kendall

April 26, 2022

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
sns.set_style('whitegrid')
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')

import os

import time
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import LeaveOneOut as loocv

from plotly import tools
from plotly.offline import plot
import plotly.offline as py
from plotly.graph_objs import Scatter, Layout
import plotly.graph_objs as go
import plotly.figure_factory as ff
```

[2]:
```
# load in training and test data
train = pd.read_csv('training.csv')
test = pd.read_csv('testing.csv')
print("Rows and Columns(Train): ",train.shape)
print("Rows and Columns(Test) : ",test.shape)
```

```
Rows and Columns(Train):  (168, 148)
Rows and Columns(Test) :  (507, 148)
```

[3]:
```
# check for missing values although it is clear there are none
train.isnull().any().any()
```

[3]: False

[4]:
```
# duplicate function of pandas returns a duplicate row as true and others as␣
 ↪false
sum(train.duplicated())
```

[4]: 0

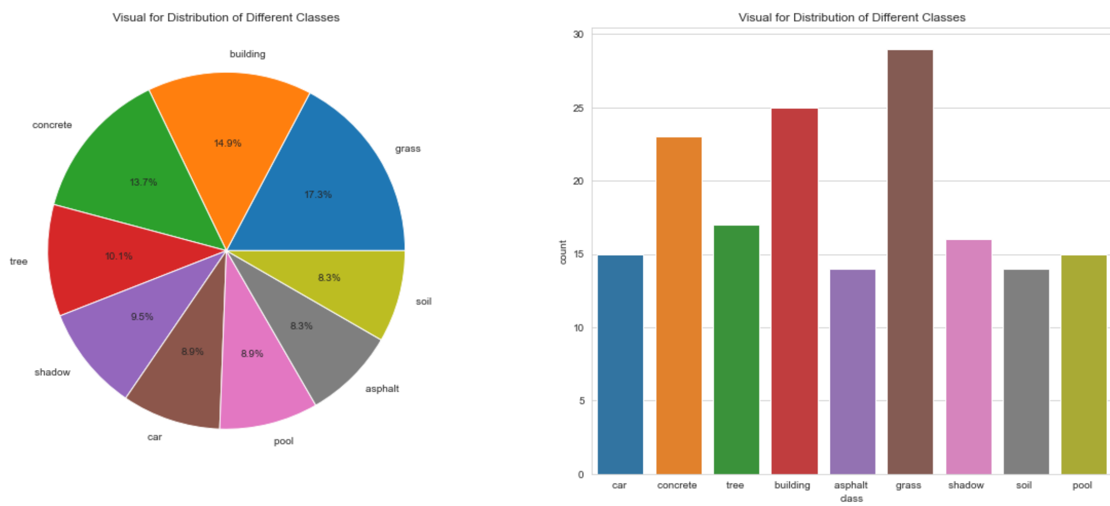
[5]:
```
# basic statistical details
fig = train.describe().T
fig = fig.round(5)  # round to 5 decimal places
table = go.Table(
    columnwidth=[0.8]+[0.5]*8,
    header=dict(
        values=['Attribute'] + list(fig.columns),
        line = dict(color='darkslategray'),
        fill = dict(color='royalblue'),
    ),
    cells=dict(
        values=[fig.index] + [fig[k].tolist() for k in fig.columns[:]],
        line = dict(color='darkslategray'),
        fill = dict(color=['paleturquoise', 'white'])
    )
)
plot([table], filename='table-of-data')
```

[5]: 'table-of-data.html'

[6]:
```
# more general data exploration
print(train['class'].value_counts())

f,axes=plt.subplots(1,2,figsize=(20,8))
train['class'].value_counts().plot.pie(autopct='%1.1f%%',ax=axes[0])
axes[0].set_title('Visual for Distribution of Different Classes')
axes[0].set_ylabel('')
```

```
sns.countplot('class',data=train,ax=axes[1]) # sns.countplot is used like a␣
↪histogram but for categorical data
axes[1].set_title('Visual for Distribution of Different Classes')
plt.show()
```

```
grass       29
building    25
concrete    23
tree        17
shadow      16
car         15
pool        15
asphalt     14
soil        14
Name: class, dtype: int64
```




