ComS574_HW3_sol

March 7, 2020

ComS 574

HW 3

Kanak Choudhury

1 Problem 1 & 2:

[367]:
```
from IPython.display import IFrame, display
path  = 'D:/ISU/COMS 574 - Introduction to Machine Learning/HW/HW3/'
pdf = IFrame(path+'ComS_574_HW3_1-2.pdf', width=700, height=400)
print(pdf)
```

```
<IPython.lib.display.IFrame object at 0x000002671FC76518>
```

2 Problem 3: Ch 9 #1

2.1 (a)

Sketch the hyperplane $1 + 3X_1 - X_2 = 0$. Indicate the set of points for which $1 + 3X_1 - X_2 > 0$, as well as the set of points for which $1 + 3X_1 - X_2 < 0$.

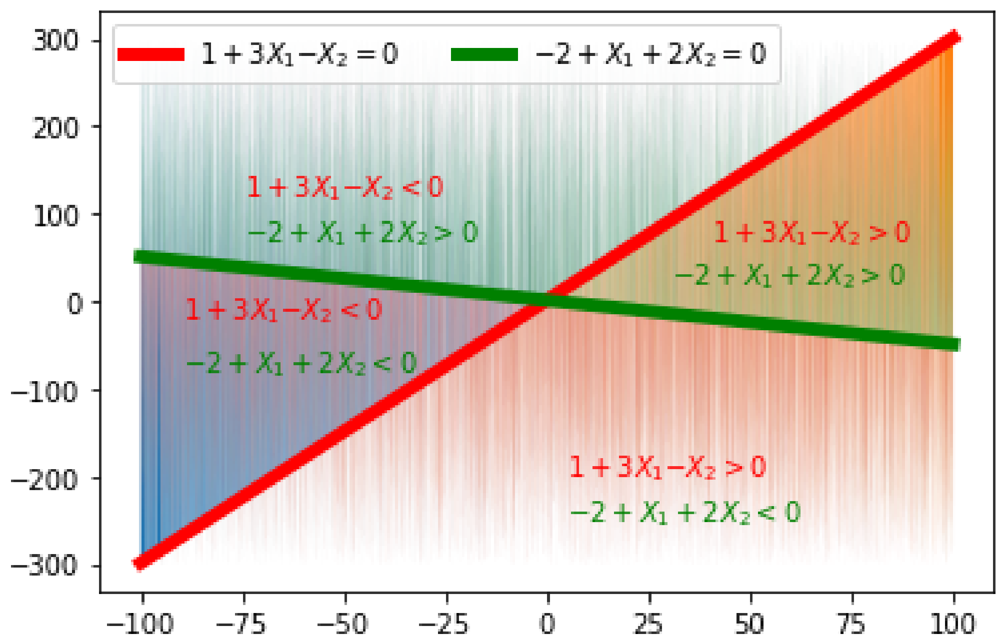
2.2 (b)

On the same plot, sketch the hyperplane $-2 + X_1 + 2X_2 = 0$. Indicate the set of points for which $-2 + X_1 + 2X_2 > 0$, as well as the set of points for which $-2 + X_1 + 2X_2 < 0$.

[368]:
```
import numpy as np
from matplotlib import pyplot as plt
import turtle
```

[369]:
```
x1 = np.linspace(-100, 100, 50000)
x2 = 1+3*x1
x22 = np.random.uniform(301, -301, 50000)
x2b = 1-x1/2

plt.plot(x1, x2, linewidth=5, color = 'red', label = '$1 + 3X_1 - X_2=0$')
plt.plot(x1, x2b, linewidth=5, color = 'green', label = '$-2+ X_1 +2X_2 = 0$')
```

```
plt.fill_between(x1, x2, x22, where=x22>x2,  alpha = 1)
plt.fill_between(x1, x2, x22, where=x22<x2,  alpha = 1)

plt.fill_between(x1, x2b, x22, where=x22>x2b,  alpha = 1)
plt.fill_between(x1, x2b, x22, where=x22<x2b,  alpha = 1)

plt.text(-75, 120, '$1 + 3X_1 - X_2 < 0$', color = 'red')
plt.text(5, -200, '$1 + 3X_1 - X_2 > 0$', color = 'red')
plt.text(40, 70, '$1 + 3X_1 - X_2 > 0$', color = 'red')
plt.text(-90, -20, '$1 + 3X_1 - X_2 < 0$', color = 'red')

plt.text(-75, 70, '$-2+ X_1 +2X_2 > 0$', color = 'green')
plt.text(5, -250, '$-2+ X_1 +2X_2 < 0$', color = 'green')
plt.text(30, 20, '$-2+ X_1 +2X_2 > 0$', color = 'green')
plt.text(-90, -80, '$-2+ X_1 +2X_2 < 0$', color = 'green')
plt.legend(loc = 'upper left', ncol=2)
plt.show()
```



3 Problem 4: Ch 9 #2

3.1 (a)

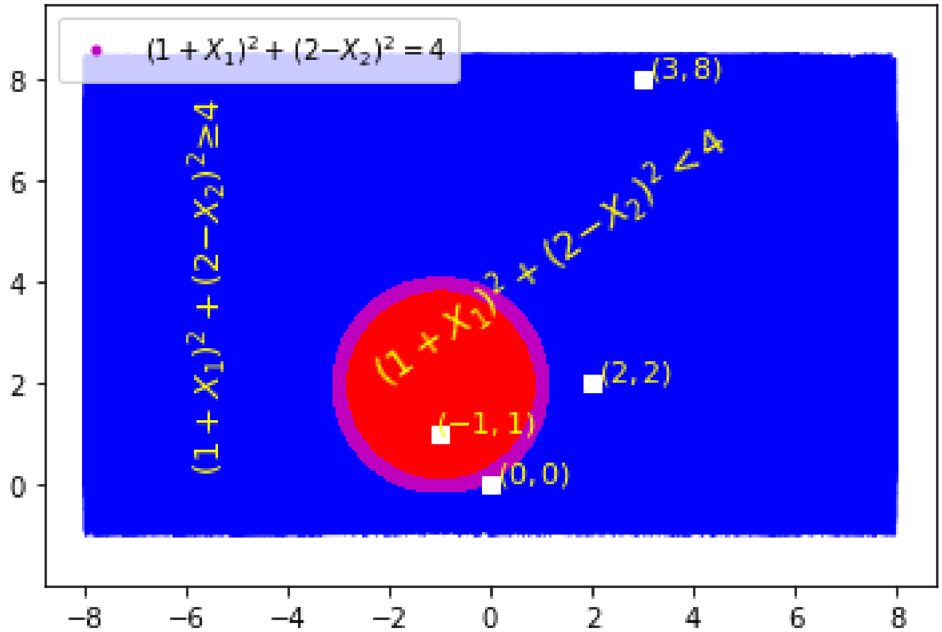
Sketch the curve $(1 + X_1)^2 + (2 - X_2)^2 = 4$.

3.2 (b)

On your sketch, indicate the set of points for which $(1 + X_1)^2 + (2{-}X_2)^2 > 4$, as well as the set of points for which $(1 + X_1)^2 + (2{-}X_2)^2$ 4.

[370]:

```
x1 = np.linspace(-5, 5, 50000)
aa = 4-(1+x1)**2
x1 = np.concatenate((x1[aa>0], x1[aa>0]), axis =0)
aa = aa[aa>0]
x2 = np.concatenate((2-np.sqrt(aa), 2+np.sqrt(aa)), axis =0)
x22 = np.random.uniform(-1, 8.5, x1.shape[0])
x3 = np.random.uniform(-8, 8, x1.shape[0])
plt.plot(x1,x2,'.', label = '$(1 + X_1)^2 + (2 - X_2)^2 = 4$', color = 'm')
plt.axis('equal')
plt.fill_between(x3, x2, x22, where=((1+x1)**2 + (2-x22)**2)>=4 ,  color =␣
 ↪'blue')
plt.fill_between(x1, x2, x22, where=((1+x1)**2 + (2-x22)**2)<4 ,  color = 'red')

plt.text(-6, 0.5, '$(1 + X_1)^2 + (2 - X_2)^2 \u2265 4$', color = 'yellow',␣
 ↪fontsize=13, rotation=90)
plt.text(-2.5, 2, '$(1 + X_1)^2 + (2 - X_2)^2 < 4$', color = 'yellow',␣
 ↪fontsize=14, rotation = 35)



plt.plot(0,0, 's',color = 'w')
plt.text(.1, .1, '$(0,0)$', color = 'yellow', fontsize=11)

plt.plot(-1,1, 's',color = 'w')
plt.text(-1.1, 1.1, '$(-1,1)$', color = 'yellow', fontsize=11)

plt.plot(2,2, 's',color = 'w')
plt.text(2.1, 2.1, '$(2,2)$', color = 'yellow', fontsize=11)

plt.plot(3,8, 's',color = 'w')
plt.text(3.1, 8.1, '$(3,8)$', color = 'yellow', fontsize=11)


plt.legend(loc = 'upper left')
plt.show()
```



3.3 (c)

Suppose that a classifier assigns an observation to the blue class if $(1 + X_1)^2 + (2-X_2)^2 > 4$, and to the red class otherwise. To what class is the observation $(0, 0)$ classified? $(-1, 1)$? $(2, 2)$? $(3, 8)$?

For $(0, 0)$: $(1 + X_1)^2 + (2-X_2)^2 = 5 > 4$ Class: Blue

For $(-1, 1)$: $(1 + X_1)^2 + (2-X_2)^2 = 1 < 4$ Class: Red

For $(2, 2)$: $(1 + X_1)^2 + (2-X_2)^2 = 9 > 4$ Class: Blue

For $(3, 8)$: $(1 + X_1)^2 + (2-X_2)^2 = 52 > 4$ Class: Blue

3.4 (d)

Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2 , X_2, and X_2^2.

