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 (README.md) for homework submission instructions

Related Tutorials

- Decision Tree with KFold Cross Validation (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)
- Decision Tree with Bagging (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html#sklearn.ensemble.BaggingF
- Support Vector Machine (https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)

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 [2]:
```
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[2]:

	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_class
0	121.156250	48.372971	0.375485	-0.013165	3.168896	18.399367	7.449874	65.159298	0.0
1	76.968750	36.175557	0.712898	3.388719	2.399666	17.570997	9.414652	102.722975	0.0
2	130.585938	53.229534	0.133408	-0.297242	2.743311	22.362553	8.508364	74.031324	0.0
3	156.398438	48.865942	-0.215989	-0.171294	17.471572	NaN	2.958066	7.197842	0.0
4	84.804688	36.117659	0.825013	3.274125	2.790134	20.618009	8.405008	76.291128	0.0
5	121.007812	47.176944	0.229708	0.091336	2.036789	NaN	9.546051	112.131721	0.0
6	79.343750	42.402174	1.063413	2.244377	141.641304	NaN	-0.700809	-1.200653	0.0
7	109.406250	55.912521	0.565106	0.056247	2.797659	19.496527	9.443282	97.374578	0.0
8	95.007812	40.219805	0.347578	1.153164	2.770067	18.217741	7.851205	70.801938	0.0
9	109.156250	47.002234	0.394182	0.190296	4.578595	NaN	5.702532	36.342493	0.0

In [3]:
```
# check if there is NaN in the dataset
df.isnull().sum()
```

Out[3]:
```
 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 [4]:
```
#Drop NaNs if there is any
df.dropna(inplace=True)

# Count number of entries for different target_class
df['target_class'].value_counts()
```

Out[4]:
```
target_class
0.0    8423
1.0     850
Name: count, dtype: int64
```

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 [6]:
```
# Assign values of ```target_class``` column to y, note you have to use .values me
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
# View shape of x and y
print(x.shape)
print(y.shape)

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

```
(9273, 8)
(9273,)
```

Decision Tree

Decision Tree with different depth

Q3 Train DecisionTreeClassifier Model at different depths

1. Create four [DecisionTreeClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) 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 [10]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error

#create decision tree classifier
clf_1 = DecisionTreeClassifier(max_depth=1, max_leaf_nodes=5, random_state=30, cri
clf_2 = DecisionTreeClassifier(max_depth=2, max_leaf_nodes=10, random_state=30, cr
clf_3 = DecisionTreeClassifier(max_depth=5, max_leaf_nodes=15, random_state=30, cr
clf_4 = DecisionTreeClassifier(max_depth=25, max_leaf_nodes=25, random_state=30, c

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

#predict
predictions_1 = clf_1.predict(xtest)
predictions_2 = clf_2.predict(xtest)
predictions_3 = clf_3.predict(xtest)
predictions_4 = clf_4.predict(xtest)

#calculate mean_squared_error
mse_1 = mean_squared_error(ytest, predictions_1)
mse_2 = mean_squared_error(ytest, predictions_2)
mse_3 = mean_squared_error(ytest, predictions_3)
mse_4 = mean_squared_error(ytest, predictions_4)

# Print MSE for each model
print("MSE for clf_1:", mse_1)
print("MSE for clf_2:", mse_2)
print("MSE for clf_3:", mse_3)
print("MSE for clf_4:", mse_4)
```

