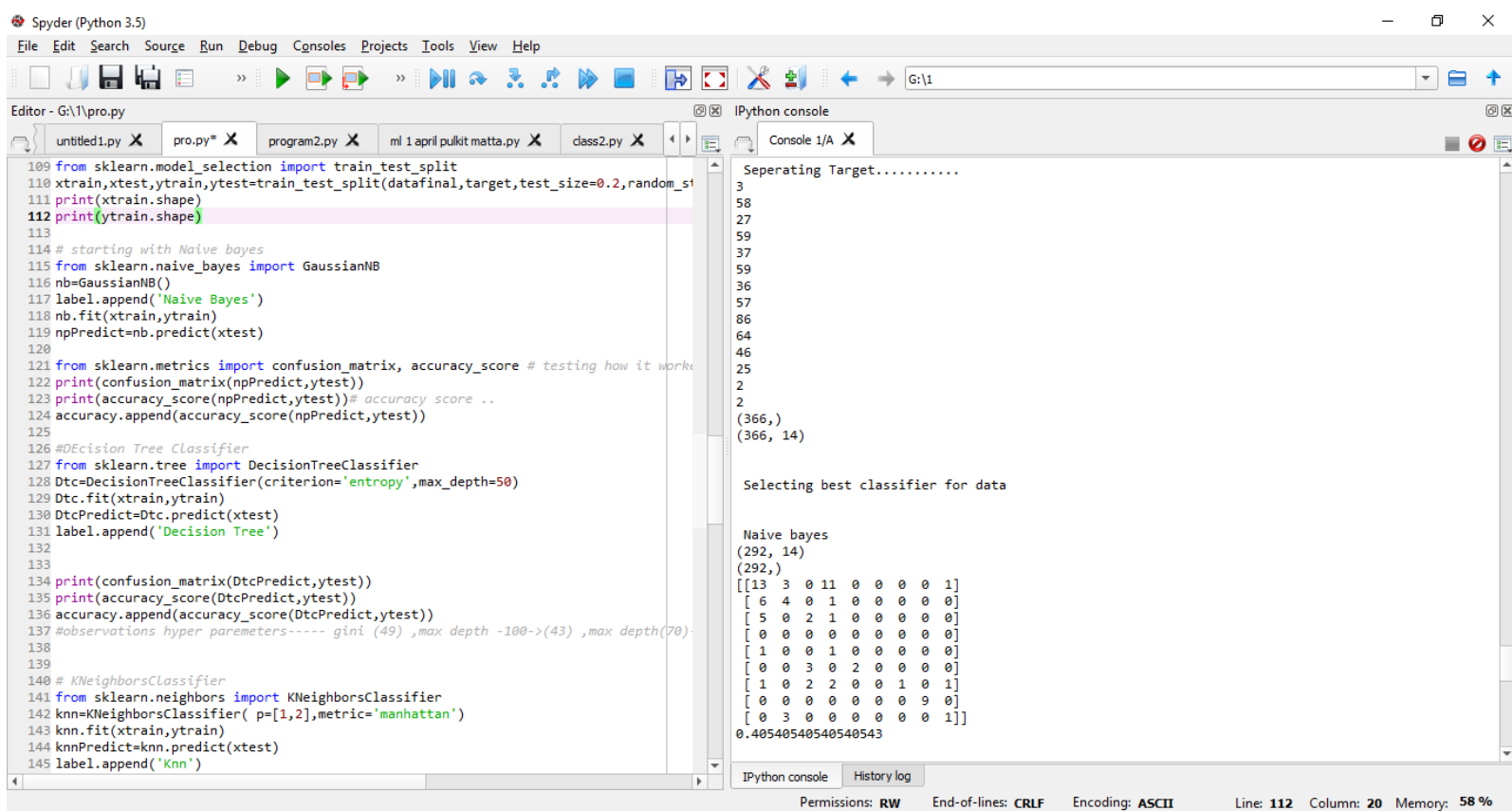
As I discovered from heat map correlation coefficient of features like height, program type, medium gender, direction, scholar type are very close to zero when compared with feature number 3 which are grades which means

- they contribute very little in classification
- but increase over fitting
- therefore I removed them from dataset
- interesting fact: weight of the person was actually related to grades more than above features

```
# separating data from target
# testing Different classification algorithms
#splitting data
```

- **Classification**

1. Starting with Naive Bayes



The screenshot shows the Spyder Python IDE with a script named `pro.py` and its output in the IPython console.

Script Code (pro.py):

```
109 from sklearn.model_selection import train_test_split
110 xtrain,xtest,ytrain,ytest=train_test_split(datafinal,target,test_size=0.2,random_st
111 print(xtrain.shape)
112 print(ytrain.shape)
113
114 # starting with Naive bayes
115 from sklearn.naive_bayes import GaussianNB
116 nb=GaussianNB()
117 label.append('Naive Bayes')
118 nb.fit(xtrain,ytrain)
119 npPredict=nb.predict(xtest)
120
121 from sklearn.metrics import confusion_matrix, accuracy_score # testing how it work
122 print(confusion_matrix(npPredict,ytest))
123 print(accuracy_score(npPredict,ytest))# accuracy score ..
124 accuracy.append(accuracy_score(npPredict,ytest))
125
126 #DEcision Tree Classifier
127 from sklearn.tree import DecisionTreeClassifier
128 Dtc=DecisionTreeClassifier(criterion='entropy',max_depth=50)
129 Dtc.fit(xtrain,ytrain)
130 DtcPredict=Dtc.predict(xtest)
131 label.append('Decision Tree')
132
133
134 print(confusion_matrix(DtcPredict,ytest))
135 print(accuracy_score(DtcPredict,ytest))
136 accuracy.append(accuracy_score(DtcPredict,ytest))
137 #observations hyper parameters----- gini (49) ,max depth -100->(43) ,max depth(70)
138
139
140 # KNeighborsClassifier
141 from sklearn.neighbors import KNeighborsClassifier
142 knn=KNeighborsClassifier( p=[1,2],metric='manhattan')
143 knn.fit(xtrain,ytrain)
144 knnPredict=knn.predict(xtest)
145 label.append('Knn')
```

