1. Perform a 10-fold Cross-Validation for SVC on BostonHousing_full.csv data using Pipeline. Specifically,

```
(i) report average accuracy, confusion matrix, precision, recall, and F1 score; and

(ii) use grid search to find the best C from C = [1, 5, 10, 50, 100, 500, 1000].
```

2. Load salary1.arff file and convert record array to Dataframe and byte string to string. Build an SVR model, Plot the data, SVR and regression models.
3. What is Weighted Nearest Neighbor Model? Explain briefly.

```
import pandas as pd
from matplotlib import style
from matplotlib import pyplot as plt
#import graphviz as gr
%matplotlib inline
style.use("fivethirtyeight")
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", 60)
pd.set_option('display.max_rows', 50)
pd.set_option('display.width', 1000)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Load BostonHousing_full.csv file and convert record array to dataframe and byte string to string
import pandas as pd
import numpy as np
data = pd.read_csv('/content/drive/MyDrive/DataMining/Files/BostonHousing_full.csv')
```

```
data.corr()
```

```
import seaborn as sns
plt.figure(figsize=(20, 10))
sns.heatmap(data.corr().abs(),  annot=True)
```

From correlation matrix, we see TAX and RAD are highly correlated features.

```
rawdf = pd.DataFrame.from_records(data)
#rawdf['CATMEDV'] = rawdf['CATMEDV'].str.decode('utf-8') # convert byte string to string
rawdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  CATMEDV  506 non-null    object
dtypes: float64(10), int64(3), object(1)
memory usage: 55.5+ KB
```

```
rawdf
```

```
rawdf.replace({'CATMEDV':{'low':0,'high':1}}, inplace = True)
```

```
# Build SVC model
X = rawdf.drop (['CATMEDV','RAD','TAX','CHAS'], axis='columns')
y = rawdf['CATMEDV']
```

```
X.head(3)
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   NOX      506 non-null    float64
 4   RM       506 non-null    float64
 5   AGE      506 non-null    float64
 6   DIS      506 non-null    float64
 7   PTRATIO  506 non-null    float64
 8   B        506 non-null    float64
 9   LSTAT    506 non-null    float64
dtypes: float64(10)
memory usage: 39.7 KB
```

```
y.head(3)
```

```
0    0
1    0
2    1
Name: CATMEDV, dtype: int64
```

```
from sklearn.model_selection import train_test_split
#Splitting into test and train
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# 10-fold cross validation - using cross_val_score

# Pipelines to normalize numeric values and build SVM classifier

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler #for normalization
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV #for figuring out the best parameters from the listed hyperparameters
#gridsearchCV automates the manual process of having to test various values/ types of "C", "epsilon", "gamma", and "kernel" and choose the best

#scaler = MinMaxScaler()


svc1 = SVC(C=[1, 5, 10, 50, 100, 500, 1000], kernel='linear')
#svc1 = SVC(C=[0.1], kernel='linear')

svc_pipe=Pipeline(steps=[('scaler', MinMaxScaler()),
                 ('svc', SVC(C=[1, 5, 10, 50, 100, 500, 1000], kernel='linear'))]) #creating a pipeline



param_grid = {'svc__C':[1, 5, 10, 50, 100, 500, 1000],'svc__gamma':[1,0.1,0.01,0.001,0.0001], 'svc__kernel':['linear']} #setting up a parameter grid

grid_search = GridSearchCV(estimator=svc_pipe, param_grid=param_grid, cv=10, scoring="neg_mean_squared_error")


grid_search.fit(X_train,y_train)
```

```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scaler', MinMaxScaler()),
                                       ('svc',
                                        SVC(C=[1, 5, 10, 50, 100, 500, 1000],
                                            kernel='linear'))]),
             param_grid={'svc__C': [1, 5, 10, 50, 100, 500, 1000],
                         'svc__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'svc__kernel': ['linear']},
             scoring='neg_mean_squared_error')
```

```
grid_search.best_params_ #use grid search to find the best C from C = [1, 5, 10, 50, 100, 500, 1000]
```

```
{'svc__C': 50, 'svc__gamma': 1, 'svc__kernel': 'linear'}
```

