svm

November 16, 2021

1 LINEAR SVM

Adam Carriker

[1]:
```
import sys
# !conda install --yes --prefix {sys.prefix} graphviz
# !conda install --yes --prefix {sys.prefix}  dl8.5
# !conda install --yes --prefix {sys.prefix}  chefboost
# !conda install --yes --prefix {sys.prefix}  sklearn
# !{sys.executable} -m pip install libsvm
```

[2]:
```
import sys
# !conda install --yes --prefix {sys.prefix} pandas
# !{sys.executable} -m pip install sklearn
```

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as io
import libsvm
import pandas as pd
from libsvm.svmutil import *
from pprint import pprint

%matplotlib inline
```

[4]:
```
debug_mode = False
```

1.1 3 Coding

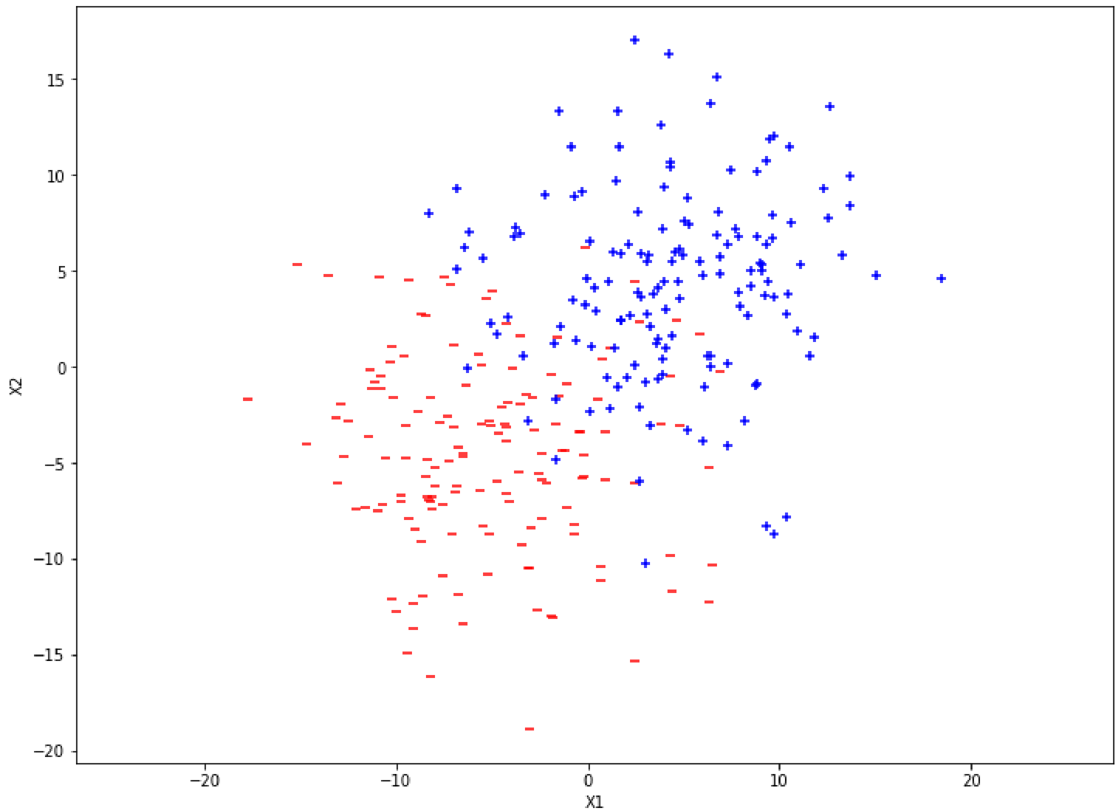
1.1.1 A. Toy Dataset

1.1.2 3.1

Generate a training set of size 300 with two-dimensional features (X) drawn at random with 150 positive and 150 negative examples as follows: • Xneg N ([-5, -5], $25 \cdot$ I2) and corresponds to negative labels (-1) • Xpos N ([5, 5], $25 \cdot$ I2) and corresponds to positive labels $(+1)$ Accordingly, X = [Xneg, Xpos] is a 300×2 array, and Y is a 300×1 array of values $\{-1, 1\}$. Here I2 represents 2×2 identity matrix.

[5]:

```
# Generate binary class dataset
random_seed = 0
np.random.seed(random_seed)

n_samples = 300
center_1 = [-5, -5]
center_2 = [5, 5]

# Generate Data:
Xneg = np.random.multivariate_normal([-5, -5], [[25, 0], [0, 25]], 150).T
Xpos = np.random.multivariate_normal([5, 5], [[25, 0], [0, 25]], 150).T
figure, axes = plt.subplots()
figure.set_size_inches(12, 9)
X = np.swapaxes(np.append(Xneg, Xpos,axis=1),0,1)
Y = np.array([-1]*150+[1]*150)
XY = np.array([[X[i][0],X[i][1],Y[i]] for i in range(0,len(X))])

if debug_mode:
    print(X)

# Scatter plot:
plt.scatter([i[0] for i in X[:150]],[i[1] for i in X[:150]], color='red',␣
 ↪marker='_')
plt.scatter([i[0] for i in X[150:]],[i[1] for i in X[150:]], color='blue',␣
 ↪marker="+")
plt.xlabel("X1")
plt.ylabel("X2")
plt.axis('equal')
plt.show()
```



1.1.3 3.1.2

Randomly sample 9/10ths of the toy data as the training set, and the other 1/10th of data as the test set.

[6]:
```
from sklearn.model_selection import train_test_split

XYdf = pd.DataFrame(XY,columns=['X1','X2','Y'])

# X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.1,␣
↪random_state=random_seed, shuffle=True)
train,test = train_test_split(XYdf, test_size=0.1, random_state=random_seed,␣
↪shuffle=True)



if debug_mode:
    print("Test\n",test)
    print("\nTrain\n",train)
```

1.1.4 3.1.3

Train an SVM classifier with the radial basis kernel on the toy dataset for a reasonable value for . Make a contour plot of the decision function f and plot the decision boundary, on top of the scatter plot of the data you made earlier, marking the test points in a different color than the training points (perhaps gray for the test points and black for the training points). Mark the support vectors separately (e.g., a circle around the point, or make its symbol bold or larger). Also please report the accuracy on the training and test sets.

[7]:
```
def print_model(m):
  svm_type = m.get_svm_type()
  nr_class = m.get_nr_class()
  svr_probability = m.get_svr_probability()
  class_labels = m.get_labels()
  sv_indices = m.get_sv_indices()
  nr_sv = m.get_nr_sv()
  is_prob_model = m.is_probability_model()
  sv_coefficients = m.get_sv_coef()
  svs = m.get_SV()

  print("svm_type:", svm_type)
  print("nr_class:", nr_class)
  print("svr_probability:", svr_probability)
  print("labels:", class_labels)
  print("sv_indices:", sv_indices)
  print("nr_sv:", nr_sv)
  print("probability_model:",is_prob_model)
  print("SVs (10):")
  pprint(svs[:10])
  print("SV coefficients, SVs (10):")
  pprint(list(zip(sv_coefficients[:10],svs[:10])))
```

[8]:
```
# Define the SVM problem
p = svm_problem(train['Y'].to_numpy(), train[['X1','X2']].to_numpy())

# Define the hyperparameters
sigma = 1/4
h = svm_parameter(f'-s 0 -t 2 -g {sigma} -q')

# Train the model
m = svm_train(p,h)

# Print Information about Model
if(debug_mode):
    print_model(m)

# Predict with model on train dataset
print("Training Accuracy:")
```

```
p_label_train, p_acc_train, p_val_train = svm_predict(train['Y'].to_numpy(),␣
↪train[['X1','X2']].to_numpy(), m)