```
[7]: # Lets take a look at any outliers that could be potential issues
from collections import Counter
def examine_outliers(train_data, n, features):
    outlier_indicator = []
    for out in features:
        Q1 = np.percentile(train_data[out], 25)
        Q3 = np.percentile(train_data[out], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR # IQR method of dealing with  outliers, 1 of 2␣
↪methods
        outlier_list_out = train_data[
            (train_data[out] < Q1 - outlier_step) | (train_data[out] > Q3 +␣
↪outlier_step)].index
        outlier_indicator.extend(outlier_list_out)
```

```
    outlier_indices = Counter(outlier_indicator)
    multiple_outliers = list(k for k, j in outlier_indices.items() if j > n)

    return multiple_outliers
```

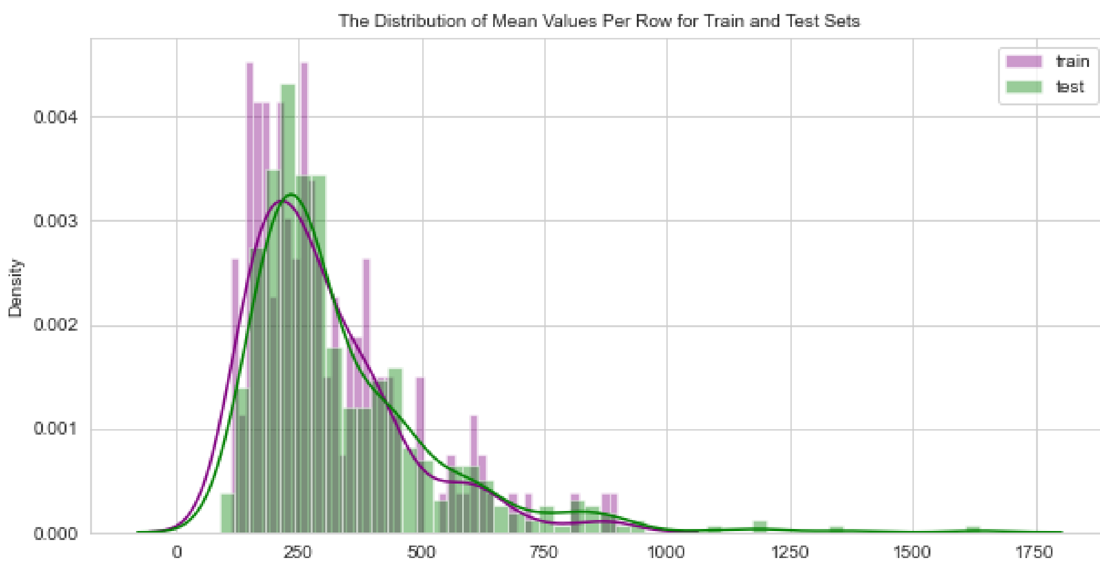
[8]:
```
# find outliers that should be removed
list_atributes = train.drop('class', axis=1).columns
outliers_to_remove = examine_outliers(train, 2, list_atributes)
train.loc[outliers_to_remove]
```

[8]:
```
         class  BrdIndx  Area  Round  Bright  Compact  ShpIndx  Mean_G  \
11     asphalt     4.19   418   2.48   83.35     4.21     4.30   68.07
10    building     1.57  3552   0.46  213.22     1.32     1.60  173.03
18    building     1.21  2797   0.78  244.70     1.34     1.23  229.52
32    building     1.48  3084   0.93  230.71     1.33     2.52  215.62
71    building     1.38  1482   0.54  145.95     1.42     1.42  122.53
..         …       …    …     …      …        …       …
99       grass     1.76   423   1.09  152.96     1.63     2.09  210.94
134    asphalt     2.22   116   1.29  100.77     2.72     2.37   87.45
13       grass     1.14   289   0.38  173.16     1.21     1.21  213.71
158    asphalt     2.25   542   1.49   76.52     2.09     2.32   60.50
132    asphalt     2.37   642   1.46   63.67     2.00     2.53   51.76

