# DSCI5340_HW4_Group2

November 15, 2023

[ ]:
```
pip install dmba
```

```
Requirement already satisfied: dmba in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (0.2.4)
Requirement already satisfied: graphviz in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (0.20.1)
Requirement already satisfied: matplotlib in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (3.7.3)
Requirement already satisfied: numpy in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (1.24.4)
Requirement already satisfied: pandas in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (1.4.2)
Requirement already satisfied: scikit-learn in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (1.3.1)
Requirement already satisfied: scipy in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
dmba) (1.11.3)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (1.1.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (21.3)
Requirement already satisfied: pillow>=6.2.0 in
```

```
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (10.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib->dmba) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pandas->dmba) (2022.1)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
scikit-learn->dmba) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
scikit-learn->dmba) (3.2.0)
Requirement already satisfied: six>=1.5 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
python-dateutil>=2.7->matplotlib->dmba) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Ignoring invalid distribution -atplotlib
(c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib
(c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages)
```

[ ]:
```
pip install ISLP
```

```
Requirement already satisfied: ISLP in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages
(0.3.21)
Requirement already satisfied: numpy<1.25,>=1.7.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.24.4)
Requirement already satisfied: scipy>=0.9 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.11.3)
Requirement already satisfied: pandas<=1.9,>=0.20 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.4.2)
Requirement already satisfied: lxml in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (4.9.3)
Requirement already satisfied: scikit-learn>=1.2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.3.1)
Requirement already satisfied: joblib in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.3.2)
Requirement already satisfied: statsmodels>=0.13 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (0.14.0)
Requirement already satisfied: lifelines in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (0.27.8)
Requirement already satisfied: pygam in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (0.9.0)
Requirement already satisfied: torch in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (2.1.0)
Requirement already satisfied: pytorch-lightning in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (2.1.0)
Requirement already satisfied: torchmetrics in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
ISLP) (1.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pandas<=1.9,>=0.20->ISLP) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pandas<=1.9,>=0.20->ISLP) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
scikit-learn>=1.2->ISLP) (3.2.0)
Requirement already satisfied: patsy>=0.5.2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
statsmodels>=0.13->ISLP) (0.5.3)
Requirement already satisfied: packaging>=21.3 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
statsmodels>=0.13->ISLP) (21.3)
Requirement already satisfied: matplotlib>=3.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
lifelines->ISLP) (3.7.3)
Requirement already satisfied: autograd>=1.5 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
lifelines->ISLP) (1.6.2)
Requirement already satisfied: autograd-gamma>=0.3 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
lifelines->ISLP) (0.5.0)
Requirement already satisfied: formulaic>=0.2.2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
lifelines->ISLP) (0.6.6)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pygam->ISLP) (4.2.0)
Requirement already satisfied: tqdm>=4.57.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pytorch-lightning->ISLP) (4.66.1)
Requirement already satisfied: PyYAML>=5.4 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pytorch-lightning->ISLP) (6.0.1)
Requirement already satisfied: fsspec>2021.06.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (2023.10.0)
Requirement already satisfied: typing-extensions>=4.0.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pytorch-lightning->ISLP) (4.8.0)
Requirement already satisfied: lightning-utilities>=0.8.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
pytorch-lightning->ISLP) (0.9.0)
Requirement already satisfied: filelock in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
torch->ISLP) (3.13.1)
Requirement already satisfied: sympy in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
torch->ISLP) (1.12)
Requirement already satisfied: networkx in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
torch->ISLP) (3.2)
Requirement already satisfied: jinja2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
torch->ISLP) (3.1.2)
Requirement already satisfied: future>=0.15.2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
autograd>=1.5->lifelines->ISLP) (0.18.3)
Requirement already satisfied: astor>=0.8 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
formulaic>=0.2.2->lifelines->ISLP) (0.8.1)
Requirement already satisfied: interface-meta>=1.2.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
formulaic>=0.2.2->lifelines->ISLP) (1.3.0)
Requirement already satisfied: wrapt>=1.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
formulaic>=0.2.2->lifelines->ISLP) (1.14.1)
Requirement already satisfied: requests in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (2.27.1)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (3.8.6)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (1.1.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (10.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
matplotlib>=3.0->lifelines->ISLP) (3.0.9)
Requirement already satisfied: six in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
patsy>=0.5.2->statsmodels>=0.13->ISLP) (1.16.0)
Requirement already satisfied: python-utils>=3.0.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
progressbar2<5.0.0,>=4.2.0->pygam->ISLP) (3.8.1)
Requirement already satisfied: colorama in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
tqdm>=4.57.0->pytorch-lightning->ISLP) (0.4.4)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
jinja2->torch->ISLP) (2.1.1)
Requirement already satisfied: mpmath>=0.19 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
sympy->torch->ISLP) (1.3.0)
Requirement already satisfied: attrs>=17.3.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(21.4.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(2.0.12)
Requirement already satisfied: multidict<7.0,>=4.5 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in
```

```
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning->ISLP)
(1.3.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (1.26.9)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (2022.5.18.1)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning->ISLP) (3.3)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Ignoring invalid distribution -atplotlib
(c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib
(c:\users\sudhe\appdata\local\programs\python\python310\lib\site-packages)
```