# Predict with model on test dataset
print("Testing Accuracy:")
p_label, p_acc, p_val = svm_predict(test['Y'].to_numpy(), test[['X1','X2']].
↪to_numpy(), m)
```

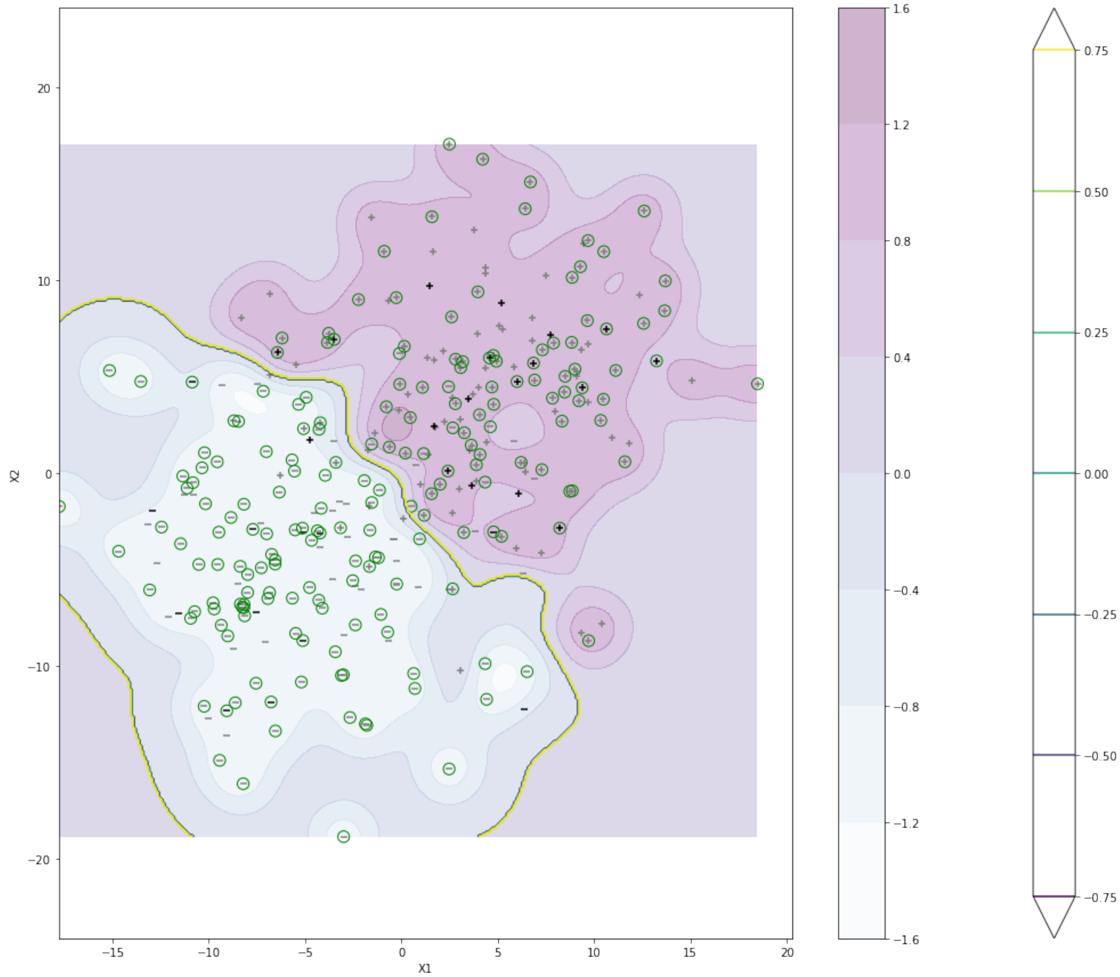
```
Training Accuracy:
Accuracy = 91.1111% (246/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```

[9]:
```
def draw_Contour_Plot(m):
    # Make grids for contour plot
    [max1, max2] = np.amax(X, axis=0)
    [min1, min2] = np.amin(X, axis=0)

    x1_scale = np.arange(min1, max1, 0.1)
    x2_scale = np.arange(min2, max2, 0.1)

    x_grid, y_grid = np.meshgrid(x1_scale, x2_scale)

    # flatten each grid to a vector
    x_g, y_g = x_grid.flatten(), y_grid.flatten()
    x_g, y_g = x_g.reshape((len(x_g), 1)), y_g.reshape((len(y_g), 1))

    grid = np.hstack((x_g, y_g))

    # predict with model on the grid
    p_label_grid, p_acc_grid, p_val_grid = svm_predict([],grid, m, options="-q")

    # reshape the boundary vector
    boundary = np.array(p_label_grid)
    boundary_grid = boundary.reshape(x_grid.shape)

    # plot the boundary grid of x, y and z values as a surface
    figure, axes = plt.subplots()
    figure.set_size_inches(18, 15)
    surface = plt.contour(x_grid, y_grid, boundary_grid, extend='both')
    plt.colorbar(surface)

    # reshape the probability vector
    p_pred = np.array(p_val_grid)
    p_pred = p_pred[:, 0]
    pp_grid = p_pred.reshape(x_grid.shape)
```

```
    # plot the probability grid of x, y and z values as a surface
    surface = plt.contourf(x_grid, y_grid, pp_grid, cmap='BuPu', alpha=0.3)
    plt.colorbar(surface)

    # create scatter plot for samples from each class
    train_plus = train.loc[train['Y'] == 1].to_numpy().tolist()
    train_neg = train.loc[train['Y'] == -1].to_numpy().tolist()
    test_plus = test.loc[test['Y'] == 1].to_numpy().tolist()
    test_neg = test.loc[test['Y'] == -1].to_numpy().tolist()
    plt.scatter([i[0] for i in train_plus],[i[1] for i in train_plus],␣
↪color='gray', marker='+')
    plt.scatter([i[0] for i in test_plus],[i[1] for i in test_plus],␣
↪color='black', marker='+')
    plt.scatter([i[0] for i in train_neg],[i[1] for i in train_neg],␣
↪color='gray', marker='_')
    plt.scatter([i[0] for i in test_neg],[i[1] for i in test_neg],␣
↪color='black', marker='_')
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.axis('equal')

    # show support vectors with green circle
    for sv_idx in m.get_sv_indices():
      circle = plt.Circle((X[sv_idx -1, 0], X[sv_idx -1, 1]), 0.3,␣
↪color='green', fill=False)
      axes.set_aspect(1)
      axes.add_artist(circle)
    plt.show()
```