     Mean_R  Mean_NIR  …  SD_NIR_140  LW_140  GLCM1_140  Rect_140  \
11    88.80     93.17  …       26.40    1.50       0.77      0.68
10   229.84    236.80  …        6.53    1.54       0.33      0.94
18   252.21    252.37  …        6.84    1.27       0.52      0.85
32   252.64    223.88  …       12.08    5.19       0.68      0.65
71   156.16    159.16  …        8.45    1.20       0.54      0.87
..      …        …  …         …       …         …         …
99   114.01    133.93  …       36.11    2.89       0.90      0.43
134  105.32    109.53  …       28.19    1.07       0.86      0.46
13   145.56    160.23  …       16.13    1.80       0.50      0.92
158   82.17     86.88  …       27.72    1.45       0.80      0.56
132   67.20     72.05  …       21.07    2.19       0.81      0.46

     GLCM2_140  Dens_140  Assym_140  NDVI_140  BordLngth_140  GLCM3_140
11        8.19      1.85       0.46     -0.11           1342    1294.14
10        6.40      2.20       0.46     -0.14            410    3132.13
18        6.72      2.18       0.44     -0.04            264    2605.29
32        7.16      1.01       0.98     -0.07            682    1965.50
71        6.70      2.20       0.41     -0.12            238    2345.76
..         …        …         …        …             …          …
99        9.11      1.07       0.88      0.20           2022     680.93
134       8.55      1.26       0.35     -0.02           1156    1087.71
13        6.91      2.04       0.52      0.18             84    2915.26
```

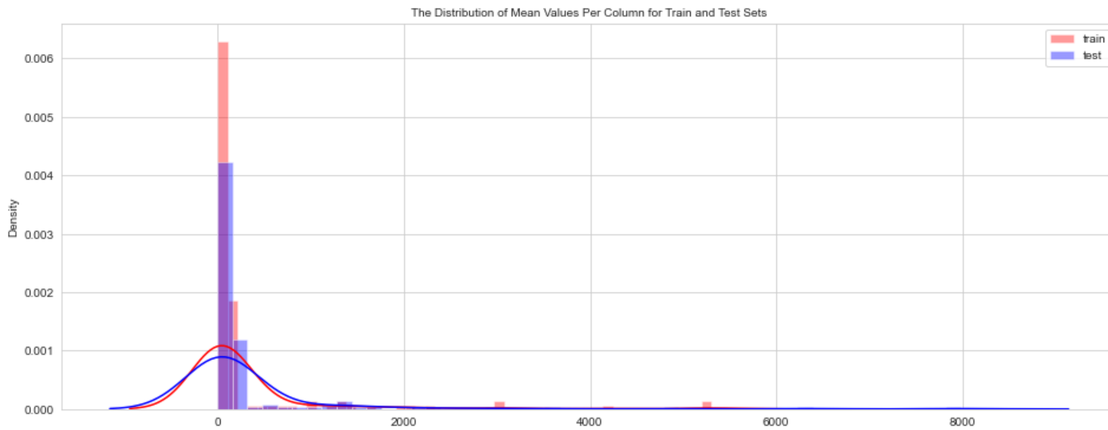
158	8.60	1.44	0.62	-0.09	3026	1297.05
132	8.05	0.90	0.91	-0.06	3092	1181.12

[63 rows x 148 columns]

[9]:
```
# lets look at mean values per row and column for train test data
# mean for rows
plt.figure(figsize=(10,5))
features = train.columns.values[1:148]
plt.title("The Distribution of Mean Values Per Row for Train and Test␣
