index

March 25, 2022

1 Kernels in scikit-learn - Lab

1.1 Introduction

In this lab, you'll explore applying several types of kernels on some more visual data. At the end of the lab, you'll then apply your knowledge of SVMs to a real-world dataset!

1.2 Objectives

In this lab you will:

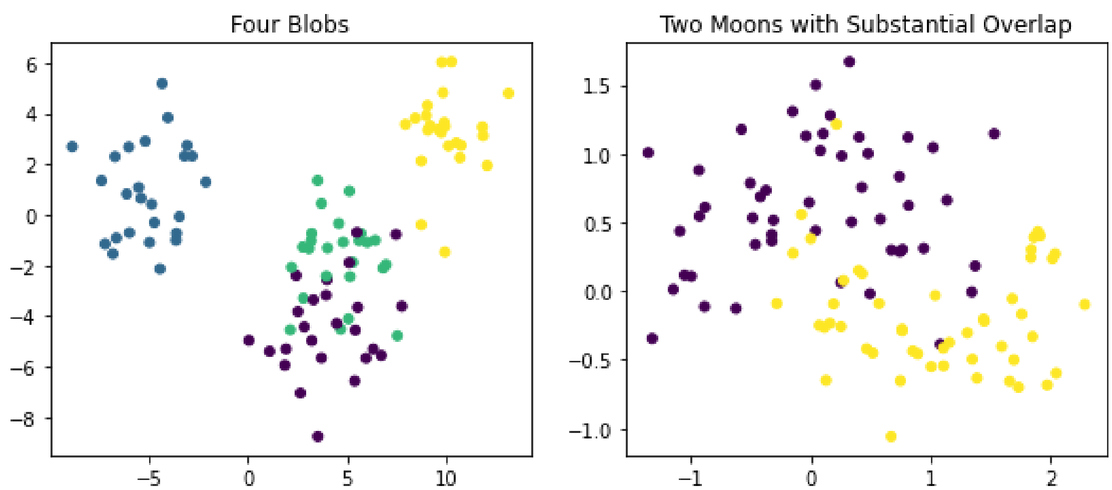
- Create and evaluate a non-linear SVM model in scikit-learn using real-world data
- Interpret the prediction results of an SVM model by creating visualizations

1.3 The data

To start, reexamine the final datasets from the previous lab:

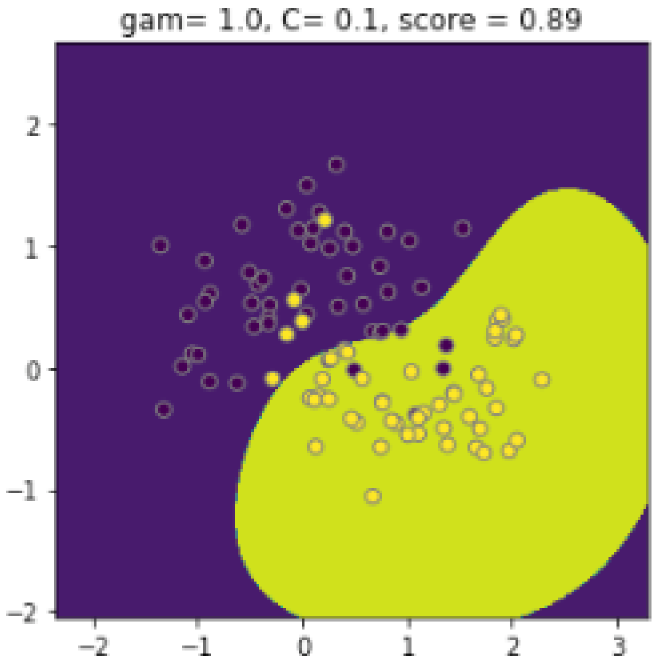
[26]:
```
from sklearn.datasets import make_blobs
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn import svm
from sklearn.model_selection import train_test_split

import numpy as np

plt.figure(figsize=(10, 4))
plt.subplot(121)

plt.title('Four Blobs')

X_3, y_3 = make_blobs(n_samples=100,
                      n_features=2,
                      centers=4,
                      cluster_std=1.6,
                      random_state=123)

plt.scatter(X_3[:, 0], X_3[:, 1], c=y_3, s=25)
```

```
plt.subplot(122)
plt.title('Two Moons with Substantial Overlap')
X_4, y_4 = make_moons(n_samples=100,
                      shuffle=False,
                      noise=0.3,
                      random_state=123)

plt.scatter(X_4[:, 0], X_4[:, 1], c=y_4, s=25)

plt.show()
```



1.4 Explore the RBF kernel

Recall how a radial basis function kernel has 2 hyperparameters: `C` and `gamma`. To further investigate tuning, you'll generate 9 subplots with varying parameter values and plot the resulting decision boundaries. Take a look at this example from scikit-learn for inspiration. Each of the 9 plots should look like this:



Note that the score represents the percentage of correctly classified instances according to the model.