Importing Required Packages

[ ]:
```python
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import dmba
from pathlib import Path
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
import sklearn.model_selection as skm
from matplotlib.pyplot import subplots, cm
from ISLP.svm import plot as plot_svm
```

Loading Dataset

[ ]:
```python
Auto = pd.read_csv(r'Auto.csv')
Auto.head()
```

```
[ ]:     mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
     0  18.0          8         307.0         130    3504          12.0    70
     1  15.0          8         350.0         165    3693          11.5    70
     2  18.0          8         318.0         150    3436          11.0    70
     3  16.0          8         304.0         150    3433          12.0    70
     4  17.0          8         302.0         140    3449          10.5    70

        origin                       name
     0       1  chevrolet chevelle malibu
     1       1          buick skylark 320
     2       1         plymouth satellite
     3       1              amc rebel sst
     4       1                ford torino
```

Create binary variable for gas mileage

```
[ ]: median_mpg = Auto['mpg'].median()
     print(median_mpg)
     Auto['high_mileage'] = (Auto['mpg'] > median_mpg).astype(int)
     Auto['high_mileage']
```

```
22.75
```

```
[ ]: 0      0
     1      0
     2      0
     3      0
     4      0
           ..
     387    1
     388    1
     389    1
     390    1
     391    1
     Name: high_mileage, Length: 392, dtype: int32
```

```
[ ]: Auto.head()
```

```
[ ]:     mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
     0  18.0          8         307.0         130    3504          12.0    70
     1  15.0          8         350.0         165    3693          11.5    70
     2  18.0          8         318.0         150    3436          11.0    70
     3  16.0          8         304.0         150    3433          12.0    70
     4  17.0          8         302.0         140    3449          10.5    70

        origin                       name  high_mileage
     0       1  chevrolet chevelle malibu             0
     1       1          buick skylark 320             0
```

```
2       1         plymouth satellite             0
3       1              amc rebel sst             0
4       1                ford torino             0
```

[ ]:
```
Auto.isnull().sum()
```

[ ]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
high_mileage    0
dtype: int64
```

[ ]:
```
Auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   year          392 non-null    int64
 7   origin        392 non-null    int64
 8   name          392 non-null    object
 9   high_mileage  392 non-null    int32
dtypes: float64(3), int32(1), int64(5), object(1)
memory usage: 29.2+ KB
```

[ ]:
```
Auto = pd.get_dummies(Auto, columns=['origin'] ,drop_first= True)
```

[ ]:
```
Auto.head()
```

[ ]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70
```

```
                        name  high_mileage  origin_2  origin_3
0  chevrolet chevelle malibu             0         0         0
1          buick skylark 320             0         0         0
2         plymouth satellite             0         0         0
3              amc rebel sst             0         0         0
4                ford torino             0         0         0
```

Standardizing

[ ]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(Auto[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',␣
 ↪'acceleration']])
```

[ ]:
```
StandardScaler()
```

Transform the full dataset

[ ]:
```
AutoNorm = pd.concat([pd.DataFrame(scaler.transform(Auto[['mpg', 'cylinders',␣
 ↪'displacement', 'horsepower', 'weight', 'acceleration']]),
                                    columns=['zmpg', 'zcylinders',␣
 ↪'zdisplacement', 'zhorsepower', 'zweight', 'zacceleration']),
                                    Auto[['year','name', 'high_mileage',␣
 ↪'origin_2', 'origin_3']]], axis=1)
```

[ ]:
```
AutoNorm.head()
```

[ ]:
```
       zmpg  zcylinders  zdisplacement  zhorsepower   zweight  zacceleration  \
0 -0.698638    1.483947       1.077290     0.664133  0.620540      -1.285258
1 -1.083498    1.483947       1.488732     1.574594  0.843334      -1.466724
2 -0.698638    1.483947       1.182542     1.184397  0.540382      -1.648189
3 -0.955212    1.483947       1.048584     1.184397  0.536845      -1.285258
4 -0.826925    1.483947       1.029447     0.924265  0.555706      -1.829655

