AMS 580 - Team Project 2 - Python - SVM

Eshan Shakrani, Vishnu Teja Sardee, Priyansh Desai, Mustafa Isik

Importing Necessary Libraries

In [ ]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_scor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

from sklearn.svm import SVC
```

Problem 1

Read in + Clean the Data

In [ ]:
```
# read in the data
df = pd.read_csv('../Data/Titanic.csv')

df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [ ]:
```
print("Original Columns:", df.columns.tolist())

# remove the "Name", "Ticket", and "Cabin" columns
df = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis = 1)

print("After removing:", df.columns.tolist())
```

```
Original Columns: ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'Sib
Sp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
After removing: ['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Em
barked']
```

In [ ]:
```
print(f"There were originally {len(df)} observations.")

# remove observations with missing values in the "Age" column
df = df.dropna(subset=['Age'])

print(f'After removing NA from "Age", there are {len(df)} observations left.')
```

```
There were originally 891 observations.
After removing NA from "Age", there are 714 observations left.
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

In [ ]:
```
# there are still NA values in the "Embarked" column so remove those as well
df = df.dropna()

print(f'There are now {len(df)} observations left.')
```

```
There are now 712 observations left.
```

In [ ]:
```
stats = df.describe()
stats
```

Out[ ]:

	Survived	Pclass	Age	SibSp	Parch	Fare
count	712.000000	712.000000	712.000000	712.000000	712.000000	712.000000
mean	0.404494	2.240169	29.642093	0.514045	0.432584	34.567251
std	0.491139	0.836854	14.492933	0.930692	0.854181	52.938648
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	20.000000	0.000000	0.000000	8.050000
50%	0.000000	2.000000	28.000000	0.000000	0.000000	15.645850
75%	1.000000	3.000000	38.000000	1.000000	1.000000	33.000000
max	1.000000	3.000000	80.000000	5.000000	6.000000	512.329200

Data Standardization

In [ ]:
```
# create a StandardScaler object
ss = StandardScaler()

# fit the scaler to the "Age" and "Fare" columns and transform
df[['Age_Scaled', 'Fare_Scaled']] = ss.fit_transform(df[['Age', 'Fare']])

df.head()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Age_Scaled	Fare_Scaled
0	0	3	male	22.0	1	0	7.2500	S	-0.527669	-0.516380
1	1	1	female	38.0	1	0	71.2833	C	0.577094	0.694046
2	1	3	female	26.0	0	0	7.9250	S	-0.251478	-0.503620
3	1	1	female	35.0	1	0	53.1000	S	0.369951	0.350326
4	0	3	male	35.0	0	0	8.0500	S	0.369951	-0.501257

In [ ]:
```
# remove the original "Age" and "Fare" columns
df = df.drop(['Age', 'Fare'], axis = 1)

df.head()
```

Out[ ]:

	Survived	Pclass	Sex	SibSp	Parch	Embarked	Age_Scaled	Fare_Scaled
0	0	3	male	1	0	S	-0.527669	-0.516380
1	1	1	female	1	0	C	0.577094	0.694046
2	1	3	female	0	0	S	-0.251478	-0.503620
3	1	1	female	1	0	S	0.369951	0.350326
4	0	3	male	0	0	S	0.369951	-0.501257

In [ ]:
```
print(f'There are currently {len(df)} passengers.')
```

```
There are currently 712 passengers.
```

Encoding Categorical Variables

In [ ]:
```
# 'Embarked' and 'Parch' features are both categorical
df['Embarked'].unique()
```

Out[ ]:
```
array(['S', 'C', 'Q'], dtype=object)
```

In [ ]:
```
df['Sex'].unique()
```

Out[ ]:
```
array(['male', 'female'], dtype=object)
```

In [ ]:
```
# use a One Hot Encoder to encode the categorical features
ohe = OneHotEncoder(sparse=False)
transformed = ohe.fit_transform(df[['Sex', 'Embarked']])
features_transformed = ohe.get_feature_names_out()
features_transformed
```

Out[ ]:
```
array(['Sex_female', 'Sex_male', 'Embarked_C', 'Embarked_Q', 'Embarked_S'],
      dtype=object)
```

In [ ]:
```
# add the encoded features to the dataset
df[features_transformed] = transformed
```

In [ ]:
```
df.head()
```

Out[ ]:

	Survived	Pclass	Sex	SibSp	Parch	Embarked	Age_Scaled	Fare_Scaled	Sex_female	Sex_m
0	0	3	male	1	0	S	-0.527669	-0.516380	0.0	
1	1	1	female	1	0	C	0.577094	0.694046	1.0	
2	1	3	female	0	0	S	-0.251478	-0.503620	1.0	
3	1	1	female	1	0	S	0.369951	0.350326	1.0	
4	0	3	male	0	0	S	0.369951	-0.501257	0.0	

In [ ]:
```
# drop the original 'Sex' and 'Embarked' columns
df = df.drop(['Sex', 'Embarked'], axis = 1)
df.head()
```

Out[ ]:

	Survived	Pclass	SibSp	Parch	Age_Scaled	Fare_Scaled	Sex_female	Sex_male	Embarked_C	E
0	0	3	1	0	-0.527669	-0.516380	0.0	1.0	0.0	
1	1	1	1	0	0.577094	0.694046	1.0	0.0	1.0	
2	1	3	0	0	-0.251478	-0.503620	1.0	0.0	0.0	
3	1	1	1	0	0.369951	0.350326	1.0	0.0	0.0	
4	0	3	0	0	0.369951	-0.501257	0.0	1.0	0.0	

Partitioning the Data

In [ ]:
```
# split the data into features (X) and target (y)
X = df.drop('Survived', axis = 1)
y = df['Survived']
```

In [ ]:
```
# split the data into 75% training and 25% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

Problem 2

SVM with Linear Kernel

In [ ]:
```
# SVM classifier with linear kernel and default cost "C"
svc2 = SVC(kernel='linear')
```

In [ ]:
```
# train the SVC on the training data
svc2.fit(X_train, y_train)
```

Out[ ]:
```
SVC(kernel='linear')
```

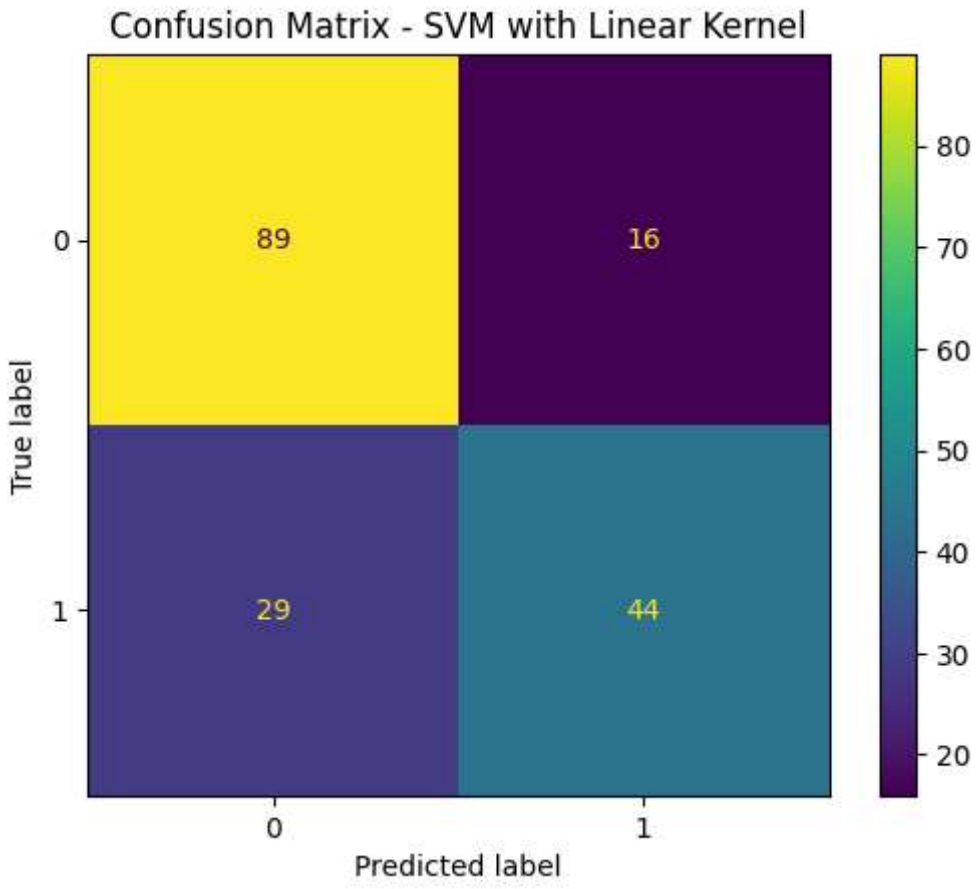
In [ ]:
```
# use the SVC to make predictions on the testing data
y_pred2 = svc2.predict(X_test)
```

Evaluate Performance

In [ ]:
```
# generate the confusion matrix
cm2 = confusion_matrix(y_test, y_pred2, labels = svc2.classes_)
cm2
```

Out[ ]:
```
array([[89, 16],
       [29, 44]], dtype=int64)
```

In [ ]:
```
# better visualization of confusion matrix
disp_cm2 = ConfusionMatrixDisplay(cm2, display_labels=svc2.classes_)
disp_cm2.plot()
plt.title("Confusion Matrix - SVM with Linear Kernel")
plt.show()
```



In [ ]:
```
# extract the elements of the confusion matrix
tn2, fp2, fn2, tp2 = cm2.ravel()
```

In [ ]:
```
# Sensitivity
sens2 = tp2 / (tp2 + fn2)
print(f'The probability that a passenger who survived is predicted to have survived
```

```
The probability that a passenger who survived is predicted to have survived is 0.6
027
```

In [ ]:
```
# Specificity
spec2 = tn2 / (tn2 + fp2)
print(f'The probability that a passenger who did not survive is predicted to not ha
```

```
The probability that a passenger who did not survive is predicted to not have surv
ived is 0.8476
```

In [ ]:
```
# Accuracy
acc2 = sum([tp2, tn2]) / sum([tn2, tp2, fn2, fp2])
print(f'The overall accuracy of the model is {acc2:.4}')
```

