# HW 5

This assignment covers Comparision of Decision Trees and Support Vector Machine. **DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission**

- **Q** - QUESTION
- **A** - Where to input your answer

## Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom.
- Objective of this assignment is to help you master python and scikit-learn package.
- See README.md for homework submission instructions

## Related Tutorials

- Decision Tree with KFold Cross Validation
- Decision Tree with Bagging
- Support Vector Machine

# Data Processing

**Q1** Get training data from the dataframe

1. Load HW5_data.csv from ```data'' folder into the dataframe
2. Check if there is any NaN in the dataset
3. Remove the rows with NaN values.
4. Print how many examples belong to each class in the data frame.

**A1** Replace ??? with code in the code cell below

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

#Read the data file using the appropriate separator as input to read_csv()

df = pd.read_csv('../data/HW5_data.csv', na_values='', sep=",")
df.head(10)
```

Out[ ]:

| | Mean of the integrated profile | Standard deviation of the integrated profile | Excess kurtosis of the integrated profile | Skewness of the integrated profile | Mean of the DM-SNR curve | Standard deviation of the DM-SNR curve | Excess kurtosis of the DM-SNR curve | Skewness of the DM-SNR curve | ta |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 121.156250 | 48.372971 | 0.375485 | -0.013165 | 3.168896 | 18.399367 | 7.449874 | 65.159298 | |
| **1** | 76.968750 | 36.175557 | 0.712898 | 3.388719 | 2.399666 | 17.570997 | 9.414652 | 102.722975 | |
| **2** | 130.585938 | 53.229534 | 0.133408 | -0.297242 | 2.743311 | 22.362553 | 8.508364 | 74.031324 | |
| **3** | 156.398438 | 48.865942 | -0.215989 | -0.171294 | 17.471572 | NaN | 2.958066 | 7.197842 | |
| **4** | 84.804688 | 36.117659 | 0.825013 | 3.274125 | 2.790134 | 20.618009 | 8.405008 | 76.291128 | |
| **5** | 121.007812 | 47.176944 | 0.229708 | 0.091336 | 2.036789 | NaN | 9.546051 | 112.131721 | |
| **6** | 79.343750 | 42.402174 | 1.063413 | 2.244377 | 141.641304 | NaN | -0.700809 | -1.200653 | |
| **7** | 109.406250 | 55.912521 | 0.565106 | 0.056247 | 2.797659 | 19.496527 | 9.443282 | 97.374578 | |
| **8** | 95.007812 | 40.219805 | 0.347578 | 1.153164 | 2.770067 | 18.217741 | 7.851205 | 70.801938 | |
| **9** | 109.156250 | 47.002234 | 0.394182 | 0.190296 | 4.578595 | NaN | 5.702532 | 36.342493 | |

In [ ]:
```
# check if there is NaN in the dataset
df.isnull().sum()
```

Out[ ]:
```
 Mean of the integrated profile                   0
 Standard deviation of the integrated profile     0
 Excess kurtosis of the integrated profile     1735
 Skewness of the integrated profile               0
 Mean of the DM-SNR curve                         0
 Standard deviation of the DM-SNR curve        1178
 Excess kurtosis of the DM-SNR curve              0
 Skewness of the DM-SNR curve                   625
target_class                                      0
dtype: int64
```

In [ ]:
```
#Drop NaNs if there is any
df.dropna(inplace=True)

# Count number of entries for different target_class
df.target_class.count()
```

Out[ ]: 9273

**Q2** Separate training and testing data from the dataframe

1. Assign values of `target_class` column to `y`, note you have to use `.values` method
2. Drop `target_class` column from data frame,
3. Assign df values to x
4. Split dataset into train and test data use train_test_split with test_size = 0.25, stratify y and random_state = 1238

**A2** Replace ??? with code in the code cell below

In [ ]:
```
# Assign values of ```target_class``` column to y, note you have to use .values met
y = df.target_class.values
# Drop 'target_class' column from data frame,
df.drop(columns=['target_class'], inplace=True)
# Assign df values to x
x = df.values
# View shape of x and y
print('Shape of x = ', x.shape)
print('Shape of y = ', y.shape)

xtrain, xtest, ytrain, ytest =  train_test_split(x, y, test_size=0.25, random_state
```

```
Shape of x =  (9273, 8)
Shape of y =  (9273,)
```

In [ ]:
```
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

Out[ ]:
```
((6954, 8), (2319, 8), (6954,), (2319,))
```

# Decision Tree

## Decision Tree with different depth

**Q3** Train DecisionTreeClassifier Model at different depths

1. Create four DecisionTreeClassifier models with different parameters. Use max_depth size = 1, 2, 5, 25 & max_leaf_nodes=5, 10, 15, 25 respectively
2. Use random_state=30 & criterion='entropy' for all models
3. Fit the four different models with the train data.
4. Predict the test data using trained models
5. Calculate the Mean Squared Error(MSE) of each model's prediction
6. Print precision recall curve for the test data with the minimum MSE value from four trianed models.