   year                       name  high_mileage  origin_2  origin_3
0    70  chevrolet chevelle malibu             0         0         0
1    70          buick skylark 320             0         0         0
2    70         plymouth satellite             0         0         0
3    70              amc rebel sst             0         0         0
4    70                ford torino             0         0         0
```

Splitting the data into Target and Predictive Variables

[ ]:
```
X = AutoNorm[['zcylinders', 'zdisplacement', 'zhorsepower', 'zweight',␣
 ↪'zacceleration',
       'year', 'origin_2', 'origin_3']]
y = AutoNorm['high_mileage']
```

Fit a Support Vector Classfier on Linear Kernel

[ ]:
```
from sklearn.model_selection import cross_val_score
c_values = [0.01, 0.1, 1, 10, 100]
kfold = skm.KFold(5,
random_state=123,
shuffle=True)
cv_scores = []
for c in c_values:
    svm = SVC(kernel='linear', C=c)
    svm.fit(X,y)
    cv_scores.append(np.mean(cross_val_score(svm, X, y, cv=kfold,␣
 ↪scoring='accuracy')))
    print("For Cost Parameter:",c, ", Accuracy =", svm.score(X,y))
```

```
For Cost Parameter: 0.01 , Accuracy = 0.9158163265306123
For Cost Parameter: 0.1 , Accuracy = 0.9158163265306123
For Cost Parameter: 1 , Accuracy = 0.9209183673469388
For Cost Parameter: 10 , Accuracy = 0.923469387755102
For Cost Parameter: 100 , Accuracy = 0.9260204081632653
```

Based on the results, For C=100, we got the highest accuracy of 0.9260204081632653. As the cost parameter is increasing we observed the accuracy of the model is increasing which shows that the model is well fitting on the data. A higher value of C means the model is less regularized and more complex, which can lead to overfitting

Justification for Best Cost Parameter on Linear Kernel

[ ]:
```
kfold = skm.KFold(5,
                  random_state = 123,
                  shuffle=True)
grid = skm.GridSearchCV(svm,
                        {'C':[0.01, 0.1, 1, 10, 100]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')
grid.fit(X,y)

# what's the best model?
grid.best_params_
```

[ ]:
```
{'C': 10}
```

[ ]:
```
cv_errors = []
for c in c_values:
    svm.C = c
    scores = cross_val_score(svm, X, y, scoring='accuracy', cv=5)
    cv_errors.append(1 - np.mean(scores))
for i in range(len(c_values)):
    print("CV Error for C={}: {}".format(c_values[i],cv_errors[i]))
```

```
CV Error for C=0.01: 0.170561506004544
CV Error for C=0.1: 0.15280753002271974
CV Error for C=1: 0.150210970464135
CV Error for C=10: 0.14511522233041219
CV Error for C=100: 0.14511522233041219
```

Though for C=100 we got the higher accuracy value. But the during the evaluation of cross validation errors low value of error is obtained for C=10 and C=100. while C=100 offers slightly higher accuracy on data, C=10 was identified as the best parameter by GridSearchCV. It implies, C=10 might be offering a better balance between model complexity and generalization ability. The model with C=10 is less complex and more likely to generalize well, it would be preferable to choose C=10.

SVM WITH RADIAL KERNAL

```
[ ]: gamma_values = [0.1, 0.5, 1, 2, 3, 4]
     # Radial Basis Kernel (RBF)
     kernel_rbf = 'rbf'
     for i in c_values:
         for g in gamma_values:
             rbf = SVC(kernel=kernel_rbf, C=i, gamma=g, random_state=123)
             scores_rbf = cross_val_score(rbf, X, y, cv=5, scoring='accuracy')
             print(f"Kernel={kernel_rbf}, c_values={i}, Gamma={g}: Mean Accuracy =␣
      ↪{scores_rbf.mean():.3f}")
```