IPython console output:

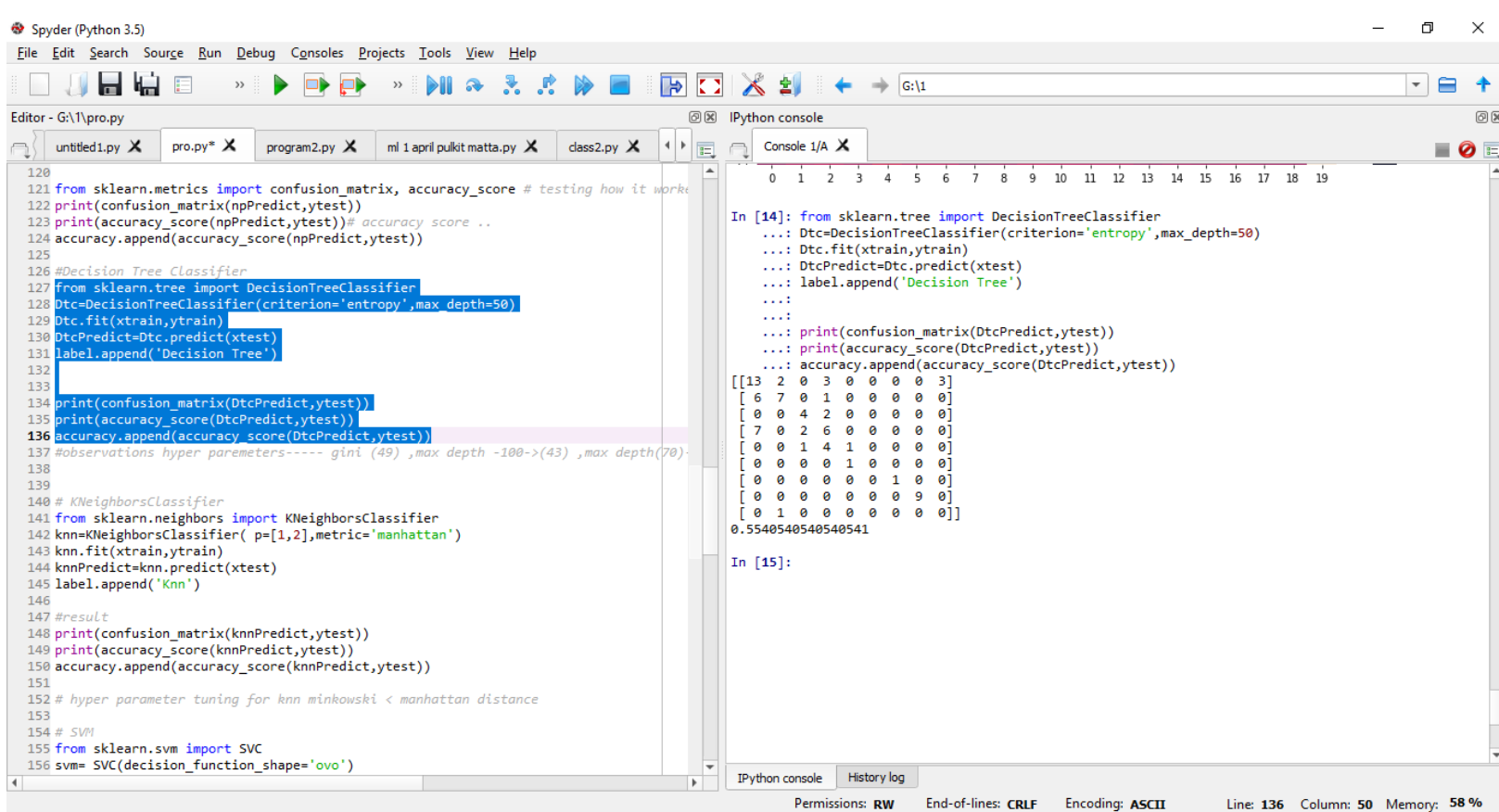
```
Seperating Target.....
3
58
27
59
37
59
36
57
86
64
46
25
2
2
(366,)
(366, 14)

Selecting best classifier for data

Naive bayes
(292, 14)
(292,)
[[13  3  0 11  0  0  0  0  1]
 [ 6  4  0  1  0  0  0  0  0]
 [ 5  0  2  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0]
 [ 1  0  0  1  0  0  0  0  0]
 [ 0  0  3  0  2  0  0  0  0]
 [ 1  0  2  2  0  0  1  0  1]
 [ 0  0  0  0  0  0  0  9  0]
 [ 0  3  0  0  0  0  0  0  1]]
0.40540540540540543
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 112, Column: 20, Memory: 58 %.

2. Decision Tree Classifier