[27]:
```
C_range = np.array([0.1, 1, 10])
gamma_range = np.array([0.1, 1, 100])
param_grid = dict(gamma=gamma_range, C=C_range)
details = []

# Create a loop that builds a model for each of the 9 combinations

for i in C_range:
    for j in gamma_range:
        svc = svm.SVC(C = i, gamma = j)
        svc.fit(X_4, y_4)
        details.append((i, j, svc))
```

[28]:
```
details[3][2]
```

[28]:
```
SVC(gamma=0.1)
```

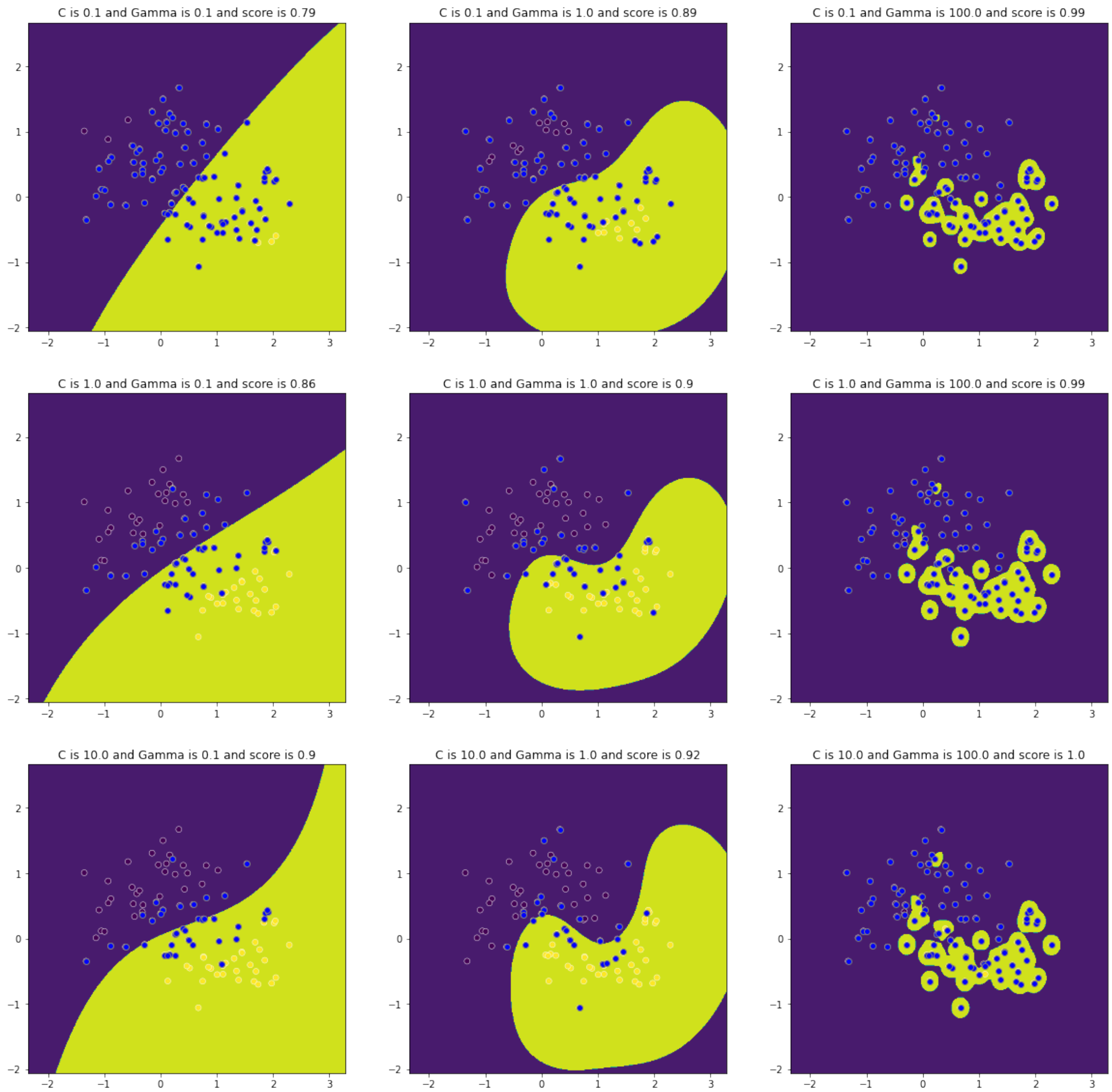
[43]:
```
i = 3

i%3
```

[43]:
```
0
```

[56]:

```
# Prepare your data for plotting
import warnings
warnings.filterwarnings('ignore')

fig, axes = plt.subplots(nrows=3, ncols=3, figsize = (20, 20))

X1 = X_4[:,0]
X2 = X_4[:,1]

X1_min, X1_max = X1.min() - 1 , X1.max() + 1
X2_min, X2_max = X2.min() - 1 , X2.max() + 1

x1_coord = np.linspace(X1_min, X1_max, num  = 500)
x2_coord = np.linspace(X2_min, X2_max, num  = 500)


X2_C, X1_C = np.meshgrid(x2_coord, x1_coord)
x1x2 = np.c_[X1_C.ravel(), X2_C.ravel()]


for i in range(len(C_range)*len(gamma_range)):
    c   = details[i][0]
    gam = details[i][1]
    svc = svm.SVC(C = c, gamma = gam)
    svc.fit(X_4, y_4)
    ax = axes[i//3][i%3]

    ax.contourf(X1_C, X2_C, svc.predict(x1x2).reshape(X1_C.shape), alpha=1)
    sns.scatterplot(X1,
                    X2,
                    c=y_4,
                    edgecolors='k',
                    ax = ax).set(
        title=f"C is {c} and Gamma is {gam} and score is {svc.score(X_4, y_4)}")
    ax.scatter(
             svc.support_vectors_[:, 0],
             svc.support_vectors_[:, 1],
             facecolors='blue',
             edgecolors='gray'
            )



#### Better than my code is below from the solution

# for (k, (C, gamma, clf)) in enumerate(details):
#     # evaluate the predictions in a grid
#     Z = clf.predict(x1x2)
```

```
#     Z = Z.reshape(X1_C.shape)

#     # visualize decision function for these parameters
#     plt.subplot(3, 3, k + 1)
#     plt.title("gam= %r, C= %r, score = %r"  % (gamma, C, round(clf.score(X_4,␣
↪y_4), 2)))

#     # visualize parameter's effect on decision function
#     plt.contourf(X1_C, X2_C, Z, alpha=1)
#     plt.scatter(X_4[:, 0], X_4[:, 1], c=y_4,  edgecolors='gray')
#     plt.axis('tight')
```