[10]:

```
draw_Contour_Plot(m)
```



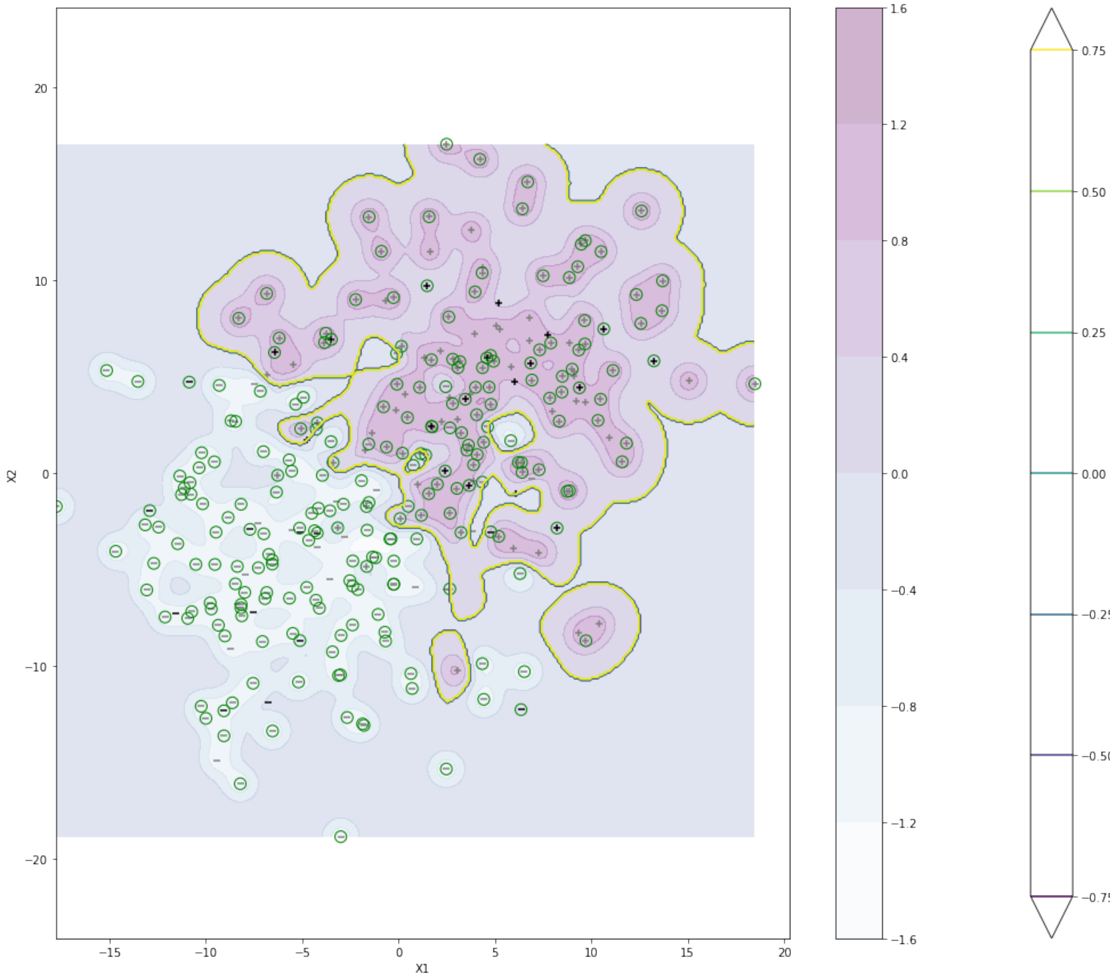
1.1.5 3.1.4

Repeat the above step a few times on separate figures: make the same plot for 6 different 2 values (1, 2, 4, 8, 32, 128). Also please report the accuracy on the training and test sets. On a separate figure, plot the number of support vectors vs. 2 (plot the horizontal axis on a log scale).

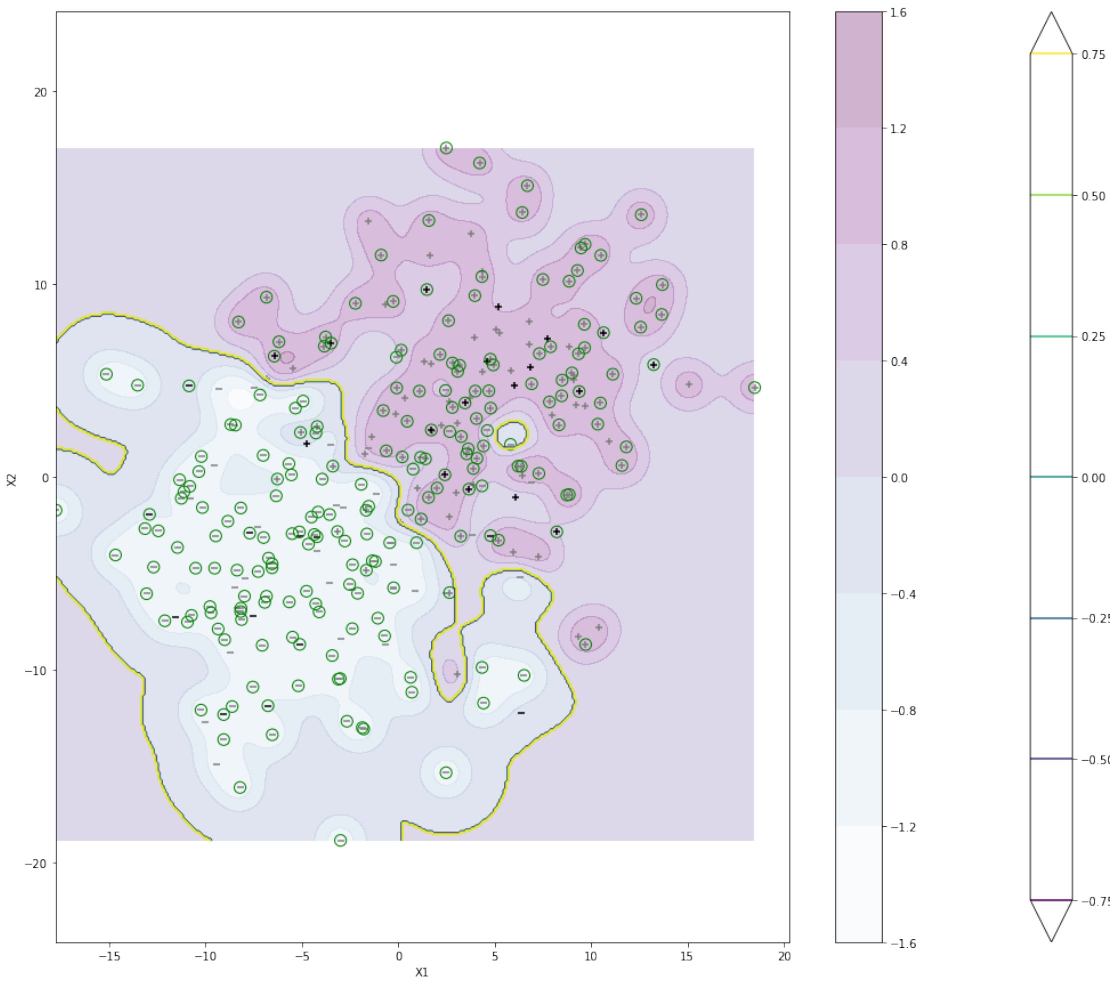
[11]:
```python
import math

sigmas_squared =[1, 2, 4, 8, 32, 128]
# sigmas = [math.sqrt(sigma) for sigma in sigmas_squared]
support_vector_count = []
for sigma_squared in sigmas_squared:

    print(f"\n\nsigma^2 = {sigma_squared}")

    # Define the hyperparameters
    h = svm_parameter(f'-s 0 -t 2 -g {1/sigma_squared} -q')
```

```
    # Train the model
    m = svm_train(p,h)

    # Record number of support vectors
    support_vector_count.append(m.get_nr_sv())

    # Print Information about Model
    if(debug_mode):
        print_model(m)

    # Predict with model on train dataset
    print("Training Accuracy:")
    p_label_train, p_acc_train, p_val_train = svm_predict(train['Y'].
↪to_numpy(), train[['X1','X2']].to_numpy(), m)

    # Predict with model on test dataset
    print("Testing Accuracy:")
    p_label, p_acc, p_val = svm_predict(test['Y'].to_numpy(), test[['X1','X2']].
↪to_numpy(), m)

    draw_Contour_Plot(m)
```