↪Sets",fontsize=10)
sns.distplot(train[features].mean(axis=1),color="purple", kde=True,bins=50,␣
↪label='train') # kde is kernel density estimation
sns.distplot(test[features].mean(axis=1),color="green", kde=True,bins=50,␣
↪label='test')
plt.legend()
plt.show()
```



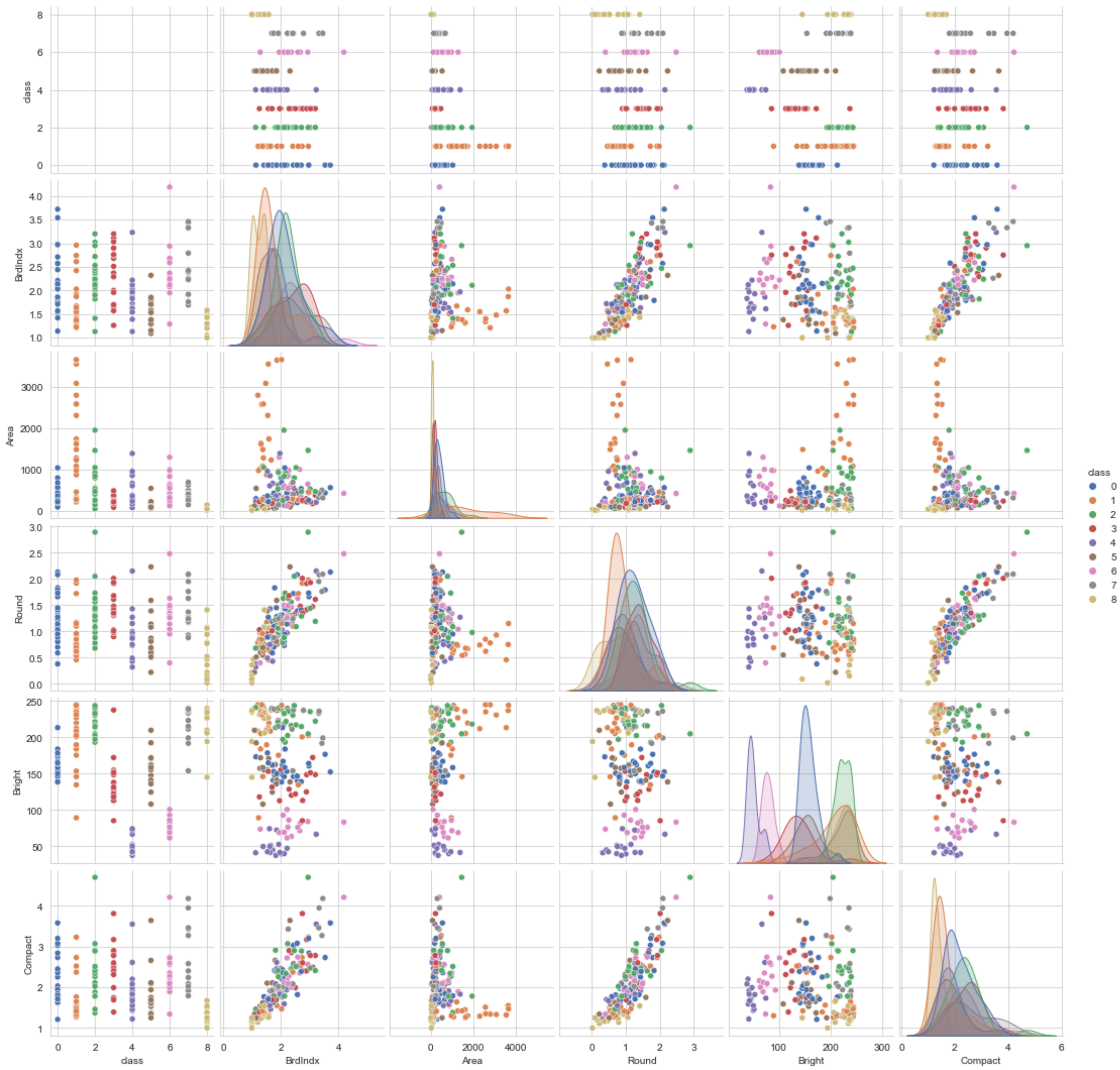
[10]:
```
# mean for columns
plt.figure(figsize=(16,6))
plt.title("The Distribution of Mean Values Per Column for Train and Test␣
↪Sets",fontsize=10)
sns.distplot(train[features].mean(axis=0),color="red",kde=True,bins=50,␣
↪label='train')
sns.distplot(test[features].mean(axis=0),color="blue", kde=True,bins=50,␣
↪label='test')
plt.legend()
plt.show()
```



[12]:
```
# lets examine correlations between features
# first the categorical variable 'class' needs to be changed to numerical␣
↪variable so...
group_map = {"grass ":0,"building ":1,'concrete ':2,'tree ':3,'shadow ':4,'pool␣
↪':5,'asphalt ':6,'soil ':7,'car ':8}
train['class'] = train['class'].map(group_map)
test['class'] = test['class'].map(group_map)
train['class'].unique()
```

[12]:
```
array([8, 2, 3, 1, 6, 0, 4, 7, 5], dtype=int64)
```

[13]:
```
# now lets look at the correlation bewteen a few variables
sns.pairplot(train, vars=['class',␣
↪'BrdIndx','Area','Round','Bright','Compact'], hue='class', palette='deep')
plt.show()
```



[15]:
```
# correlation of features with target
corr = train.corr().abs().unstack().sort_values(kind="quicksort").reset_index()
corr = corr[corr['level_0'] != corr['level_1']]
corr.head()
correlations = corr.loc[corr[0] == 1]
features_to_be_removed = set(list(correlations['level_1']))
correlations.shape
```

[15]: (42, 3)

[16]:
```
# prepare to try different classification algorithms
X_train = train.drop(['class'], axis=1)
y_train = pd.DataFrame(train['class'].values)
X_test = test.drop(['class'], axis=1)
```

