

HW 5

This assignment covers Comparison 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 prropriate separator as input to read_csv()
```

```
df = pd.read_csv('data/HW5_data.csv')
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	target
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 [ ]: print(f'Rows in Dataframe: {len(df)}') # check the number of rows in the dataframe
```

Rows in Dataframe: 12528

```
In [ ]: #Drop NaNs if there is any
#df.?
df.dropna(axis=0, inplace=True)

df.head()
print(f'Rows in Dataframe: {len(df)}') # ensure the rows got dropped
```

Rows in Dataframe: 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 method
# y = ?
# Drop 'target_class' column from data frame,
# df.?
# Assign df values to x
# x =?
# View shape of x and y
# print(?)
# print(?)

y = df['target_class']
x = df.drop(columns=['target_class']) # I usually like to preserve the dataframe. I co
print(f'X Shape: {x.shape}')
print(f'Y Shape: {y.shape}')

# xtrain, xtest, ytrain, ytest = ?
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.25, random_state=1238)

X Shape: (9273, 8)
Y Shape: (9273,)
```

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 trained 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 = ?
#clf_2 = ?
#clf_3 = ?
```

```

#clf_4 = ?
clf_1 = DecisionTreeClassifier(max_depth=1, max_leaf_nodes=5, random_state=30, criteri
clf_2 = DecisionTreeClassifier(max_depth=2, max_leaf_nodes=10, random_state=30, criter
clf_3 = DecisionTreeClassifier(max_depth=5, max_leaf_nodes=15, random_state=30, criter
clf_4 = DecisionTreeClassifier(max_depth=25, max_leaf_nodes=25, random_state=30, crite

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

#predict
#?
y1_pred = clf_1.predict(xtest)
y2_pred = clf_2.predict(xtest)
y3_pred = clf_3.predict(xtest)
y4_pred = clf_4.predict(xtest)
#calculate mean_squared_error
#?
mse1 = mean_squared_error(ytest, y1_pred)
mse2 = mean_squared_error(ytest, y2_pred)
mse3 = mean_squared_error(ytest, y3_pred)
mse4 = mean_squared_error(ytest, y4_pred)

print(f'Model 1 MSE: {mse1}') # Models 1, 2, 3 tie when running on my computer. I chos
print(f'Model 2 MSE: {mse2}')
print(f'Model 3 MSE: {mse3}')
print(f'Model 4 MSE: {mse4}')

```

```

Model 1 MSE: 0.017248814144027597
Model 2 MSE: 0.017248814144027597
Model 3 MSE: 0.017248814144027597
Model 4 MSE: 0.018111254851228976

```

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
```

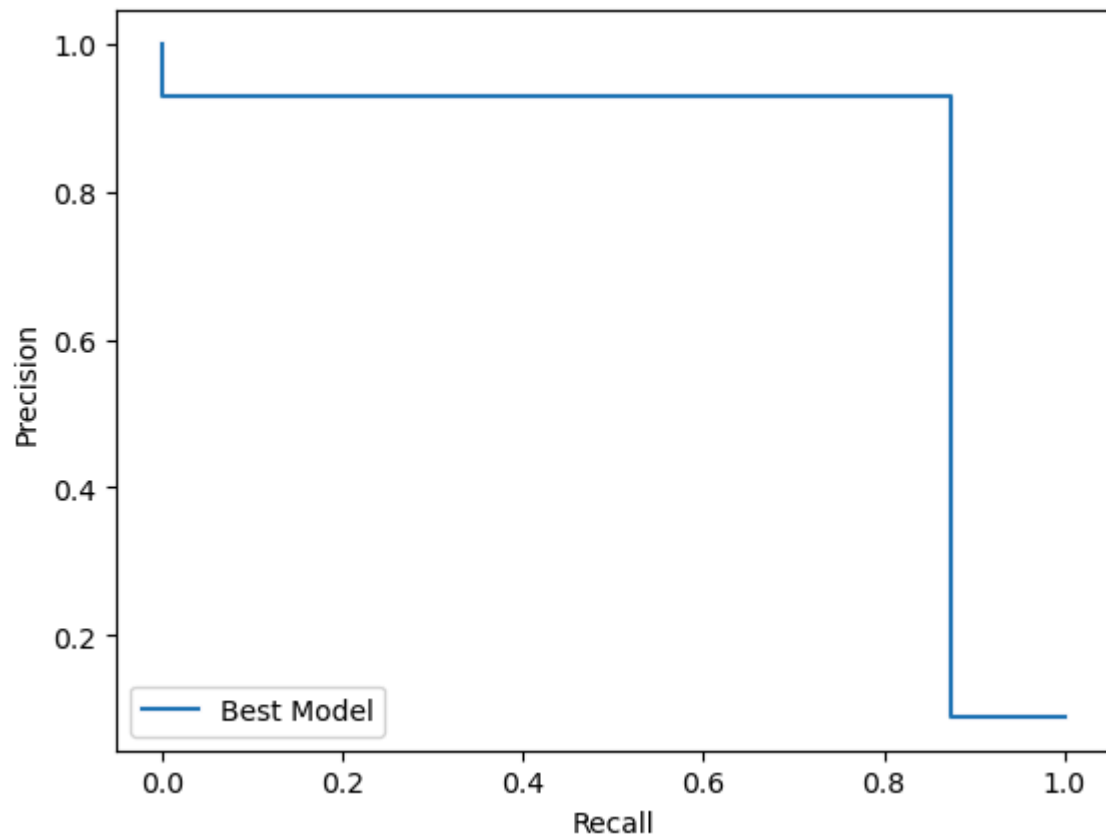
```

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

#?

