Support Vector Machines

First part

The objective of the first part of this practice is to get familiar with the use of SVM, coming from sckit-learn.

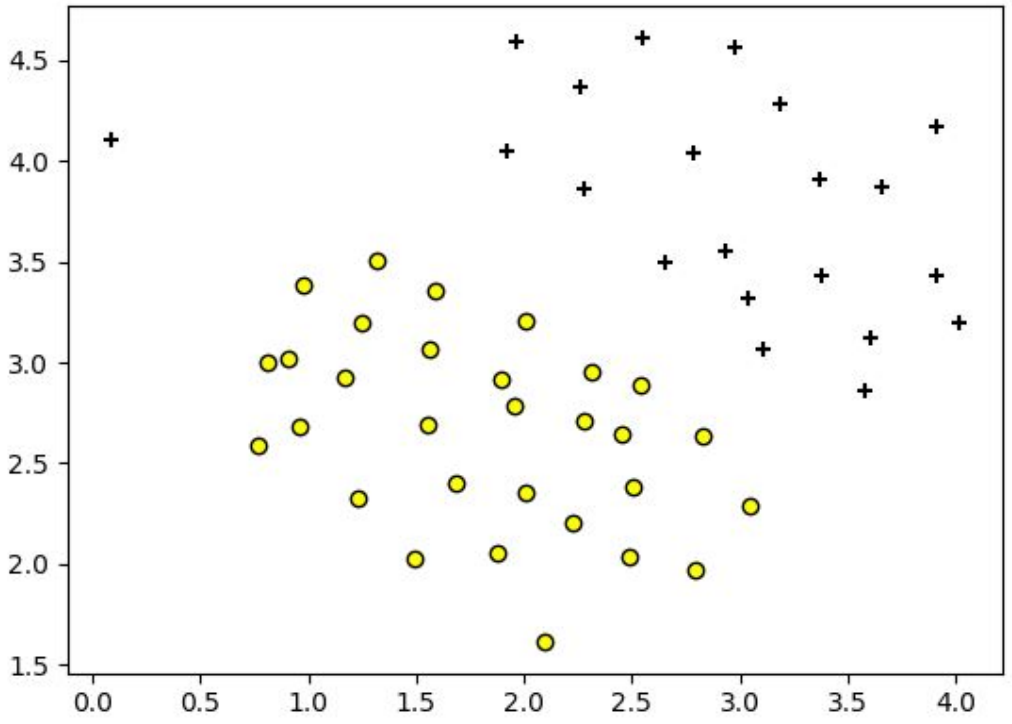
1.1. Lineal Kernel

Our code:

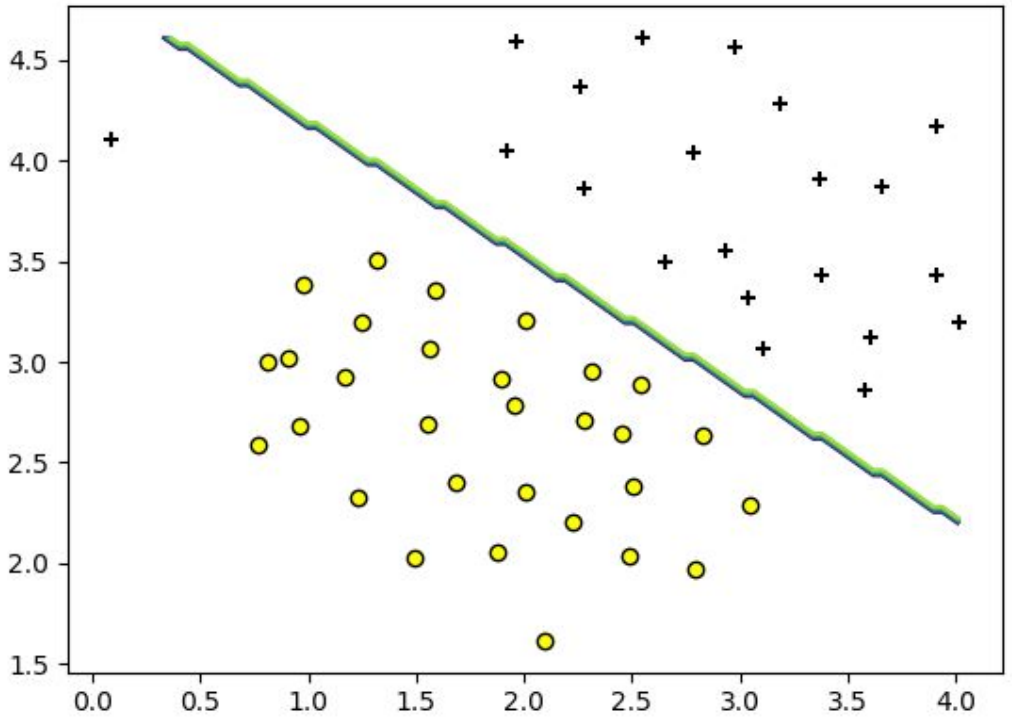
```
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm as svm

data1 = loadmat("ex6data1.mat")
data2 = loadmat("ex6data2.mat")
data3 = loadmat("ex6data3.mat")

y1 = data1['y']
X1 = data1['X']

y2 = data2['y']
X2 = data2['X']

y3 = data3['y']
X3 = data3['X']

X_val = data3['Xval']
y_val = data3['yval']


def draw_graphic(X, y):
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()

    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='black')
    plt.scatter(X[neg, 0], X[neg, 1], c='yellow', edgecolors='black')



def draw_line(X, y, c):
```

```
    clf = svm.SVC(kernel='linear', C=c)
    clf.fit(X, y.ravel())

    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)

    y = clf.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape)
    plt.contour(x1, x2, y)

draw_graphic(X1, y1)

#draw_line(X1, y1, 1)
draw_line(X1, y1, 100)
```