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

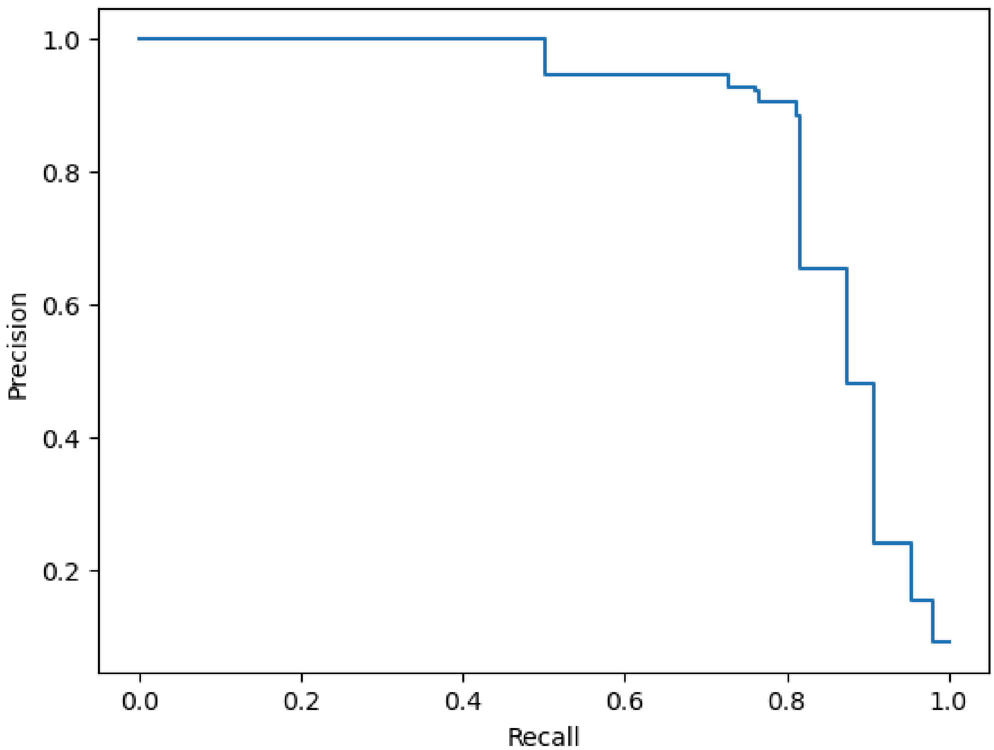
- If you are using conda, you can upgrade with

conda upgrade -c conda-forge scikit-learn

- or, with pip,

python -m pip install scikit-learn --upgrade

In [12]:
```
# 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

best_model = clf_3
probabilities = best_model.predict_proba(xtest)[:, 1]

precision, recall, _ = precision_recall_curve(ytest, probabilities)

# Plot the Precision-Recall curve
disp = PrecisionRecallDisplay(precision=precision, recall=recall)
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 [13]:
```
from sklearn.model_selection import KFold, cross_val_score

kf = KFold(n_splits=5, shuffle=True, random_state=30)

scores = cross_val_score(best_model, xtest, ytest, cv=kf)
print("Cross-validation scores: {}".format(scores))
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

```
Cross-validation scores: [0.96767241 0.9612069  0.96982759 0.97413793 0.97192225]
Average cross-validation score: 0.97
```