```

```
#?
precision1, recall1, _ = precision_recall_curve(ytest, y1_pred)
PrecisionRecallDisplay(precision1, recall1, estimator_name='Best Model').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 = ?
#print("Cross-validation scores: {}".format(?))
#print("Average cross-validation score: {:.2f}".format(?))

scores = cross_val_score(clf_1, xtrain, ytrain, cv=5)
print(f'Scores {scores}')
avg_cross_val = sum(scores)/len(scores)
print(f'Average cross-validation score: {avg_cross_val}')
```

```
Scores [0.97843278 0.97915169 0.97483825 0.97915169 0.96690647]
Average cross-validation score: 0.9756961763443307
```

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

previous_best = clf_1
# Use BaggingRegressor to fit the training data
# Calculate the mean squared error

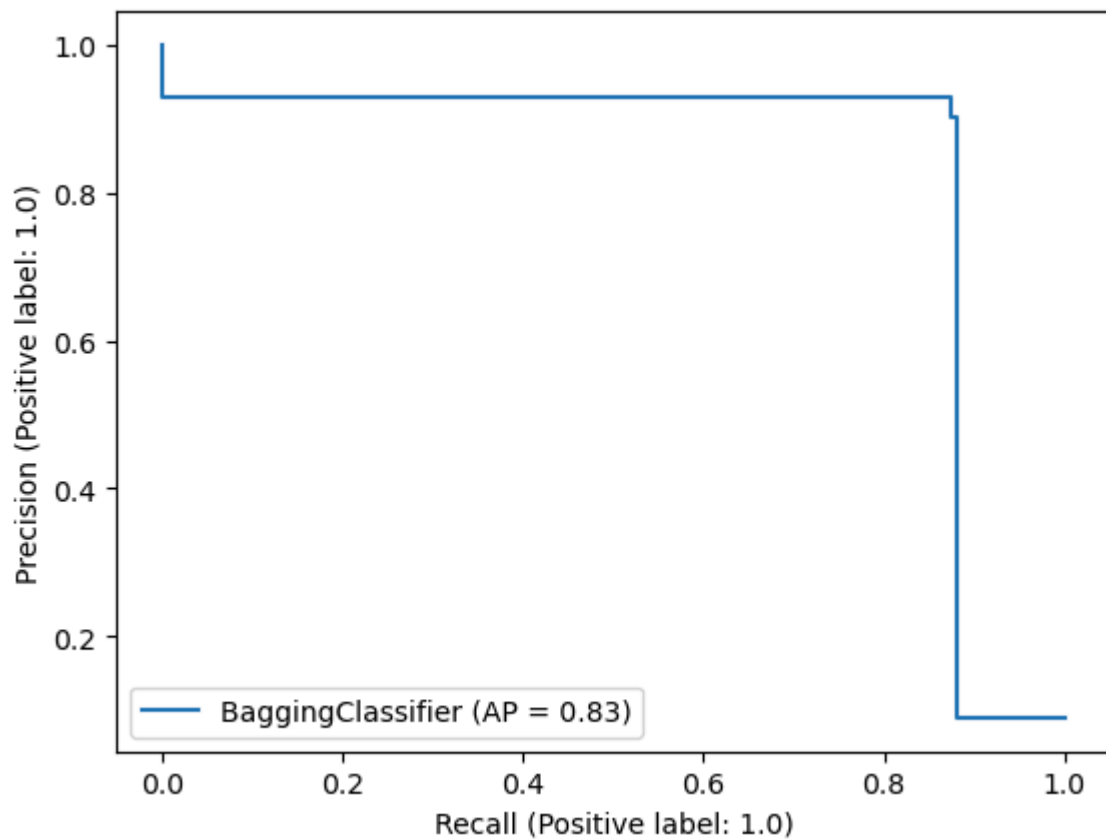
#Load BaggingRegressor model and pass n_estimators=10, random_state=1
#bagged_clf = ?
#bagged_clf.?
#pred = ?
#?