Clearly, from

$$(1 + X_1)^2 + (2-X_2)^2 = 4X_1^2 + X_2^2 + 2X_1 - 4X_2 + 1 = 0$$

which is linear in terms of X_1, X_1^2 , X_2, and X_2^2

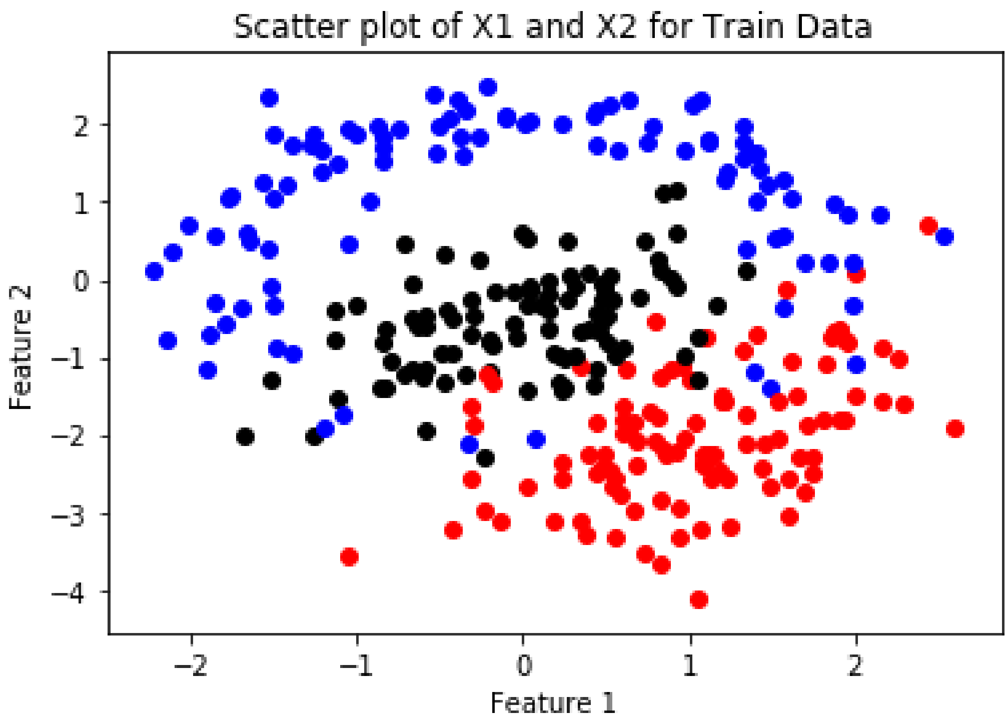
4 Problem 5

4.1 A.

[371]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LogisticRegression as lr
from sklearn.neighbors import KNeighborsClassifier as kn
from matplotlib.colors import ListedColormap
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
from sklearn.svm import SVC

import sys
import gc
from itertools import product

import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.simplefilter("ignore", ConvergenceWarning)
```

[372]:
```
def plot_scatter(x_df, y_df, title=None):
    for x,y in zip(x_df, y_df):
    #     print(x1,x2,y)
        if y==1:
            col = 'blue'
        if y==2:
            col = 'red'
        if y==3:
            col = 'black'
        plt.scatter(x[0], x[1],  color=col)

    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()


df_train = pd.read_csv(path + 'HW3train.csv', sep=',',
                       header=None, names=['Y', 'X1', 'X2'])
df_test = pd.read_csv(path + 'HW3test.csv', sep=',',
                      header=None, names=['Y', 'X1', 'X2'])
tr_size = df_train.shape
ts_size = df_test.shape

x_train = np.array(df_train[['X1', 'X2']])
```

```
y_train = np.array(df_train['Y'])

x_test = np.array(df_test[['X1', 'X2']])
y_test = np.array(df_test['Y'])


plot_scatter(x_train, y_train, title = 'Scatter plot of X1 and X2 for Train␣
 ↪Data')

plot_scatter(x_test, y_test, title = 'Scatter plot of X1 and X2 for Test Data')
```

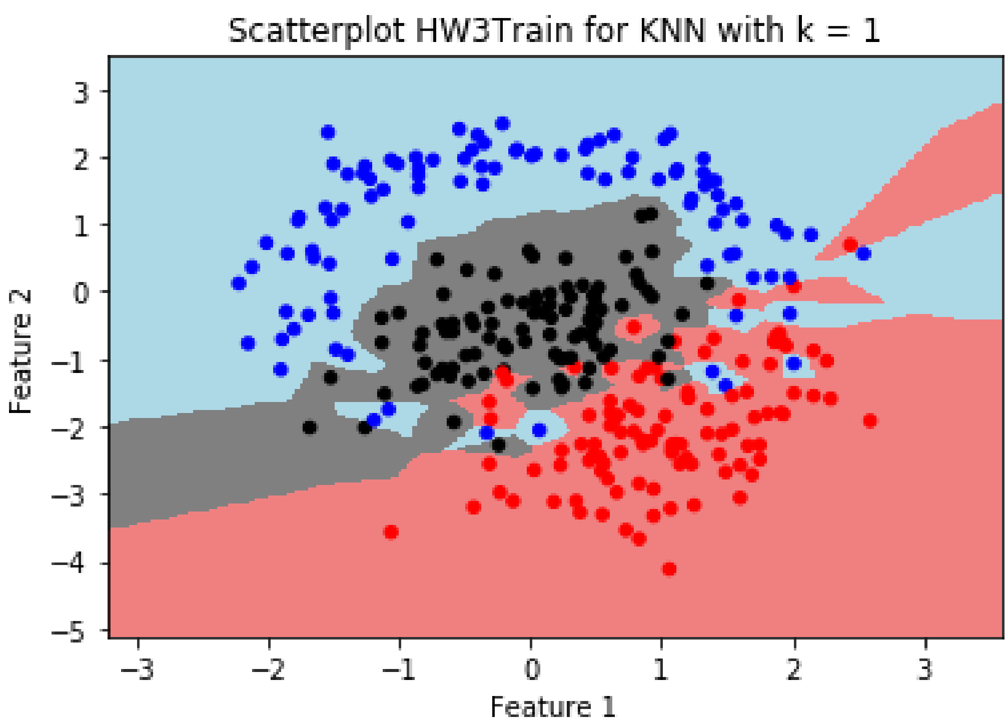


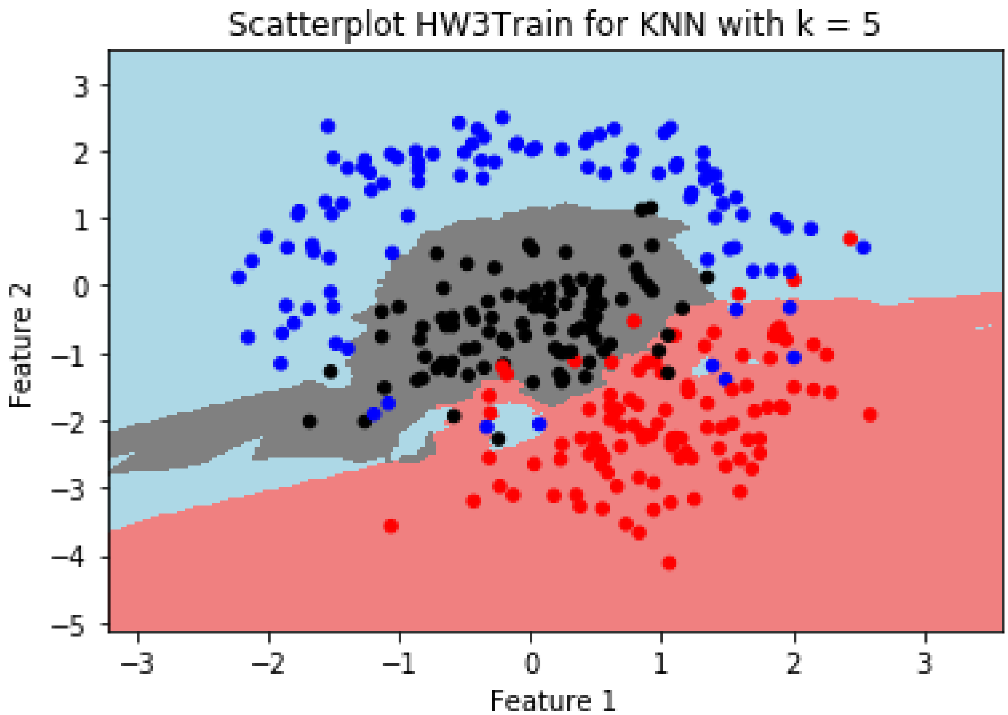
4.2 B. KNN

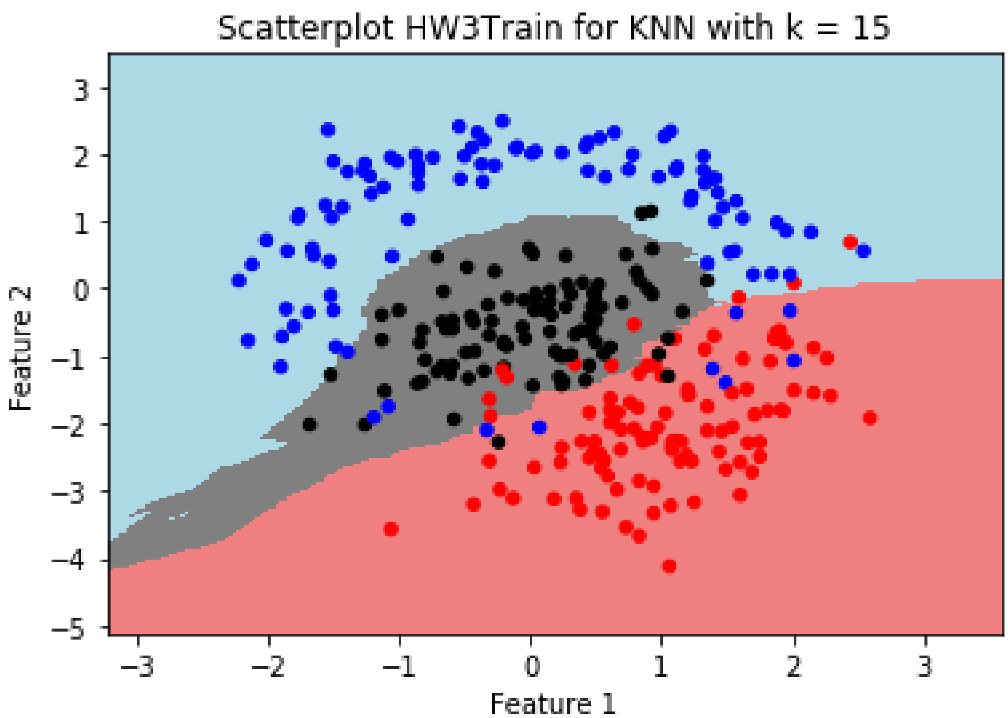
4.2.1 (1)

[373]:
```
def plot_func(x_df, y_df, pred, title = None):