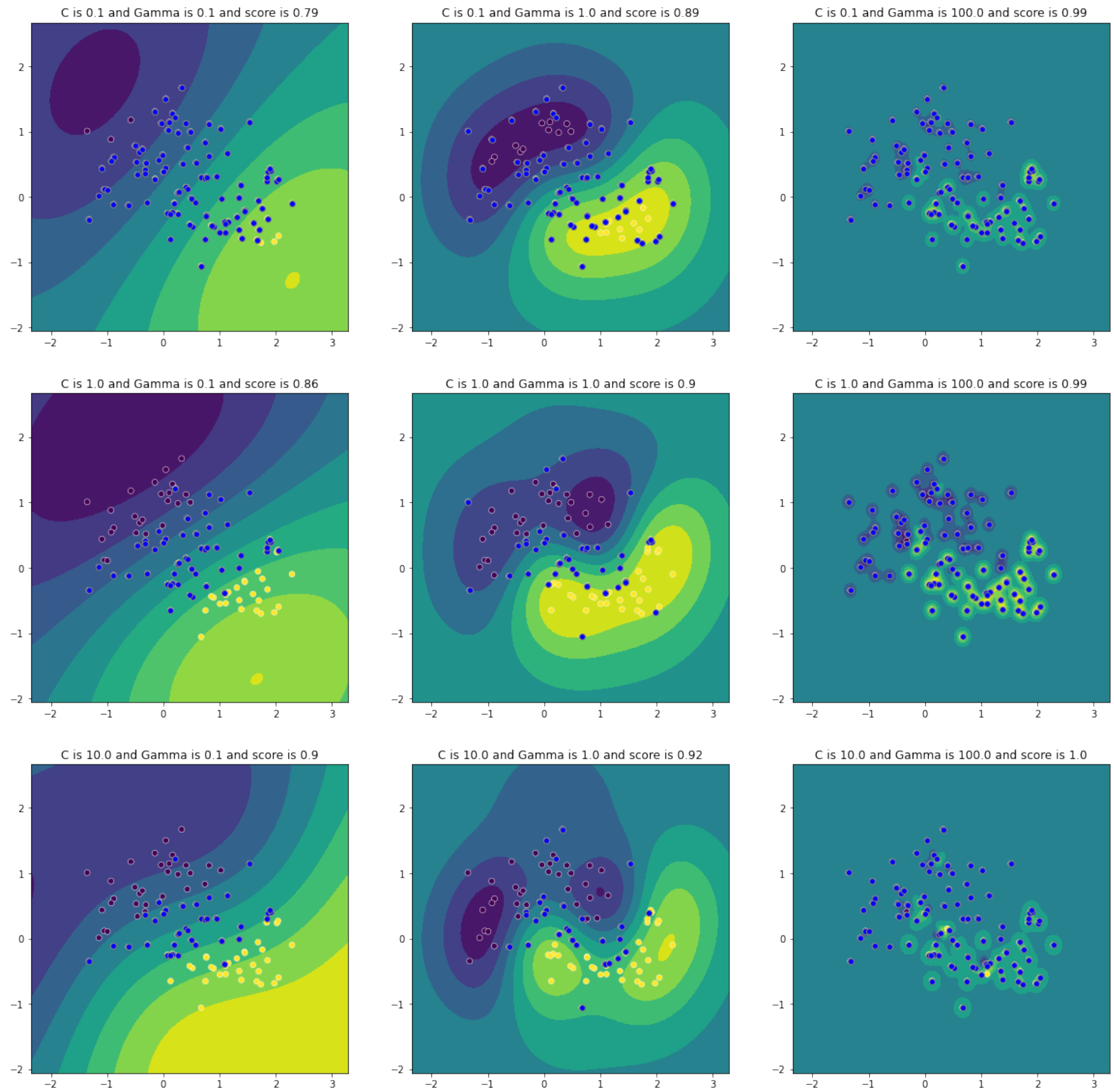


[ ]:
```
# Plot the prediction results in 9 subplots
```

Repeat what you did before but now, use `decision_function()` instead of `predict()`. What do you see?

[59]:
```
# Plot the decision function results in 9 subplots

import warnings
warnings.filterwarnings('ignore')

fig, axes = plt.subplots(nrows=3, ncols=3, figsize = (20, 20))

X1 = X_4[:,0]
X2 = X_4[:,1]
```

```
X1_min, X1_max = X1.min() - 1 , X1.max() + 1
X2_min, X2_max = X2.min() - 1 , X2.max() + 1

x1_coord = np.linspace(X1_min, X1_max, num  = 500)
x2_coord = np.linspace(X2_min, X2_max, num  = 500)


X2_C, X1_C = np.meshgrid(x2_coord, x1_coord)
x1x2 = np.c_[X1_C.ravel(), X2_C.ravel()]


for i in range(len(C_range)*len(gamma_range)):
    c   = details[i][0]
    gam = details[i][1]
    svc = svm.SVC(C = c, gamma = gam)
    svc.fit(X_4, y_4)
    ax = axes[i//3][i%3]

    ax.contourf(X1_C, X2_C, svc.decision_function(x1x2).reshape(X1_C.shape),␣
 ↪alpha=1)
    sns.scatterplot(X1,
                    X2,
                    c=y_4,
                    edgecolors='k',
                    ax = ax).set(
        title=f"C is {c} and Gamma is {gam} and score is {svc.score(X_4, y_4)}")
    ax.scatter(
             svc.support_vectors_[:, 0],
             svc.support_vectors_[:, 1],
             facecolors='blue',
             edgecolors='gray'
            )
```