bagged_clf = BaggingClassifier(previous_best, n_estimators=10, random_state=1)
bagged_clf.fit(xtrain, ytrain)
pred = bagged_clf.predict(xtest)
print(mean_squared_error(ytest, pred))

0.017680034497628287
```

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

#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?

Bagging is called an ensembled technique because it will create multiple models and aggregate them to determine the classification. With Decision Trees, instead of creating 1 tree, multiple trees would be created, and the classification is chosen from the aggregation of the scores from the trees. A single decision tree is prone to overfitting, and a bag of trees increases the score while reducing variance.

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

More estimators means more models to train. Training models is a timely, CPU intensive process. Also, a large number of estimators does not mean that the score will improve. If a BaggingClassifier creates a collection of Decision Trees, where 5 trees may have been optimal, adding additional trees would only give diminishing returns and may even overfit the overall model.

Your answer goes here.

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 = SVC()

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

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

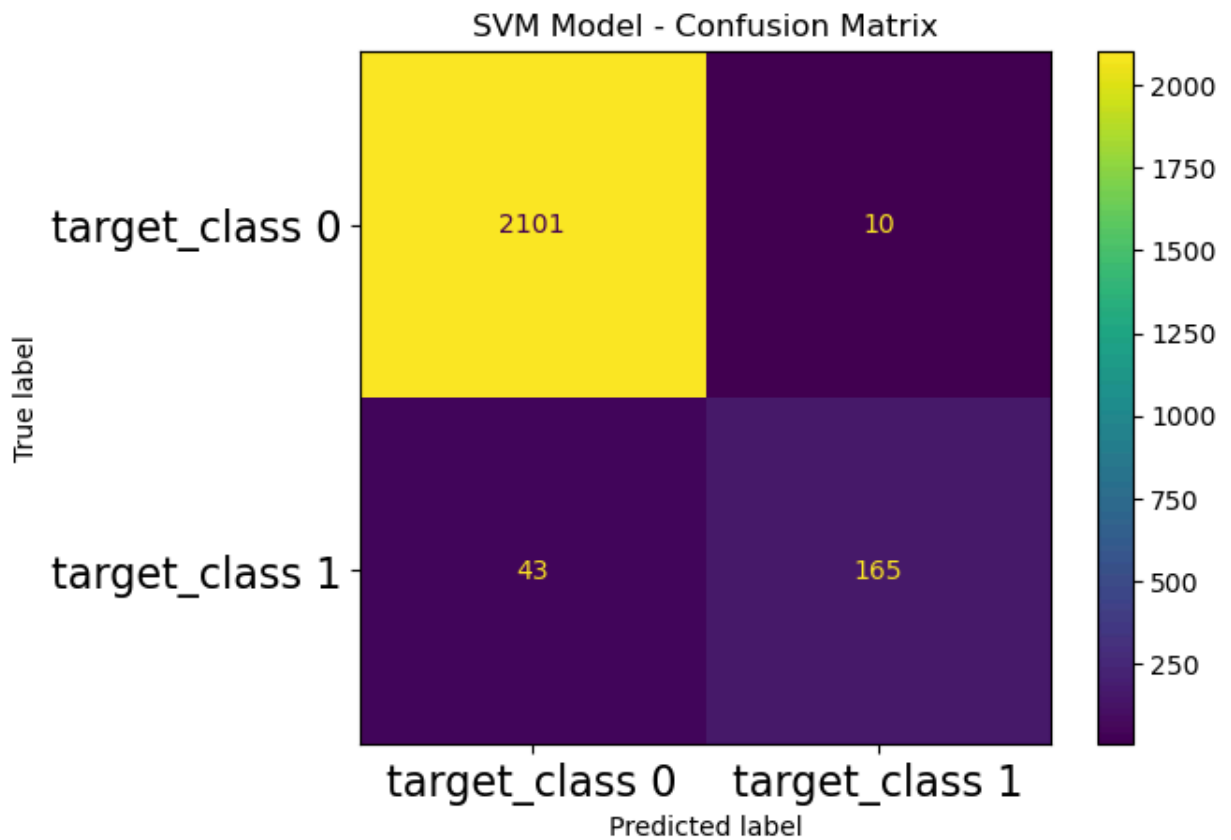
# compute and print accuracy score
#print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(?))
svc_accuracy = accuracy_score(ytest, ySVC_pred)
print(f'SVM Accuracy: {svc_accuracy}')
```