```
120 from sklearn.metrics import confusion_matrix, accuracy_score # testing how it works
121 print(confusion_matrix(npPredict,ytest))
122 print(accuracy_score(npPredict,ytest)) # accuracy score ..
123 accuracy.append(accuracy_score(npPredict,ytest))
124
125 #Decision Tree Classifier
126 from sklearn.tree import DecisionTreeClassifier
127 Dtc=DecisionTreeClassifier(criterion='entropy',max_depth=50)
128 Dtc.fit(xtrain,ytrain)
129 DtcPredict=Dtc.predict(xtest)
130 label.append('Decision Tree')
131
132
133
134 print(confusion_matrix(DtcPredict,ytest))
135 print(accuracy_score(DtcPredict,ytest))
136 accuracy.append(accuracy_score(DtcPredict,ytest))
137 #observations hyper parameters----- gini (49) ,max depth -100->(43) ,max depth(70)-
138
139
140 # KNeighborsClassifier
141 from sklearn.neighbors import KNeighborsClassifier
142 knn=KNeighborsClassifier( p=[1,2],metric='manhattan')
143 knn.fit(xtrain,ytrain)
144 knnPredict=knn.predict(xtest)
145 label.append('Knn')
146
147 #result
148 print(confusion_matrix(knnPredict,ytest))
149 print(accuracy_score(knnPredict,ytest))
150 accuracy.append(accuracy_score(knnPredict,ytest))
151
152 # hyper parameter tuning for knn minkowski < manhattan distance
153
154 # SVM
155 from sklearn.svm import SVC
156 svm= SVC(decision_function_shape='ovo')
```

```
In [14]: from sklearn.tree import DecisionTreeClassifier
...: Dtc=DecisionTreeClassifier(criterion='entropy',max_depth=50)
...: Dtc.fit(xtrain,ytrain)
...: DtcPredict=Dtc.predict(xtest)
...: label.append('Decision Tree')
...:
...: print(confusion_matrix(DtcPredict,ytest))
...: print(accuracy_score(DtcPredict,ytest))
...: accuracy.append(accuracy_score(DtcPredict,ytest))

[[13  2  0  3  0  0  0  0  3]
 [ 6  7  0  1  0  0  0  0  0]
 [ 0  4  2  0  0  0  0  0  0]
 [ 7  0  2  6  0  0  0  0  0]
 [ 0  1  4  1  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  0  0  9  0  0]
 [ 0  1  0  0  0  0  0  0  0]]
0.5540540540540541

In [15]:
```

IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 136 Column: 50 Memory: 58 %

#observations hyper parameters----- gini ->(49) ,max depth -100->(43) , max depth(70)->44 , max depth(49), using entropy ->(50)

3. KNeighborsClassifier

hyper parameter tuning for knn minkowski < manhattan distance

```
139
140 # KNeighborsClassifier
141 from sklearn.neighbors import KNeighborsClassifier
142 knn=KNeighborsClassifier( p=[1,2],metric='manhattan')
143 knn.fit(xtrain,ytrain)
144 knnPredict=knn.predict(xtest)
145 label.append('Knn')
146
147 #result
148 print(confusion_matrix(knnPredict,ytest))
149 print(accuracy_score(knnPredict,ytest))
150 accuracy.append(accuracy_score(knnPredict,ytest))
151
152 # hyper parameter tuning for knn minkowski < manhattan distance
153
154 # SVM
155 from sklearn.svm import SVC
156 svm= SVC(decision_function_shape='ovo')
157 svm.fit(xtrain,ytrain)
158 svmPredict=svm.predict(xtest)
159 label.append('SVM')
160
161 #result
162 print(confusion_matrix(svmPredict,ytest))
163 print(accuracy_score(svmPredict,ytest))
164 accuracy.append(accuracy_score(svmPredict,ytest))
165
166 # hyperparameter tuning result using decision_function_shape='ovo' because of mult
167
168 #Logistic Regression
169 from sklearn.linear_model import LogisticRegression
170 lg= LogisticRegression(multi_class='auto',solver='lbfgs')
171 lg.fit(xtrain,ytrain)
172 lgPredict=lg.predict(xtest)
173 label.append('Log. Regression')
174
175
```

```
classification
...:
...: #Logistic Regression
...: from sklearn.linear_model import LogisticRegression
...: lg= LogisticRegression(multi_class='auto',solver='lbfgs')
...: lg.fit(xtrain,ytrain)
...: lgPredict=lg.predict(xtest)
...: label.append('Log. Regression')
[[13 2 3 9 0 0 0 0 2]
 [4 4 0 0 0 0 0 0 1]
 [1 1 2 3 2 0 0 0 0]
 [8 2 2 4 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 9 0]
 [0 0 0 0 0 0 0 0 0]]
0.43243243243243246
[[10 6 2 9 2 1 2 2]
 [5 3 1 2 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [10 1 4 5 0 0 0 1]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 7 0]
 [0 0 0 0 0 0 0 0]]
0.33783783783783783
c:\users\acer\anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default
value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
"avoid this warning.", FutureWarning)
c:\users\acer\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:757:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
"of iterations.", ConvergenceWarning)