```
The overall accuracy of the model is 0.7472
```

NOTE: Adding the predictions for each model to the testing data at the end

Problem 3

SVM with Linear Kernel

In [ ]:
```
# SVM classifier with linear kernel
svc3 = SVC(kernel='linear')
```

In [ ]:
```
# set up the Grid Search parameters
param_grid3 = {
    "C": np.linspace(0, 2, 20)
}
```

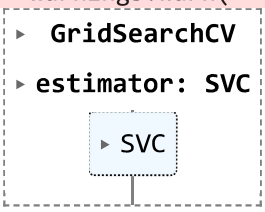
In [ ]:
```
# perform the grid search
grid_search3 = GridSearchCV(svc3, param_grid=param_grid3, cv=5)
grid_search3.fit(X_train, y_train)
```

```
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_validation.py:378: FitFailedWarning:
5 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\mode
l_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 251, in fit
    fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 333, in _dense_fit
    ) = libsvm.fit(
  File "sklearn\svm\_libsvm.pyx", line 192, in sklearn.svm._libsvm.fit
ValueError: C <= 0

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_search.py:953: UserWarning: One or more of the test scores are non-finite: [
nan 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839
 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839
 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839 0.7902839]
  warnings.warn(
```

Out[ ]:



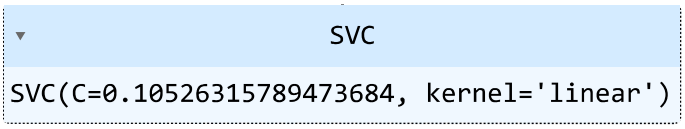
In [ ]:
```
# get the optimal value of C from the GridSearch
optimal_C3 = grid_search3.best_params_["C"]
print(f"Optimal value for C: {optimal_C3:.4}")
```

```
Optimal value for C: 0.1053
```

In [ ]:
```
# make the SVM classifier with Linear Kernel and the optimal C value
svc3_opt = SVC(kernel="linear", C=optimal_C3)
```

In [ ]:
```
# train the SVC on the training data
svc3_opt.fit(X_train, y_train)
```

Out[ ]:



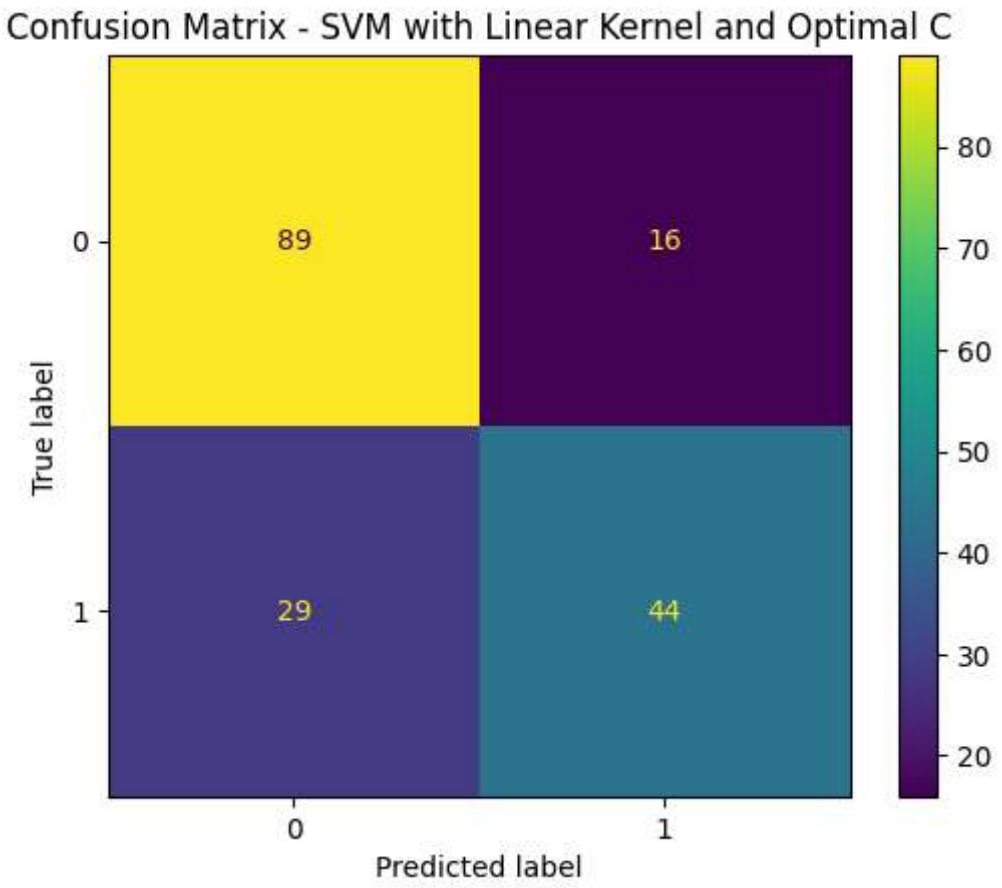
In [ ]:
```
# use the SVC to make predictions on the testing data
y_pred3 = svc3_opt.predict(X_test)
```

Evaluate Performance

In [ ]:
```
# generate the confusion matrix
cm3 = confusion_matrix(y_test, y_pred3, labels = svc3_opt.classes_)
cm3
```

Out[ ]:
```
array([[89, 16],
       [29, 44]], dtype=int64)
```

In [ ]:
```
# better visualization of confusion matrix
disp_cm3 = ConfusionMatrixDisplay(cm3, display_labels=svc3_opt.classes_)
disp_cm3.plot()
plt.title("Confusion Matrix - SVM with Linear Kernel and Optimal C")
plt.show()
```



In [ ]:
```
# extract the elements of the confusion matrix
tn3, fp3, fn3, tp3 = cm3.ravel()
```

In [ ]:
```
# Sensitivity
sens3 = tp3 / (tp3 + fn3)
print(f'The probability that a passenger who survived is predicted to have survived
```