**A3** Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error

#create decision tree classifier
clf_1 = DecisionTreeClassifier(criterion="entropy", max_depth=1, max_leaf_nodes=5,
clf_2 = DecisionTreeClassifier(criterion="entropy", max_depth=2, max_leaf_nodes=10,
clf_3 = DecisionTreeClassifier(criterion="entropy", max_depth=5, max_leaf_nodes=15,
clf_4 = DecisionTreeClassifier(criterion="entropy", max_depth=25, max_leaf_nodes=25

#fit classifier model
clf_1.fit(xtrain,ytrain)
clf_2.fit(xtrain,ytrain)
clf_3.fit(xtrain,ytrain)
clf_4.fit(xtrain,ytrain)

#predict
pred_1 = clf_1.predict(xtest)
pred_2 = clf_2.predict(xtest)
pred_3 = clf_3.predict(xtest)
pred_4 = clf_4.predict(xtest)

#calculate mean_squared_error
print('clf_1 MSE: ', mean_squared_error(ytest, pred_1))
print('clf_2 MSE: ', mean_squared_error(ytest, pred_2))
print('clf_3 MSE: ', mean_squared_error(ytest, pred_3))
print('clf_4 MSE: ', mean_squared_error(ytest, pred_4))
```

```
clf_1 MSE:  0.0258732212160414
clf_2 MSE:  0.0258732212160414
clf_3 MSE:  0.02501078050884002
clf_4 MSE:  0.0258732212160414
```

## Precision-Recall Curve for Best Above

`Important Note:` If `from_estimator()` function gives Attribute error then it means your sklearn is not updated.