In [17]:
```

IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 174 Column: 1 Memory: 58 %

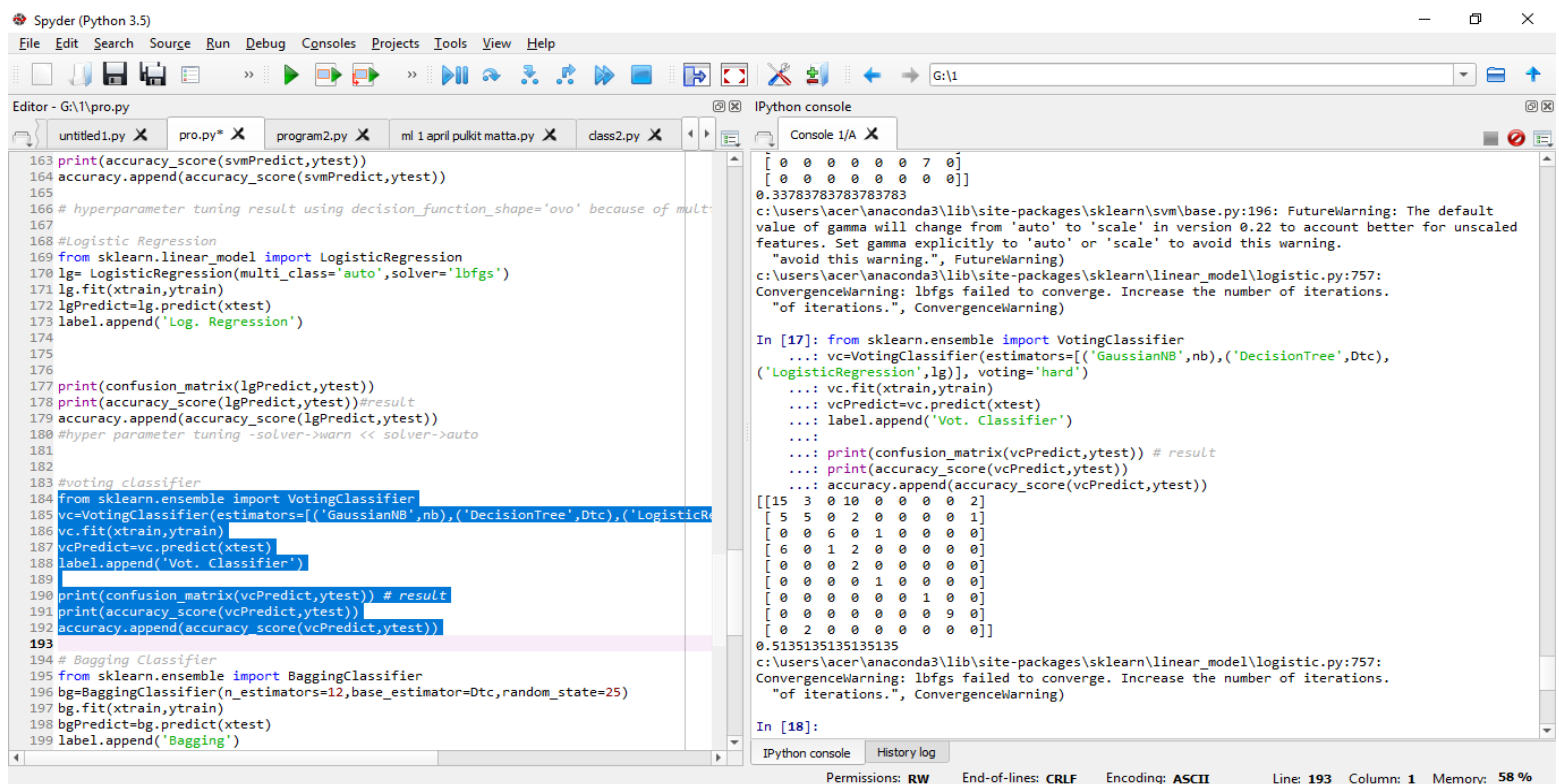
4. SVM

hyperparameter tuning result using `decision_function_shape='ovo'` because of multiclass classification

#Logistic Regression

#hyper parameter tuning -solver->warn << solver->auto

1) voting classifier



```
163 print(accuracy_score(svmPredict,ytest))
164 accuracy.append(accuracy_score(svmPredict,ytest))
165
166 # hyperparameter tuning result using decision_function_shape='ovo' because of mult
167
168 #Logistic Regression
169 from sklearn.linear_model import LogisticRegression
170 lg= LogisticRegression(multi_class='auto',solver='lbfgs')
171 lg.fit(xtrain,ytrain)
172 lgPredict=lg.predict(xtest)
173 label.append('Log. Regression')
174
175
176
177 print(confusion_matrix(lgPredict,ytest))
178 print(accuracy_score(lgPredict,ytest))#result
179 accuracy.append(accuracy_score(lgPredict,ytest))
180 #hyper parameter tuning -solver->warn << solver->auto
181
182
183 #voting classifier
184 from sklearn.ensemble import VotingClassifier
185 vc=VotingClassifier(estimators=[('GaussianNB',nb),('DecisionTree',Dtc),('LogisticR
186 vc.fit(xtrain,ytrain)
187 vcPredict=vc.predict(xtest)
188 label.append('Vot. Classifier')
189
190 print(confusion_matrix(vcPredict,ytest)) # result
191 print(accuracy_score(vcPredict,ytest))
192 accuracy.append(accuracy_score(vcPredict,ytest))
193
194 # Bagging Classifier
195 from sklearn.ensemble import BaggingClassifier
196 bg=BaggingClassifier(n_estimators=12,base_estimator=Dtc,random_state=25)
197 bg.fit(xtrain,ytrain)
198 bgPredict=bg.predict(xtest)
199 label.append('Bagging')
```

```
[ 0 0 0 0 0 0 7 0]
[ 0 0 0 0 0 0 0 0]]
0.33783783783783783
c:\users\acer\anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default
value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
"avoid this warning.", FutureWarning)
c:\users\acer\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:757:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
"of iterations.", ConvergenceWarning)