```
The probability that a passenger who survived is predicted to have survived is 0.6
027
```

In [ ]:
```
# Specificity
spec3 = tn3 / (tn3 + fp3)
print(f'The probability that a passenger who did not survive is predicted to not ha
```

```
The probability that a passenger who did not survive is predicted to not have surv
ived is 0.8476
```

In [ ]:
```
# Accuracy
acc3 = sum([tp3, tn3]) / sum([tn3, tp3, fn3, fp3])
print(f'The overall accuracy of the model is {acc3:.4}')
```

```
The overall accuracy of the model is 0.7472
```

Problem 4

SVM with Radial Basis Kernel

In [ ]:
```
# SVM classifier with radial basis kernel
svc4 = SVC(kernel='rbf')
```

In [ ]:
```
# set up the Grid Search parameters
param_grid4 = {
    "C": np.linspace(0, 2, 20),
    'gamma': np.linspace(0, 5, 20)
}
```

In [ ]:
```
# perform the grid search
grid_search4 = GridSearchCV(svc4, param_grid=param_grid4, cv=5)
grid_search4.fit(X_train, y_train)
```

```
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_validation.py:378: FitFailedWarning:
195 fits failed out of a total of 2000.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
100 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\mode
l_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 237, in fit
    raise ValueError(msg)
ValueError: gamma value must be > 0; 0.0 is invalid. Use a positive number or use
'auto' to set gamma to a value of 1 / n_features.

--------------------------------------------------------------------------------
95 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\mode
l_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 251, in fit
    fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 333, in _dense_fit
    ) = libsvm.fit(
  File "sklearn\svm\_libsvm.pyx", line 192, in sklearn.svm._libsvm.fit
ValueError: C <= 0

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_search.py:953: UserWarning: One or more of the test scores are non-finite: [
nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.81837418 0.81463587 0.77346147
 0.69100688 0.63105272 0.61611709 0.61424793 0.61424793 0.61236114
 0.60675366 0.60299771 0.59924176 0.59924176 0.59924176 0.5973726
 0.5973726  0.5973726  0.5973726  0.5973726         nan 0.82398166
 0.81839182 0.8221125  0.81648739 0.77718215 0.74348439 0.69663199
 0.67603597 0.66101217 0.65353553 0.6460589  0.63667783 0.6292012
 0.62170693 0.61796861 0.61609945 0.61609945 0.61609945 0.61423029
        nan 0.82399929 0.82775525 0.8221125  0.82024334 0.81087992
 0.80333275 0.77152178 0.74528302 0.72659143 0.70784694 0.70784694
 0.69474519 0.67973902 0.67785223 0.66476812 0.65729148 0.65168401
 0.65355317 0.65729148        nan 0.82586845 0.82585082 0.82398166
 0.82398166 0.82398166 0.81461823 0.80710633 0.79770764 0.7808323
 0.75464645 0.74902134 0.73593722 0.72093105 0.70973373 0.70786457
 0.70973373 0.70596015 0.70035267 0.69850115        nan 0.8296244
 0.82771998 0.82398166 0.82585082 0.82022571 0.81461823 0.80338565
```