    pred = pred.reshape(x1mesh.shape)

    plt.figure()
    plt.pcolormesh(x1mesh, x2mesh, pred, cmap=cmap_light)
    ytrain_colors = [y-1 for y in y_df]
    plt.scatter(x_df[:, 0], x_df[:, 1], c=ytrain_colors, cmap=cmap_bold, s=20)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()


h = .03
x1_min, x1_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
x2_min, x2_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
```

```
x1mesh, x2mesh = np.meshgrid(np.arange(x1_min, x1_max, h), np.arange(x2_min,␣
↪x2_max, h))

cmap_light = ListedColormap(['lightblue', 'lightcoral', 'grey'])
cmap_bold = ListedColormap(['blue', 'red', 'black'])

for i in [1,5,15]:
    model_kn = kn(n_neighbors=i, weights='uniform', algorithm='auto').
↪fit(x_train, y_train)
    Z = model_kn.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])

    plot_func(x_train, y_train, Z, title = 'Scatterplot HW3Train for KNN with k␣
↪= '+str(i))
```




Scatterplot HW3Train for KNN with k = 5
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3

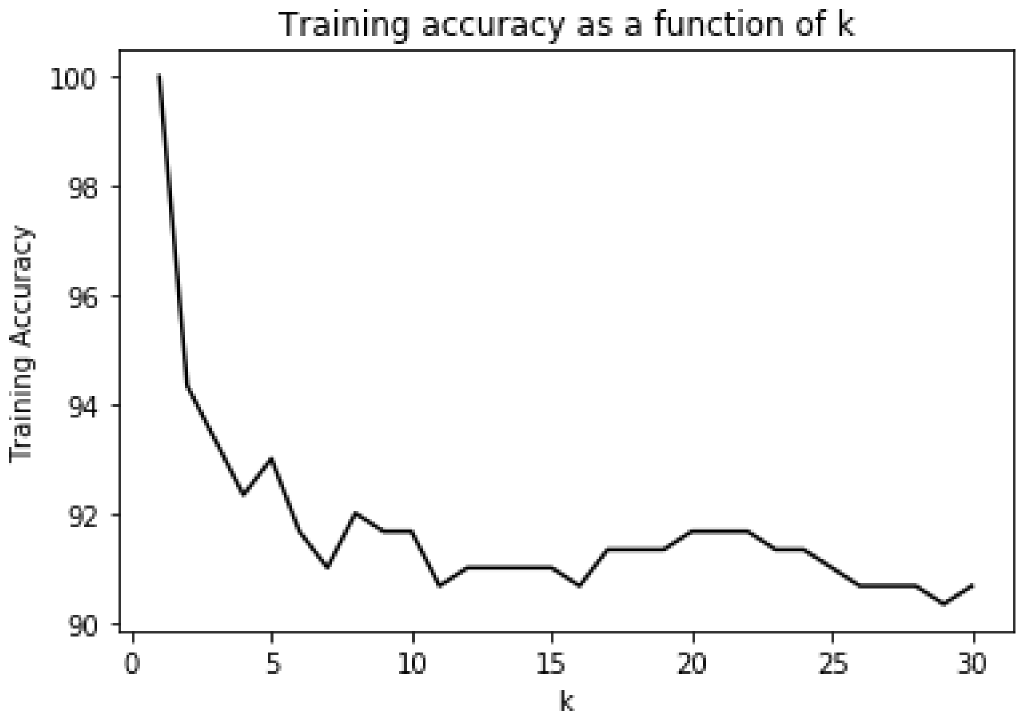

Scatterplot HW3Train for KNN with k = 15
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3

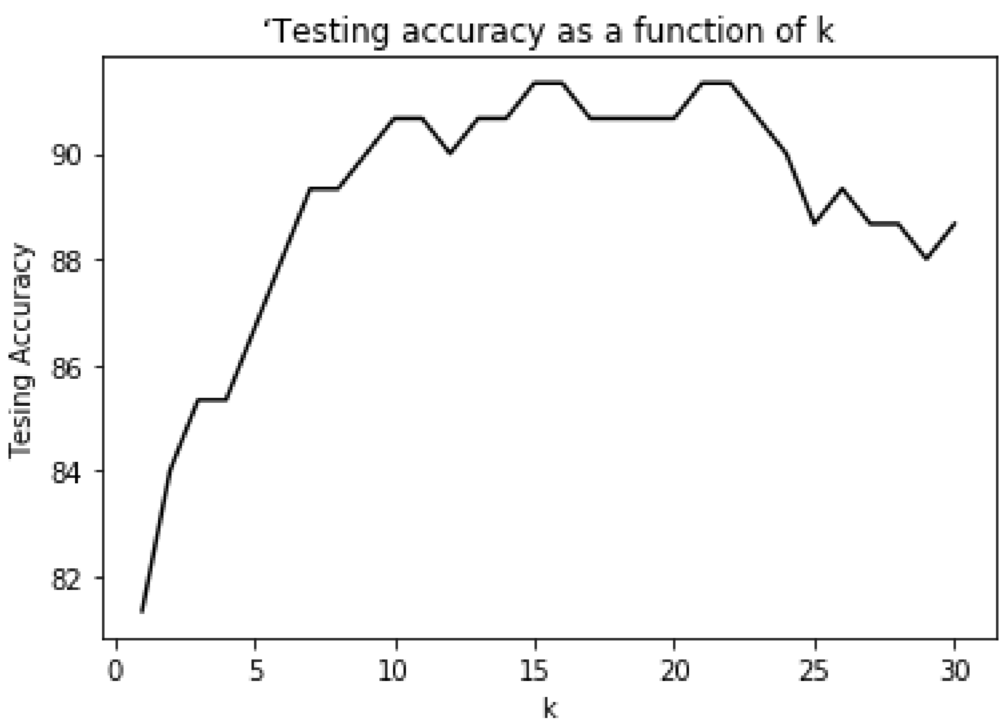
4.2.2 (2)

Clearly, for $k = 1$ generates the most complex model. It creates a region around each data points. As a result, it is very likely to have a overfitted model. However, as the value of k increases, it creates smoother that is less complex boundary. Similarly, for $k = 1$, training error will be 0 and it is highly likely to have a high-test error. As we increase k, it makes a trade-off (Training and testing accuracy plots below) between training and testing error.

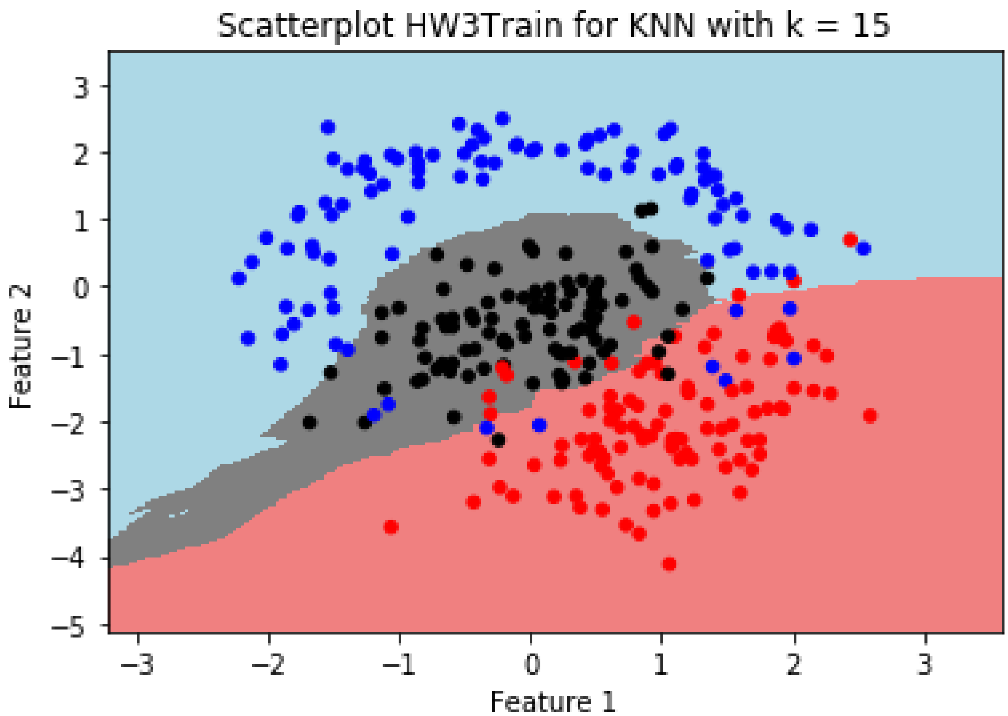
4.2.3 (3) - (7)

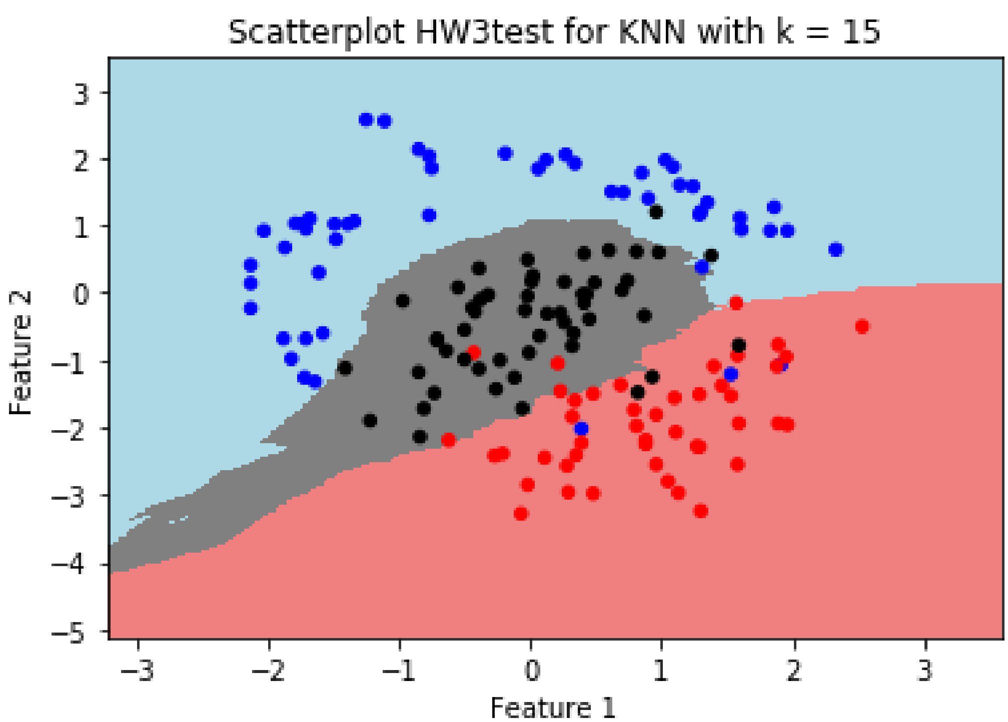
[374]:
```
tr_acc = []
ts_acc = []
k = range(1,31)
for i in k:
    model_kn = kn(n_neighbors=i, weights='uniform', algorithm='auto').
 ↪fit(x_train, y_train)
    tr_acc.append(model_kn.score(x_train, y_train)*100)
    ts_acc.append(model_kn.score(x_test, y_test)*100)