```
Kernel=rbf, c_values=0.01, Gamma=0.1: Mean Accuracy = 0.636
Kernel=rbf, c_values=0.01, Gamma=0.5: Mean Accuracy = 0.610
Kernel=rbf, c_values=0.01, Gamma=1: Mean Accuracy = 0.559
Kernel=rbf, c_values=0.01, Gamma=2: Mean Accuracy = 0.569
Kernel=rbf, c_values=0.01, Gamma=3: Mean Accuracy = 0.595
Kernel=rbf, c_values=0.01, Gamma=4: Mean Accuracy = 0.508
Kernel=rbf, c_values=0.1, Gamma=0.1: Mean Accuracy = 0.771
Kernel=rbf, c_values=0.1, Gamma=0.5: Mean Accuracy = 0.733
Kernel=rbf, c_values=0.1, Gamma=1: Mean Accuracy = 0.584
Kernel=rbf, c_values=0.1, Gamma=2: Mean Accuracy = 0.569
Kernel=rbf, c_values=0.1, Gamma=3: Mean Accuracy = 0.595
Kernel=rbf, c_values=0.1, Gamma=4: Mean Accuracy = 0.508
Kernel=rbf, c_values=1, Gamma=0.1: Mean Accuracy = 0.771
Kernel=rbf, c_values=1, Gamma=0.5: Mean Accuracy = 0.755
Kernel=rbf, c_values=1, Gamma=1: Mean Accuracy = 0.720
Kernel=rbf, c_values=1, Gamma=2: Mean Accuracy = 0.643
Kernel=rbf, c_values=1, Gamma=3: Mean Accuracy = 0.546
Kernel=rbf, c_values=1, Gamma=4: Mean Accuracy = 0.515
Kernel=rbf, c_values=10, Gamma=0.1: Mean Accuracy = 0.771
Kernel=rbf, c_values=10, Gamma=0.5: Mean Accuracy = 0.714
Kernel=rbf, c_values=10, Gamma=1: Mean Accuracy = 0.707
Kernel=rbf, c_values=10, Gamma=2: Mean Accuracy = 0.633
```

```
Kernel=rbf, c_values=10, Gamma=3: Mean Accuracy = 0.551
Kernel=rbf, c_values=10, Gamma=4: Mean Accuracy = 0.515
Kernel=rbf, c_values=100, Gamma=0.1: Mean Accuracy = 0.745
Kernel=rbf, c_values=100, Gamma=0.5: Mean Accuracy = 0.694
Kernel=rbf, c_values=100, Gamma=1: Mean Accuracy = 0.709
Kernel=rbf, c_values=100, Gamma=2: Mean Accuracy = 0.630
Kernel=rbf, c_values=100, Gamma=3: Mean Accuracy = 0.548
Kernel=rbf, c_values=100, Gamma=4: Mean Accuracy = 0.515
```

The following combination have same Mean Accuracy values:

Kernel=rbf, c_values=0.1, Gamma=0.1: Mean Accuracy = 0.771

Kernel=rbf, c_values=10, Gamma=0.1: Mean Accuracy = 0.771

Justification

[ ]:
```python
kfold = skm.KFold(5,
random_state=123,
shuffle=True)
grid = skm.GridSearchCV(rbf,
                        {'C':[0.01, 0.1, 1, 10, 100],
                        'gamma':[0.1, 0.5, 1, 2, 3, 4]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')


grid.fit(X, y)
grid.best_params_
```

[ ]:
```
{'C': 10, 'gamma': 0.1}
```

By observing the results obtained from Radial Basis kernal and Grid search: The best combination from the output is "Kernel=rbf, c_values=10, Gamma=0.1: Mean Accuracy = 0.771" beacuse it has the highest mean accuracy(0.771), the c value(10) will allows less complexity and simpler model, with lower gamma value(0.1) will avoid the overfitting and have smoother decision boundaries.

SVM WITH POLYNOMIAL KERNAL

[ ]:
```python
# Polynomial Kernel
degree_values = [2, 3, 4, 5]
kernel_poly = 'poly'
for i in c_values:
    for d in degree_values:
        poly = SVC(kernel=kernel_poly, C=i, degree=d, random_state=123)
        scores_poly = cross_val_score(poly, X, y, cv=5, scoring='accuracy')
        print(f"Kernel={kernel_poly}, c_values={i}, Degree={d}: Mean Accuracy =␣
  ↪{scores_poly.mean():.3f}")
```