SVM Accuracy: 0.9771453212591634

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

#cm = ?
matrix = confusion_matrix(ytest, ySVC_pred)
cm = ConfusionMatrixDisplay(confusion_matrix=matrix)
cm.plot()
plt.title('SVM Model - Confusion Matrix')
# What are we mapping?
plt.xticks(range(2), ['target_class 0', 'target_class 1'], fontsize=16)
plt.yticks(range(2), ['target_class 0', 'target_class 1'], 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 situation?

Nearly 20% of Class 1 is predicted as Class 0. In terms of a Support Vector Classifier, this would suggest that a large number of Class 1's are on the Class 0 side of the hyperplane.

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

The margins could be changed to better classify the classes.

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 = SVC(kernel='rbf', C=100)

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

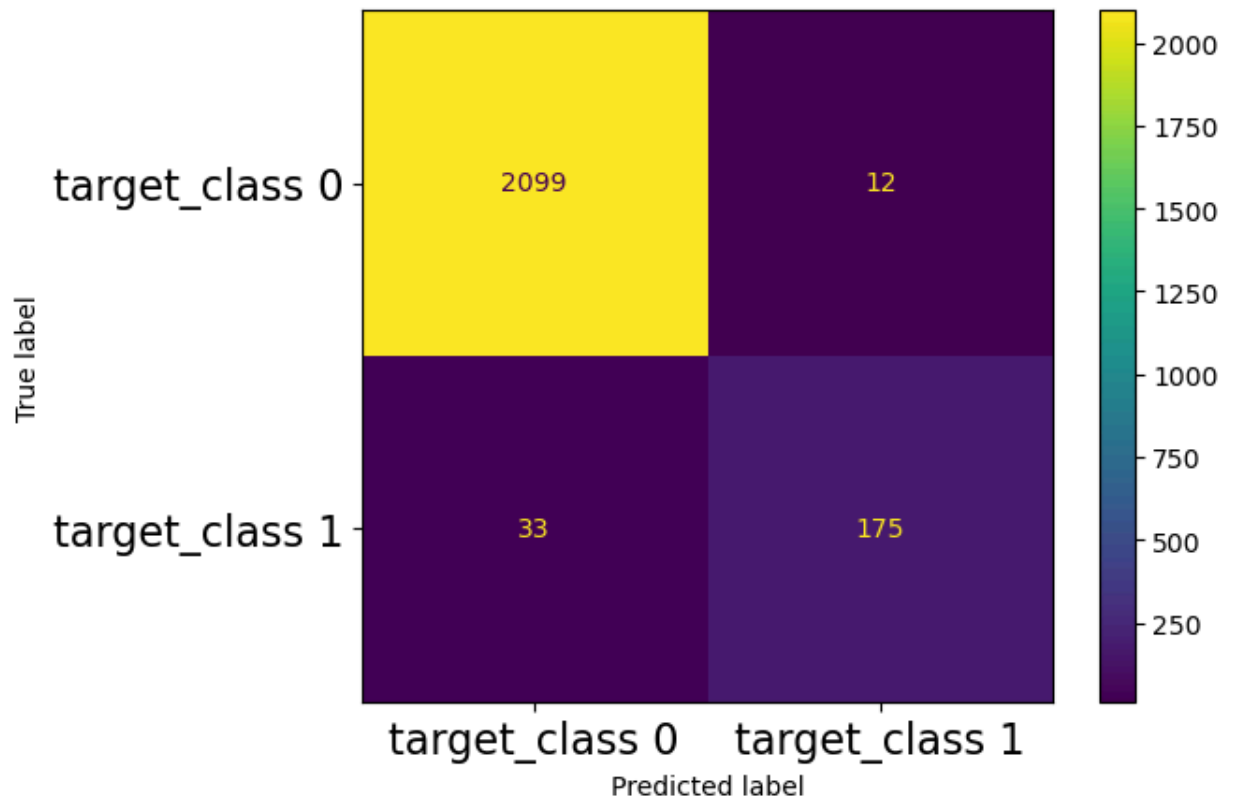
# make predictions on test set
#?
ySVC_pred = svc.predict(xtest)