```
0.79217069 0.782772   0.78275436 0.77150414 0.75842003 0.75279492
0.74531829 0.73969318 0.7340857  0.7247399  0.72100159 0.71350732
       nan 0.83336272 0.82585082 0.81837418 0.82209487 0.82022571
0.81274907 0.80151649 0.79215306 0.7902839  0.78845001 0.78846764
0.77719979 0.7734262  0.75658614 0.75468171 0.73972844 0.72851349
0.7191677  0.71542938        nan 0.83336272 0.83147593 0.82213014
0.82207724 0.81835655 0.81461823 0.79962969 0.79964733 0.79776054
0.79030153 0.77721742 0.77908658 0.77349674 0.76035972 0.75473461
0.74537119 0.74161524 0.73410333 0.73410333        nan 0.83336272
0.82960677 0.82026098 0.82022571 0.82022571 0.81086228 0.80336801
0.80151649 0.80153412 0.79403985 0.79030153 0.7828249  0.76413331
0.76039499 0.76039499 0.75478752 0.74918004 0.73978134 0.7341386
       nan 0.83521425 0.82960677 0.82024334 0.82020808 0.81833892
0.81646976 0.81086228 0.80338565 0.80153412 0.79592664 0.79030153
0.78469406 0.78095574 0.77160995 0.7734791  0.76039499 0.75476988
0.75476988 0.74729325        nan 0.83708341 0.82960677 0.82209487
0.81833892 0.81833892 0.8146006  0.81273144 0.80338565 0.80153412
0.80153412 0.79405749 0.78845001 0.7828249  0.78469406 0.77908658
0.77159231 0.76785399 0.7585082  0.75290072        nan 0.83521425
0.82960677 0.82209487 0.81646976 0.8146006  0.81835655 0.81273144
0.80525481 0.80153412 0.80153412 0.79592664 0.78658085 0.78658085
0.78656322 0.78469406 0.77346147 0.7585082  0.75663904 0.75103156
       nan 0.83708341 0.82960677 0.82022571 0.81833892 0.81646976
0.81835655 0.81273144 0.80338565 0.80153412 0.80153412 0.79592664
0.78845001 0.78845001 0.78471169 0.78469406 0.76787163 0.76039499
0.75478752 0.75480515        nan 0.83708341 0.82960677 0.82020808
0.81833892 0.8146006  0.81835655 0.81273144 0.80338565 0.80153412
0.7977958  0.79592664 0.79031917 0.78845001 0.7828249  0.78469406
0.76787163 0.76041263 0.75480515 0.75103156        nan 0.83708341
0.82773761 0.82207724 0.82020808 0.81648739 0.82022571 0.81273144
0.80338565 0.80153412 0.7977958  0.7977958  0.79031917 0.78658085
0.7828249  0.78658085 0.76788926 0.75667431 0.75291836 0.75103156
       nan 0.83519661 0.82398166 0.82207724 0.82020808 0.81648739
0.82022571 0.81086228 0.80338565 0.79966496 0.7977958  0.79592664
0.79031917 0.78658085 0.78471169 0.78471169 0.76041263 0.75667431
0.75291836 0.7510492         nan 0.83334509 0.82398166 0.82019044
0.82207724 0.81835655 0.81833892 0.81086228 0.80338565 0.79966496
0.7977958  0.79592664 0.78845001 0.78471169 0.78284253 0.78097337
0.75852583 0.75291836 0.75291836 0.75291836        nan 0.83521425
0.82209487 0.81645212 0.82207724 0.82022571 0.81833892 0.81086228
0.80338565 0.79966496 0.7977958  0.79592664 0.7903368  0.78471169
0.78097337 0.77160995 0.75665667 0.75478752 0.75291836 0.7510492
       nan 0.83147593 0.82209487 0.82207724 0.82020808 0.82022571
0.81646976 0.81086228 0.80151649 0.79966496 0.7977958  0.79781344
0.7903368  0.78471169 0.77534826 0.76787163 0.75665667 0.75478752
0.75478752 0.74729325        nan 0.82960677 0.82209487 0.82020808
0.82020808 0.82022571 0.81833892 0.81086228 0.79962969 0.79966496
0.7977958  0.79407512 0.78845001 0.78095574 0.7734791  0.76226415
0.75665667 0.75665667 0.75291836 0.74542409]
 warnings.warn(
```

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

In [ ]:
```
# get the optimal values of C and sigma from the GridSearch
optimal_C4, optimal_sigma4 = grid_search4.best_params_["C"], grid_search4.best_para

print(f'Optimal value for C {optimal_C4:.4}')
print(f'Optimal value for sigma: {optimal_sigma4:.4}')
```