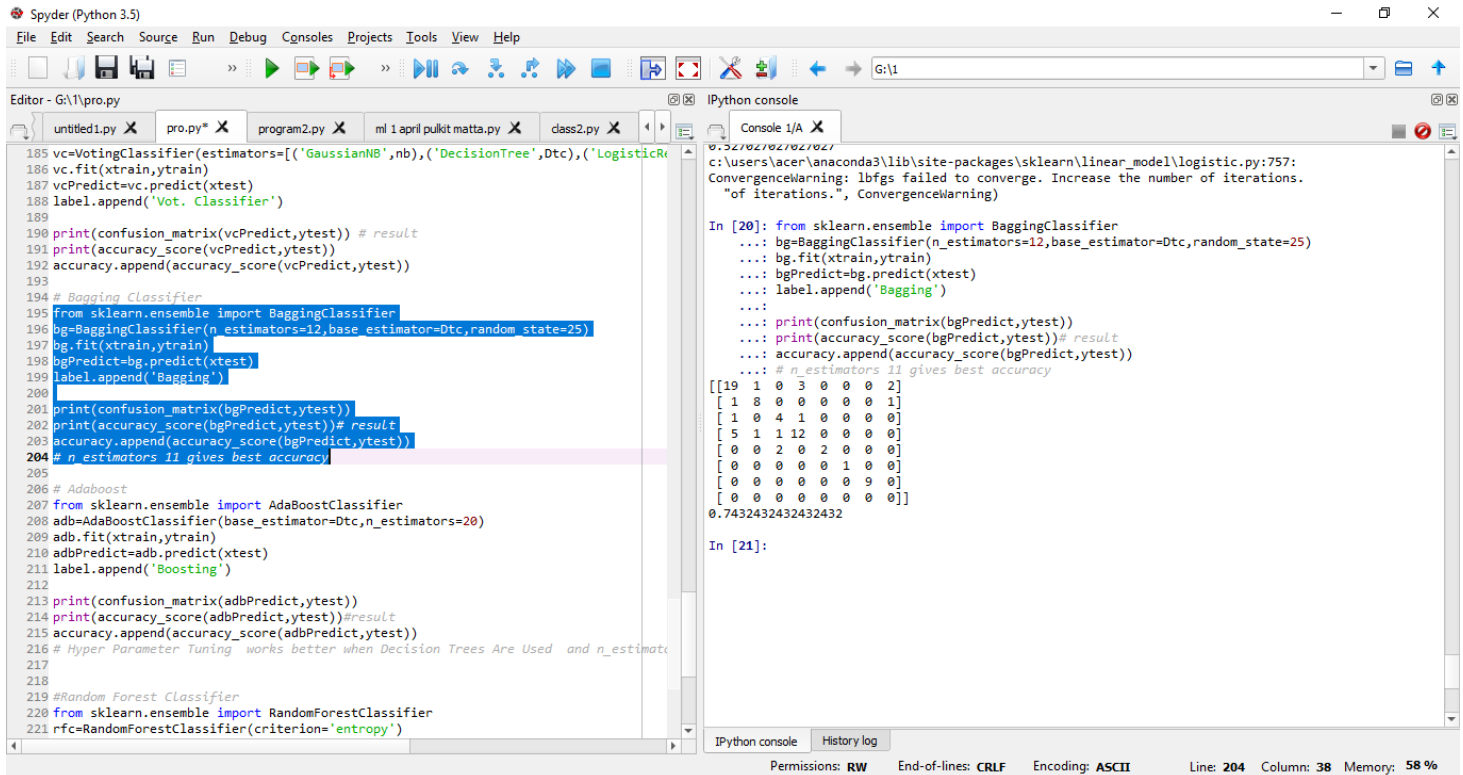
In [17]: from sklearn.ensemble import VotingClassifier
... vc=VotingClassifier(estimators=[('GaussianNB',nb),('DecisionTree',Dtc),
('LogisticRegression',lg)], voting='hard')
... vc.fit(xtrain,ytrain)
... vcPredict=vc.predict(xtest)
... label.append('Vot. Classifier')
...
... print(confusion_matrix(vcPredict,ytest)) # result
... print(accuracy_score(vcPredict,ytest))
... accuracy.append(accuracy_score(vcPredict,ytest))

[[15 3 0 10 0 0 0 0 2]
 [ 5 5 0 2 0 0 0 0 1]
 [ 0 0 6 0 1 0 0 0 0]
 [ 6 0 1 2 0 0 0 0 0]
 [ 0 0 0 2 0 0 0 0 0]
 [ 0 0 0 0 1 0 0 0 0]
 [ 0 0 0 0 0 0 1 0 0]
 [ 0 0 0 0 0 0 0 0 9]
 [ 0 2 0 0 0 0 0 0 0]]
0.5135135135135135
c:\users\acer\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:757:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
"of iterations.", ConvergenceWarning)

In [18]:
```

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 193 Column: 1 Memory: 58 %

5. Bagging Classifier



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script for training and evaluating a Bagging Classifier. The code includes imports for VotingClassifier, BaggingClassifier, AdaBoostClassifier, and RandomForestClassifier. It defines a list of estimators, fits a voting classifier, and then fits a bagging classifier with 12 estimators. The output in the IPython console shows the confusion matrix and accuracy score for the bagging classifier, indicating a best accuracy of 0.7432432432432432.

```
185 vc=VotingClassifier(estimators=[('GaussianNB',nb),('DecisionTree',Dtc),('LogisticR
186 vc.fit(xtrain,ytrain)
187 vcPredict=vc.predict(xtest)
188 label.append('Vot. Classifier')
189
190 print(confusion_matrix(vcPredict,ytest)) # result
191 print(accuracy_score(vcPredict,ytest))
192 accuracy.append(accuracy_score(vcPredict,ytest))
193
194 # Bagging Classifier
195 from sklearn.ensemble import BaggingClassifier
196 bg=BaggingClassifier(n_estimators=12,base_estimator=Dtc,random_state=25)
197 bg.fit(xtrain,ytrain)
198 bgPredict=bg.predict(xtest)
199 label.append('Bagging')
200
201 print(confusion_matrix(bgPredict,ytest))
202 print(accuracy_score(bgPredict,ytest))# result
203 accuracy.append(accuracy_score(bgPredict,ytest))
204 # n_estimators 11 gives best accuracy
205
206 # Adaboost
207 from sklearn.ensemble import AdaBoostClassifier
208 adb=AdaBoostClassifier(base_estimator=Dtc,n_estimators=20)
209 adb.fit(xtrain,ytrain)
210 adbPredict=adb.predict(xtest)
211 label.append('Boosting')
212
213 print(confusion_matrix(adbPredict,ytest))
214 print(accuracy_score(adbPredict,ytest))#result
215 accuracy.append(accuracy_score(adbPredict,ytest))
216 # Hyper Parameter Tuning works better when Decision Trees Are Used and n_estimators
217
218
219 #Random Forest Classifier
220 from sklearn.ensemble import RandomForestClassifier
221 rfc=RandomForestClassifier(criterion='entropy')
```