1.5 Explore the Polynomial kernel

Recall that the polynomial kernel has 3 hyperparameters: - γ, which can be specified using parameter `gamma` - r, which can be specified using parameter `coef0` - d, which can be specified using parameter `degree`

Build 8 different plots using all the possible combinations: - $r = 0.1$ and 2 - $\gamma = 0.1$ and 1 - $d = 3$ and 4

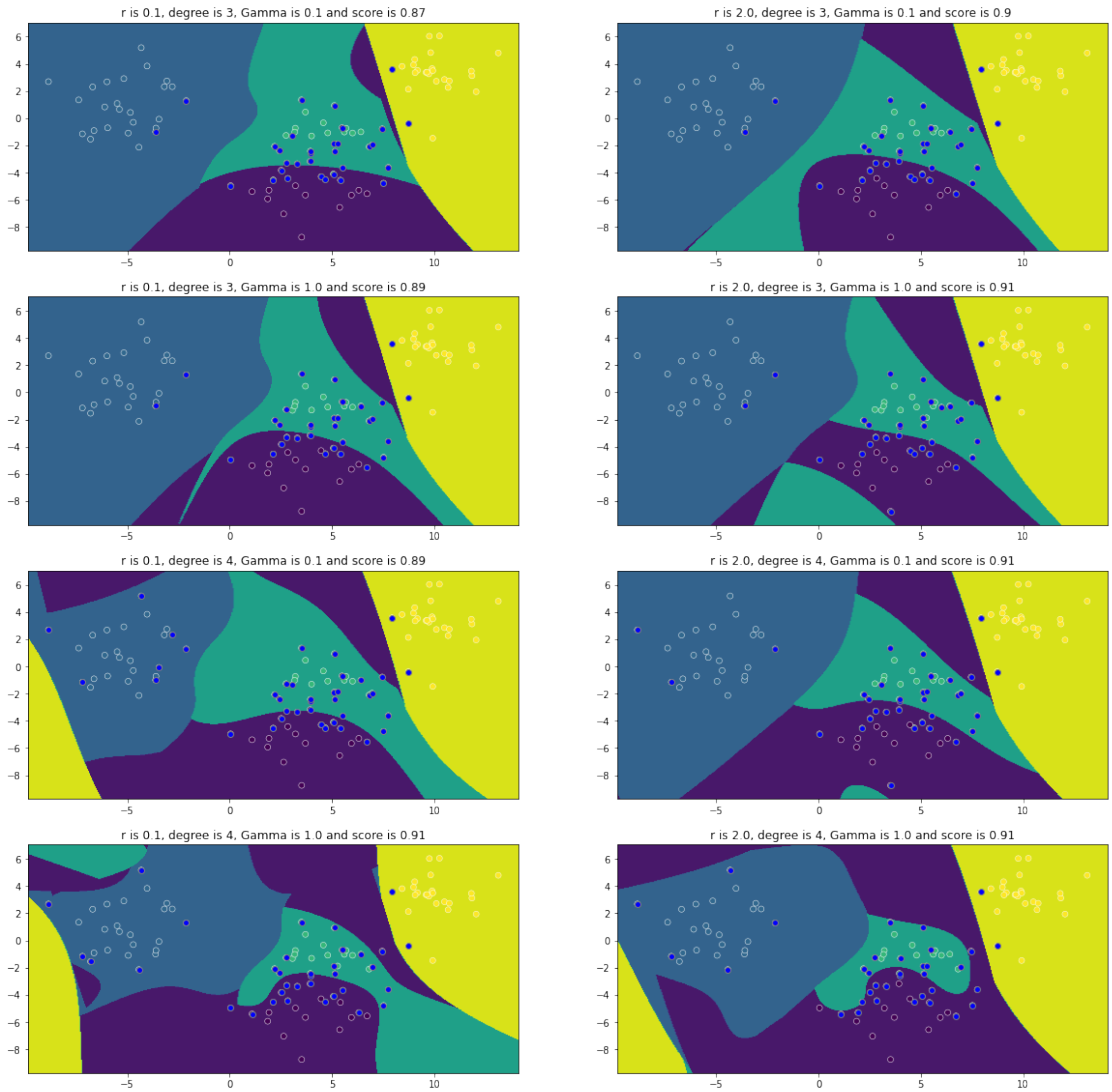
Note that `decision_function()` cannot be used on a classifier with more than two classes, so simply use `predict()` again.

[94]:
```
r_range =  np.array([0.1, 2])
gamma_range =  np.array([0.1, 1])
d_range = np.array([3, 4])
```

```
param_grid = dict(gamma=gamma_range, degree=d_range, coef0=r_range)
details = []

# Create a loop that builds a model for each of the 8 combinations
for d in d_range:
    for g in gamma_range:
        for r in r_range:
            details.append((r, d, g))
```

[96]:
```
# Prepare your data for plotting

import warnings
warnings.filterwarnings('ignore')

fig, axes = plt.subplots(nrows=4, ncols=2, figsize = (20, 20))

X1 = X_3[:,0]
X2 = X_3[:,1]

X1_min, X1_max = X1.min() - 1 , X1.max() + 1
X2_min, X2_max = X2.min() - 1 , X2.max() + 1

x1_coord = np.linspace(X1_min, X1_max, num  = 500)
x2_coord = np.linspace(X2_min, X2_max, num  = 500)


X2_C, X1_C = np.meshgrid(x2_coord, x1_coord)
x1x2 = np.c_[X1_C.ravel(), X2_C.ravel()]


for i in range(len(r_range)*len(gamma_range)*len(d_range)):
    r   = details[i][0]
    d   = details[i][1]
    g   = details[i][2]
    svc = svm.SVC(kernel='poly', coef0=r, degree = d, gamma = g)
    svc.fit(X_3, y_3)
    ax = axes[i//2][i%2]

    ax.contourf(X1_C, X2_C, svc.predict(x1x2).reshape(X1_C.shape), alpha=1)
    sns.scatterplot(X1,
                    X2,
                    c=y_3,
                    edgecolors='k',
                    ax = ax).set(
```

```
        title=f"r is {r}, degree is {d}, Gamma is {g} and score is {svc.
↪score(X_3, y_3)}")
    ax.scatter(
            svc.support_vectors_[:, 0],
            svc.support_vectors_[:, 1],
            facecolors='blue',
            edgecolors='gray'
           )
```