plt.plot(k, tr_acc, color = 'black')
plt.xlabel("k")
plt.ylabel("Training Accuracy")
plt.title("Training accuracy as a function of k")
plt.show()

plt.plot(k, ts_acc, color = 'black')
plt.xlabel("k")
plt.ylabel("Tesing Accuracy")
plt.title("‘Testing accuracy as a function of k")
plt.show()

print('Best k based on maximum testing accuracy: k = %d with training_accuracy␣
 ↪= %.2f and testing_accuracy = %.2f'
     %(np.argmax(ts_acc)+1, tr_acc[np.argmax(ts_acc)], ts_acc[np.
 ↪argmax(ts_acc)]))

model_kn = kn(n_neighbors=np.argmax(ts_acc)+1, weights='uniform',␣
 ↪algorithm='auto').fit(x_train, y_train)
# pred_tr = model_kn.predict(x_train)
# pred_ts = model_kn.predict(x_test)
Z = model_kn.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])

plot_func(x_train, y_train, Z, title = 'Scatterplot HW3Train for KNN with k =␣
 ↪'+str(np.argmax(ts_acc)+1))
plot_func(x_test, y_test, Z, title = 'Scatterplot HW3test for KNN with k =␣
 ↪'+str(np.argmax(ts_acc)+1))
```


Training accuracy as a function of k
Training Accuracy
100
98
96
94
92
90
0
5
10
15
20
25
30
k


'Testing accuracy as a function of k
Tesing Accuracy
90
88
86
84
82
0
5
10
15
20
25
30
k

```
Best k based on maximum testing accuracy: k = 15 with training_accuracy = 91.00
and testing_accuracy = 91.33
```

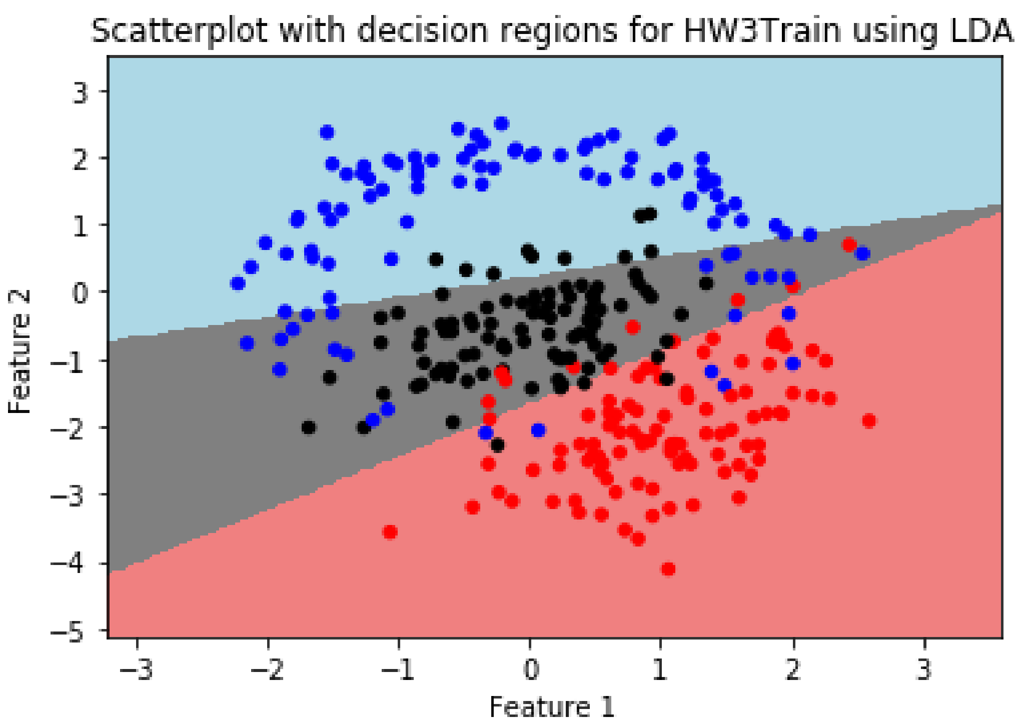


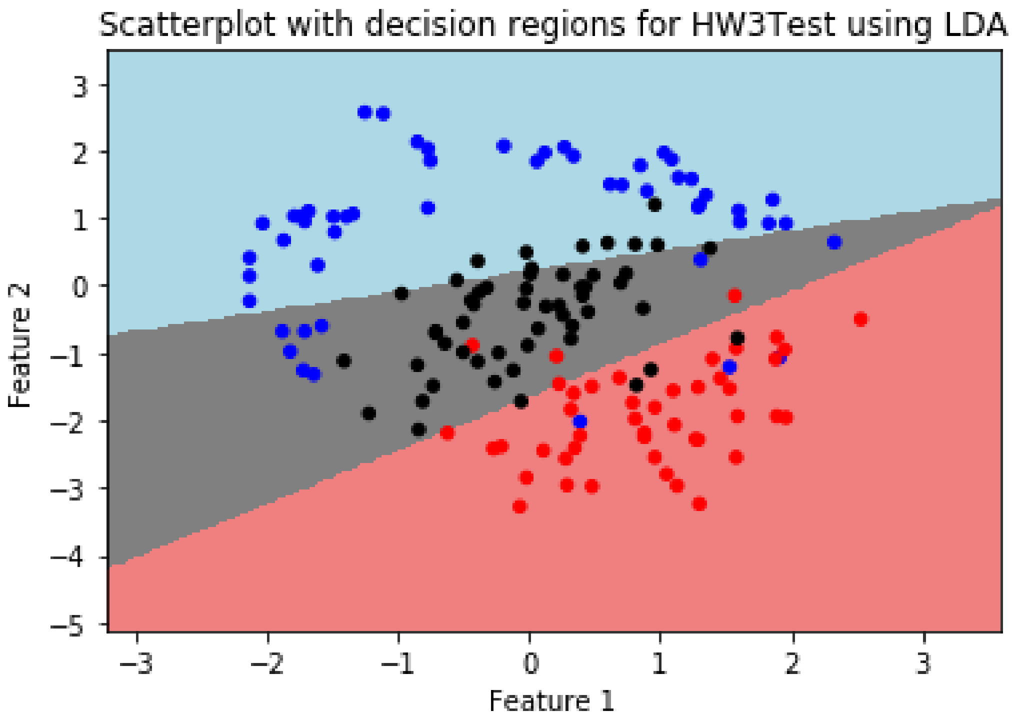


4.3 C. LDA

[375]:
```
lda1 = lda(solver='svd', shrinkage=None, priors= None)
fit_lda = lda1.fit(x_train, y_train)