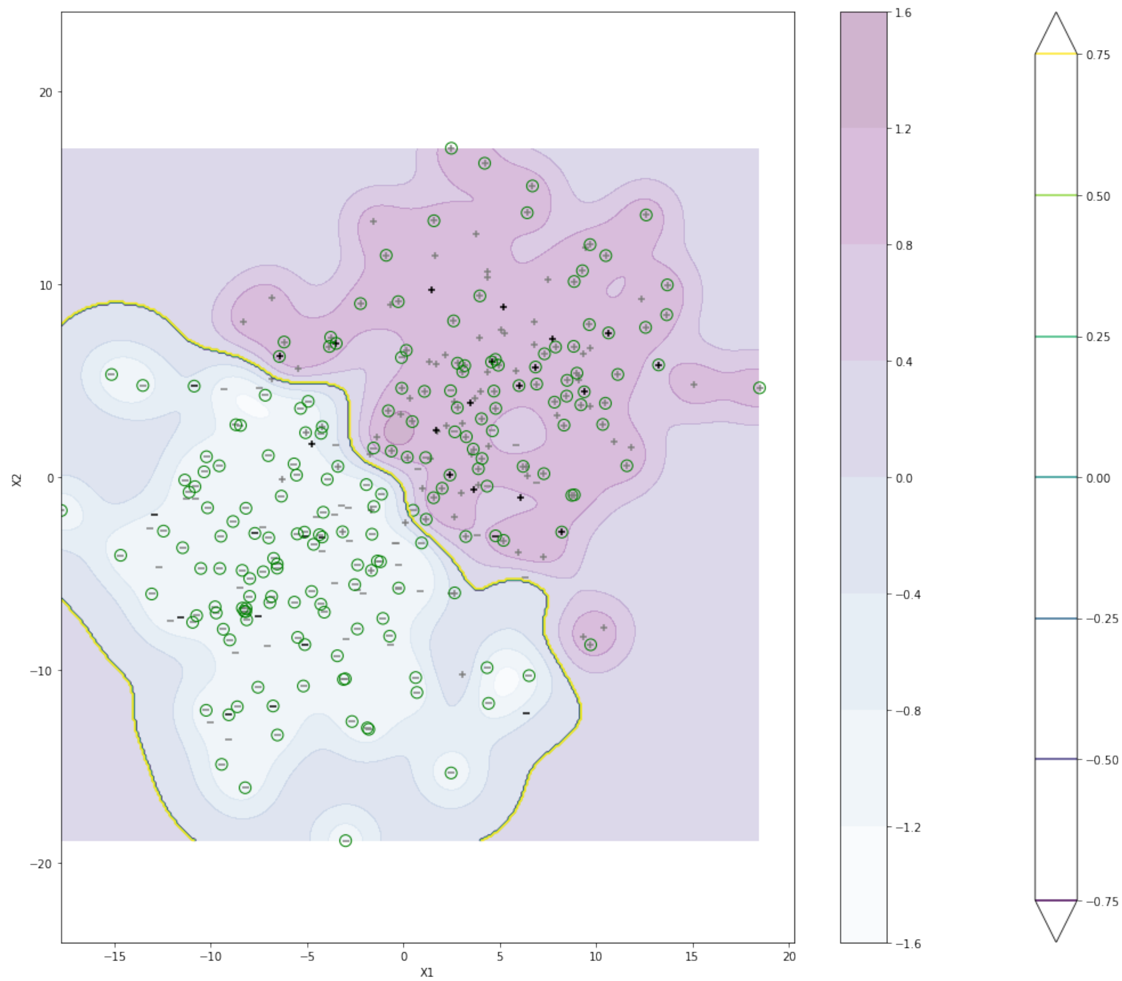
```
sigma^2 = 1
Training Accuracy:
Accuracy = 95.1852% (257/270) (classification)
Testing Accuracy:
Accuracy = 96.6667% (29/30) (classification)
```



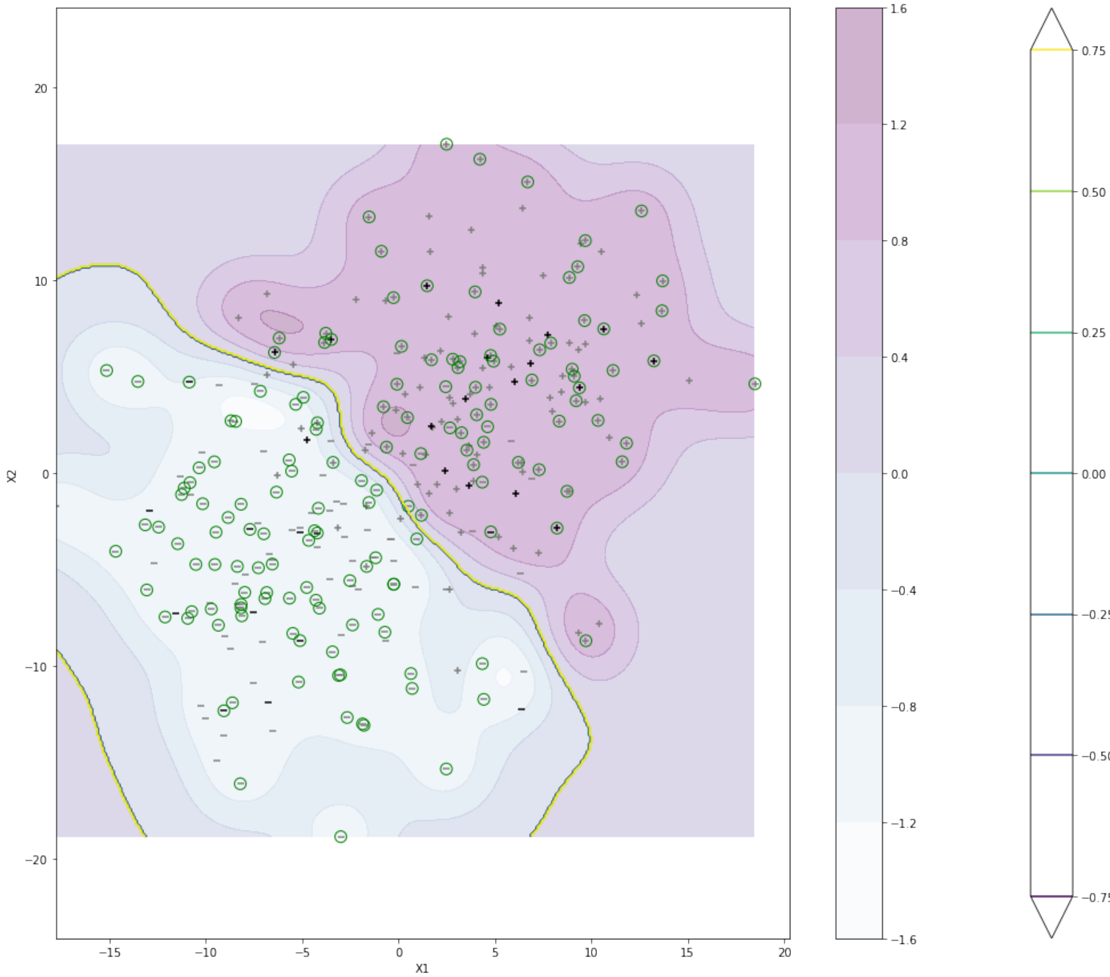

```
sigma^2 = 2
Training Accuracy:
Accuracy = 92.963% (251/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```



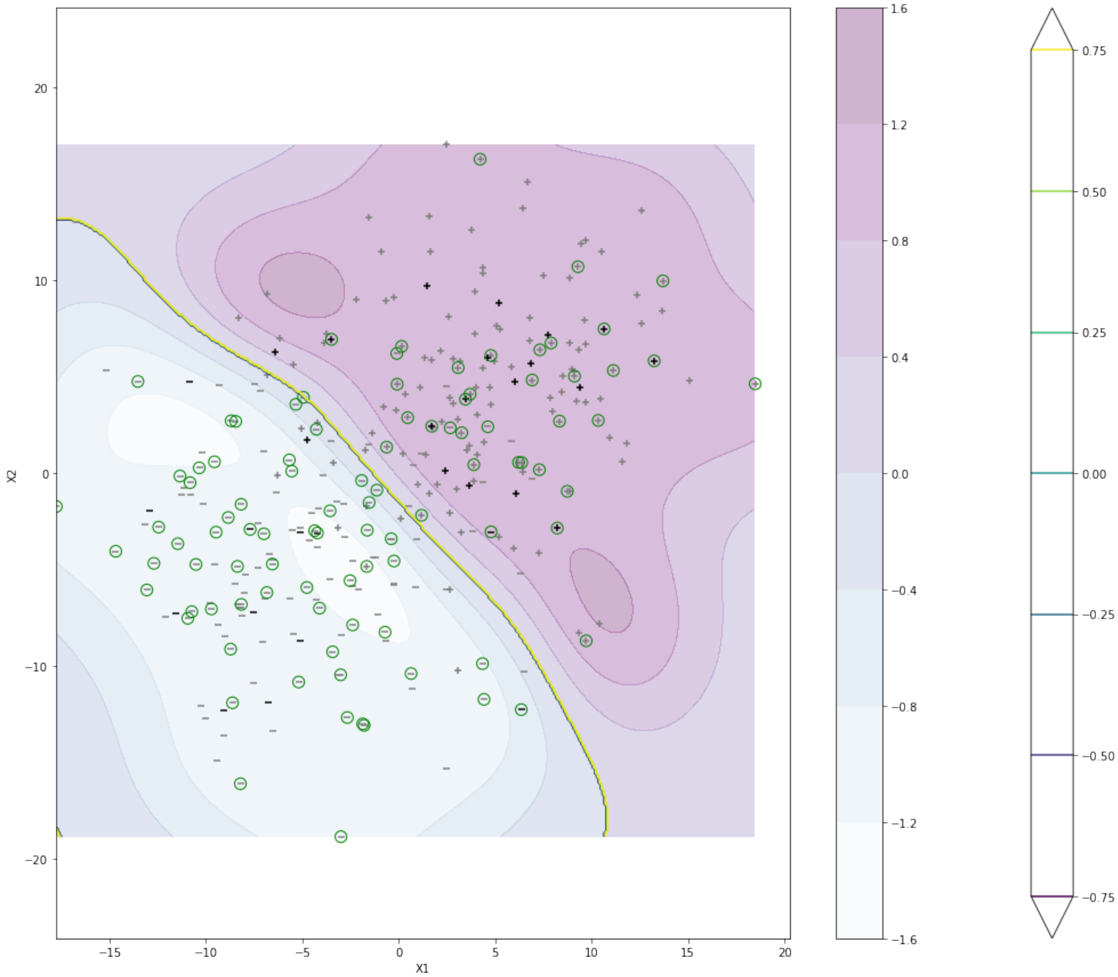