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

Accuracy: 0.9805950840879689

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

matrix = confusion_matrix(ytest, ySVC_pred)
#cm = ?
cm = ConfusionMatrixDisplay(confusion_matrix=matrix)
cm.plot()
#?
#?
plt.xticks(range(2), ['target_class 0', 'target_class 1'], fontsize = 16)
plt.yticks(range(2), ['target_class 0', 'target_class 1'], 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 = SVC(kernel='linear', C=1)

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

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

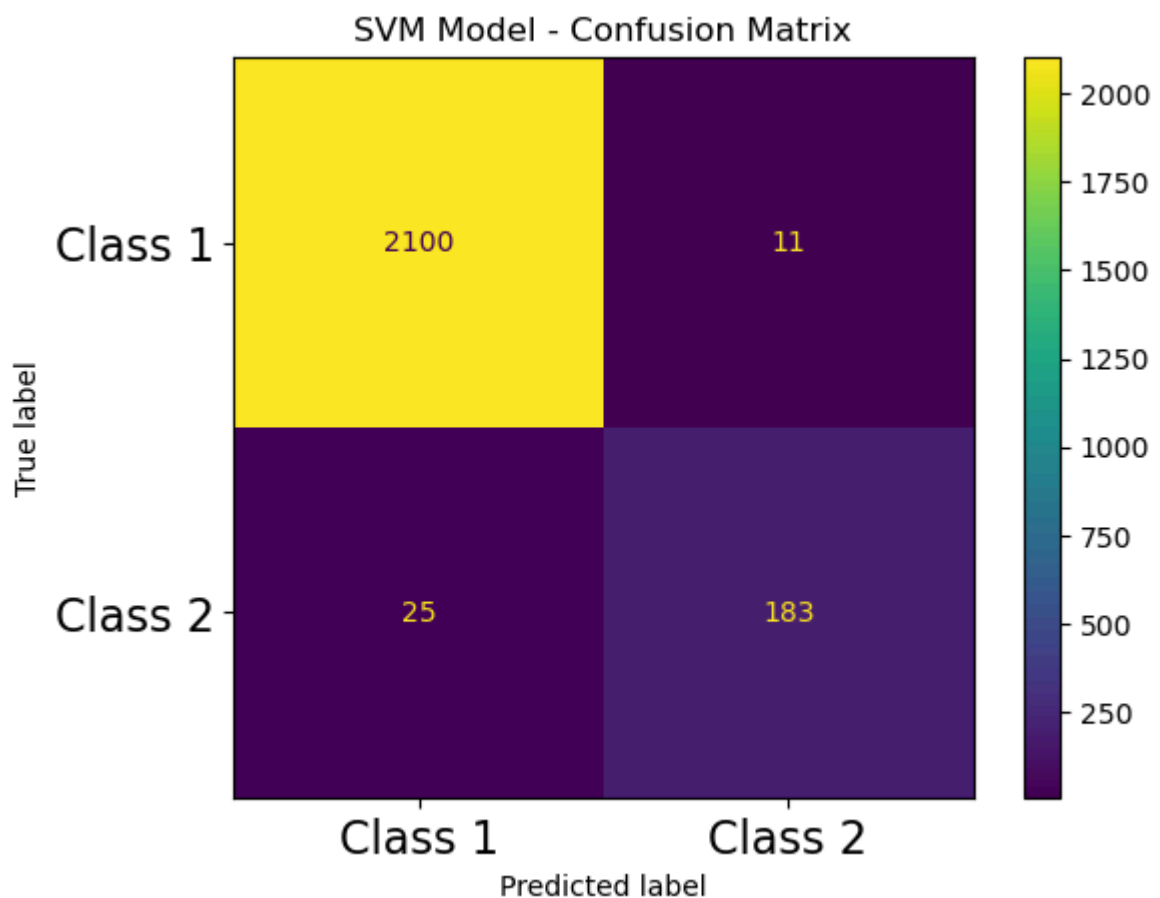
# compute and print accuracy score
#print('Model accuracy score with Linear kernel and C=1.0 : {0:0.4f}'.format(?))
```