Z = fit_lda.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])

print('For LDA: training_accuracy = %.2f and testing_accuracy = %.2f'
      %(lda1.score(x_train, y_train)*100, lda1.score(x_test, y_test)*100))

plot_func(x_train, y_train, Z, title = 'Scatterplot with decision regions for␣
 ↪HW3Train using LDA')
plot_func(x_test, y_test, Z, title = 'Scatterplot with decision regions for␣
 ↪HW3Test using LDA')
```

```
For LDA: training_accuracy = 83.33 and testing_accuracy = 81.33
```

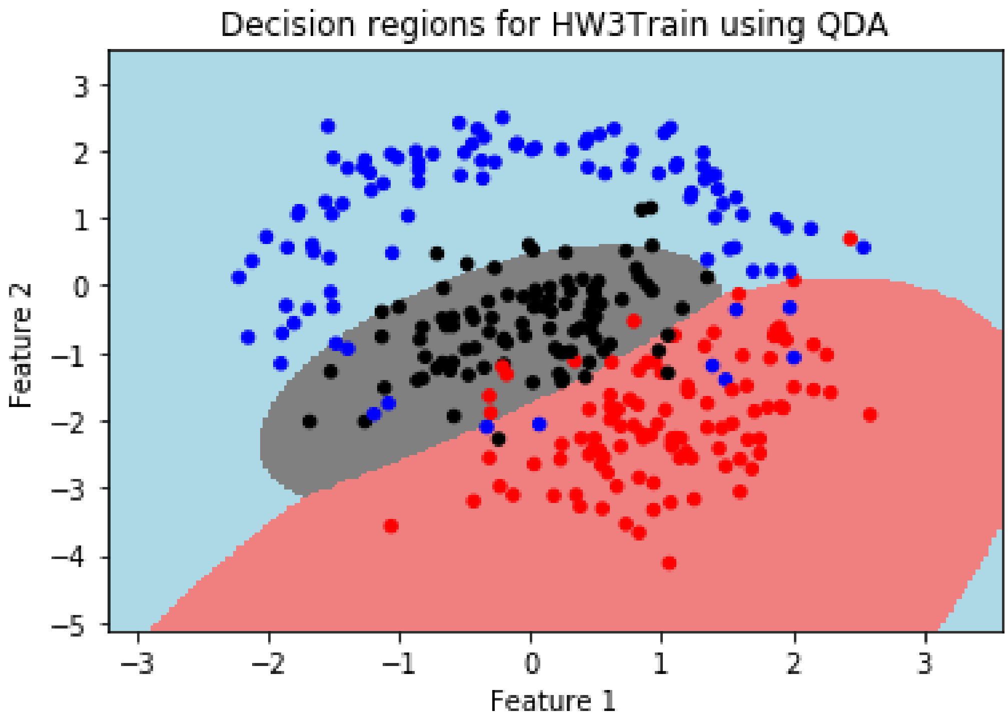


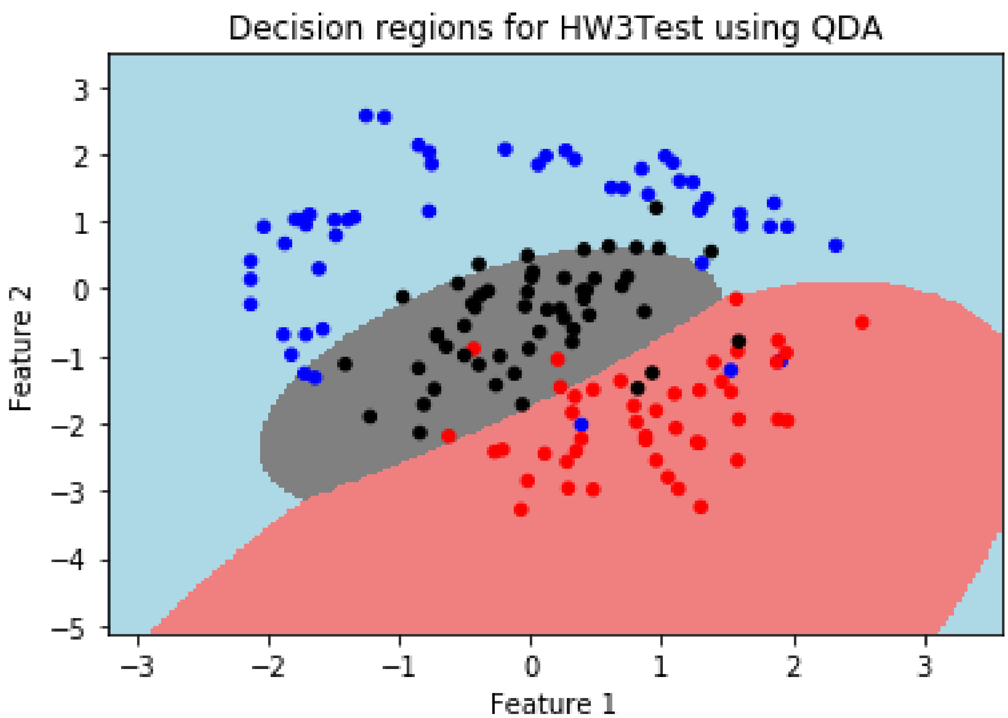


4.4 D. QDA

[376]:
```
qda1 = qda(priors=None,reg_param=0.0)
fit_qda = qda1.fit(x_train, y_train)
Z = fit_qda.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])
print('For QDA: training_accuracy = %.2f and testing_accuracy = %.2f'
      %(qda1.score(x_train, y_train)*100, qda1.score(x_test, y_test)*100))

plot_func(x_train, y_train, Z, title = 'Decision regions for HW3Train using␣
 ↪QDA')
plot_func(x_test, y_test, Z, title = 'Decision regions for HW3Test using QDA')
```

```
For QDA: training_accuracy = 89.67 and testing_accuracy = 86.67
```


Decision regions for HW3Train using QDA
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3


Decision regions for HW3Test using QDA
Feature 2
Feature 1
3
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3

4.5 E.

LDA generates linear boundary among different groups (LDA plots) as a result it has the smoothest boundaries. QDA generates quadratic boundary among groups that is boundaries can be nonlinear, but it cannot generate any region for a group within another group's region like KNN for $k = 1$. However, KNN can create linear or non-linear boundary depending on the k value. For a given y, KNN in general (but not always) disjoint. However, LDA and QDA are connected.

4.6 F.

4.6.1 (1) - (2)

[377]:
```
def expand_grid(dictionary):
   return pd.DataFrame([row for row in product(*dictionary.values())],
                       columns=dictionary.keys())

Cvals=np.logspace(-4,2,25,base=10)
degree = [1,2,3,4]

dictionary = {'c': Cvals,
              'degree': degree}

prem1 = expand_grid(dictionary)
prem1['train_accuracy'] = np.NaN
prem1['test_accuracy'] = np.NaN

size_prem1 = prem1.shape[0]

gamma=1.0
max_iter=1000
coef0=1.0

for i in range(0,size_prem1):
    clf=SVC(C=prem1.iloc[i,0], kernel='poly', degree=prem1.iloc[i,1],␣
 ↪gamma=gamma,
            coef0=coef0, shrinking=True, probability=False, max_iter=max_iter)
    fit_svm = clf.fit(x_train, y_train)
    prem1.iloc[i, 2:4] = [fit_svm.score(x_train, y_train)*100, fit_svm.
 ↪score(x_test, y_test)*100]
    sys.stdout.write("\r Progress: %.2f%%" %round(float(i+1)/size_prem1*100,2))
    sys.stdout.flush()

idx = prem1.groupby(['degree'])['test_accuracy'].transform(max) ==␣
 ↪prem1['test_accuracy']
idx2 = prem1[idx].groupby(['degree'])['train_accuracy'].transform(max) ==␣
 ↪prem1[idx]['train_accuracy']
idx3 = prem1[idx][idx2].groupby(['degree'])['c'].transform(min) ==␣
 ↪prem1[idx][idx2]['c']
```

```
prem = prem1[idx][idx2][idx3]
prem = prem.sort_values(by=['degree'])

print('Best C for each Degree with train and test accuracy\n\n')
print(prem)
print('\n\nNote: Best C has been selected based on maximum test accuracy. If␣
↪there are more than ' +
      'one for each degree, then by maximum train accuracy, if still more than␣
↪one, then by minimum C')
```

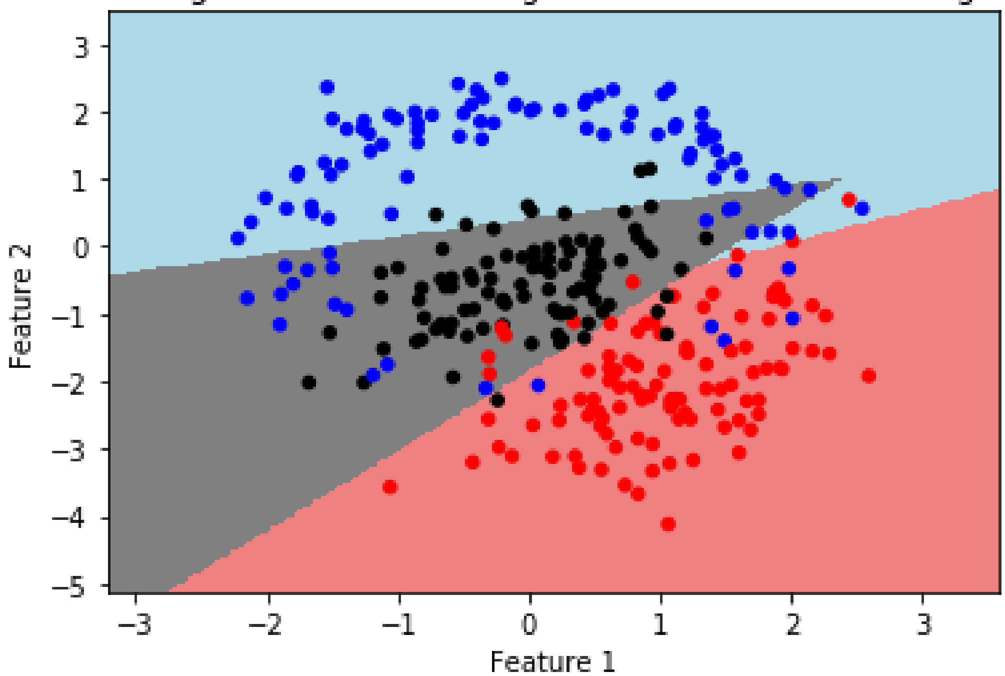
```
Progress: 100.00%Best C for each Degree with train and test accuracy


          c  degree  train_accuracy  test_accuracy
76  5.623413       1       85.333333      84.666667
49  0.100000       2       90.666667      92.000000
58  0.316228       3       92.333333      92.000000
47  0.056234       4       92.333333      90.666667