```
Optimal value for C 1.053
Optimal value for sigma: 0.2632
```

In [ ]:
```
# make the SVM classifier with Radial Basis Kernel and the optimal values for C and
svc4_opt = SVC(kernel="rbf", C=optimal_C4, gamma=optimal_sigma4)
```

In [ ]:
```
# train the SVC on the training data
svc4_opt.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC(C=1.0526315789473684, gamma=0.2631578947368421)
```

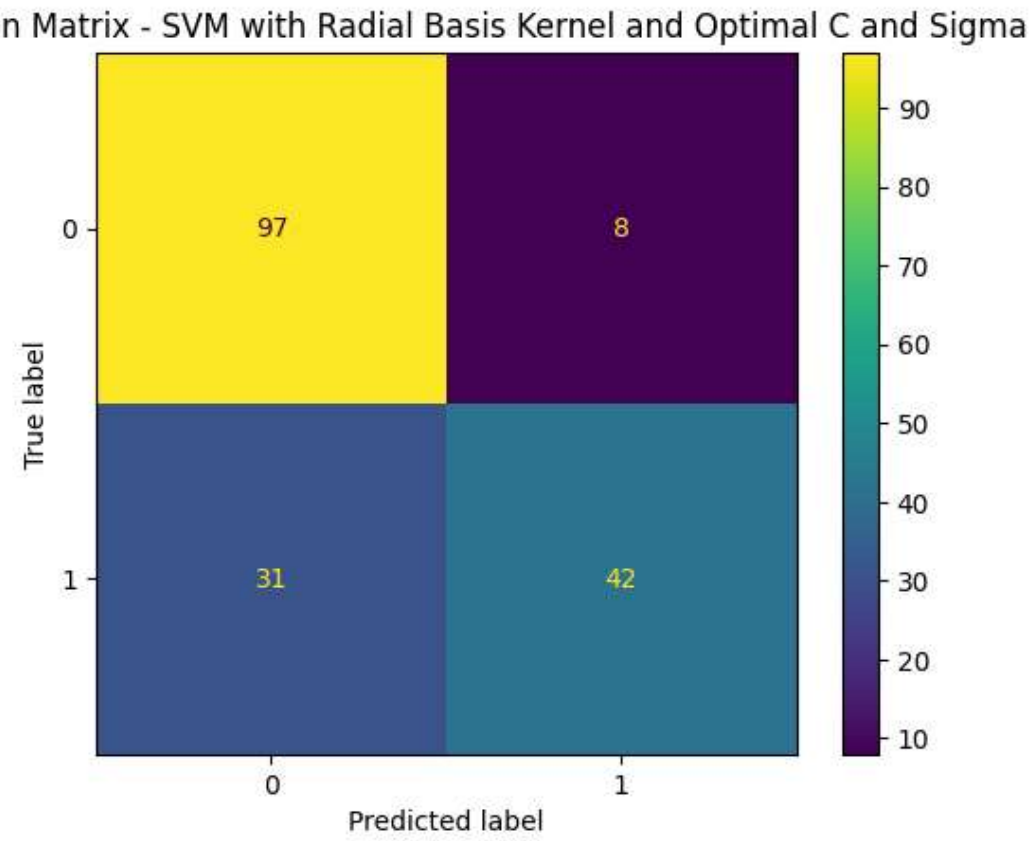
In [ ]:
```
# use the SVC to make predictions on the testing data
y_pred4 = svc4_opt.predict(X_test)
```

Evaluate Performance

In [ ]:
```
# generate the confusion matrix
cm4 = confusion_matrix(y_test, y_pred4, labels = svc4_opt.classes_)
cm4
```

Out[ ]:
```
array([[97,  8],
       [31, 42]], dtype=int64)
```

In [ ]:
```
# better visualization of confusion matrix
disp_cm4 = ConfusionMatrixDisplay(cm4, display_labels=svc4_opt.classes_)
disp_cm4.plot()
plt.title("Confusion Matrix - SVM with Radial Basis Kernel and Optimal C and Sigma"
plt.show()
```



In [ ]:
```
# extract the elements of the confusion matrix
tn4, fp4, fn4, tp4 = cm4.ravel()
```

In [ ]:
```
# Sensitivity
sens4 = tp4 / (tp4 + fn4)
print(f'The probability that a passenger who survived is predicted to have survived
```

```
The probability that a passenger who survived is predicted to have survived is 0.5
753
```

In [ ]:
```
# Specificity
spec4 = tn4 / (tn4 + fp4)
print(f'The probability that a passenger who did not survive is predicted to not ha
```

```
The probability that a passenger who did not survive is predicted to not have surv
ived is 0.9238
```

In [ ]:
```
# Accuracy
acc4 = sum([tp4, tn4]) / sum([tn4, tp4, fn4, fp4])
print(f'The overall accuracy of the model is {acc4:.4}')
```

```
The overall accuracy of the model is 0.7809
```

Problem 5

SVM with Polynomial Kernel

In [ ]:
```
# SVM classifier with polynomial kernel
svc5 = SVC(kernel='poly')
```

In [ ]:
```
# set up the Grid Search parameters
param_grid5 = {
    "C": np.linspace(0, 2, 20),
    'degree': list(range(1,6)),
    'coef0': [-1, 0, 1]
}
```

In [ ]:
```
# perform the grid search
grid_search5 = GridSearchCV(svc5, param_grid=param_grid5, cv=5)
grid_search5.fit(X_train, y_train)
```

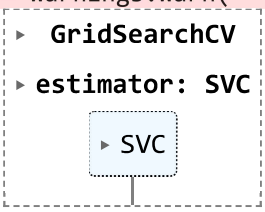
```
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_validation.py:378: FitFailedWarning:
75 fits failed out of a total of 1500.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
75 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\mode
l_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 251, in fit
    fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
  File "C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\svm
\_base.py", line 333, in _dense_fit
    ) = libsvm.fit(
  File "sklearn\svm\_libsvm.pyx", line 192, in sklearn.svm._libsvm.fit
ValueError: C <= 0