The best C from C = [1, 5, 10, 50, 100, 500, 1000] is C=50

```
grid_search.best_score_ #best score out of all the scores of multilple train test splits
```

```
-0.04695121951219512
```

```
grid_search.best_estimator_.steps[-1][1].coef_
```

```
array([[  3.24255609,   0.89305404,  -2.35445643,   0.63743685,
         10.37926746,   0.92304309,  -1.88273637,  -3.42418396,
          5.17269506, -10.99058788]])
```

Confusion Matrix

```
grid_pred=grid_search.predict(X_test)

from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, grid_pred)
print(confusion_matrix)

#Confusion Matrix
```

```
[[86  4]
 [ 3  9]]
```

Precision, Recall, F1 score

```
from sklearn.metrics import classification_report
print(classification_report(y_test, grid_pred))

#avg precision, recall, f1-score
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96        90
           1       0.69      0.75      0.72        12

    accuracy                           0.93       102
   macro avg       0.83      0.85      0.84       102
weighted avg       0.93      0.93      0.93       102
```

Load salary1.arff file and convert record array to Dataframe and byte string to string. Build an SVR model, Plot the data, SVR and regression models.

```
# Load salary1.arff file and convert record array to dataframe and byte string to string
import pandas as pd
import numpy as np
from scipy.io import arff
data, meta = arff.loadarff('/content/drive/MyDrive/DataMining/Files/salary1.arff')
print(data,meta)
```

```
[(10., 33.) (12., 36.) (12., 50.) (13., 51.) (14., 42.) (14., 45.)
 (15., 59.) (16., 49.) (16., 60.) (17., 72.) (18., 69.) (20., 77.)] Dataset: salary1
        education's type is numeric
        salary's type is numeric
```

Byte string conversion is not needed as the data is numerical and not categorical

```
rawdf1 = pd.DataFrame.from_records(data)
rawdf1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   education  12 non-null     float64
 1   salary     12 non-null     float64
dtypes: float64(2)
memory usage: 320.0 bytes
```

```
rawdf1
```

```
# Build SVR model
X = rawdf1.drop('salary', axis='columns')
y = rawdf1['salary']
```

Evaluation of SVR with Pipeline

```
# Pipelines to normalize numeric values and build SVM classifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

#parameters = {'C':[1,5,10,50,100,500,1000], 'epsilon':[0.05, 0.1, 0.15, 0.2]}
#parameters = {'kernel': ('linear', 'rbf','poly'), 'C':[1.5, 10],'gamma': [1e-7, 1e-4],'epsilon':[0.05, 0.1, 0.15, 0.2]}
#parameters = {'kernel': ('linear'), 'C':[1,5,10,50,100,500,1000], 'epsilon':[0.05, 0.1, 0.15, 0.2]}
#svr = SVR()
#clf = GridSearchCV(svr, parameters)
#clf.fit(X,y)
#clf.best_params_
```

```
svr_pipe = Pipeline([('scaler', MinMaxScaler()), ('linear_svr', SVR(epsilon=5, kernel='linear'))])
#svr_pipe = Pipeline([('scaler', MinMaxScaler()), ('linear_svr', SVR(kernel='linear', gamma=1.0E-12))])
#svr_pipe = Pipeline([('scaler', MinMaxScaler()), ('linear_svr', SVR(kernel='poly'))])

#svr.fit(X, y)
```

```
# 10-fold cross validation - using cross_val_score
from sklearn.model_selection import GridSearchCV
#param_grid = {'linear_svr__kernel': ('linear', 'rbf','poly'), 'linear_svr__C': [1,5,10,50,100,500,1000], 'linear_svr__epsilon': [0.05, 0.1, 0.15, 0.2], 'linear_svr__gamma':[1e-7, 1e-4, 1.0
param_grid = {'linear_svr__C': [1,5,10,50,100,500,1000], 'linear_svr__epsilon': [0.05, 0.1, 0.15, 0.2]}
#param_grid = {'linear_svr__C': [5], 'linear_svr__epsilon': [0.15]}
grid_search = GridSearchCV(svr_pipe, param_grid, cv=10, scoring="neg_mean_squared_error")
#grid_search = GridSearchCV(svr_pipe, param_grid, cv=10)
grid_search.fit(X, y)
```

```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scaler', MinMaxScaler()),
                                       ('linear_svr',
                                        SVR(epsilon=5, kernel='linear'))]),
             param_grid={'linear_svr__C': [1, 5, 10, 50, 100, 500, 1000],
                         'linear_svr__epsilon': [0.05, 0.1, 0.15, 0.2]},
             scoring='neg_mean_squared_error')
```

```
grid_search.best_params_
```

```
{'linear_svr__C': 500, 'linear_svr__epsilon': 0.2}
```