Note: Best C has been selected based on maximum test accuracy. If there are more
than one for each degree, then by maximum train accuracy, if still more than
one, then by minimum C
```

4.6.2 (3) - (4)

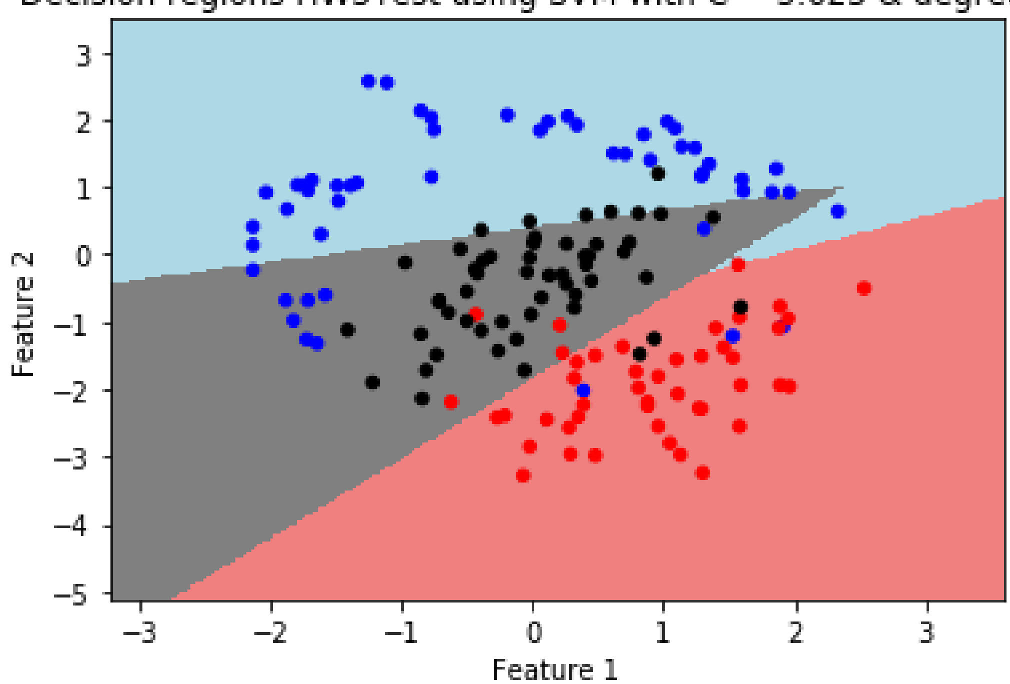
[378]:
```
for i in range(0, prem.shape[0]):
    clf=SVC(C=prem.iloc[i,0], kernel='poly', degree=prem.iloc[i,1], gamma=gamma,
            coef0=1.0, shrinking=True, probability=False, max_iter=max_iter)
    fit_svm = clf.fit(x_train, y_train)
    Z = fit_svm.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])
    print('For C = %.4f and degree = %d, Training_accuracy = %.2f%% and␣
↪Testing_accuracy = %.2f%%'
          %(prem.iloc[i,0], prem.iloc[i,1],
            fit_svm.score(x_train, y_train)*100,
            fit_svm.score(x_test, y_test)*100))
    plot_func(x_train, y_train, Z, title = 'Decision regions HW3Train using SVM␣
↪with C = ' +
              str(round(prem.iloc[i,0],3)) + ' & degree = ' +str(round(prem.
↪iloc[i,1],3)))
    plot_func(x_test, y_test, Z, title = 'Decision regions HW3Test using SVM␣
↪with C = ' +
              str(round(prem.iloc[i,0],3)) + ' & degree = ' +str(round(prem.
↪iloc[i,1],3)))
```