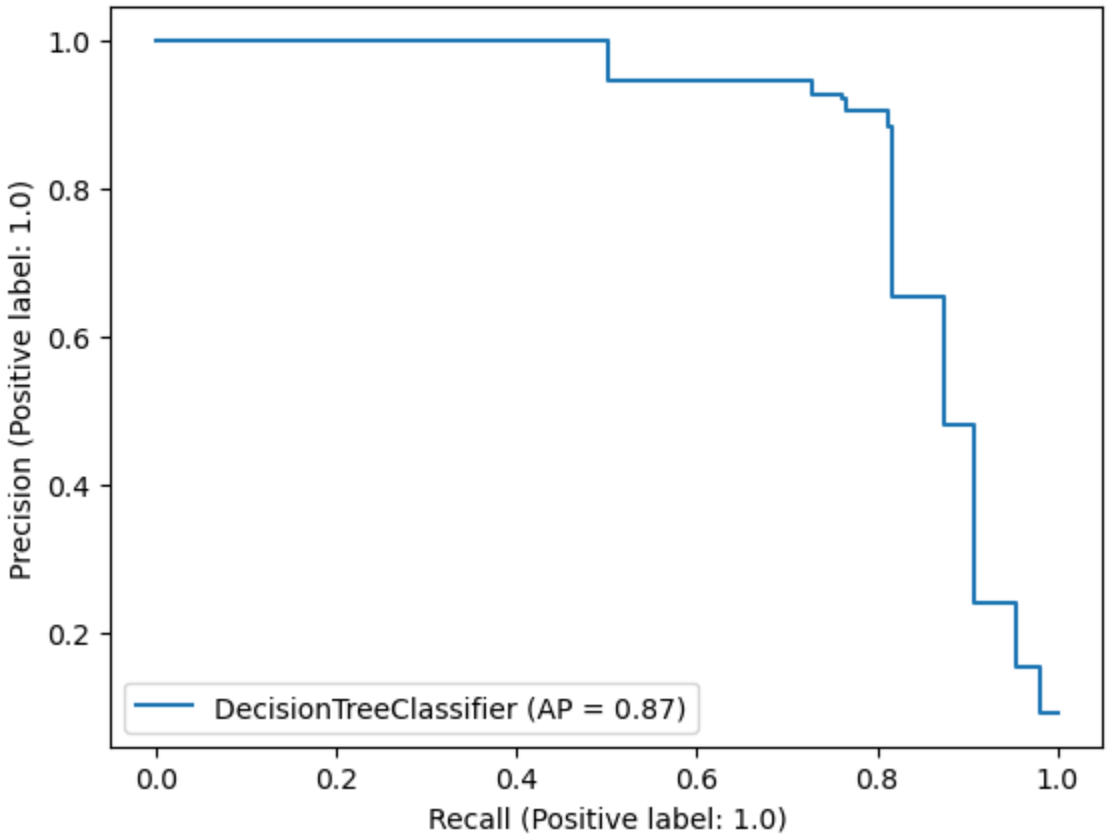
- If you are using conda, you can upgrade with

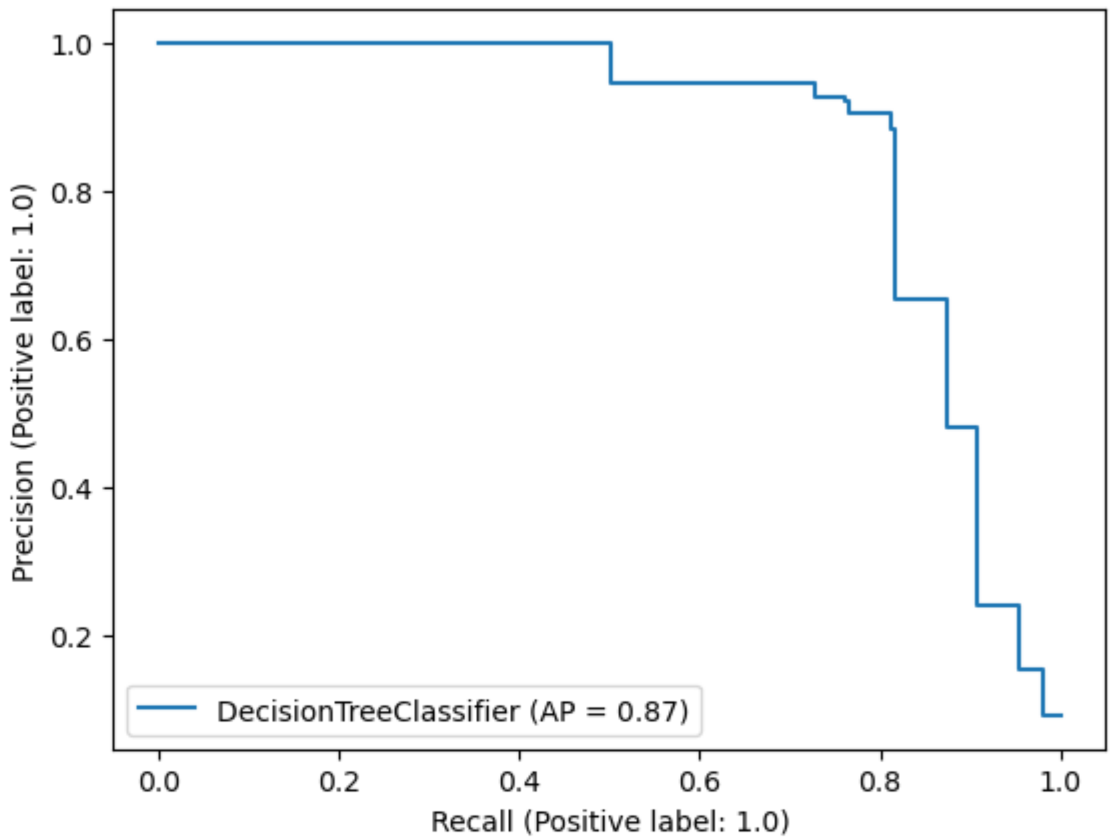
conda upgrade -c conda-forge scikit-learn

- or, with pip,

python -m pip install scikit-learn --upgrade

In [ ]:
```
# Use the below one
from sklearn.metrics import precision_recall_curve

# Or this below one, whichever suits you
from sklearn.metrics import PrecisionRecallDisplay
import matplotlib.pyplot as plt

# using prediction from clf_3 because it had the MSE closest to 0
disp = PrecisionRecallDisplay.from_estimator(clf_3, xtest, ytest)
disp.plot()
plt.show()
```






## Decision Tree with K-fold cross validation

**Q4** Use Kfold on the test dataset, and evaluate the best model

1. Use cross_val_score and fit your best model with k = 5 fold size on test data
2. Calculate average scores in kfold

**A4** Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.model_selection import KFold, cross_val_score

scores = cross_val_score(clf_4, xtest, ytest, cv=5)
print("Cross-validation scores: {}".format(scores))
print("Average cross-validation score: {}".format(scores.sum()/5))
```

```
Cross-validation scores: [0.95689655 0.9762931  0.96982759 0.95258621 0.9524838 ]
Average cross-validation score: 0.9616174499143517
```