```
y_test = test['class']
scaler = StandardScaler() #standardize data values into standard format
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

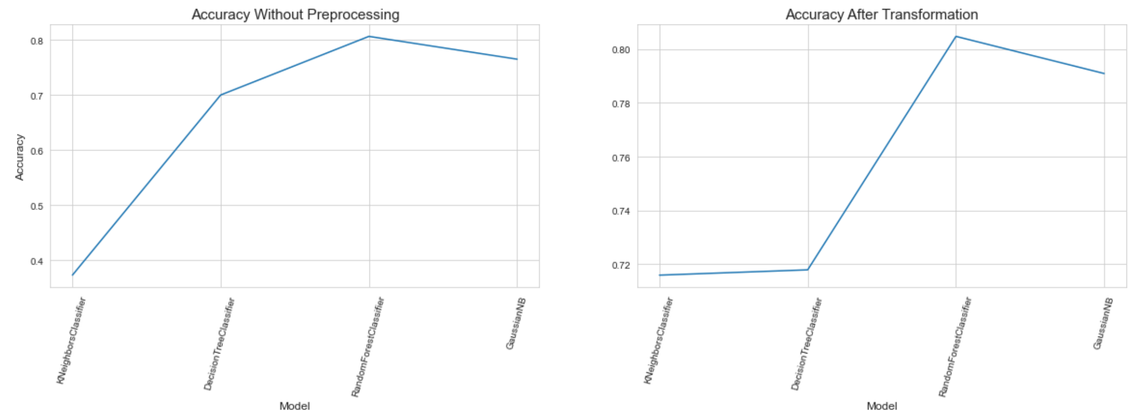
[23]:
```
X_train.columns
```

[23]:
```
Index(['BrdIndx', 'Area', 'Round', 'Bright', 'Compact', 'ShpIndx', 'Mean_G',
       'Mean_R', 'Mean_NIR', 'SD_G',
       …
       'SD_NIR_140', 'LW_140', 'GLCM1_140', 'Rect_140', 'GLCM2_140',
       'Dens_140', 'Assym_140', 'NDVI_140', 'BordLngth_140', 'GLCM3_140'],
      dtype='object', length=147)
```

[17]:
```
# classification algorithms
classification_choice = [KNeighborsClassifier(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), GaussianNB(),]
accuracy = {}
accuracy_std = {}
for choice in classification_choice:
    choice.fit(X_train, y_train)
    pred = choice.predict(X_test)
    accuracy[str((str(choice).split('(')[0]))] = accuracy_score(pred, y_test)

for choice in classification_choice:
    choice.fit(X_train_std, y_train)
    prediction = choice.predict(X_test_std)
    accuracy_std[str((str(choice).split('(')[0]))] = accuracy_score(prediction,␣
 ↪y_test)

data = accuracy.values()
labels = accuracy.keys()
data_std = accuracy_std.values()
labels_std = accuracy_std.keys()
```

[18]:
```
# plot accuracy for visualization
fig = plt.figure(figsize=(20,5))
plt.subplot(121)
plt.plot([i for i, e in enumerate(data)], data); plt.xticks([i for i, e in␣
 ↪enumerate(labels)], [l[:] for l in labels])
plt.title("Accuracy Without Preprocessing",fontsize = 15)
plt.xlabel('Model',fontsize = 12)
plt.xticks(rotation = 75)
plt.ylabel('Accuracy',fontsize = 12)

plt.subplot(122)
```

```
plt.plot([i for i, e in enumerate(data_std)], data_std); plt.xticks([i for i, e in enumerate(labels_std)], [l[:] for l in labels_std])
plt.title("Accuracy After Transformation",fontsize = 15)
plt.xlabel('Model',fontsize = 12)
plt.xticks(rotation =75)
plt.show()
# above results show that random forest classifier seems to perform the best
```