```
For C = 5.6234 and degree = 1, Training_accuracy = 85.33% and Testing_accuracy =
84.67%
```

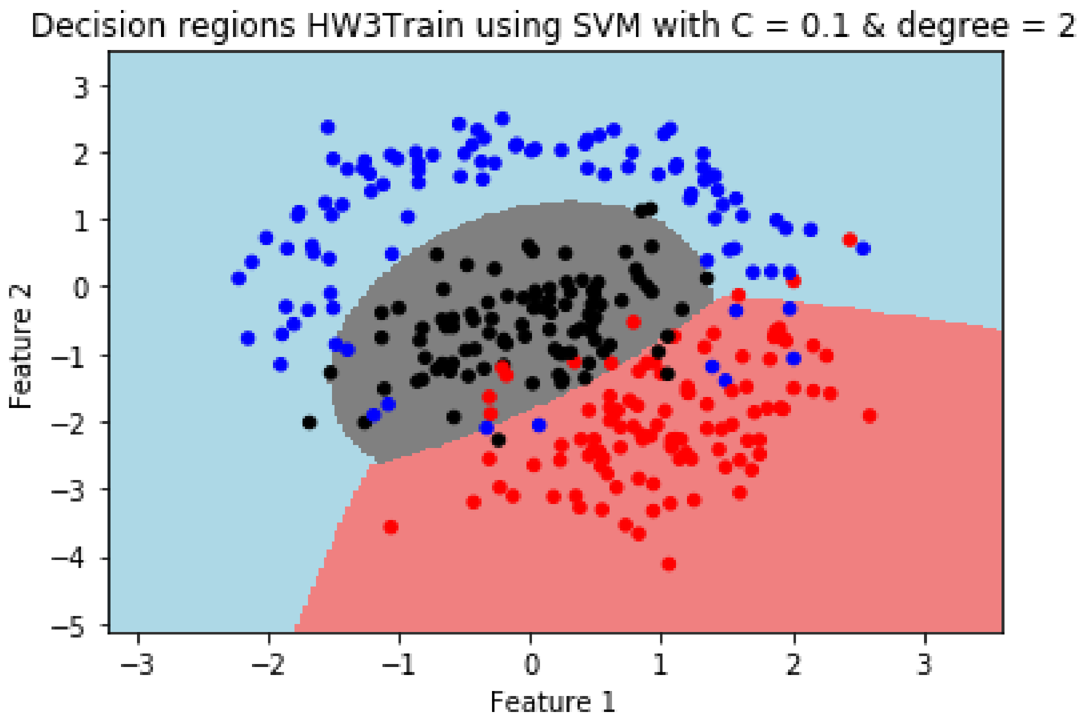
Decision regions HW3Train using SVM with C = 5.623 & degree = 1

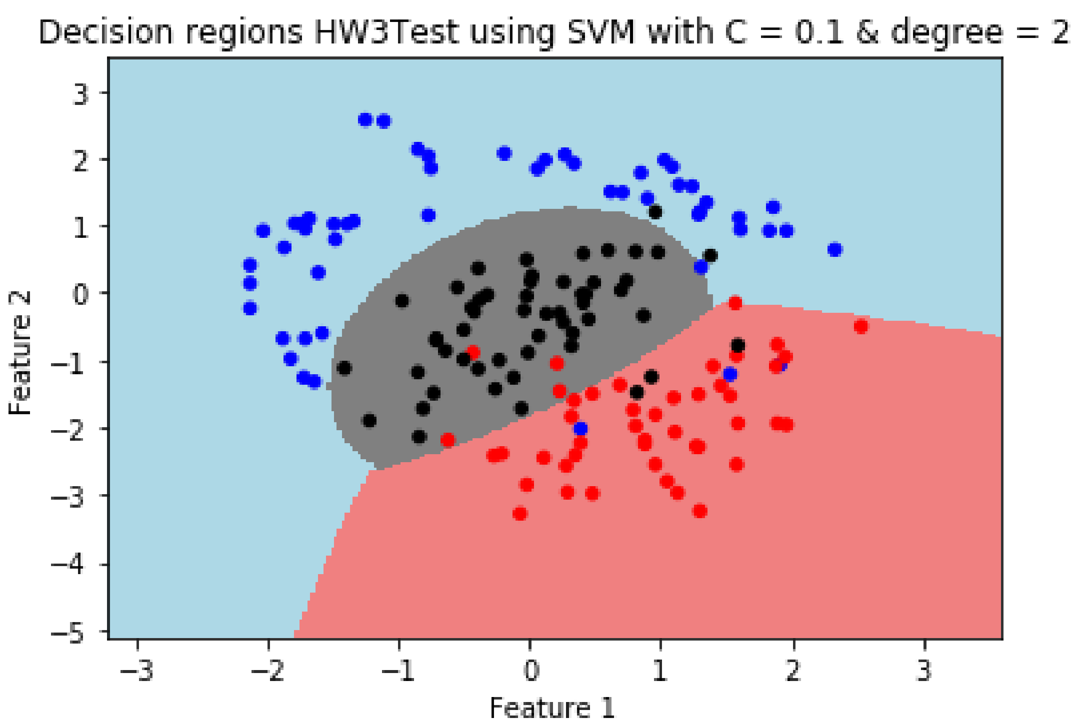


Decision regions HW3Test using SVM with C = 5.623 & degree = 1

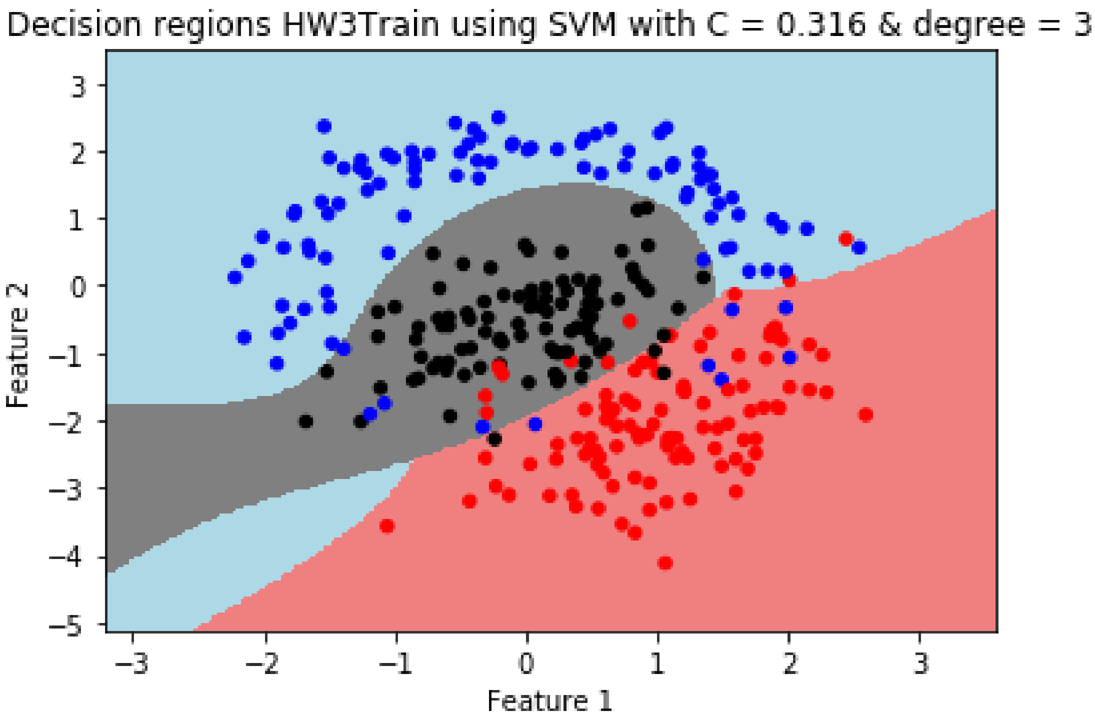


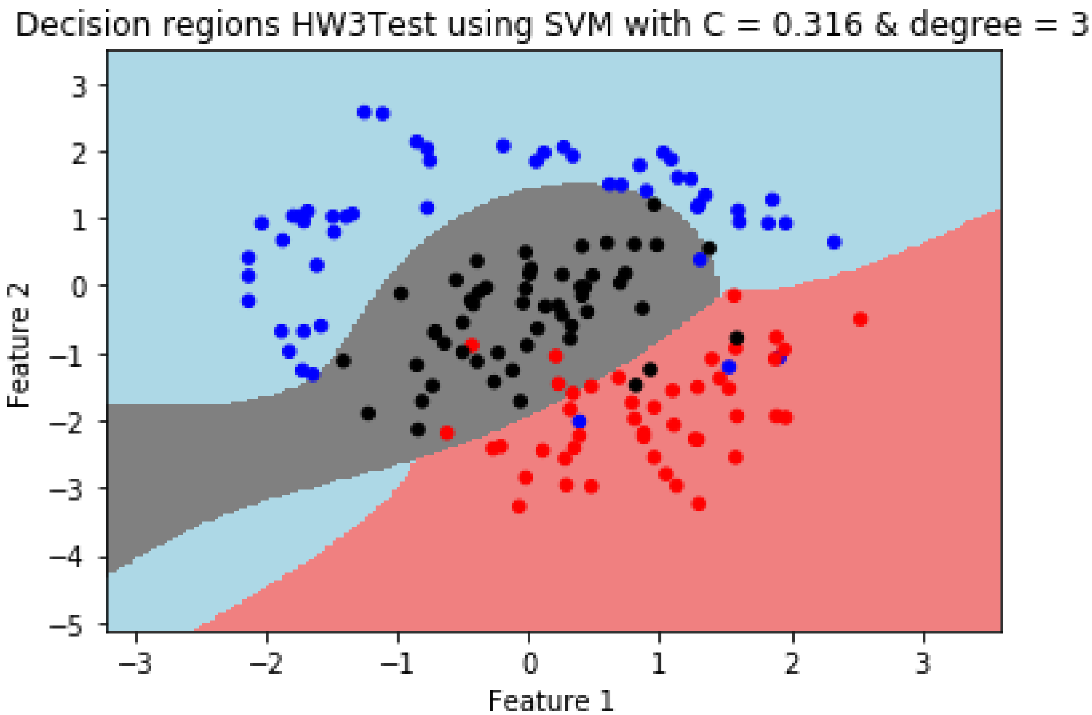
```
For C = 0.1000 and degree = 2, Training_accuracy = 90.67% and Testing_accuracy =
92.00%
```



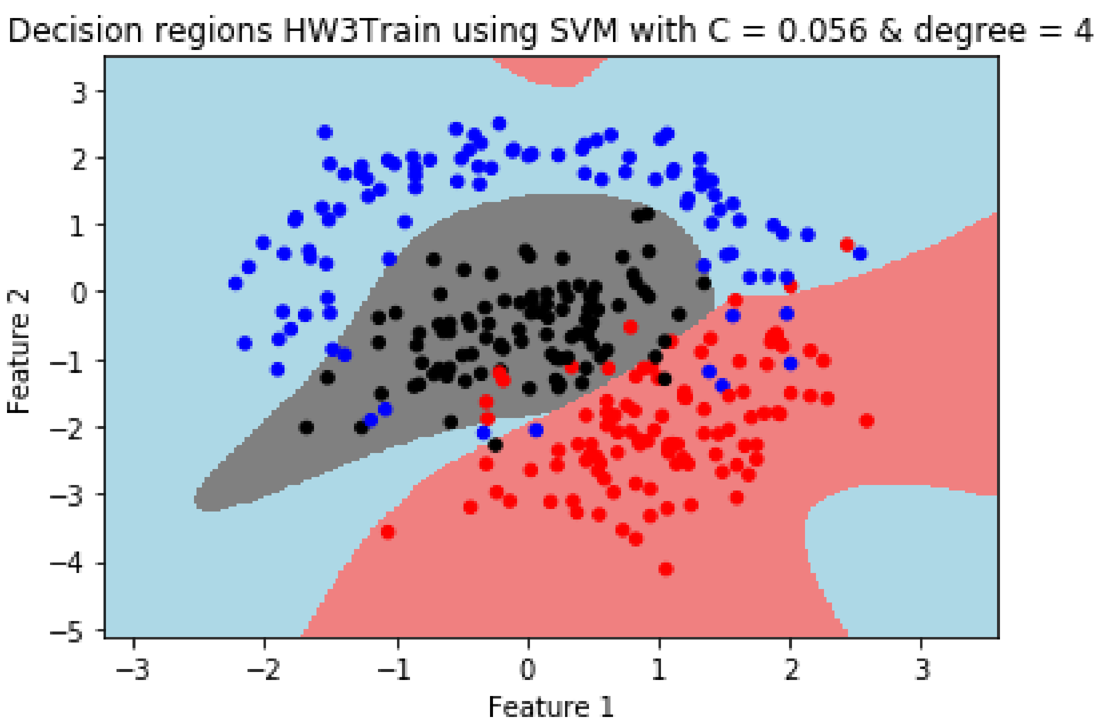


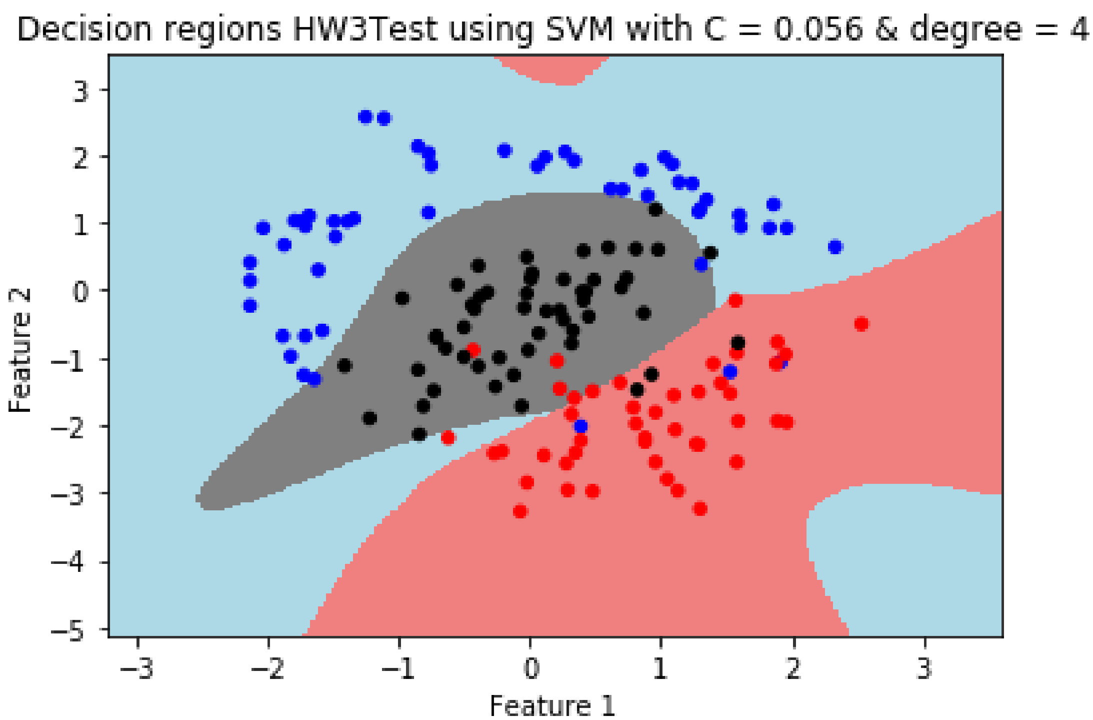
For C = 0.3162 and degree = 3, Training_accuracy = 92.33% and Testing_accuracy = 92.00%





For C = 0.0562 and degree = 4, Training_accuracy = 92.33% and Testing_accuracy = 90.67%





4.7 G.

4.7.1 (1) - (2)

[379]:

```
Cvals=np.logspace(-4,2,25,base=10)
gamma_vals=np.logspace(-2,2,25,base=10)

dictionary = {'c': Cvals,
              'gamma': gamma_vals}

prem1 = expand_grid(dictionary)
prem1['train_accuracy'] = np.NaN
prem1['test_accuracy'] = np.NaN

size_prem1 = prem1.shape[0]

max_iter=1000

for i in range(0,size_prem1):
    clf=clf=SVC(C=prem1.iloc[i,0], kernel='rbf', gamma=prem1.iloc[i,1],
                shrinking=True, probability=False, max_iter=max_iter)
```

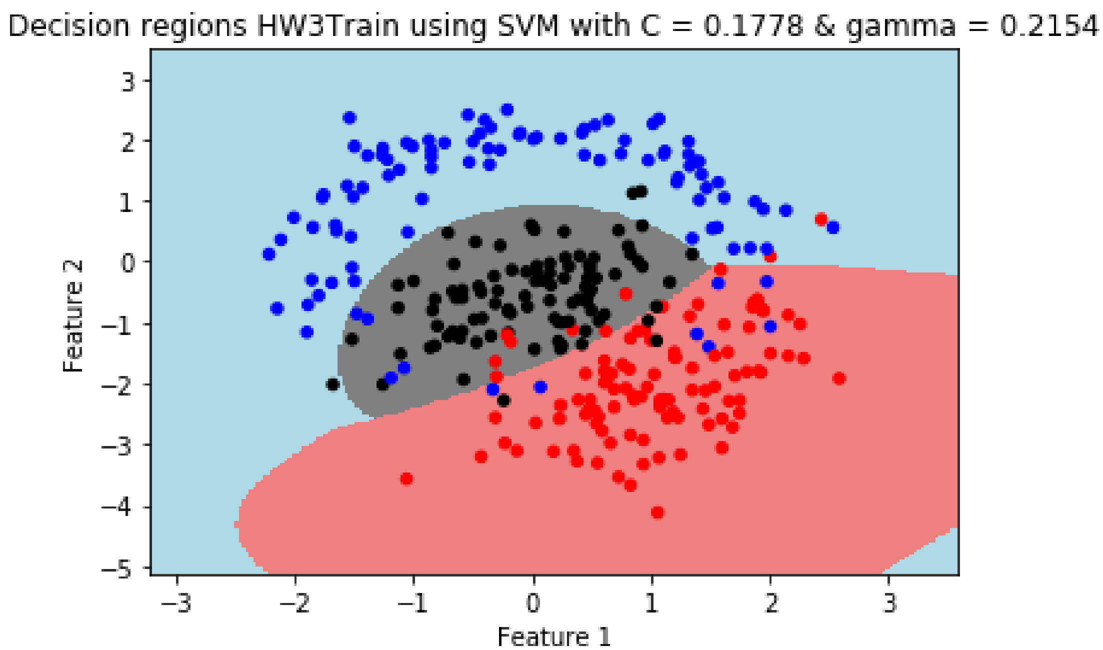
```
    fit_svm = clf.fit(x_train, y_train)
    prem1.iloc[i, 2:4] = [fit_svm.score(x_train, y_train)*100, fit_svm.
 ↪score(x_test, y_test)*100]
    sys.stdout.write("\r Progress: %.2f%%" %round(float(i+1)/size_prem1*100,2))
    sys.stdout.flush()


prem = prem1.iloc[333,:]
print('Best C and gamma pair with train and test accuracy\n\n')
print(prem)
print('\n\nNote: Best C and gamma pair has been selected based on maximum test␣
 ↪accuracy.')
```

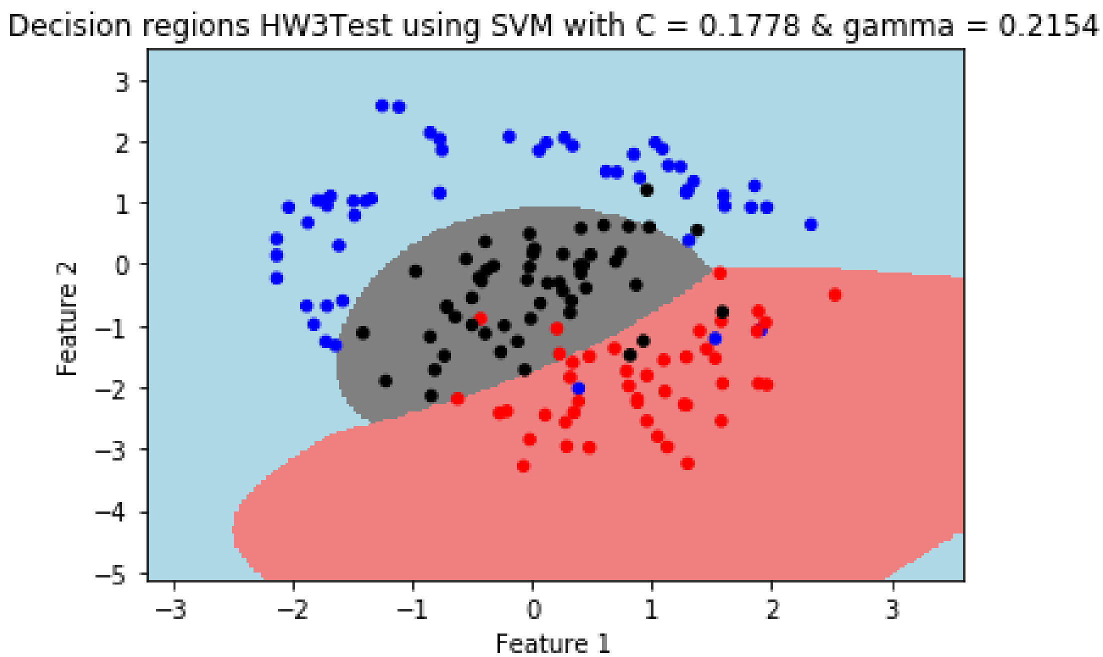
```
 Progress: 100.00%Best C and gamma pair with train and test accuracy


c                    0.177828
gamma                0.215443
train_accuracy      91.000000
test_accuracy       92.666667
Name: 333, dtype: float64


Note: Best C and gamma pair has been selected based on maximum test accuracy.
```

4.7.2 (3) - (4)

[380]:
```
clf=SVC(C=prem[0], kernel='rbf', gamma=prem[1],
                shrinking=True, probability=False, max_iter=max_iter)
fit_svm = clf.fit(x_train, y_train)
Z = fit_svm.predict(np.c_[x1mesh.ravel(), x2mesh.ravel()])
print('For C = %.4f and gamma = %.4f, Training_accuracy = %.2f%% and␣
 ↪Testing_accuracy = %.2f%%'
      %(prem[0], prem[1],
        fit_svm.score(x_train, y_train)*100,
        fit_svm.score(x_test, y_test)*100))
plot_func(x_train, y_train, Z, title = 'Decision regions HW3Train using SVM␣
 ↪with C = ' +
          str(round(prem[0],4)) + ' & gamma = ' +str(round(prem[1],4)))
plot_func(x_test, y_test, Z, title = 'Decision regions HW3Test using SVM with C␣
 ↪= ' +
          str(round(prem[0],4)) + ' & gamma = ' +str(round(prem[1],4)))
```

```
For C = 0.1778 and gamma = 0.2154, Training_accuracy = 91.00% and
Testing_accuracy = 92.67%
```





4.8 H.

Using SVM model, it is possible to find linear (polynomial kernel) and non-linear (polynomial with higher order and RBF kernel) data pattern that is not possible for LDA (linear separation) and

QDA (quadratic). SVM finds comparatively smooth pattern than KNN.

In general, SVM should best capture the shapes of the data. Though, it may seem KNN with low k values can capture any shape of the data, but it generally overestimates data. And for some certain data structure, it may be difficult to find best KNN model with moderately high values. On the other hand, SVM can capture linear and non-linear data pattern. However, there is no model that can outperform for all data.