```
0.521021021021021
c:\users\acer\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:757:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)

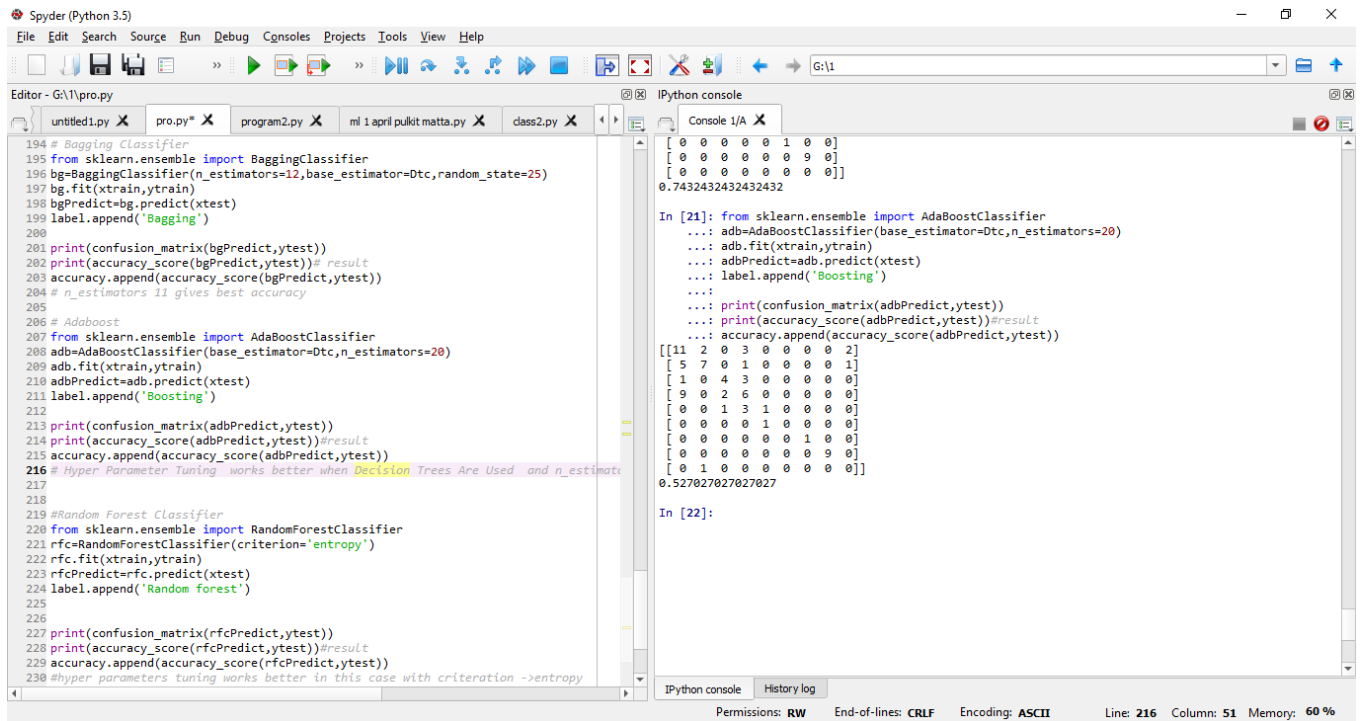
In [20]: from sklearn.ensemble import BaggingClassifier
...: bg=BaggingClassifier(n_estimators=12,base_estimator=Dtc,random_state=25)
...: bg.fit(xtrain,ytrain)
...: bgPredict=bg.predict(xtest)
...: label.append('Bagging')
...:
...: print(confusion_matrix(bgPredict,ytest))
...: print(accuracy_score(bgPredict,ytest))# result
...: accuracy.append(accuracy_score(bgPredict,ytest))
...: # n_estimators 11 gives best accuracy
[[19  1  0  3  0  0  0  2]
 [ 1  8  0  0  0  0  0  1]
 [ 1  0  4  1  0  0  0  0]
 [ 5  1  1 12  0  0  0  0]
 [ 0  0  2  0  2  0  0  0]
 [ 0  0  0  0  0  1  0  0]
 [ 0  0  0  0  0  0  9  0]
 [ 0  0  0  0  0  0  0  0]]
0.7432432432432432

In [21]:
```

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 204 Column: 38 Memory: 58 %

#hyperparameter tuning result: - n_estimators (11) ->(74.32)

6. Adaboost



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script with the following code:

```
194 # Bagging Classifier
195 from sklearn.ensemble import BaggingClassifier
196 bg=BaggingClassifier(n_estimators=12,base_estimator=Dtc,random_state=25)
197 bg.fit(xtrain,ytrain)
198 bgPredict=bg.predict(xtest)
199 label.append('Bagging')
200
201 print(confusion_matrix(bgPredict,ytest))
202 print(accuracy_score(bgPredict,ytest))# result
203 accuracy.append(accuracy_score(bgPredict,ytest))
204 # n_estimators 11 gives best accuracy
205
206 # Adaboost
207 from sklearn.ensemble import AdaBoostClassifier
208 adb=AdaBoostClassifier(base_estimator=Dtc,n_estimators=20)
209 adb.fit(xtrain,ytrain)
210 adbPredict=adb.predict(xtest)
211 label.append('Boosting')
212
213 print(confusion_matrix(adbPredict,ytest))
214 print(accuracy_score(adbPredict,ytest))#result
215 accuracy.append(accuracy_score(adbPredict,ytest))
216 # Hyper Parameter Tuning works better when Decision Trees Are Used and n_estimators
217
218
219 #Random Forest Classifier
220 from sklearn.ensemble import RandomForestClassifier
221 rfc=RandomForestClassifier(criterion='entropy')
222 rfc.fit(xtrain,ytrain)
223 rfcPredict=rfc.predict(xtest)
224 label.append('Random Forest')
225
226
227 print(confusion_matrix(rfcPredict,ytest))
228 print(accuracy_score(rfcPredict,ytest))#result
229 accuracy.append(accuracy_score(rfcPredict,ytest))
230 #hyper parameters tuning works better in this case with criterion ->entropy
```

The IPython console shows the output of the code:

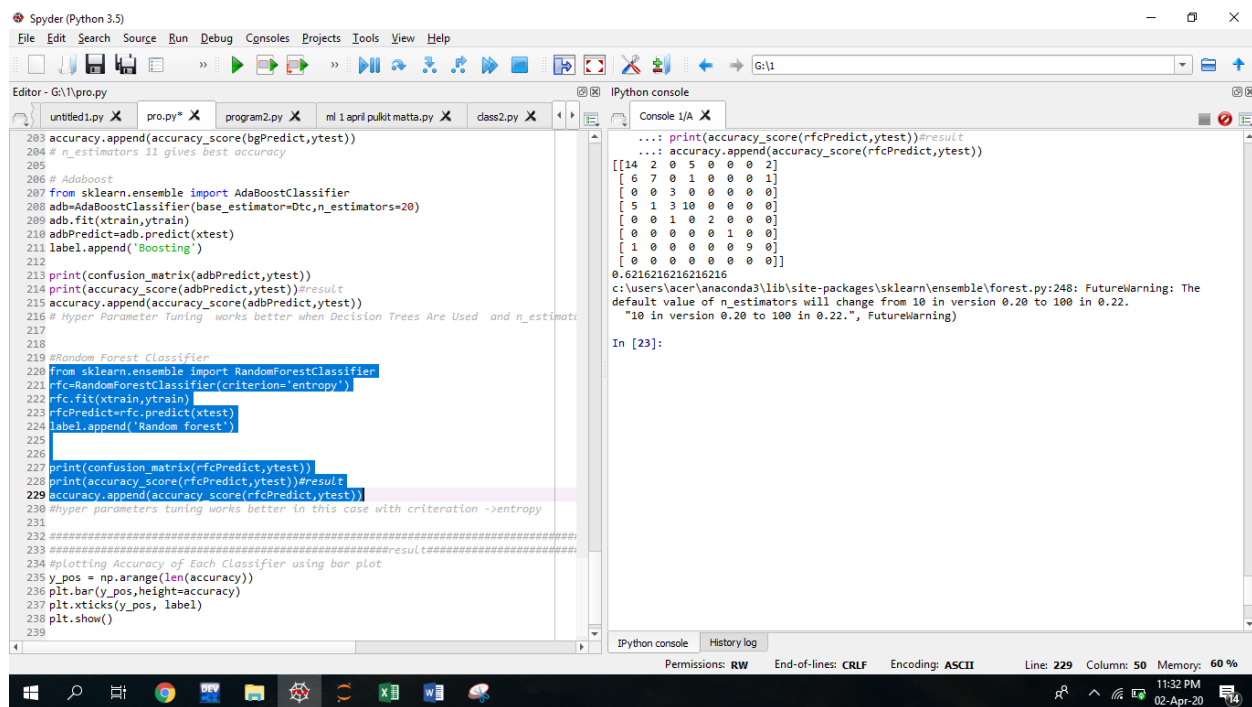
```
In [21]: from sklearn.ensemble import AdaBoostClassifier
...: adb=AdaBoostClassifier(base_estimator=Dtc,n_estimators=20)
...: adb.fit(xtrain,ytrain)
...: adbPredict=adb.predict(xtest)
...: label.append('Boosting')
...:
...: print(confusion_matrix(adbPredict,ytest))
...: print(accuracy_score(adbPredict,ytest))#result
...: accuracy.append(accuracy_score(adbPredict,ytest))
[[11 2 0 3 0 0 0 2]
 [5 7 0 1 0 0 0 1]
 [1 0 4 3 0 0 0 0]
 [9 0 2 6 0 0 0 0]
 [0 0 1 3 1 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 9 0]
 [0 1 0 0 0 0 0 0]]
0.527027027027027

In [22]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 216, Column: 51, Memory: 60%.

Hyper Parameter Tuning works better when Decision Trees Are Used and `n_estimators (20)`->52.75

7. Random Forest Classifier



The screenshot shows the Spyder Python IDE with a file named 'pro.py' open. The code implements two classifiers: AdaBoost and Random Forest. The AdaBoost section (lines 203-218) uses `AdaBoostClassifier` with `base_estimator=Dtc` and `n_estimators=20`. The Random Forest section (lines 219-229) uses `RandomForestClassifier` with `criterion='entropy'`. Both classifiers are trained on `xtrain, ytrain` and tested on `xtest, ytest`. The accuracy of each classifier is printed and stored in a list. A bar plot is generated to compare the accuracies of the two classifiers. The IPython console shows the output of the accuracy scores and a warning about the default value of `n_estimators` changing from 10 to 100 in version 0.22.