```
svc_accuracy = accuracy_score(ytest, ySVC_pred)
print(f'Accuracy: {svc_accuracy}')
```

Accuracy: 0.9844760672703752

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

matrix = confusion_matrix(ytest, ySVC_pred)
cm = ConfusionMatrixDisplay(confusion_matrix=matrix)
cm.plot()
#cm = ?
plt.title("SVM Model - Confusion Matrix")
plt.xticks(range(2), ['Class 1', 'Class 2'], fontsize=16)
plt.yticks(range(2), ['Class 1', 'Class 2'], 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 until the cell compiles

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

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

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

```
Out[ ]: ▸ GridSearchCV
        ▸ estimator: SVC
```

```
          ▸ SVC
```

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

```

Out[ ]: {'mean_fit_time': array([ 0.15799952,  0.53960299, 11.01528454, 18.8787957 , 88.64911
222]),
'std_fit_time': array([1.24296021e-02, 4.51557244e-02, 1.26260911e+00, 3.82136306e+0
0,
3.36815962e+01]),
'mean_score_time': array([0.01466735, 0.01350236, 0.01266726, 0.01283216, 0.0115010
7]),
'std_score_time': array([0.00085032, 0.00081401, 0.00085112, 0.00084893, 0.0008144
]),
'param_C': masked_array(data=[0.01, 0.1, 5, 10, 100],
mask=[False, False, False, False, False],
fill_value='?',
dtype=object),
'param_kernel': masked_array(data=['linear', 'linear', 'linear', 'linear', 'linea
r'],
mask=[False, False, False, False, False],
fill_value='?',
dtype=object),
'params': [{'C': 0.01, 'kernel': 'linear'},
{'C': 0.1, 'kernel': 'linear'},
{'C': 5, 'kernel': 'linear'},
{'C': 10, 'kernel': 'linear'},
{'C': 100, 'kernel': 'linear'}],
'split0_test_score': array([0.97540984, 0.97799827, 0.98015531, 0.9797239 , 0.979292
49]),
'split1_test_score': array([0.97411562, 0.97756687, 0.97842968, 0.97842968, 0.977135
46]),
'split2_test_score': array([0.9659189 , 0.96937015, 0.97454702, 0.97454702, 0.974547
02]),
'mean_test_score': array([0.97181478, 0.97497843, 0.97771067, 0.97756687, 0.9769916
6]),
'std_test_score': array([0.00420237, 0.00396956, 0.00234534, 0.00219975, 0.00194
]),
'rank_test_score': array([5, 4, 1, 2, 3])}

```