Decision Tree with Bagging

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

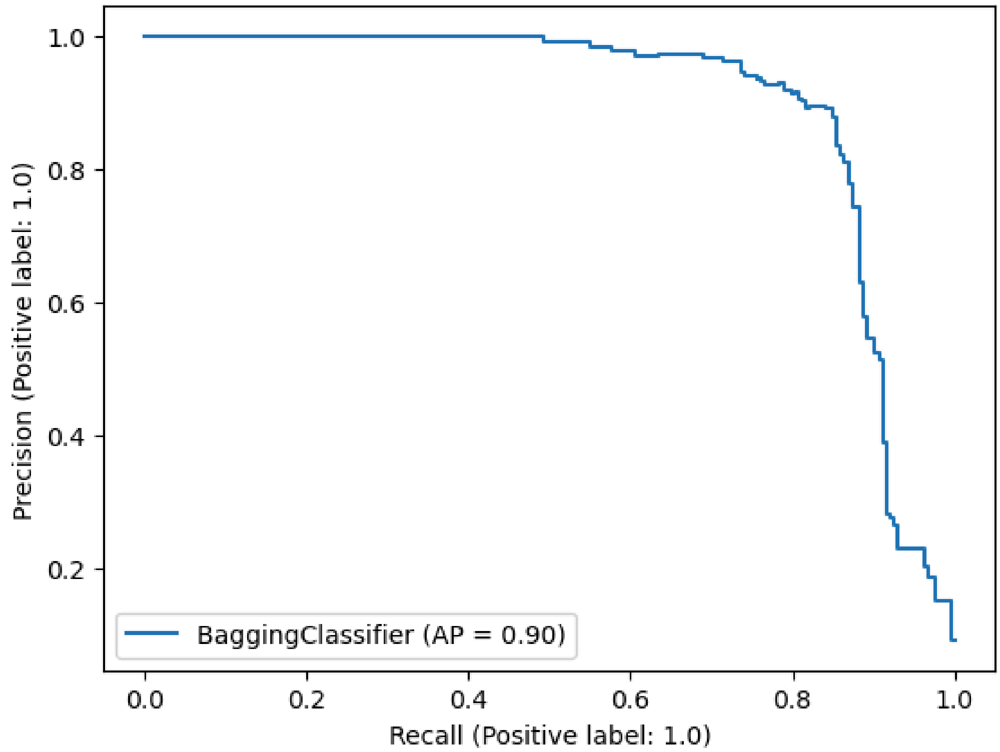
`Part 1:`

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 [19]:
```
from sklearn.ensemble import BaggingClassifier

# 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 = BaggingClassifier(estimator=best_model, n_estimators=10, random_state=
bagged_clf.fit(xtrain, ytrain)
pred = bagged_clf.predict(xtest)
test_mse = mean_squared_error(ytest, pred)
```

In [20]:
```
#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?

because it combines the predictions of serveral base estimators. It works better than most of the time compared to single model classifiers because it will have more diversity amoung base estimators, which can lead to better generalization.

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

increasing the number of estimators can lead to overfitting and having to use more computing power than needed. And example would be if with 10 decision trees the accuracy is 95% and by increasing the number of trees to 100 the accuracy is 95.5%. So even with a lot more trees the accuracy only went up .5%.

Support Vector Machine(SVM)

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

Part:1

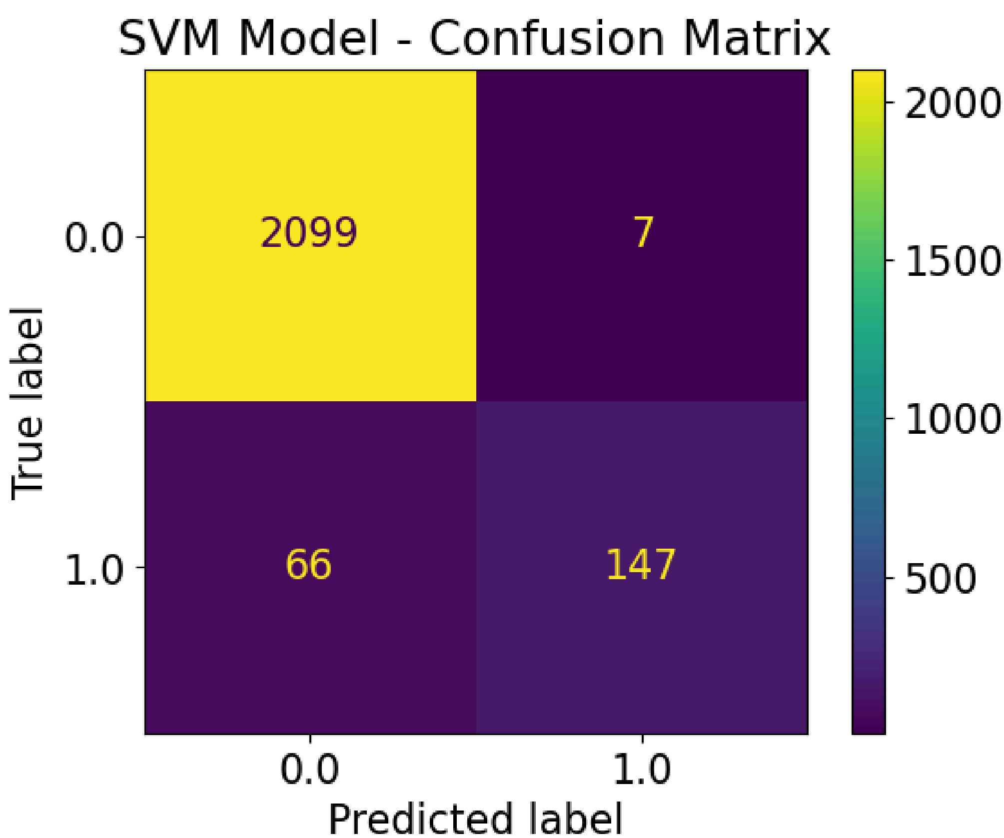
1. Now, Create a SVM Model with default parameters
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot confusion matrix on the test data (Make font size 16)

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

In [25]:
```
# 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
ypred = svc.predict(xtest)