```
sigma^2 = 4
Training Accuracy:
Accuracy = 91.1111% (246/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```



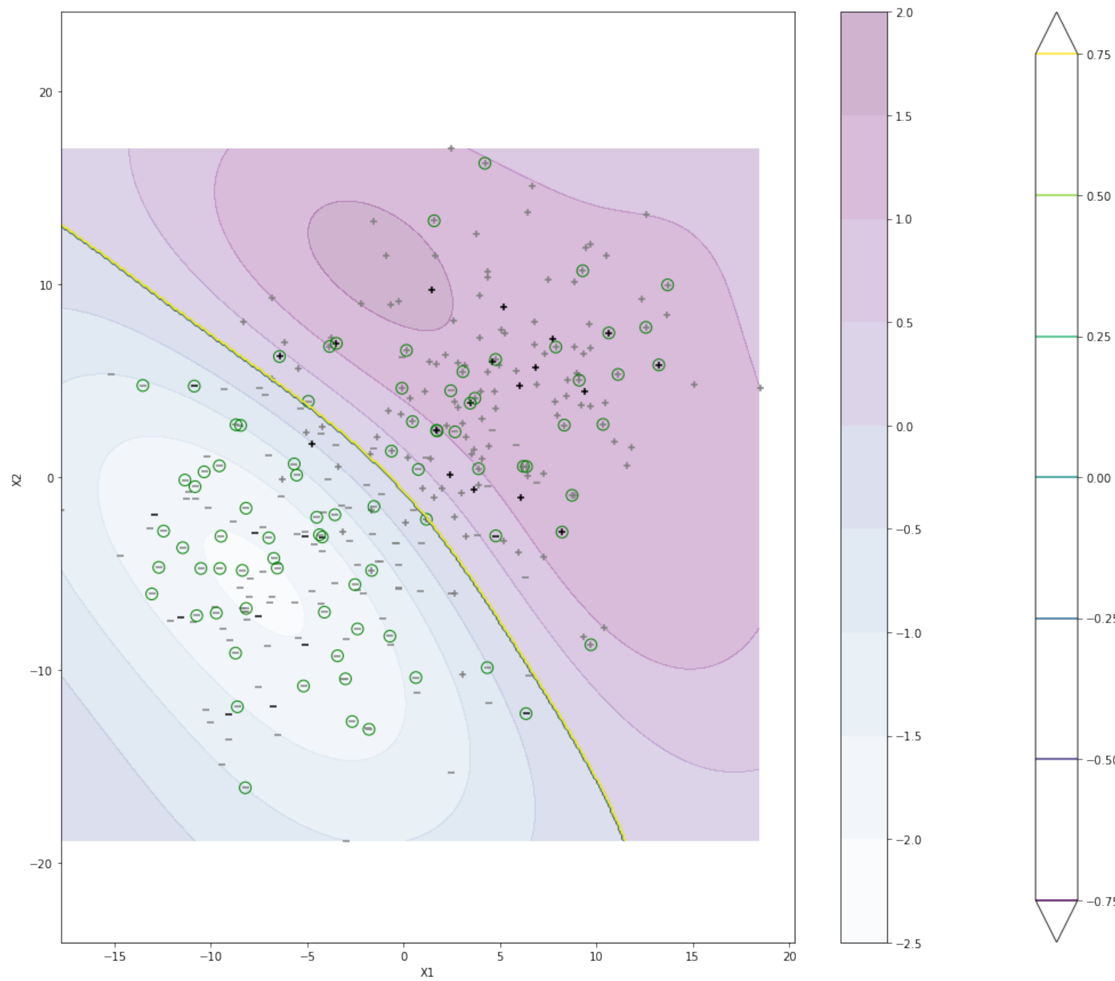

```
sigma^2 = 8
Training Accuracy:
Accuracy = 91.1111% (246/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```



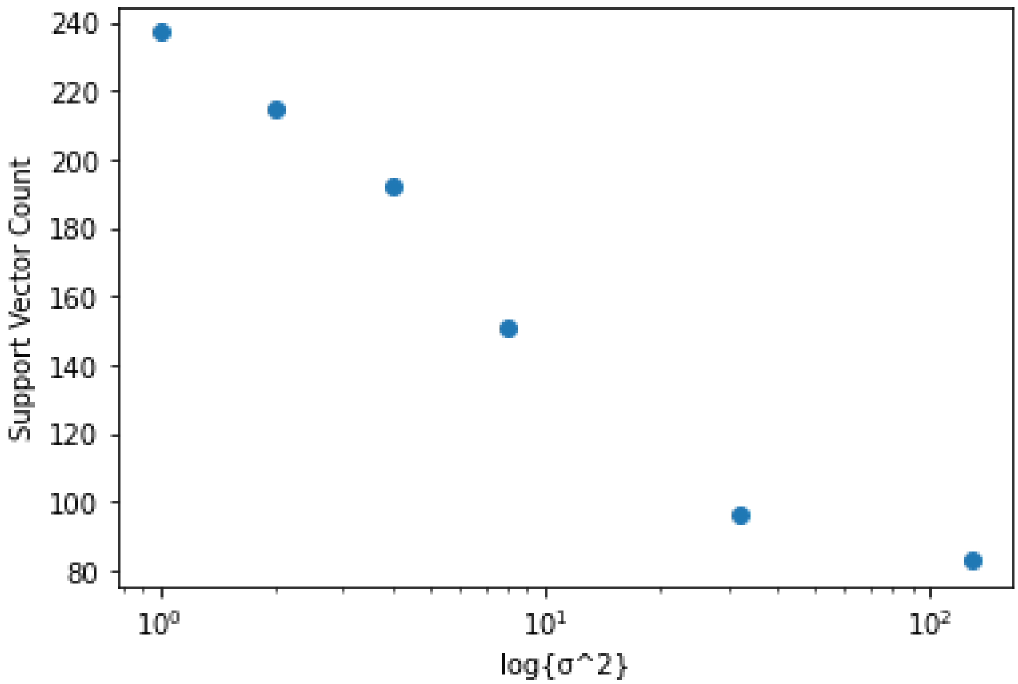

```
sigma^2 = 32
Training Accuracy:
Accuracy = 91.1111% (246/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```




```
sigma^2 = 128
Training Accuracy:
Accuracy = 91.1111% (246/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```