```

In [ ]: #clf.?
clf.best_params_

```

```

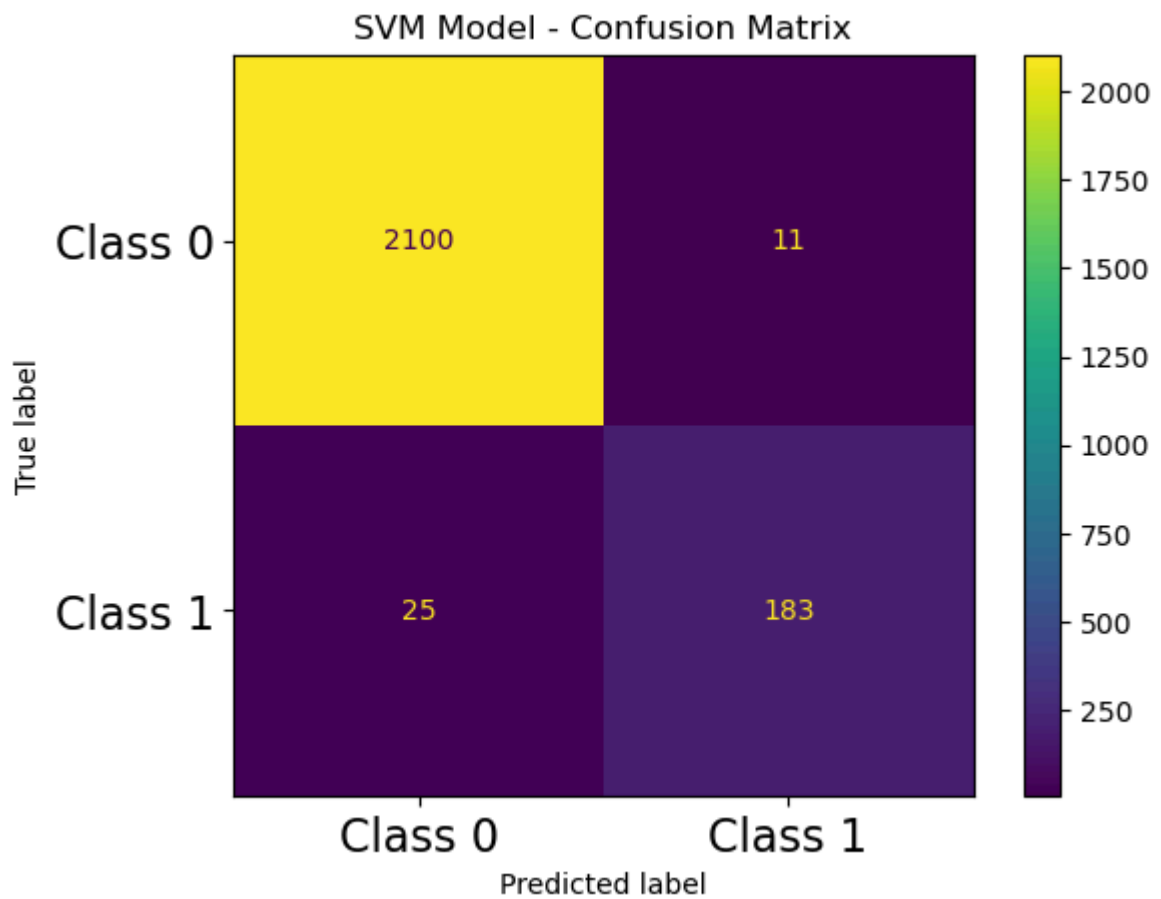
Out[ ]: {'C': 5, 'kernel': 'linear'}

```

```

In [ ]: #best_model=?
#best_model.?
#cm = ?
best_model = SVC(kernel=clf.best_params_['kernel'], C=clf.best_params_['C'])
best_model.fit(xtrain, ytrain)
yBest_pred = best_model.predict(xtest)
matrix = confusion_matrix(ytest, yBest_pred)
cm = ConfusionMatrixDisplay(confusion_matrix=matrix)
cm.plot()
plt.title("SVM Model - Confusion Matrix")
#?
plt.xticks(range(2), ['Class 0', 'Class 1'], fontsize=16)
plt.yticks(range(2), ['Class 0', 'Class 1'], 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.