```
Kernel=poly, c_values=0.01, Degree=2: Mean Accuracy = 0.664
Kernel=poly, c_values=0.01, Degree=3: Mean Accuracy = 0.674
```

```
Kernel=poly, c_values=0.01, Degree=4: Mean Accuracy = 0.680
Kernel=poly, c_values=0.01, Degree=5: Mean Accuracy = 0.690
Kernel=poly, c_values=0.1, Degree=2: Mean Accuracy = 0.664
Kernel=poly, c_values=0.1, Degree=3: Mean Accuracy = 0.677
Kernel=poly, c_values=0.1, Degree=4: Mean Accuracy = 0.700
Kernel=poly, c_values=0.1, Degree=5: Mean Accuracy = 0.713
Kernel=poly, c_values=1, Degree=2: Mean Accuracy = 0.720
Kernel=poly, c_values=1, Degree=3: Mean Accuracy = 0.717
Kernel=poly, c_values=1, Degree=4: Mean Accuracy = 0.738
Kernel=poly, c_values=1, Degree=5: Mean Accuracy = 0.761
Kernel=poly, c_values=10, Degree=2: Mean Accuracy = 0.796
Kernel=poly, c_values=10, Degree=3: Mean Accuracy = 0.814
Kernel=poly, c_values=10, Degree=4: Mean Accuracy = 0.822
Kernel=poly, c_values=10, Degree=5: Mean Accuracy = 0.829
Kernel=poly, c_values=100, Degree=2: Mean Accuracy = 0.842
Kernel=poly, c_values=100, Degree=3: Mean Accuracy = 0.837
Kernel=poly, c_values=100, Degree=4: Mean Accuracy = 0.842
Kernel=poly, c_values=100, Degree=5: Mean Accuracy = 0.850
```

Kernel=poly, c_values=100, Degree=5: Mean Accuracy = 0.850

Justification

[ ]:
```
kfold = skm.KFold(5,
random_state=123,
shuffle=True)
grid = skm.GridSearchCV(poly,
                        {'C':[0.01, 0.1, 1, 10, 100],
                        'degree':[2, 3, 4, 5]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')