## Decision Tree with Bagging

**Q5** Now we will use Bagging technique on the our previous best model, and evaluate it

```
Part 1:
```

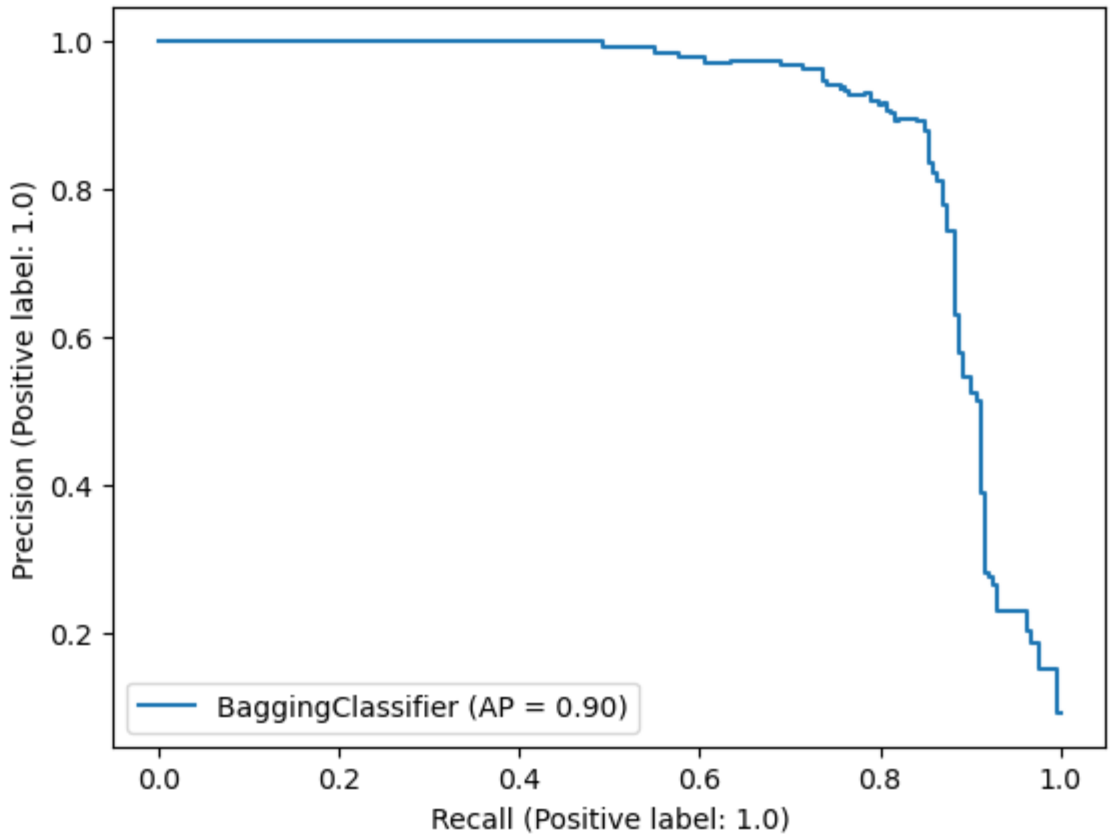
1. Now, Create a Bagged Model passing `model = previous_best, n_estimators = 10 & random_state=1 to BaggingClassifier()`
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the test MSE
5. Plot Precision-Recall Curve from the true & predicted test data

**A5** Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.ensemble import BaggingClassifier

# Use BaggingClassifier to fit the training data
# Calculate the mean squared error

#load BaggingClassifier model and pass n_estimators=10, random_state=1
bagged_clf = BaggingClassifier(estimator=clf_3, n_estimators=10, random_state=1)
bagged_clf.fit(xtrain, ytrain)
pred = bagged_clf.predict(xtest)
print('bagged_clf MSE: ', mean_squared_error(ytest, pred))
```

```
bagged_clf MSE:  0.02630444156964209
```

In [ ]:
```
#pass necessary parameters to PrecisionRecallDisplay.from_estimator()

PrecisionRecallDisplay.from_estimator(bagged_clf, xtest, ytest)
plt.show()
```




```
Part 2:
```

1. Why BaggingClassifier is called an ensembled technique? why it works better most of the time than the single model classifiers?

BaggingClassifier is called an ensemble technique because it combines the predictions of multiple models in making its final prediction. In the case of our bagging_clf, we incorporated a DecisionTreeClassifier into the "estimator" parameter of the BaggingClassifier. Ensemble techniques generally work better than single model classifiers because the combination of predictions from multiple models improves generalization as well as reducing variance and variability.

1. What is the disadvantage of increasing the number of estimators while using BaggingClassifier? Explain with an appropriate example.

Increasing the number of estimators can lead to potential overfitting, which is exactly what bagging is supposed to reduce the risk of.

# Support Vector Machine(SVM)

**Q6** Create SVM Model on the training set, and do the following