Best C = 500

```
grid_search.best_score_
```

```
-44.765445826752824
```

```
grid_search.best_estimator_.steps[-1][1].coef_
```

```
array([[44.33335644]])
```

```
import math
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
pred_y=grid_search.predict(X)

#score=grid_search.score(X,y)
#print(score)

mse=mean_squared_error(y, pred_y)
#print("Mean Squared Error:",mse)
rmse=math.sqrt(mse)
print("Root Mean Squared Error:", rmse)

mse=mean_absolute_error(y, pred_y)
#mse=mean_absolute_error(y, pred_y, multioutput='raw_values')
print("Mean Absolute Error:",mse)
```

```
Root Mean Squared Error: 5.955662955967093
Mean Absolute Error: 4.736110148390507
```

```
# print(grid_search.cv_results_)
cvresult = grid_search.cv_results_
for mean_test_score, params in zip(cvresult['mean_test_score'], cvresult['params']):
    print(mean_test_score, params)
```

```
-210.87766415458918 {'linear_svr__C': 1, 'linear_svr__epsilon': 0.05}
-210.87766415458918 {'linear_svr__C': 1, 'linear_svr__epsilon': 0.1}
-210.87766415458918 {'linear_svr__C': 1, 'linear_svr__epsilon': 0.15}
-210.87766415458918 {'linear_svr__C': 1, 'linear_svr__epsilon': 0.2}
-143.80891615596414 {'linear_svr__C': 5, 'linear_svr__epsilon': 0.05}
-144.29704129798338 {'linear_svr__C': 5, 'linear_svr__epsilon': 0.1}
-145.101853895884 {'linear_svr__C': 5, 'linear_svr__epsilon': 0.15}
-145.98310389692713 {'linear_svr__C': 5, 'linear_svr__epsilon': 0.2}
-106.92418826955557 {'linear_svr__C': 10, 'linear_svr__epsilon': 0.05}
-107.1160632695556 {'linear_svr__C': 10, 'linear_svr__epsilon': 0.1}
-107.30918826955558 {'linear_svr__C': 10, 'linear_svr__epsilon': 0.15}
-107.5035632695556 {'linear_svr__C': 10, 'linear_svr__epsilon': 0.2}
-52.10914371869321 {'linear_svr__C': 50, 'linear_svr__epsilon': 0.05}
-52.253526353640936 {'linear_svr__C': 50, 'linear_svr__epsilon': 0.1}
-52.40436397070433 {'linear_svr__C': 50, 'linear_svr__epsilon': 0.15}
-52.56165659210806 {'linear_svr__C': 50, 'linear_svr__epsilon': 0.2}
-47.31025453579236 {'linear_svr__C': 100, 'linear_svr__epsilon': 0.05}
-47.14857445749685 {'linear_svr__C': 100, 'linear_svr__epsilon': 0.1}
-46.99297476362707 {'linear_svr__C': 100, 'linear_svr__epsilon': 0.15}
-46.84345471483461 {'linear_svr__C': 100, 'linear_svr__epsilon': 0.2}
-45.62974547926026 {'linear_svr__C': 500, 'linear_svr__epsilon': 0.05}
-45.33656554398844 {'linear_svr__C': 500, 'linear_svr__epsilon': 0.1}
-45.048465631901614 {'linear_svr__C': 500, 'linear_svr__epsilon': 0.15}
-44.765445826752824 {'linear_svr__C': 500, 'linear_svr__epsilon': 0.2}
-45.6297484965322 {'linear_svr__C': 1000, 'linear_svr__epsilon': 0.05}
-45.33656862191763 {'linear_svr__C': 1000, 'linear_svr__epsilon': 0.1}
-45.048468740414634 {'linear_svr__C': 1000, 'linear_svr__epsilon': 0.15}
-44.76544888314841 {'linear_svr__C': 1000, 'linear_svr__epsilon': 0.2}
```

```
# Build SVR model
X = rawdf1.drop('salary', axis='columns')
y = rawdf1['salary']
```