grid.fit(X, y)
grid.best_params_
```

[ ]:
```
{'C': 100, 'degree': 5}
```

By observing the results of Polynomial Kernal and grid search: The best combination from the output is Kernel=poly, c_values=100, Degree=5: Mean Accuracy = 0.850 beacuse it has the highest mean accuracy(0.850) which shows good performance, higher c value(c=100) allows more complex model and higher degree(degree=5) allows the polynomial kernel to capture more complex relationships in the data.

FITTING LINEAR KERNEL WITH BEST VALUE OF C=10

[ ]:
```
svm_linear = SVC(kernel="linear", C= 10)
svm_linear.fit(X , y)
```

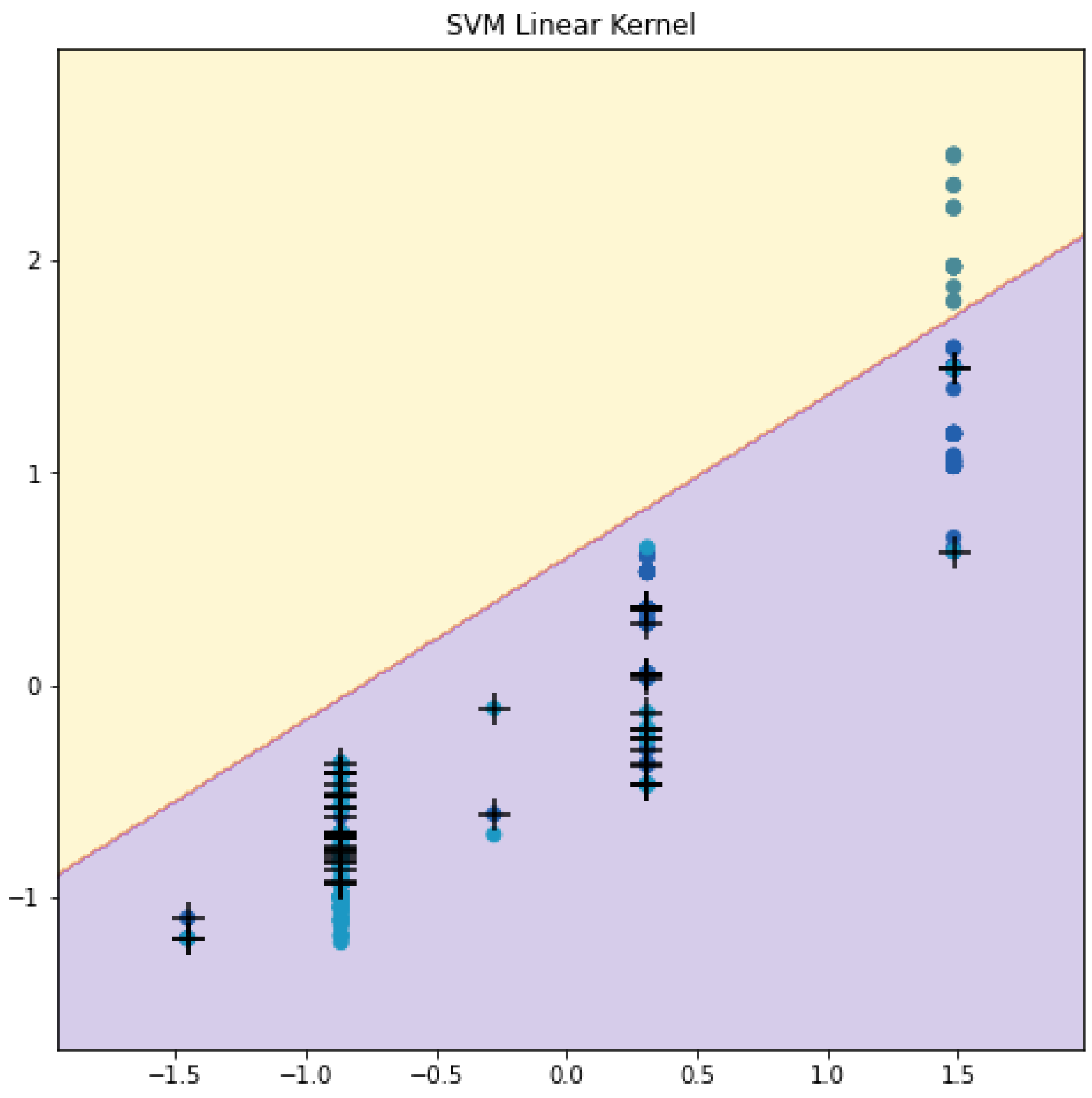
[ ]:
```
SVC(C=10, kernel='linear')
```

PLOT: Linear

```
[ ]: fig, ax = subplots(figsize=(8,8))
     plot_svm(X,
              y,
              svm_linear,
              ax=ax)
     ax.set_title('SVM Linear Kernel')
```

```
c:\Users\sudhe\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:465: UserWarning: X does not have valid feature names,
but SVC was fitted with feature names
  warnings.warn(
```

```
[ ]: Text(0.5, 1.0, 'SVM Linear Kernel')
```



Based on the above plot, the data is well-separated and there is a clear hyperplane that can be used to separate the two classes. The SVM model has learned this hyperplane and is able to accurately predict the class labels of new data points.

Overall, the plot suggests that the SVM model is a good fit for this data set. The model is able to accurately separate the two classes and is likely to generalize well to new data.

FITTING RADIAL KERNEL WITH BEST VALUE OF C=10 AND GAMMA = 0.1