```
Part:1
```

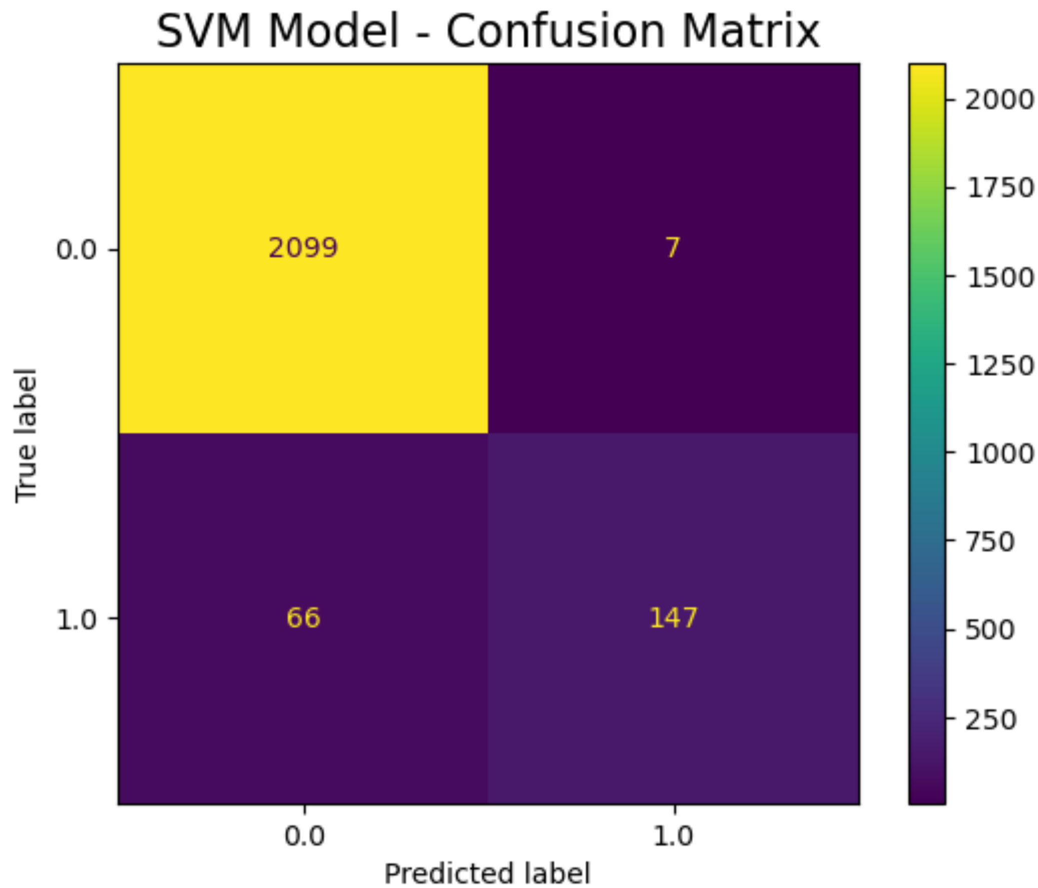
1. Now, Create a SVM Model with default parameters
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot confusion matrix on the test data (Make font size 16)

**A6** Replace ??? with code in the code cell below

In [ ]:
```
# import SVC classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

svc = SVC()

# fit classifier to training set
svc.fit(xtrain, ytrain)

# make predictions on test set
svc_pred = svc.predict(xtest)

# compute and print accuracy score
acc = accuracy_score(ytest, svc_pred)
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(acc))
```

```
Model accuracy score with default hyperparameters: 0.9685
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

cm = confusion_matrix(ytest, svc_pred, labels=svc.classes_)
cm_disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svc.classes_)

cm_disp.plot()
plt.title("SVM Model - Confusion Matrix", fontsize=16)
plt.show()
```