1.6 The Sigmoid kernel

Build a support vector machine using the Sigmoid kernel.

Recall that the sigmoid kernel has 2 hyperparameters: - γ, which can be specified using parameter `gamma` - r, which can be specified using parameter `coef0`

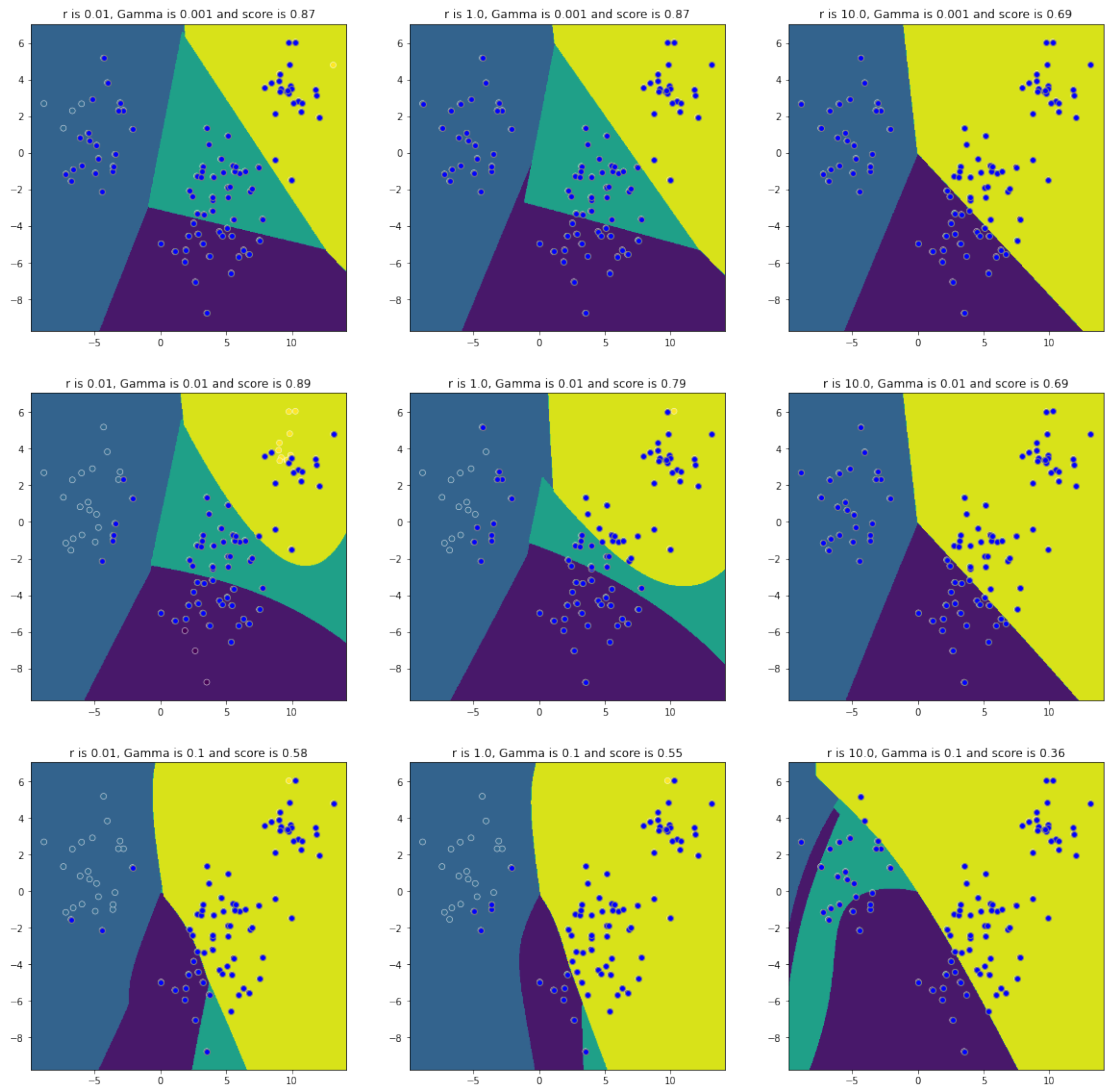
Look at 9 solutions using the following values for γ and r.