```
[ ]: svm_rbf = SVC(kernel="rbf", gamma=0.1, C= 10)
     svm_rbf.fit(X , y)
```

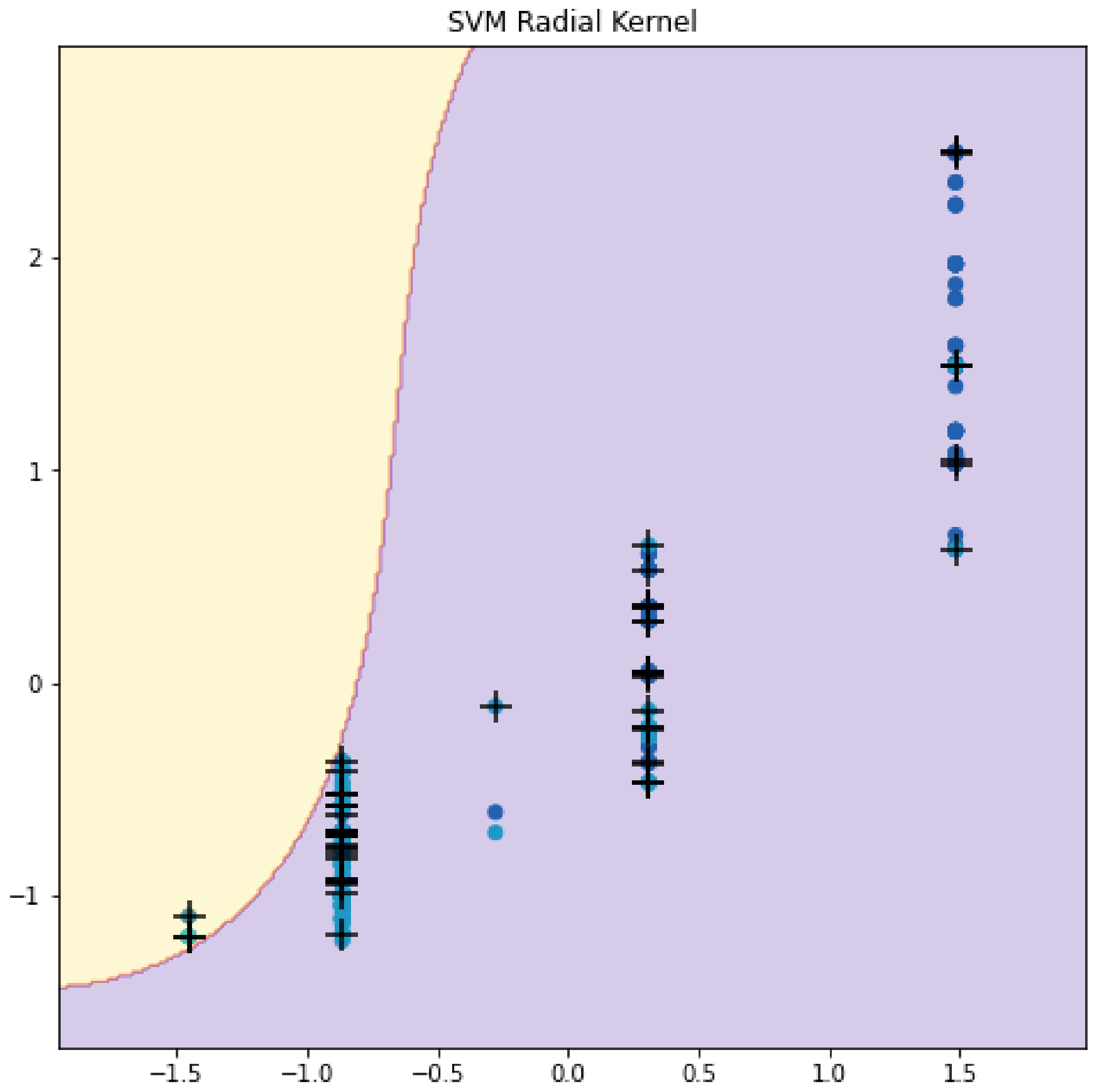
```
[ ]: SVC(C=10, gamma=0.1)
```

Plot: Radial

```
[ ]: fig, ax = subplots(figsize=(8,8))
     plot_svm(X,
              y,
              svm_rbf,
              ax=ax)
     ax.set_title('SVM Radial Kernel')
```

```
c:\Users\sudhe\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:465: UserWarning: X does not have valid feature names,
but SVC was fitted with feature names
  warnings.warn(
```