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

```
lm.fit(X,y)
```

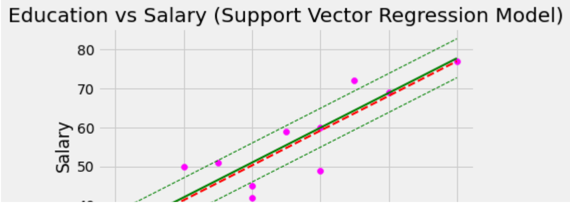
```
LinearRegression()
```

```
y_reg = lm.predict(X)
```

```
regression = SVR(kernel='linear')
regression.fit(X,y)

#Split dataset to be input here
svr_result = regression.predict(X)

plt.scatter(X, y, color = 'magenta')
plt.plot(X, svr_result+5, 'g--', color = 'green',linewidth=1)
plt.plot(X, svr_result,'k-', color = 'green',linewidth=2)
plt.plot(X, svr_result-5,'g--', color = 'green',linewidth=1)
plt.plot(X, y_reg,'r--', color = 'red',linewidth=2)
plt.title('Education vs Salary (Support Vector Regression Model)')
plt.xlabel('Education')
plt.ylabel('Salary')
plt.show()
```

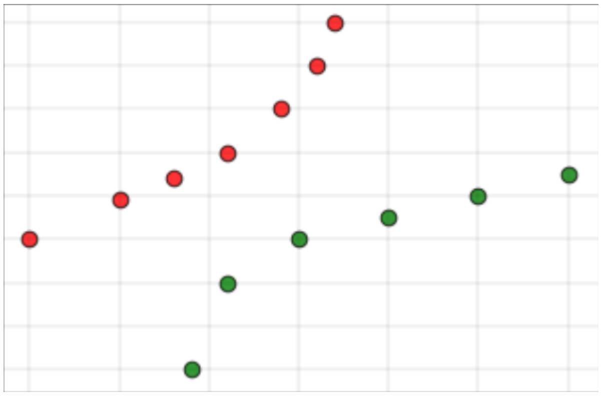


As the dataset is small we could see regression line is matching the SVM line

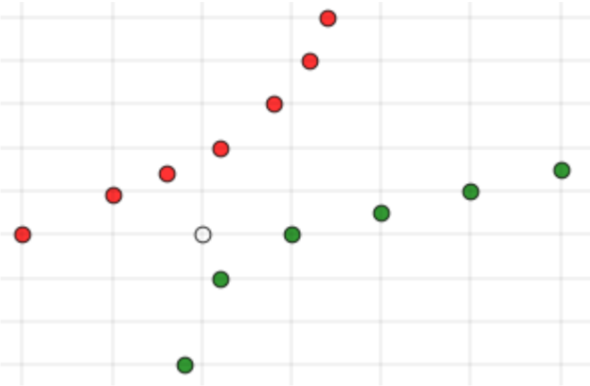
What is Weighted Nearest Neighbor Model? Explain briefly.

Weighted kNN is a modified version of k nearest neighbors. One of the many issues that affect the performance of the kNN algorithm is the choice of the hyperparameter k. If k is too small, the algorithm would be more sensitive to outliers. If k is too large, then the neighborhood may include too many points from other classes. Another issue is the approach to combining the class labels. The simplest method is to take the majority vote, but this can be a problem if the nearest neighbors vary widely in their distance and the closest neighbors more reliably indicate the class of the object.

Let's consider the below training set



The red labels indicate the class 0 points and the green labels indicate class 1 points. Let's consider the white point as the query point(the point whose class label has to be predicted)



If we give the above dataset to a kNN based classifier, then the classifier would declare the query point to belong to the class 0. But in the plot, it is clear that the point is more closer to the class 1 points compared to the class 0 points. To overcome this disadvantage, weighted kNN is used. In weighted kNN, the nearest k points are given a weight using a function called as the kernel function. The intuition behind weighted kNN, is to give more weight to the points which are nearby and less weight to the points which are farther away. Any function can be used as a kernel function for the weighted knn classifier whose value decreases as the distance increases. The simple function which is used is the inverse distance function which implies that as the distance increases weight decreases and as the distance decreases, weight increases.

1s completed at 2:20 AM