- γ= 0.001, 0.01, and 0.1
- r = 0.01, 1, and 10

[99]:
```
# Create a loop that builds a model for each of the 9 combinations
gamma_range = np.array([0.001, 0.01, 0.1])
r_range = np.array([0.01, 1, 10])

det = []
for g in gamma_range:
    for r in r_range:
        det.append((g, r))
```

[100]:
```
# Prepare your data for plotting
import warnings
warnings.filterwarnings('ignore')

fig, axes = plt.subplots(nrows=3, ncols=3, figsize = (20, 20))

X1 = X_3[:,0]
X2 = X_3[:,1]

X1_min, X1_max = X1.min() - 1 , X1.max() + 1
X2_min, X2_max = X2.min() - 1 , X2.max() + 1

x1_coord = np.linspace(X1_min, X1_max, num  = 500)
x2_coord = np.linspace(X2_min, X2_max, num  = 500)


X2_C, X1_C = np.meshgrid(x2_coord, x1_coord)
x1x2 = np.c_[X1_C.ravel(), X2_C.ravel()]


for i in range(len(r_range)*len(gamma_range)):
    r   = det[i][1]
    g   = det[i][0]
    svc = svm.SVC(kernel='sigmoid', coef0=r , gamma=g)
    svc.fit(X_3, y_3)
    ax = axes[i//3][i%3]

    ax.contourf(X1_C, X2_C, svc.predict(x1x2).reshape(X1_C.shape), alpha=1)
    sns.scatterplot(X1,
                    X2,
                    c=y_3,
                    edgecolors='k',
```

```
                  ax = ax).set(
    title=f"r is {r}, Gamma is {g} and score is {svc.score(X_3, y_3)}")
ax.scatter(
        svc.support_vectors_[:, 0],
        svc.support_vectors_[:, 1],
        facecolors='blue',
        edgecolors='gray'
       )
```



[ ]:
```
# Plot the prediction results in 9 subplots
```

1.7 What is your conclusion here?

- The polynomial kernel is very sensitive to the hyperparameter settings. Especially when setting a "wrong" gamma - this can have a dramatic effect on the model performance
- Our experiments with the Polynomial kernel were more successful

1.8 Explore the Polynomial kernel again, with a train-test split

Explore the same parameters you did before when exploring polynomial kernel

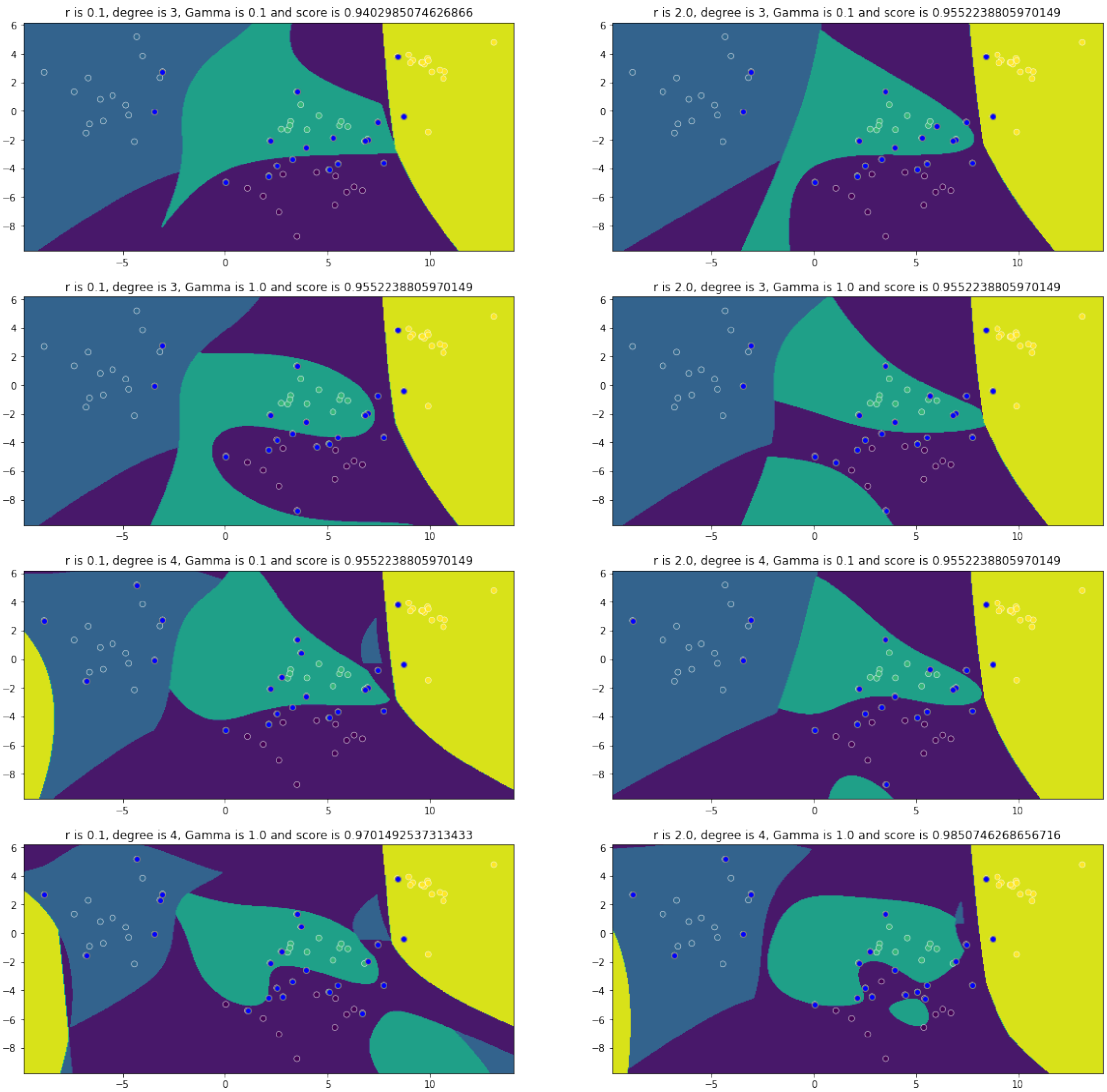
- Perform a train-test split. Assign 33% of the data to the test set and set the `random_state` to 123
- Train 8 models using the training set for each combination of different parameters
- Plot the results as above, both for the training and test sets
- Make some notes for yourself on training vs test performance and select an appropriate model based on these results