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

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

In [26]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = ConfusionMatrixDisplay.from_estimator(svc, xtest, ytest)

plt.title("SVM Model - Confusion Matrix")
plt.rcParams.update({'font.size': 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 reasoning is because default c is equal to 1 and the kernel is equal to RBF, so its not sensitive enough and is causing underfitting of the data.

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

increasing the sensitivity.

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:`

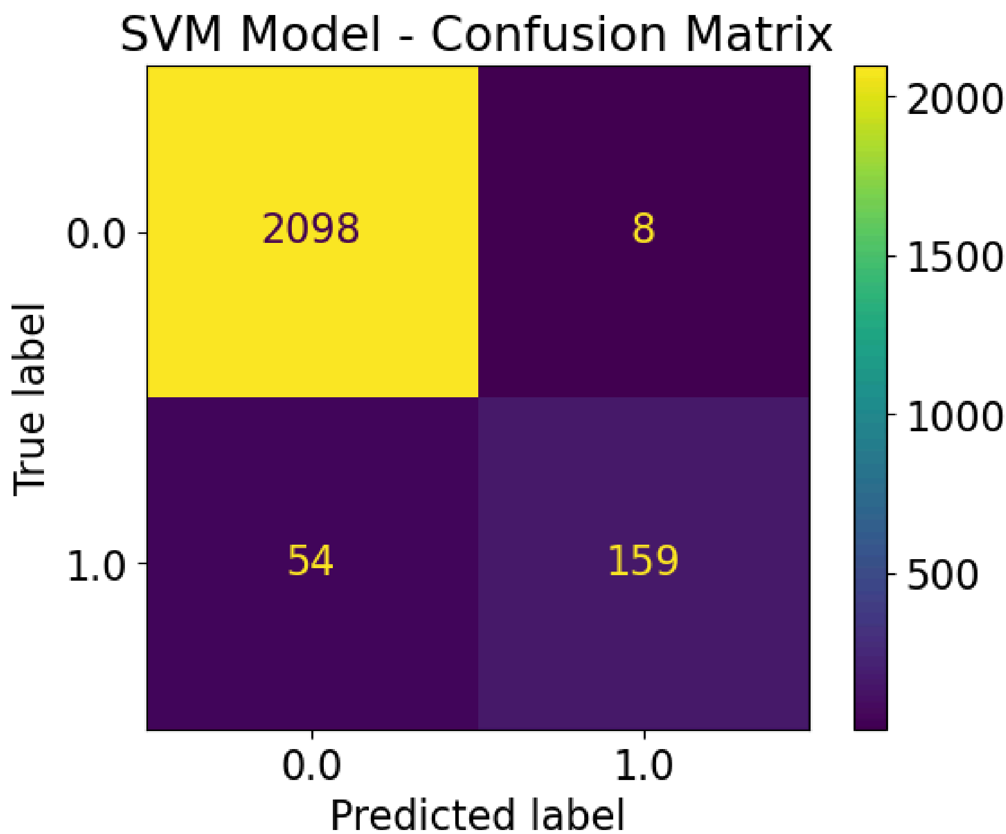
1. Now, Create a SVM Model with rbf kernel and C=100
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot Confusion Matrix from the true & predicted test data (Make font size 16)