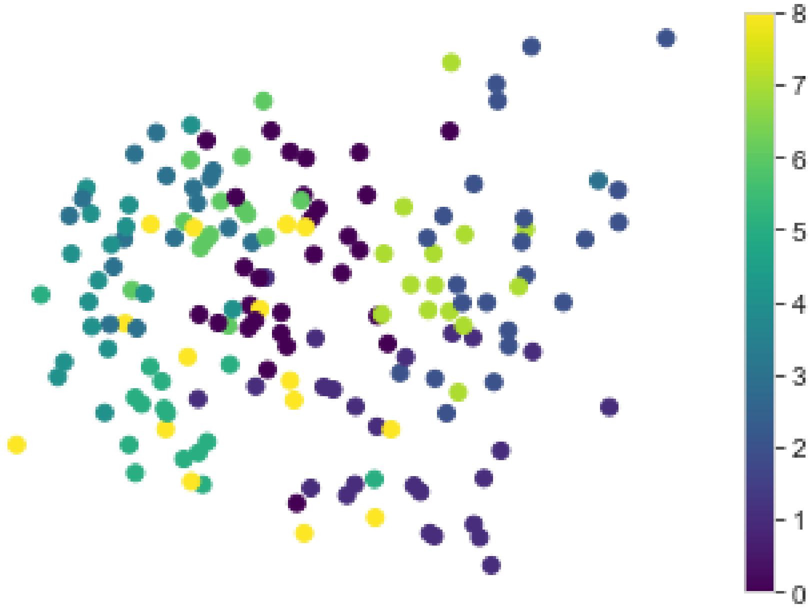
[19]:
```
# now lets perform cross validation
n_fold = 5
folds = KFold(n_splits=n_fold, shuffle=True, random_state=1)
```

[20]:
```
# Random Forest Classifier
prediction = np.zeros(len(X_test))
complete_acc = []
out_of_fold = np.zeros(len(X_train))
for fold_n, (train_index, valid_index) in enumerate(folds.split(X_train, y_train)):
    print('Fold', fold_n, 'started at', time.ctime(), end="  ")
    X_train_, X_valid = X_train.iloc[train_index], X_train.iloc[valid_index] #iloc function used to retrieve rows from a data set
    y_train_, y_valid = y_train.iloc[train_index], y_train.iloc[valid_index]

    classifier_randomforest = RandomForestClassifier(n_estimators=1000, n_jobs=-1, random_state=0) # default estimator, -1 is using all processors, 0 fixes sequence
    classifier_randomforest.fit(X_train_, y_train_)
    out_of_fold[valid_index] = classifier_randomforest.predict(X_train.iloc[valid_index])

    prediction = classifier_randomforest.predict(X_test)
    print("Validation Score: ", accuracy_score(y_test, prediction))
```

```
    complete_acc.append(accuracy_score(y_test, prediction))
print("CV score".format(accuracy_score(y_train, out_of_fold)))
print("Mean Testing Score: ", np.mean(complete_acc))
```

```
Fold 0 started at Tue Apr 26 09:33:02 2022  Validation Score:
0.7909270216962525
Fold 1 started at Tue Apr 26 09:33:04 2022  Validation Score:  0.814595660749507
Fold 2 started at Tue Apr 26 09:33:06 2022  Validation Score:
0.8067061143984221
Fold 3 started at Tue Apr 26 09:33:08 2022  Validation Score:
0.8047337278106509
Fold 4 started at Tue Apr 26 09:33:10 2022  Validation Score:  0.814595660749507
CV score
Mean Testing Score:  0.8063116370808678
```

[21]:
```
# will take a look at hyperparameters for random forest and decision tree later␣
↪in assignment
#PCA analysis
scaler = StandardScaler()
scaled_training = scaler.fit_transform(X_train)
PCA_xtrain = PCA().fit_transform(scaled_training)
plt.scatter(PCA_xtrain[:, 0], PCA_xtrain[:, 1], c=train['class'],␣
↪cmap="viridis")
plt.axis('off')
plt.colorbar()
plt.show()
```



```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```