[104]:
```
# Perform a train-test split, then create a loop that builds a model for each␣
↪of the 8 combinations
X_train, X_test, y_train, y_test = train_test_split(X_3, y_3,
                                                    test_size = 0.33,
                                                    random_state = 123)

# Create a loop that builds a model for each of the 8 combinations
r_range =  np.array([0.1, 2])
gamma_range =  np.array([0.1, 1])
d_range = np.array([3, 4])

for d in d_range:
    for g in gamma_range:
        for r in r_range:
            details.append((r, d, g))
```

[108]:
```
# Prepare your data for plotting

X1_test = X_test[:,0]
X2_test = X_test[:,1]

X1_test_min, X1_test_max = X1_test.min() - 1 , X1_test.max() + 1
X2_test_min, X2_test_max = X2_test.min() - 1 , X2_test.max() + 1

x1_coord_test = np.linspace(X1_test_min, X1_test_max, num  = 500)
x2_coord_test = np.linspace(X2_test_min, X2_test_max, num  = 500)


X2_C_test, X1_C_test = np.meshgrid(x2_coord_test, x1_coord_test)
x1x2_test = np.c_[X1_C_test.ravel(), X2_C_test.ravel()]
```

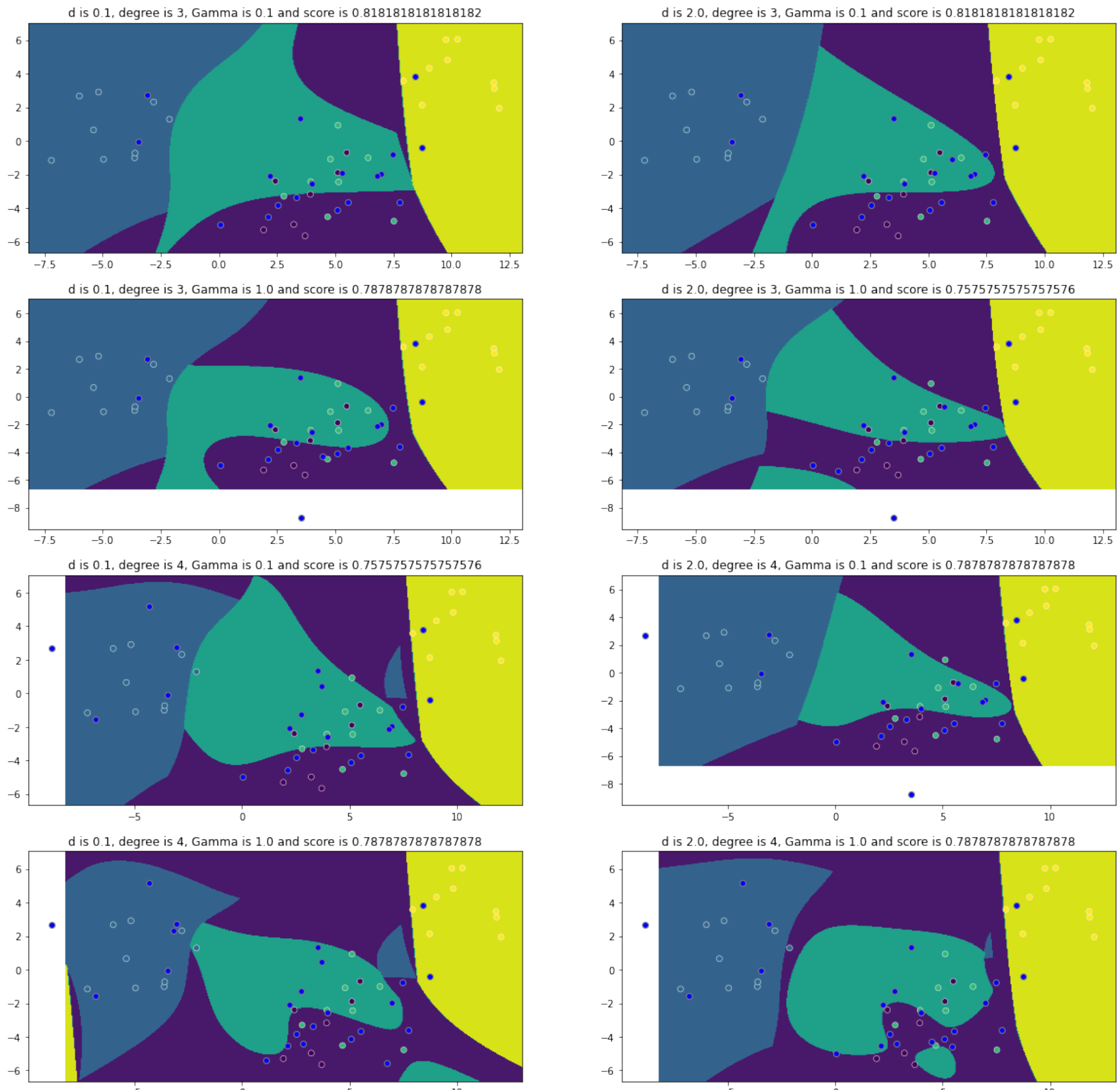
[109]:

```
fig, axes = plt.subplots(nrows=4, ncols=2, figsize = (20, 20))

# Prepare your data for plotting

X1_train = X_train[:,0]
X2_train = X_train[:,1]

X1_train_min, X1_train_max = X1_train.min() - 1 , X1_train.max() + 1
X2_train_min, X2_train_max = X2_train.min() - 1 , X2_train.max() + 1

x1_coord_train = np.linspace(X1_train_min, X1_train_max, num  = 500)
x2_coord_train = np.linspace(X2_train_min, X2_train_max, num  = 500)


X2_C_train, X1_C_train = np.meshgrid(x2_coord_train, x1_coord_train)
x1x2_train = np.c_[X1_C_train.ravel(), X2_C_train.ravel()]




for i in range(len(r_range)*len(gamma_range)*len(d_range)):
    r   = details[i][0]
    d   = details[i][1]
    g   = details[i][2]
    svc = svm.SVC(kernel='poly', coef0=r, degree = d, gamma = g)
    svc.fit(X_train, y_train)
    ax = axes[i//2][i%2]

    ax.contourf(X1_C_train,
                X2_C_train,
                svc.predict(x1x2_train).reshape(X1_C_train.shape),
                alpha=1)

    sns.scatterplot(X1_train,
                    X2_train,
                    c=y_train,
                    edgecolors='k',
                    ax = ax).set(
        title=f"r is {r}, degree is {d}, Gamma is {g} and score is {svc.
 ↪score(X_train, y_train)}")
    ax.scatter(
             svc.support_vectors_[:, 0],
             svc.support_vectors_[:, 1],
             facecolors='blue',
             edgecolors='gray'
            )
```