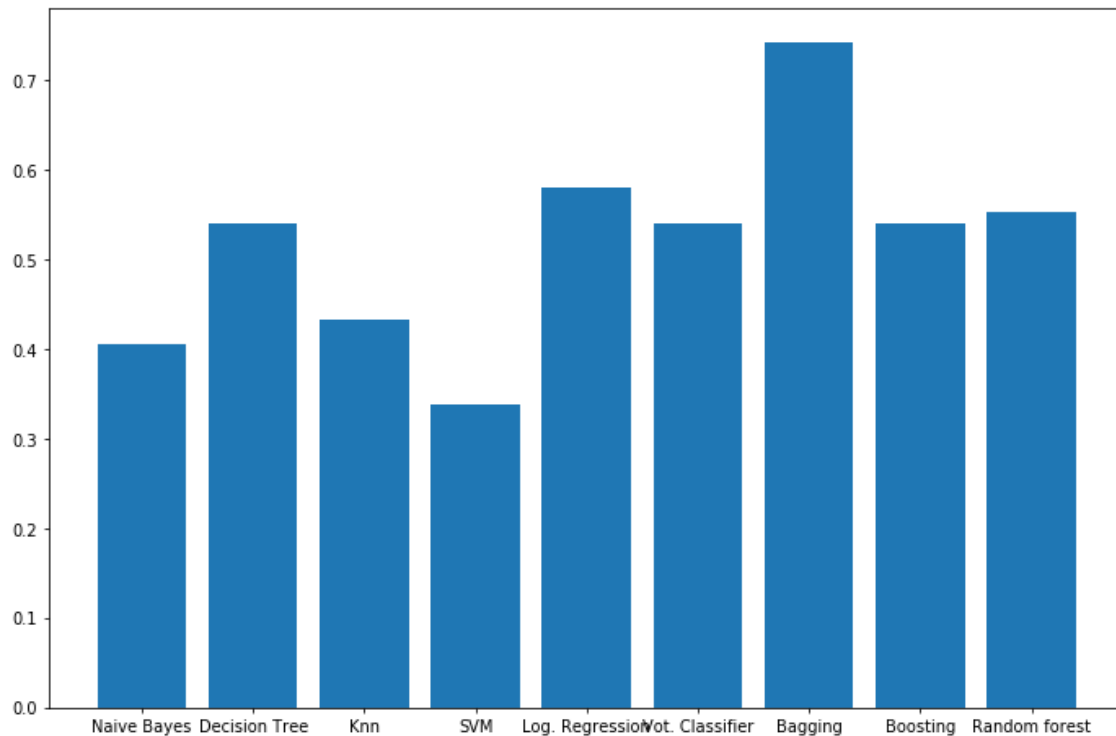
```
203 accuracy.append(accuracy_score(bgPredict,ytest))
204 # n_estimators 11 gives best accuracy
205
206 # Adaboost
207 from sklearn.ensemble import AdaBoostClassifier
208 adb=AdaBoostClassifier(base_estimator=Dtc,n_estimators=20)
209 adb.fit(xtrain,ytrain)
210 adbPredict=adb.predict(xtest)
211 label.append('Boosting')
212
213 print(confusion_matrix(adbPredict,ytest))
214 print(accuracy_score(adbPredict,ytest))#result
215 accuracy.append(accuracy_score(adbPredict,ytest))
216 # Hyper Parameter Tuning works better when Decision Trees Are Used and n_estimators
217
218
219 #Random Forest Classifier
220 from sklearn.ensemble import RandomForestClassifier
221 rfc=RandomForestClassifier(criterion='entropy')
222 rfc.fit(xtrain,ytrain)
223 rfcPredict=rfc.predict(xtest)
224 label.append('Random Forest')
225
226
227 print(confusion_matrix(rfcPredict,ytest))
228 print(accuracy_score(rfcPredict,ytest))#result
229 accuracy.append(accuracy_score(rfcPredict,ytest))
230 #hyper parameters tuning works better in this case with criterion ->entropy
231
232 #####
233 #####result#####
234 #Plotting Accuracy of Each Classifier using bar plot
235 y_pos = np.arange(len(accuracy))
236 plt.bar(y_pos,height=accuracy)
237 plt.xticks(y_pos, label)
238 plt.show()
239
```

```
...: print(accuracy_score(rfcPredict,ytest))#result
...: accuracy.append(accuracy_score(rfcPredict,ytest))
[[14  2  0  5  0  0  0  2]
 [ 6  7  0  1  0  0  0  1]
 [ 0  0  3  0  0  0  0  0]
 [ 5  1  3 10  0  0  0  0]
 [ 0  0  1  0  2  0  0  0]
 [ 0  0  0  0  1  0  0  0]
 [ 1  0  0  0  0  0  9  0]
 [ 0  0  0  0  0  0  0  0]]
0.6216216216216216
c:\users\acer\anaconda3\lib\site-packages\sklearn\ensemble\forest.py:248: FutureWarning: The
default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

In [23]:
```

#hyper parameters tuning works better in this case with criterion ->entropy as compared to gini

```
...: plt.show()
```



Accuracy graph

Final result + Observations

- According to the no free lunch theorem there is no universal classifier that can classify every problem as we observed that for this problem bagging is a better option .
- as I saw this in my project by iterating it many times with different setting of different models that minor settings can have a deep impact on results.
- This graphical result depicts that bagging performed better than every other classifier when (decision tree was used as an estimator for it)
- After tuning dataset I discovered that height, program type, medium gender, direction, scholar type, were responsible for over fitting for result as they were have correlation coefficient ~ 0 with grade of student . performance was much better after their removal
- From study of different truth of algorithm I observed that dataset was a little biased towards the grade A+.

- Scaling of non-categorical data helped graph like heat map faster