```
[ ]: Text(0.5, 1.0, 'SVM Radial Kernel')
```



Based on the above plot, the RBF SVM model has learned a more complex hyperplane than the linear SVM model. This is because the RBF kernel allows the model to learn non-linear relationships between the features.

The RBF SVM model has found a hyperplane that separates the two classes well, but not perfectly. There are a few data points on the wrong side of the hyperplane. The hyperplane is more complex than the hyperplane learned by the linear SVM model. It curves around the data points, suggesting that the model has learned non-linear relationships between the features.

Overall, the RBF SVM model is a good choice for classification tasks where the data is non-linear.

FITTING POLYNOMIAL KERNEL WITH BEST VALUE OF C=100 AND DEGREE = 5

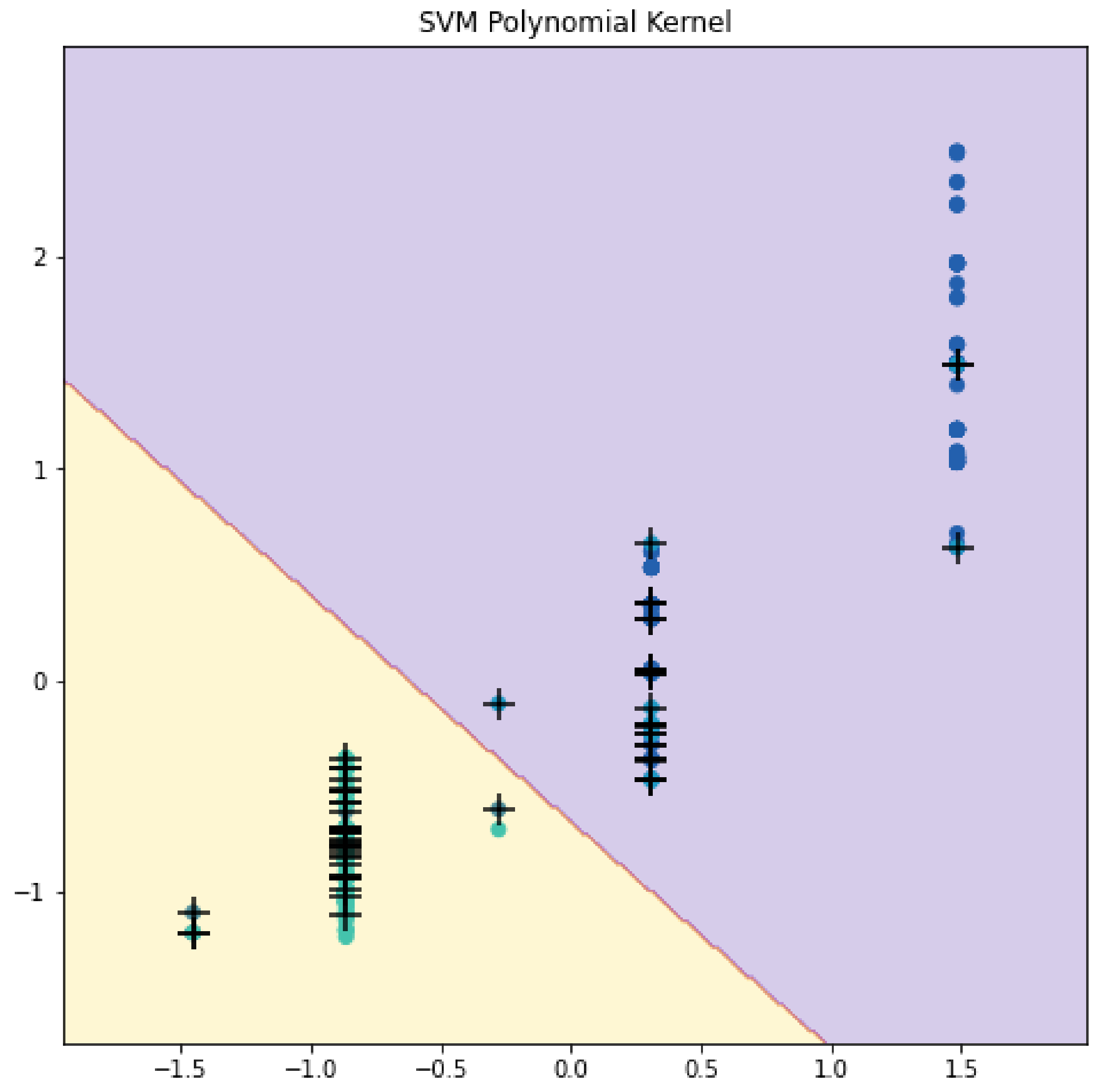
[ ]:
```
svm_poly = SVC(kernel="poly", degree=5, C= 100)
svm_poly.fit(X , y)
```

```
[ ]: SVC(C=100, degree=5, kernel='poly')
```

```
[ ]: fig, ax = subplots(figsize=(8,8))
     plot_svm(X,
              y,
              svm_poly,
              ax=ax)
     ax.set_title('SVM Polynomial Kernel')
```

```
c:\Users\sudhe\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:465: UserWarning: X does not have valid feature names,
but SVC was fitted with feature names
  warnings.warn(
```

```
[ ]: Text(0.5, 1.0, 'SVM Polynomial Kernel')
```



Based on the above plot, the polynomial SVM model has learned a very complex hyperplane that perfectly separates the two classes. This is because the polynomial kernel allows the model to learn non-linear relationships between the features of any degree.