```
[12]: # Plot number of support vectors vs sigma^2

figure, axes = plt.subplots()
axes.set_xscale('log')
plt.scatter(sigmas_squared, support_vector_count)
plt.xlabel("log{ ^2}")
plt.ylabel("Support Vector Count")
plt.show()
```



1.1.6 3.1.5

Report any patterns you find with underfitting or overfitting, as a function of sigma^2.

Overfitting seems to occur when sigma^2 is small. This can be seen in the first three plots (with sigma^2 = [1,2,4], respectivley) where we have many "islands" formed by the decision boundary, indicating the function crosses the f(x)=0 boundary many times within clusters of the data where we wouldn't expect it to generally. As we increase sigma^2 to values [8,32,128], we see that these islands disappear, and the decision boundary smoothes out to becomre more of a regular oval shape. This indicates we are getting towards overfitting as sigma^2 increases. This evidence is also supported by the trend of the number of support vectors decreasing as sigma^2 increases.

1.1.7 3.1.6

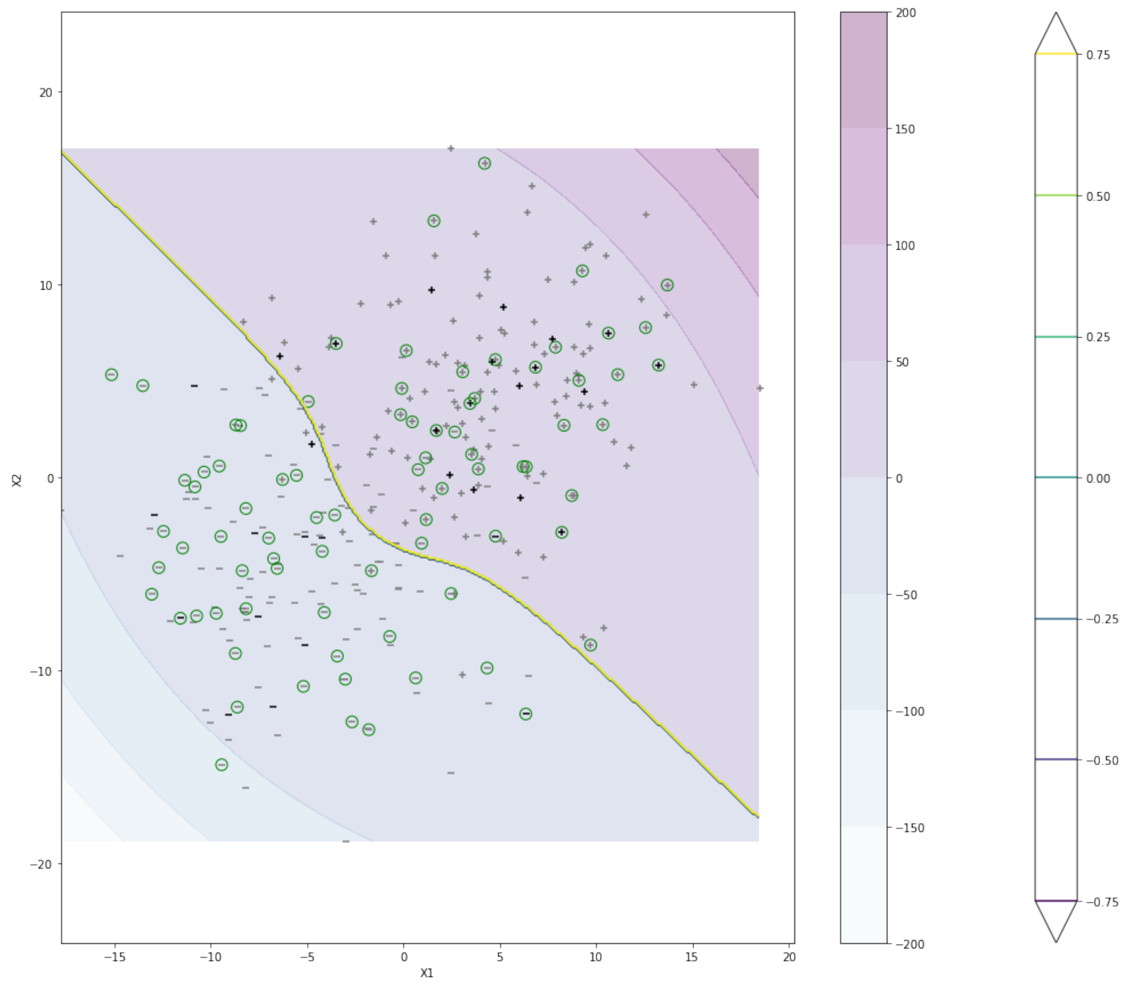
Train an SVM classifier with a polynomial kernel of degree of 3 (you may use the default parameter from LIBSVM). Again draw the same plot as before, with the data, contour of f, and decision boundary, highlighting the support vectors. Also please report the accuracy on the training and test sets.

[13]:
```
# Define the SVM problem
p = svm_problem(train['Y'].to_numpy(), train[['X1','X2']].to_numpy())

# Define the hyperparameters (polynomial of degree 3 instead of Gaussian this␣
↪time)
```

```
sigma = 1/4
h = svm_parameter('-s 0 -t 1 -d 3 -q')

# Train the model
m = svm_train(p,h)

# Print Information about Model
if(debug_mode):
    print_model(m)

# Predict with model on train dataset
print("Training Accuracy:")
p_label_train, p_acc_train, p_val_train = svm_predict(train['Y'].to_numpy(),␣
↪train[['X1','X2']].to_numpy(), m)

# Predict with model on test dataset
print("Testing Accuracy:")
p_label, p_acc, p_val = svm_predict(test['Y'].to_numpy(), test[['X1','X2']].
↪to_numpy(), m)
```

```
Training Accuracy:
Accuracy = 89.2593% (241/270) (classification)
Testing Accuracy:
Accuracy = 93.3333% (28/30) (classification)
```

[14]:
```
draw_Contour_Plot(m)
```



1.1.8 B. Credit Card Dataset

1.1.9 3.1

Train an SVM classifier with the kernel function k(x, z) = x>z on 9/10ths of the credit card data set. What is the accuracy of this classifier on the test data set? Show the ROC curves, and also report the AUC.

[15]:
```
# load data, split into test & train
df = pd.read_csv('creditCard.csv')

if debug_mode: print(df.info())

train,test = train_test_split(df, test_size=0.1, random_state=random_seed,␣
↪shuffle=True)

if debug_mode:
```

```
    print("Test\n",test)
    print("\nTrain\n",train)
```

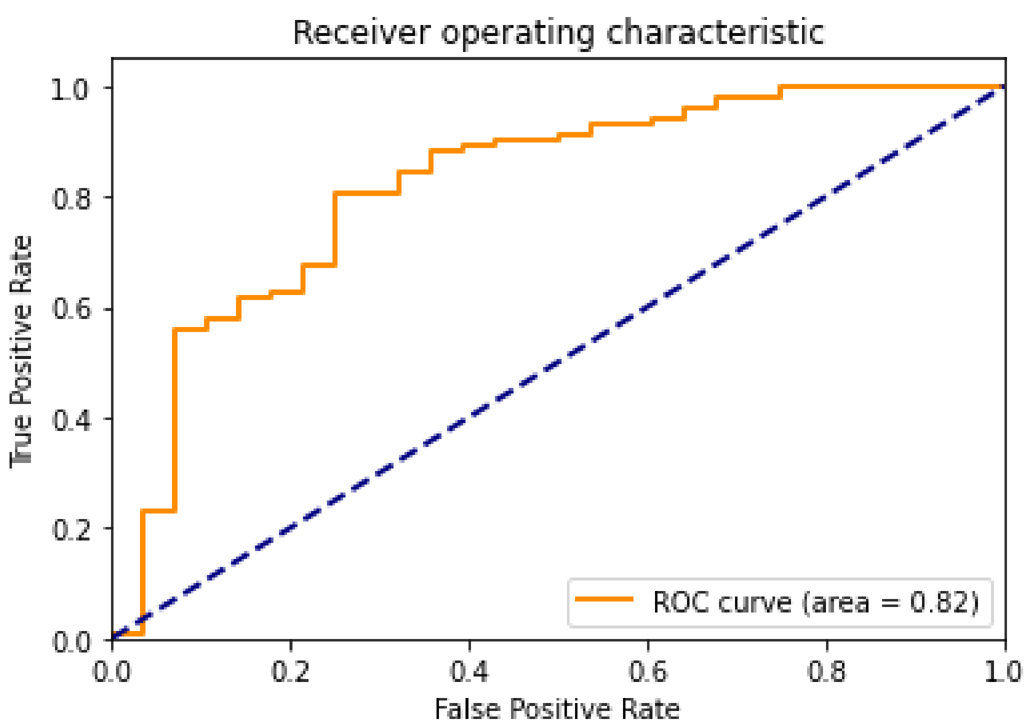
[16]:
```
# Define the SVM problem
p = svm_problem(train['Class'].to_numpy(), train.drop(['Class'], axis=1).
 ↪to_numpy())