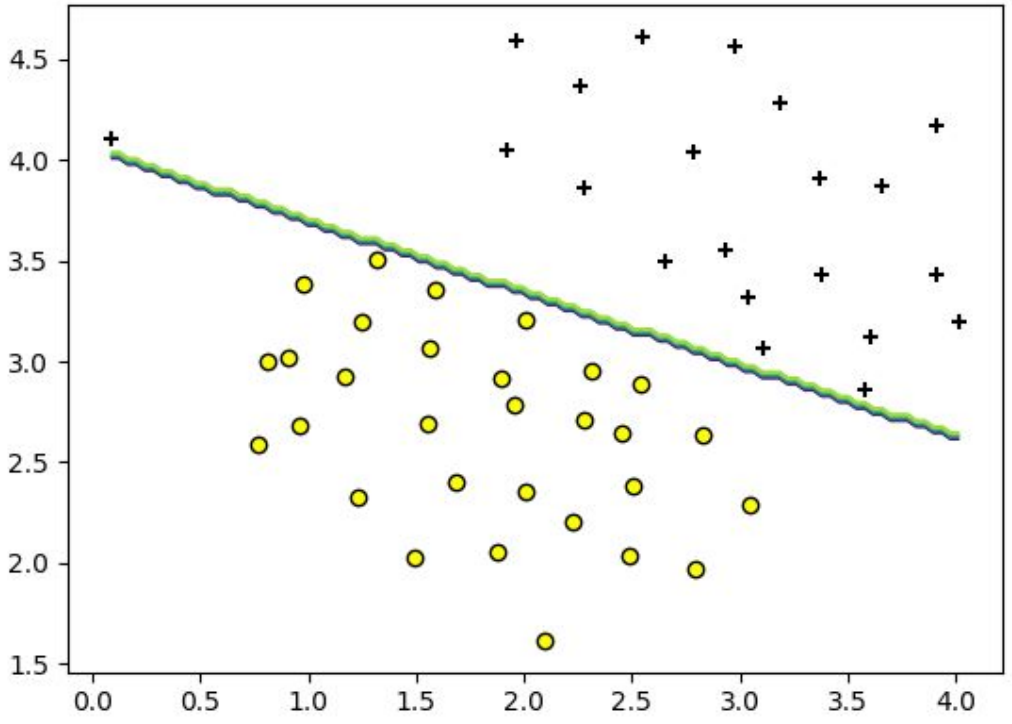
Executing the above code we have generated the following graphs.



This graph represents the data from *“ex6data1.mat”*.



The above graph represents the decision boundary line when the parameter C is 1.