[110]:
```
# Now plot the prediction results for the test set

fig, axes = plt.subplots(nrows=4, ncols=2, figsize = (20, 20))

# Prepare your data for plotting

X1_test = X_test[:,0]
X2_test = X_test[:,1]

X1_test_min, X1_test_max = X1_test.min() - 1 , X1_test.max() + 1
X2_test_min, X2_test_max = X2_test.min() - 1 , X2_test.max() + 1

x1_coord_test = np.linspace(X1_test_min, X1_test_max, num  = 500)
```

```
x2_coord_test = np.linspace(X2_test_min, X2_test_max, num  = 500)


X2_C_test, X1_C_test = np.meshgrid(x2_coord_test, x1_coord_test)
x1x2_test = np.c_[X1_C_test.ravel(), X2_C_test.ravel()]





for i in range(len(r_range)*len(gamma_range)*len(d_range)):
    r   = details[i][0]
    d   = details[i][1]
    g   = details[i][2]
    svc = svm.SVC(kernel='poly', coef0=r, degree = d, gamma = g)
    svc.fit(X_train, y_train)
    ax = axes[i//2][i%2]

    ax.contourf(X1_C_test,
                X2_C_test,
                svc.predict(x1x2_test).reshape(X1_C_test.shape),
                alpha=1)

    sns.scatterplot(X1_test,
                    X2_test,
                    c=y_test,
                    edgecolors='k',
                    ax = ax).set(
        title=f"d is {r}, degree is {d}, Gamma is {g} and score is {svc.
↪score(X_test, y_test)}")
    ax.scatter(
             svc.support_vectors_[:, 0],
             svc.support_vectors_[:, 1],
             facecolors='blue',
             edgecolors='gray'
            )
```



1.9 A higher-dimensional, real-world dataset

Until now, you've only explored datasets with two features to make it easy to visualize the decision boundary. Remember that you can use Support Vector Machines on a wide range of classification datasets, with more than two features. While you will no longer be able to visually represent decision boundaries (at least not if you have more than three feature spaces), you'll still be able to make predictions.

To do this, you'll use the salaries dataset again (in `'salaries_final.csv'`).

This dataset has six predictors:

- `Age`: continuous
- `Education`: Categorical - Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool

- `Occupation`: Categorical - Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
- `Relationship`: Categorical - Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried
- `Race`: Categorical - White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
- `Sex`: Categorical - Female, Male

Simply run the code below to import and preview the dataset.

[112]:
```
import statsmodels as sm
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import pandas as pd
salaries = pd.read_csv('salaries_final.csv', index_col=0)
salaries.head()
```

[112]:

	Age	Education	Occupation	Relationship	Race	Sex	Target
0	39	Bachelors	Adm-clerical	Not-in-family	White	Male	<=50K
1	50	Bachelors	Exec-managerial	Husband	White	Male	<=50K
2	38	HS-grad	Handlers-cleaners	Not-in-family	White	Male	<=50K
3	53	11th	Handlers-cleaners	Husband	Black	Male	<=50K
4	28	Bachelors	Prof-specialty	Wife	Black	Female	<=50K

The following cell creates dummy variables for all categorical columns and splits the data into target and predictor variables.

[113]:
```
# Create dummy variables and
# Split data into target and predictor variables
target = pd.get_dummies(salaries['Target'], drop_first=True)
xcols = salaries.columns[:-1]
data = pd.get_dummies(salaries[xcols], drop_first=True)
```

Now build a simple linear SVM using this data. Note that using SVC, some slack is automatically allowed, so the data doesn't have to perfectly linearly separable.

- Create a train-test split of 75-25. Set the `random_state` to 123
- Standardize the data
- Fit an SVM model, making sure that you set "probability = True"
- After you run the model, calculate the classification accuracy score on both the test set

[115]:
```
# Split the data into a train and test set
X_train, X_test, y_train, y_test = train_test_split(data, target,
                                                    random_state = 123,
                                                    test_size = 0.25)
```

[116]:
```python
# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Warning: It takes quite a while to build the model! The score is slightly better than the best result obtained using decision trees, but at the cost of computational resources. Changing kernels can make computation times even longer.

[ ]:
```python
# Fit SVM model
#   This cell may take several minutes to run
from sklearn.svm import SVC
clf = SVC(probability=True)
clf.fit(X_train_scaled, y_train[">50K"])
```

[ ]:
```python
# Calculate the classification accuracy score
clf.score(X_test_scaled, y_test)
```

1.10 Summary

Great, you've got plenty of practice with Support Vector Machines! In this lab, you explored kernels and applying SVMs to real-life data!