```
Part2:
```

1. From the above Confusion Matrix we can see that high number of Class 1 is predicted as Class 0 from the model. What is your reasoning behind this situtation?

The default parameters for an SVC model do not perform as well as fine-tuned parameters.

1. What can be done in order to resolve this issue?

Tinker with the parameters to determine which would yield the best results for predicting.

# SVM with high margin

**Q7** Create SVM Model on the training set, and evaluate

`Note:`

1. If we analyze our dataset using df.describe() function, we will see that there are many outliers in the dataset.
2. So, we need to increase our margin with `HIGH C` values so that the SVM model get better generalization

`Task:`

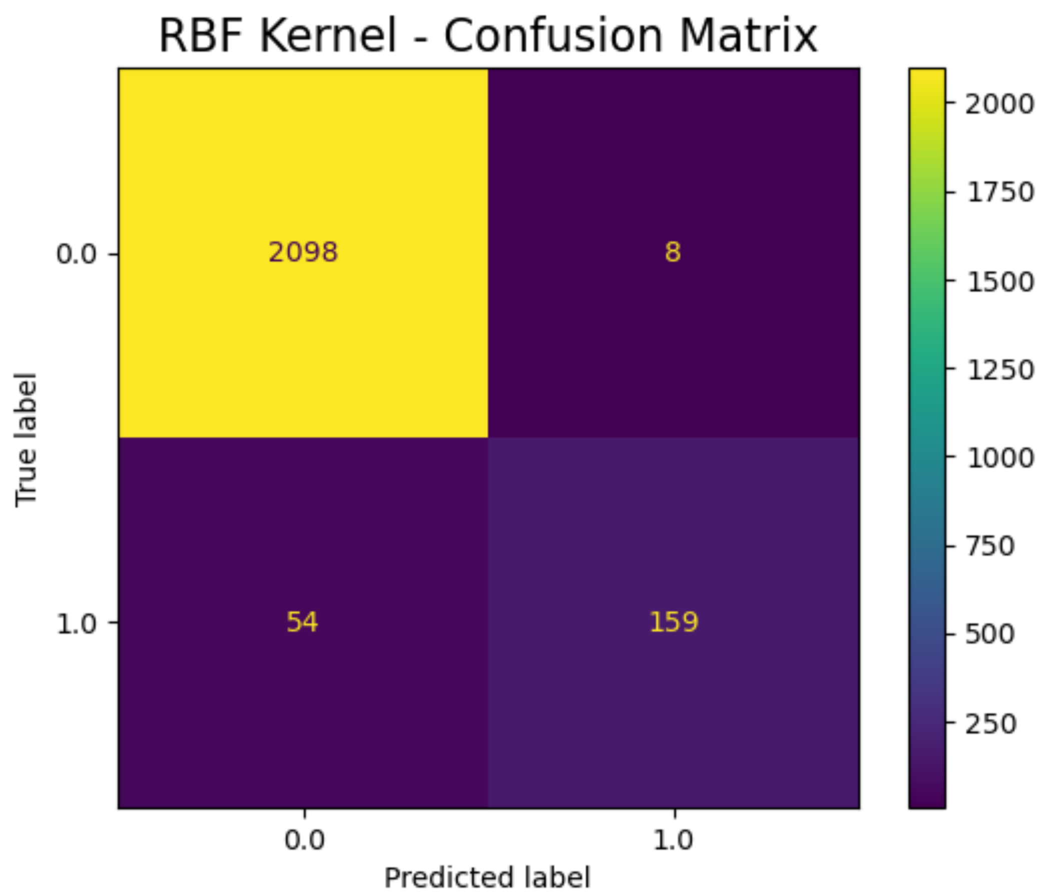
1. Now, Create a SVM Model with rbf kernel and C=100
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot Confusion Matrix from the true & predicted test data (Make font size 16)

**A7** Replace ??? with code in the code cell below

In [ ]:
```
# instantiate classifier with rbf kernel and C=100
svc = SVC(C=100.0, kernel='rbf')

# fit classifier to training set
svc.fit(xtrain, ytrain)

# make predictions on test set
rbf_pred = svc.predict(xtest)

# compute and print accuracy score
acc = accuracy_score(ytest, rbf_pred)
print('Model accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(acc))
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.9733
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as pl

cm = confusion_matrix(ytest, rbf_pred, labels=svc.classes_)
cm_disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svc.classes_)

cm_disp.plot()
plt.title("RBF Kernel - Confusion Matrix", fontsize=16)
plt.show()
```




# SVM with linear kernel

**Q8** Create SVM Model on the training set, and evaluate