# Define the hyperparameters
h = svm_parameter(f'-s 0 -t 0 -q')

# Train the model
m = svm_train(p,h)

# Print Information about Model
if(debug_mode):
    print_model(m)

# Predict with model on train dataset
print("Training Accuracy:")
p_label_train, p_acc_train, p_val_train = svm_predict(train['Class'].
 ↪to_numpy(), train.drop(['Class'], axis=1).to_numpy(), m)

# Predict with model on test dataset
print("Testing Accuracy:")
p_label, p_acc, p_val = svm_predict(test['Class'].to_numpy(), test.
 ↪drop(['Class'], axis=1).to_numpy(), m)

if(debug_mode):
    print(np.asarray([int(i) for i in p_label]))
    print(test['Class'].to_numpy())
```

```
Training Accuracy:
Accuracy = 85.1727% (1011/1187) (classification)
Testing Accuracy:
Accuracy = 83.3333% (110/132) (classification)
```

[17]:
```
from sklearn.metrics import roc_curve, auc, roc_auc_score
def compute_and_plot_roc_curve(dataset, model_output):
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in [0,1]:
        fpr[i], tpr[i], _ = roc_curve(dataset['Class'].to_numpy(), np.
 ↪asarray(model_output))
        roc_auc[i] = auc(fpr[i], tpr[i])
```

```
    if(debug_mode):
        print(fpr)
        print(tpr)
        print(roc_auc)

    # Plot ROC curve
    plt.figure()
    lw = 2
    plt.plot(
        fpr[1],
        tpr[1],
        color="darkorange",
        lw=lw,
        label="ROC curve (area = %0.2f)" % roc_auc[1],
    )
    plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver operating characteristic")
    plt.legend(loc="lower right")
    plt.show()
```

[18]:
```
compute_and_plot_roc_curve(test,p_val)
```



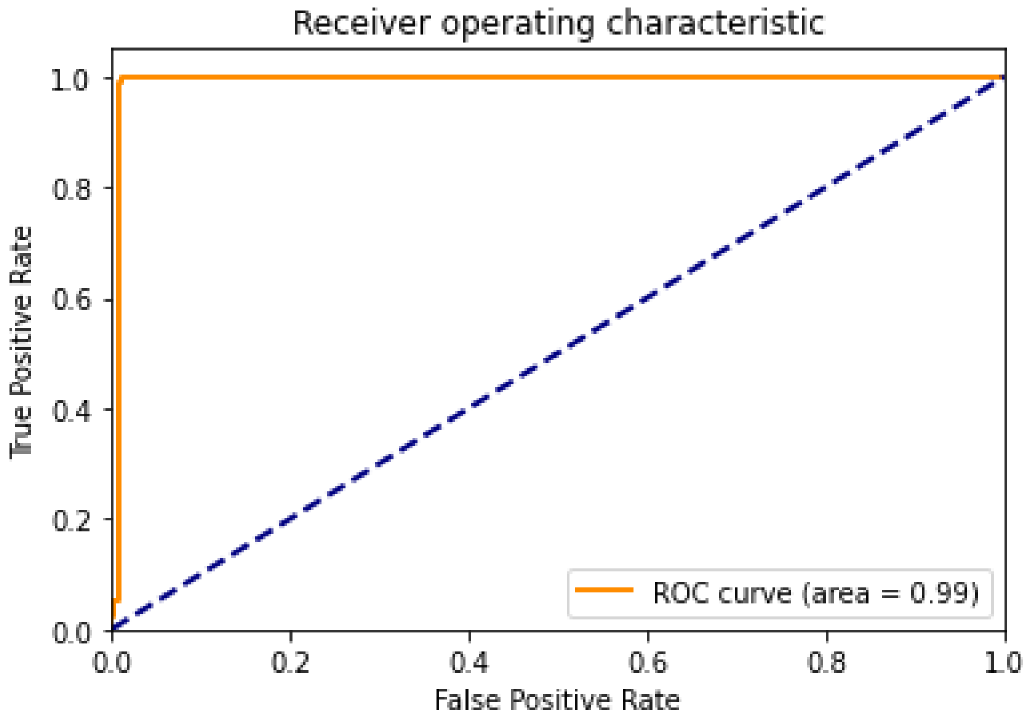
1.1.10 3.2

Train an SVM classifier with the radial basis kernel on the credit card data training set, for ˆ2 = 5 and ˆ2 = 25. Report the accuracy of these classifierson the training and test data set, show the ROC curves on the training and test sets, and also report the training and test AUCs.

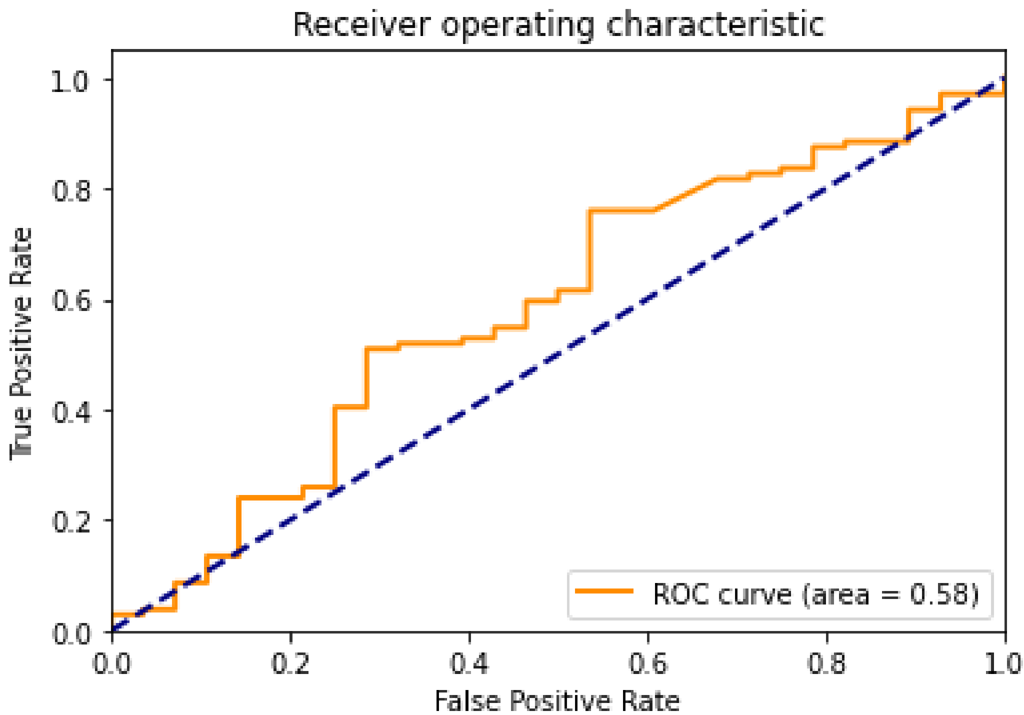
[19]:
```
sigmas_squared =[5,25]
support_vector_count = []
for sigma_squared in sigmas_squared:

    print(f"\n\n----- sigma^2 = {sigma_squared} -----\n")

    # Define the hyperparameters
    h = svm_parameter(f'-s 0 -t 2 -g {1/sigma_squared} -q')

    # Train the model
    m = svm_train(p,h)

    # Record number of support vectors
    print("Support vector count:", m.get_nr_sv())

    # Print Information about Model
    if(debug_mode):
        print_model(m)

    # Predict with model on train dataset
    print("Training Accuracy:")
    p_label_train, p_acc_train, p_val_train = svm_predict(train['Class'].
 ↪to_numpy(), train.drop(['Class'], axis=1).to_numpy(), m)

    compute_and_plot_roc_curve(train,p_val_train)