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

In [27]:
```
# instantiate classifier with rbf kernel and C=100
svc=SVC(kernel='rbf', C=100)


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


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


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

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

In [28]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as pl

cm = ConfusionMatrixDisplay.from_estimator(svc, xtest, ytest)
plt.title("SVM Model - Confusion Matrix")
plt.rcParams.update({'font.size': 16})
plt.show()
```



SVM with linear kernel

Q8 Create SVM Model on the training set, and evaluate

```
Task:
```

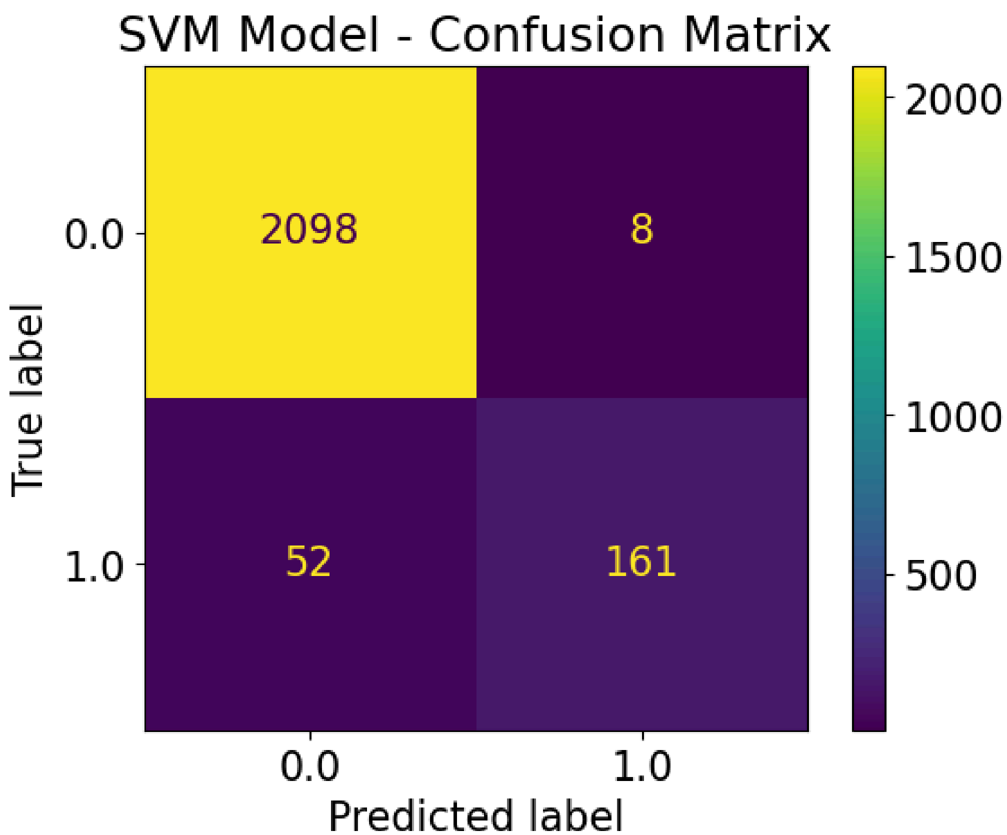
1. Now, Create a SVM Model with linear kernel and C=1.0
2. Fit the model with the train data
3. Predict the values with the test data
4. Calculate the model accuracy on test data
5. Plot Confusion Matrix from the true & predicted test data (Make font size 16)