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\eshak\AppData\Roaming\Python\Python310\site-packages\sklearn\model_select
ion\_search.py:953: UserWarning: One or more of the test scores are non-finite: [
nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.7902839  0.49987656 0.67219185
 0.53921707 0.64226768 0.7902839  0.80532534 0.76968789 0.70788221
 0.68728619 0.7902839  0.79962969 0.81461823 0.82586845 0.82213014
 0.7902839  0.47183918 0.64415447 0.49428672 0.58615764 0.7902839
 0.7996826  0.79030153 0.73599013 0.71163816 0.7902839  0.80336801
 0.82022571 0.82773761 0.82027861 0.7902839  0.45686828 0.59735496
 0.48679245 0.54307882 0.7902839  0.79776054 0.79592664 0.77719979
 0.72285311 0.7902839  0.80899312 0.81835655 0.83147593 0.81840945
 0.7902839  0.45129607 0.57109857 0.48305414 0.52807265 0.7902839
 0.80338565 0.81842709 0.78656322 0.71913243 0.7902839  0.81274907
 0.82213014 0.82213014 0.8146535  0.7902839  0.42886616 0.56739552
 0.48305414 0.52995944 0.7902839  0.80901076 0.8221654  0.79405749
 0.73225181 0.7902839  0.81461823 0.82399929 0.82214777 0.8146535
 0.7902839  0.41389526 0.55240698 0.4755775  0.52248281 0.7902839
 0.81087992 0.82590372 0.79407512 0.73600776 0.7902839  0.81274907
 0.82399929 0.82027861 0.80717686 0.7902839  0.41015694 0.55801446
 0.46997002 0.5149709  0.7902839  0.81087992 0.82401693 0.7978487
 0.74914477 0.7902839  0.81648739 0.82773761 0.81840945 0.80717686
 0.7902839  0.40453183 0.5561453  0.46434491 0.51874449 0.7902839
 0.81274907 0.82775525 0.80158702 0.7734791  0.7902839  0.81837418
 0.82399929 0.82027861 0.80904602 0.7902839  0.39887145 0.54304355
 0.45499912 0.51313701 0.7902839  0.81461823 0.83149356 0.79971786
 0.78661612 0.7902839  0.82024334 0.82399929 0.82027861 0.80902839
 0.7902839  0.38952566 0.53554929 0.43443837 0.51126785 0.7902839
 0.81650503 0.83147593 0.80160466 0.78848528 0.7902839  0.82024334
 0.82773761 0.81840945 0.80901076 0.7902839  0.38576971 0.53368013
 0.42883089 0.51313701 0.7902839  0.81650503 0.83334509 0.80536061
 0.79037207 0.7902839  0.82024334 0.83147593 0.81654029 0.80712396
```

```
 0.7902839  0.38576971 0.53554929 0.42322342 0.51126785 0.7902839
 0.81650503 0.82960677 0.80349145 0.79971786 0.7902839  0.82024334
 0.83147593 0.81840945 0.80525481 0.7902839  0.38576971 0.53366249
 0.42135426 0.51126785 0.7902839  0.81463587 0.82586845 0.80911656
 0.80532534 0.7902839  0.82024334 0.82773761 0.81467113 0.80525481
 0.7902839  0.38763886 0.53369776 0.41387762 0.51500617 0.7902839
 0.81650503 0.82773761 0.8222183  0.80158702 0.7902839  0.81835655
 0.82773761 0.81467113 0.80338565 0.7902839  0.38576971 0.52995944
 0.41387762 0.51126785 0.7902839  0.81650503 0.82773761 0.82220067
 0.80158702 0.7902839  0.82024334 0.82773761 0.8146535  0.79962969
 0.7902839  0.38390055 0.52809028 0.41012167 0.51313701 0.7902839
 0.81461823 0.82960677 0.81657556 0.80160466 0.7902839  0.82024334
 0.82960677 0.81278434 0.79962969 0.7902839  0.38390055 0.52622113
 0.40825251 0.51126785 0.7902839  0.81648739 0.82960677 0.81657556
 0.81098572 0.7902839  0.82024334 0.82773761 0.81278434 0.80148122
 0.7902839  0.38201375 0.52622113 0.40638335 0.5093987  0.7902839
 0.81648739 0.82773761 0.81844472 0.81285488 0.7902839  0.82024334
 0.82773761 0.8146535  0.80336801 0.7902839  0.38390055 0.52246517
 0.40638335 0.51126785 0.7902839  0.81648739 0.82960677 0.8221654
 0.81472403 0.7902839  0.82213014 0.82773761 0.81278434 0.80149885]
  warnings.warn(
```

Out[ ]:



In [ ]:
```
# get the optimal values of C, degree, and scale from the GridSearch
optimal_C5, optimal_degree5, optimal_scale5 = grid_search5.best_params_["C"], grid_

print(f'Optimal value for C {optimal_C5:.4}')
print(f'Optimal value for degree: {optimal_degree5}')
print(f'Optimal value for scale: {optimal_scale5}')
```

```
Optimal value for C 1.158
Optimal value for degree: 3
Optimal value for scale: 0
```

In [ ]:
```
# make the SVM classifier with Polynomial Kernel and the optimal values for C, degr
svc5_opt = SVC(kernel="poly", C=optimal_C5, degree=optimal_degree5, coef0=optimal_s
```

In [ ]:
```
# train the SVC on the training data
svc5_opt.fit(X_train, y_train)
```

Out[ ]:

```
SVC
SVC(C=1.1578947368421053, coef0=0, kernel='poly')
```

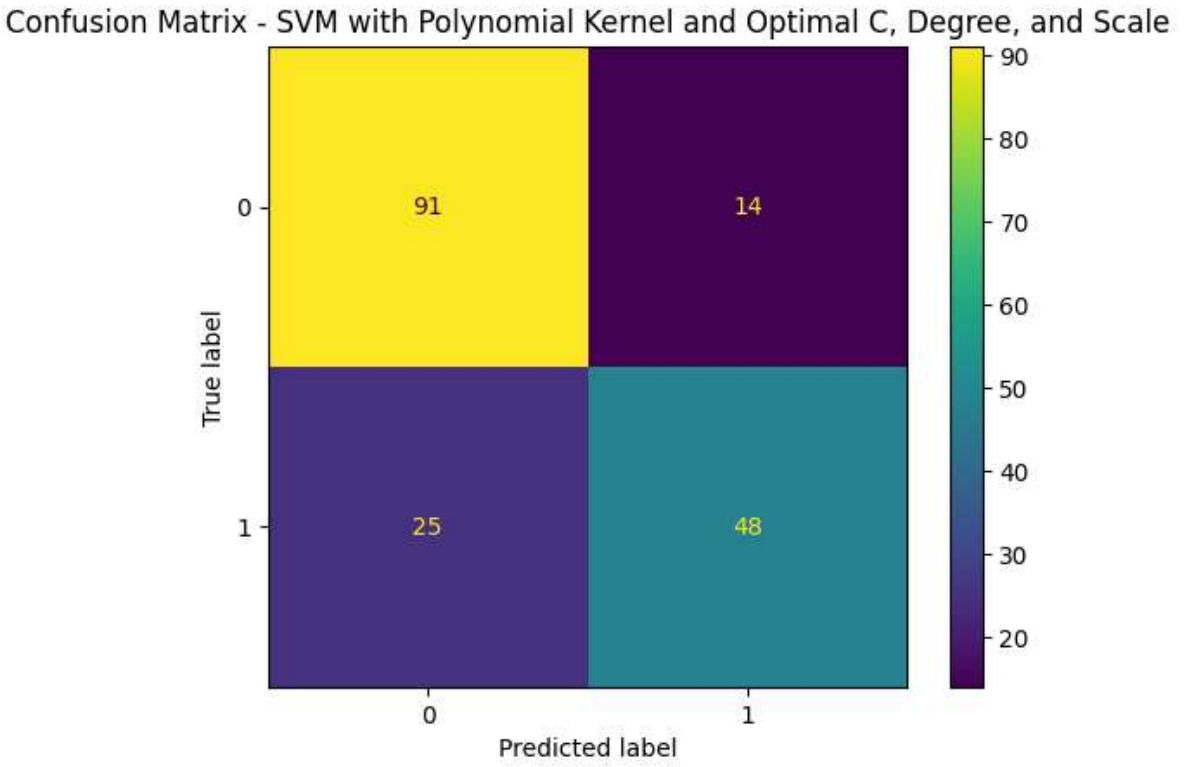
In [ ]:
```
# use the SVC to make predictions on the testing data
y_pred5 = svc5_opt.predict(X_test)
```

Evaluate Performance

In [ ]:
```
# generate the confusion matrix
cm5 = confusion_matrix(y_test, y_pred5, labels = svc5_opt.classes_)
cm5
```

Out[ ]:
```
array([[91, 14],
       [25, 48]], dtype=int64)
```

In [ ]:
```
# better visualization of confusion matrix
disp_cm5 = ConfusionMatrixDisplay(cm5, display_labels=svc5_opt.classes_)
disp_cm5.plot()
plt.title("Confusion Matrix - SVM with Polynomial Kernel and Optimal C, Degree, and
plt.show()
```



In [ ]:
```
# extract the elements of the confusion matrix
tn5, fp5, fn5, tp5 = cm5.ravel()
```

In [ ]:
```
# Sensitivity
sens5 = tp5 / (tp5 + fn5)
print(f'The probability that a passenger who survived is predicted to have survived
```

```
The probability that a passenger who survived is predicted to have survived is 0.6
575
```

In [ ]:
```
# Specificity
spec5 = tn5 / (tn5 + fp5)
print(f'The probability that a passenger who did not survive is predicted to not ha
```

```
The probability that a passenger who did not survive is predicted to not have surv
ived is 0.8667
```

In [ ]:
```
# Accuracy
acc5 = sum([tp5, tn5]) / sum([tn5, tp5, fn5, fp5])
```

```
print(f'The overall accuracy of the model is {acc5:.4}')
```

```
The overall accuracy of the model is 0.7809
```

Adding the predictions from Problem 2 to the testing data

In [ ]:
```
y_test_df = pd.DataFrame(y_test)
```

In [ ]:
```
y_test_df['Problem 2 Predictions'] = y_pred2
```

In [ ]:
```
y_test_df
```

Out[ ]:

	Survived	Problem 2 Predictions
509	1	0
640	0	0
537	1	1
526	1	1
262	0	0
...	...	...
741	0	0
562	0	0
586	0	0
88	1	1
187	1	0

178 rows × 2 columns

Problem 6

Find the model with the best accuracy

In [ ]:
```
# acc3: Linear with optimal C
# acc4: Radial with optimal C and sigma
# acc5: Polynomial with optimal C, degree, and scale
accuracies = [acc3, acc4, acc5]

models = ['Linear SVM', 'SVM with Radial Basis Kernel', 'SVM with Polynomial Kernel

# get the highest accuracy of the three
max_accuracy = max(accuracies)

# get the highest accuracy's index
max_index = accuracies.index(max_accuracy)

# get the highest accuracy's corresponding model
best_model = models[max_index]
```

```
print(f'Best model was {best_model} with an accuracy of {max_accuracy:.4}')
```

```
Best model was SVM with Radial Basis Kernel with an accuracy of 0.7809
```