    # Predict with model on test dataset
    print("Testing Accuracy:")
    p_label, p_acc, p_val = svm_predict(test['Class'].to_numpy(), test.
 ↪drop(['Class'], axis=1).to_numpy(), m)

    compute_and_plot_roc_curve(test,p_val)
```

```
----- sigma^2 = 5 -----

Support vector count: 1137
Training Accuracy:
```

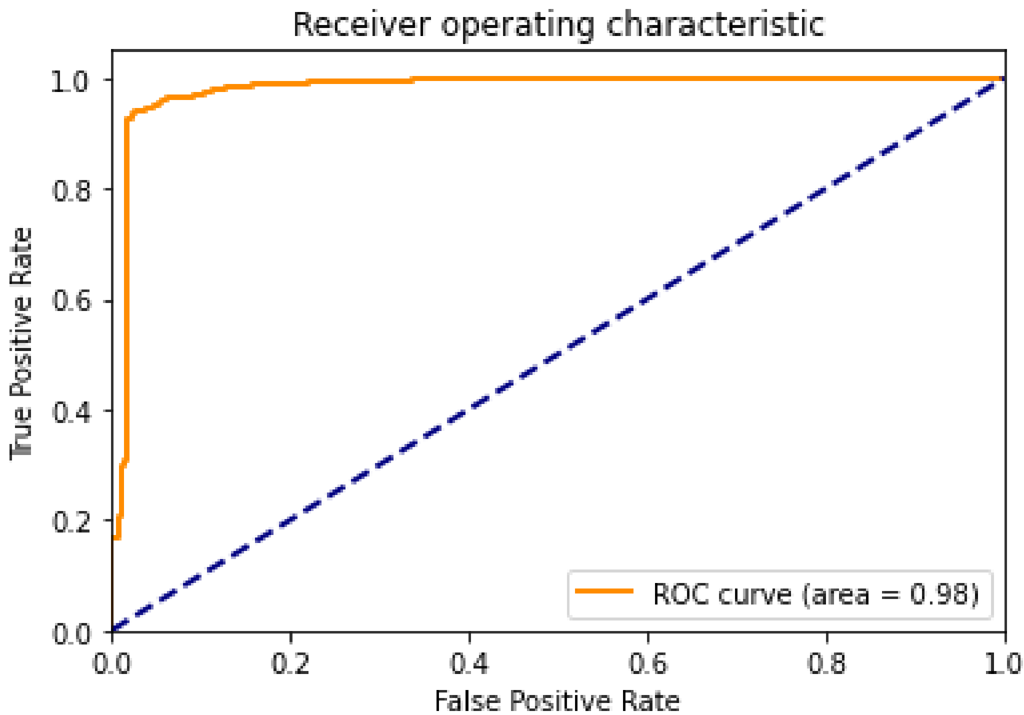
```
Accuracy = 97.2199% (1154/1187) (classification)
```



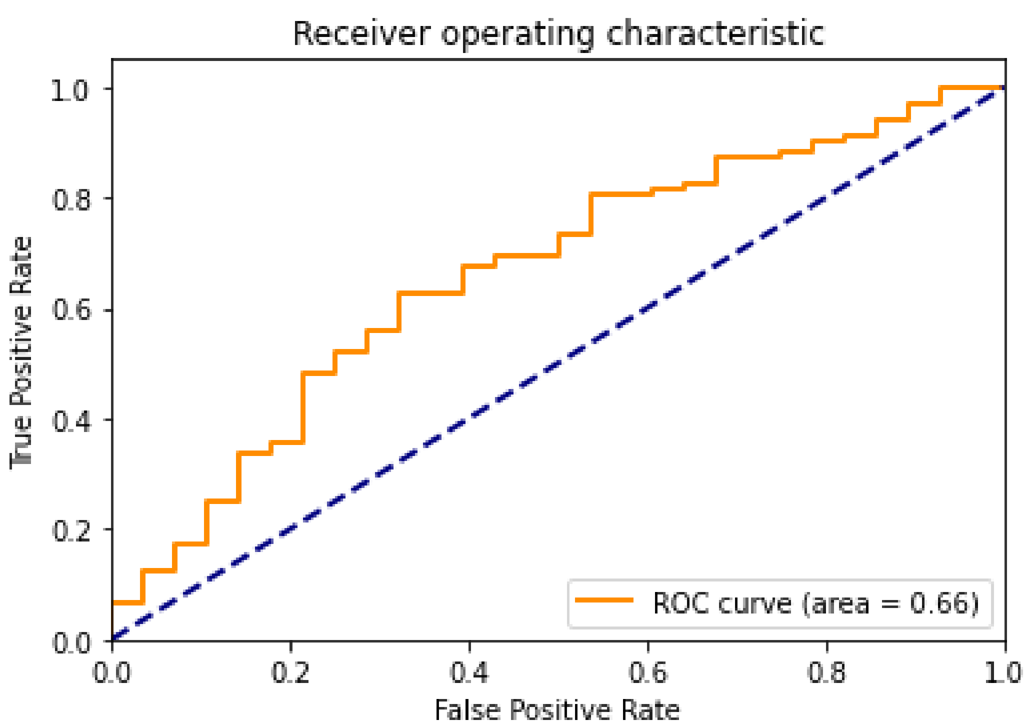

```
Testing Accuracy:
Accuracy = 78.7879% (104/132) (classification)
```




```
----- sigma^2 = 25 -----

Support vector count: 893
Training Accuracy:
Accuracy = 87.4473% (1038/1187) (classification)
```




```
Testing Accuracy:
Accuracy = 79.5455% (105/132) (classification)
```



1.1.11 3.2 Clustering

3.2.1

[20]:
```
import sys
# !conda install --yes --prefix {sys.prefix} Pillow
!{sys.executable} -m pip install Pillow
```

```
Requirement already satisfied: Pillow in
/Users/adamcarriker/opt/anaconda3/envs/MachineLearning/lib/python3.9/site-
packages (8.3.1)
```

[21]:
```
# Implements the k-means algorithm
from itertools import repeat
from math import floor, sqrt

def distance(p1,p2): # Since we are in 1 dimension, we can just use abs value␣
↪of difference between points
    return p1 - p2 if p1 > p2 else p2 - p1

def closest_center(point, centers):
    cc = None
    for i, c in enumerate(centers):
        cc = i if ( cc==None or distance(point,c) < distance(point,centers[cc])␣
↪) else cc
    return cc

def k_means(k, d):
    # Randomly initializes centers for each cluster
    centers = np.random.choice(d, k, replace=False)
    prev_centers = None
    i_ctr = 0

    while(not np.array_equal(centers,prev_centers)):
        if debug_mode:
            print("centers;",centers)
            print("prev_centers;",prev_centers)
        # Assign all points to closest cluster center
        groups = np.array(list(map(closest_center, d, repeat(centers))))
        if debug_mode:
            print("groups;",groups)
            print("groups avg:",np.mean(groups))

        # Change each cluster center to the middle of its points
        prev_centers = centers.copy()
        for i,center in enumerate(centers):
```

```
            centers[i]=floor(np.average(d[groups==i]))
        if debug_mode:
            print("new centers;",centers)
            print("prev_centers;",prev_centers)

        # Check for too many iterations
        i_ctr+=1
        if i_ctr > 100: break

    print("iterations:",i_ctr)
    return groups, centers
```

[23]:

```
from PIL import Image
from matplotlib.pyplot import imshow

%matplotlib inline

random_seed = 1
np.random.seed(random_seed)

# load the image
image = Image.open('raccoon.png')

# convert image to numpy array
data = np.asarray(image)
if debug_mode: print(data)

# flatten d to make computation easier
d = data.flatten()

# run k-means for k=[2,4,6]
for k in [2,4,6,8]:
    print(f"\n------------\nk={k}\n")
    codemap, codebook = k_means(k,d)
    if debug_mode:
        print("codemap:", np.unique(codemap))
        print("codebook:", np.unique(codebook))

    # create new image data
    new_data = np.copy(codemap)
    for i,code in enumerate(codebook):
        f = lambda x: code if x==i else x
        new_data = np.array(list(map(f, new_data)))
        if(debug_mode): print("new_data:",np.unique(new_data))

    # un-flatten image
    new_image = new_data.reshape(data.shape).astype(np.uint8)
```

```
    if debug_mode:
        print(np.unique(new_image))

    # create Pillow image
    image2 = Image.fromarray(new_image)

    # show image
    display(image2)
```

```
------------
k=2

iterations: 8
```

```
------------
k=4

iterations: 15
```

```
------------
k=6

iterations: 5
```

```
------------
k=8

iterations: 16
```

```
[ ]:
```