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

In [29]:
```
# instantiate classifier with linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0)


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


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


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

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

In [30]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as pl

cm = ConfusionMatrixDisplay.from_estimator(linear_svc, xtest, ytest)
plt.title("SVM Model - Confusion Matrix")
plt.rcParams.update({'font.size': 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 (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) 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

In [31]:
```
from sklearn.model_selection import GridSearchCV

# Select the optimal C parameter by cross-validation
tuned_parameters = [{'C': [0.01, 0.1, 1, 5, 10, 100]}]
clf = GridSearchCV(SVC(kernel='linear'), tuned_parameters, scoring='accuracy', cv=:
clf.fit(xtrain, ytrain)
```

Out[31]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

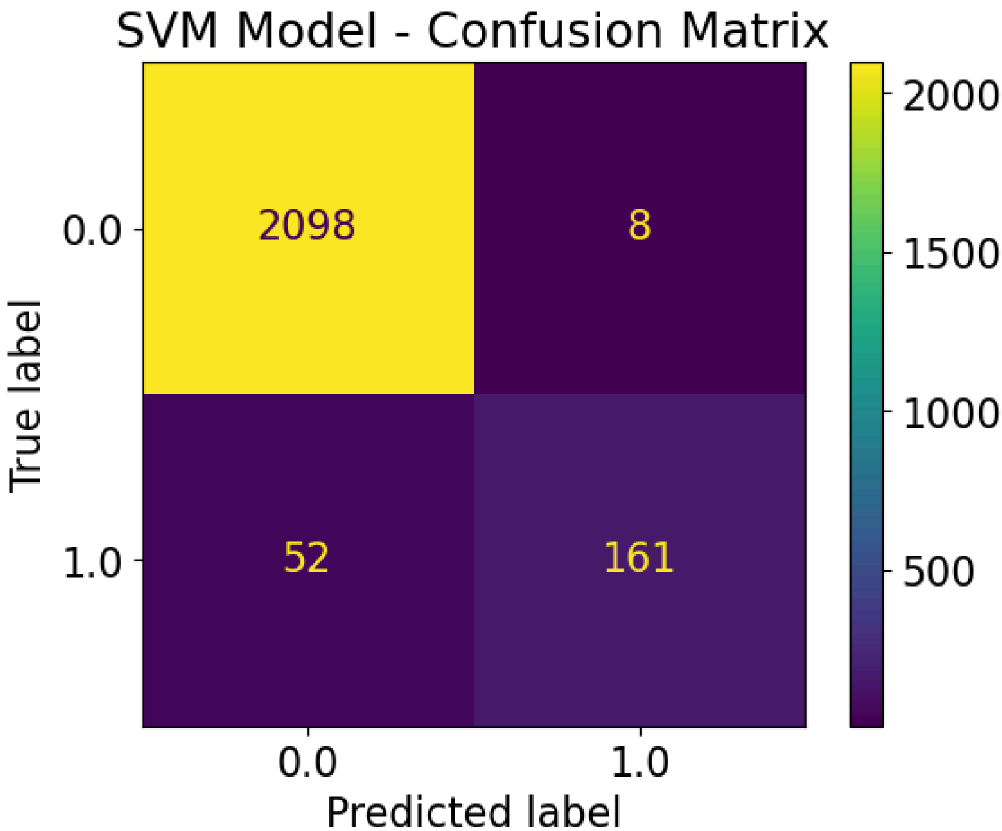
In [32]:
```
clf.best_params_
```

Out[32]:
```
{'C': 5}
```

In [33]:
```
clf.best_score_
```

Out[33]:
```
0.9805867126833476
```

In [34]:
```
best_model=clf.best_estimator_
cm = ConfusionMatrixDisplay.from_estimator(best_model, xtest, ytest)
plt.title("SVM Model - Confusion Matrix")
plt.rcParams.update({'font.size': 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.