```
Task:
```

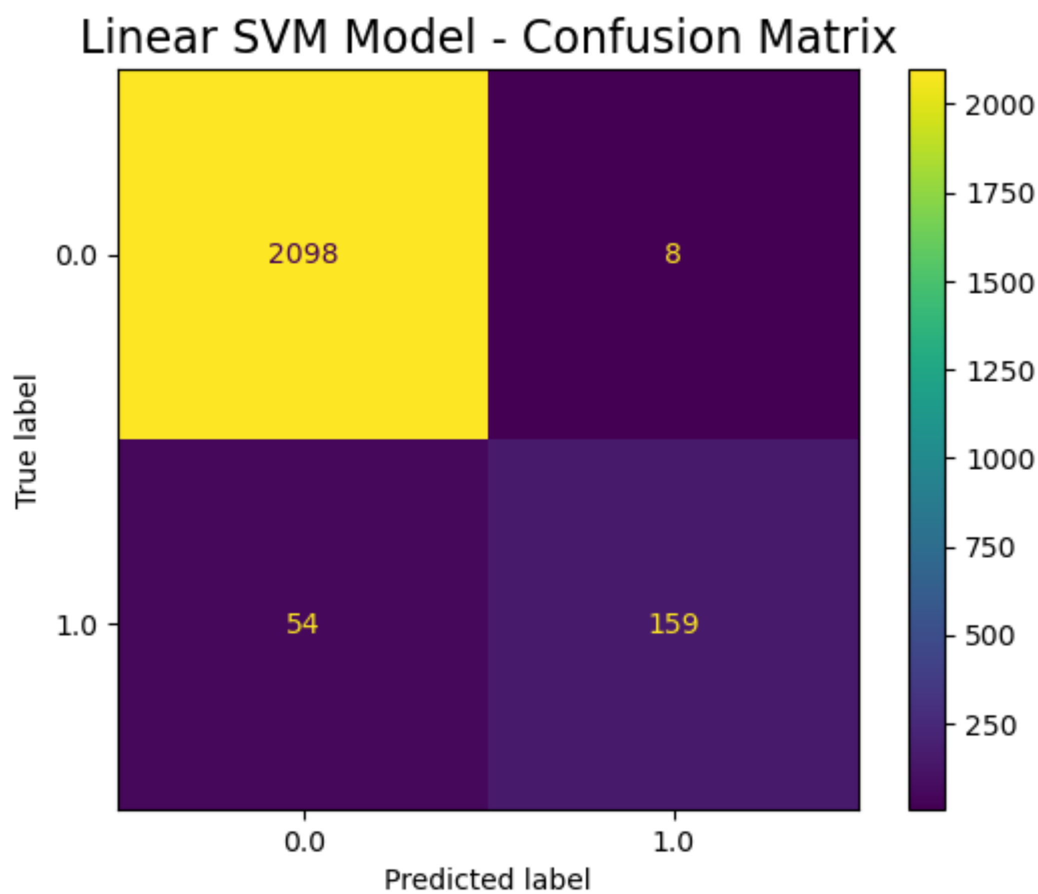
1. Now, Create a SVM Model with linear kernel and C=1.0
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot Confusion Matrix from the true & predicted test data (Make font size 16)

**A8** Replace ??? with code in the code cell below

In [ ]:
```
# instantiate classifier with linear kernel and C=1.0
linear_svc = SVC(C=1.0, kernel='linear')

# fit classifier to training set
linear_svc.fit(xtrain, ytrain)

# make predictions on test set
linear_pred = linear_svc.predict(xtest)

# compute and print accuracy score
lin_acc = accuracy_score(ytest, linear_pred)
print('Model accuracy score with linear kernel and C=1.0 : {0:0.4f}'. format(lin_ac
```

```
Model accuracy score with linear kernel and C=1.0 : 0.9741
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as pl

cm = confusion_matrix(ytest, rbf_pred, labels=svc.classes_)
cm_disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svc.classes_)

cm_disp.plot()
plt.title("Linear SVM Model - Confusion Matrix", fontsize=16)
plt.show()
```



**Q9** Create a Grid Search for finetuning the value of `C` in SVM Model on the `training set`,

`Task:`

1. Now, Create a SVM Model with linear kernel and evaluate the model for different values of C. Use `'C': [0.01, 0.1, 5, 10, 100]`
2. Use the sklearn GridSearchCV method for finetuning the `linear SVM`.
3. Use `3` fold of Cross Validation
4. Use `accuracy` as the scoring technique
5. Use `clf.cv_results_` & `clf.best_params_` for getting the fine-tuned results from the Cross Validation run.
6. Now, Plot the Confusion Matrix for test data, using the `best value of C` we found from our finetune.

Note: The Grid Search may take couple of minutes. Please wait untill the cell compiles

**A9** Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

# Select the optimal C parameter by cross-validation
tuned_parameters = {'C':[0.01, 0.1, 5, 10, 100]}
svc = svm.SVC(kernel='linear')
clf = GridSearchCV(svc, tuned_parameters, scoring='accuracy', cv=3)
```

In [ ]:
```
clf.fit(xtrain, ytrain)
```

Out[ ]:

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

In [ ]:
```
clf.cv_results_
```

Out[ ]:

```
{'mean_fit_time': array([  0.26187881,   0.80521607,  16.79767434,  28.79816222,
        137.95772982]),
 'std_fit_time': array([ 0.0424564 ,  0.04078544,  2.14925627,  7.25246291, 14.8237
9017]),
 'mean_score_time': array([0.02810756, 0.02735662, 0.02347398, 0.02578712, 0.022703
25]),
 'std_score_time': array([0.00225142, 0.00437647, 0.00208902, 0.00194161, 0.0012843
1]),
 'param_C': masked_array(data=[0.01, 0.1, 5, 10, 100],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.01}, {'C': 0.1}, {'C': 5}, {'C': 10}, {'C': 100}],
 'split0_test_score': array([0.97713546, 0.98101812, 0.98101812, 0.98101812, 0.9827
4374]),
 'split1_test_score': array([0.97670406, 0.97929249, 0.98188093, 0.98188093, 0.9801
5531]),
 'split2_test_score': array([0.97411562, 0.97799827, 0.97886109, 0.97842968, 0.9788
6109]),
 'mean_test_score': array([0.97598504, 0.9794363 , 0.98058671, 0.98044291, 0.980586
71]),
 'std_test_score': array([0.00133357, 0.00123703, 0.00127003, 0.0014665 , 0.0016141
7]),
 'rank_test_score': array([5, 4, 1, 3, 1])}
```

In [ ]:

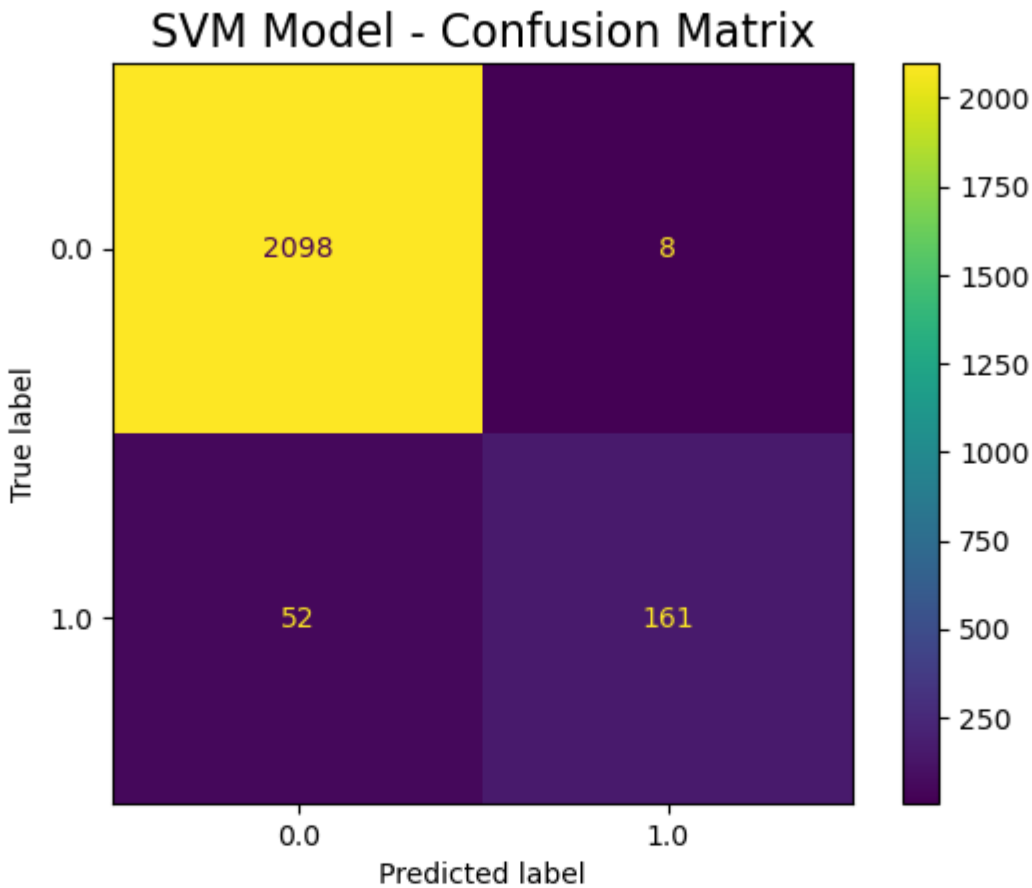
```
clf.best_params_
```

Out[ ]:

```
{'C': 5}
```

In [ ]:

```
best_model = SVC(C=5, kernel='linear')
best_model.fit(xtrain,ytrain)
best_pred = best_model.predict(xtest)

cm = confusion_matrix(ytest, best_pred, labels=best_model.classes_)
cm_disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_model.cla

cm_disp.plot()
plt.title("SVM Model - Confusion Matrix", fontsize=16)
plt.show()
```




We can see that after using the Best Value of `C`, we have less amount of false positive in our test data prediction.