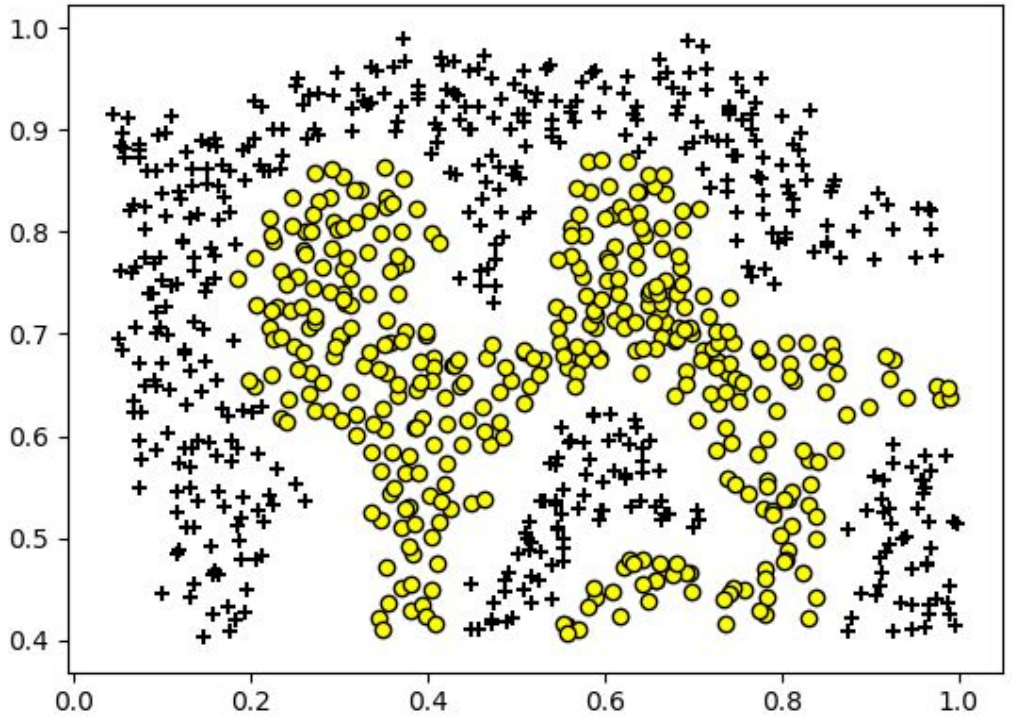
The graph represents the decision boundary line when the parameter C is 100.

1.2. Gaussian Kernel

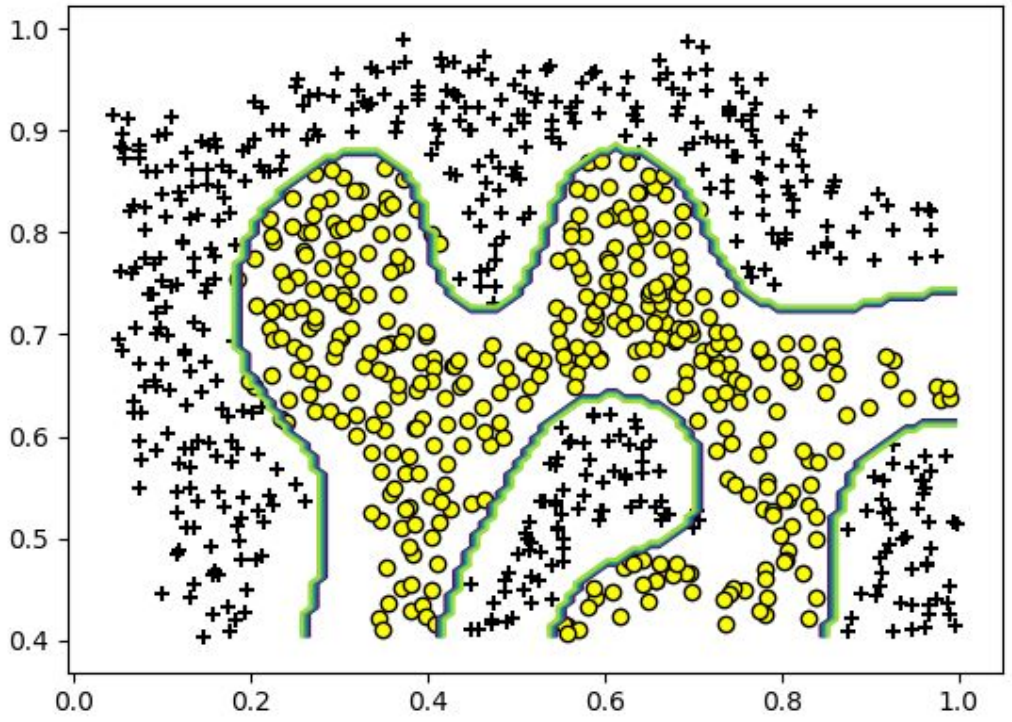
Our code:

```
def gaussian_kernel(X, y, c, sigma):
    clf = svm.SVC(kernel='rbf', C=c, gamma=1 / (2 * sigma ** 2))
    clf.fit(X, y.ravel())

    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    y =
clf.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(x1.shape)
    plt.contour(x1, x2, y)

draw_graphic(X2, y2)
gaussian_kernel(X2, y2, 1, 0.1)
```

The data from *ex6data2.mat*-our second dataset, cannot be linearly separated as seen in the graph below.



Although, using the above code we can calculate the decision boundary using SVM shown in the graph below.



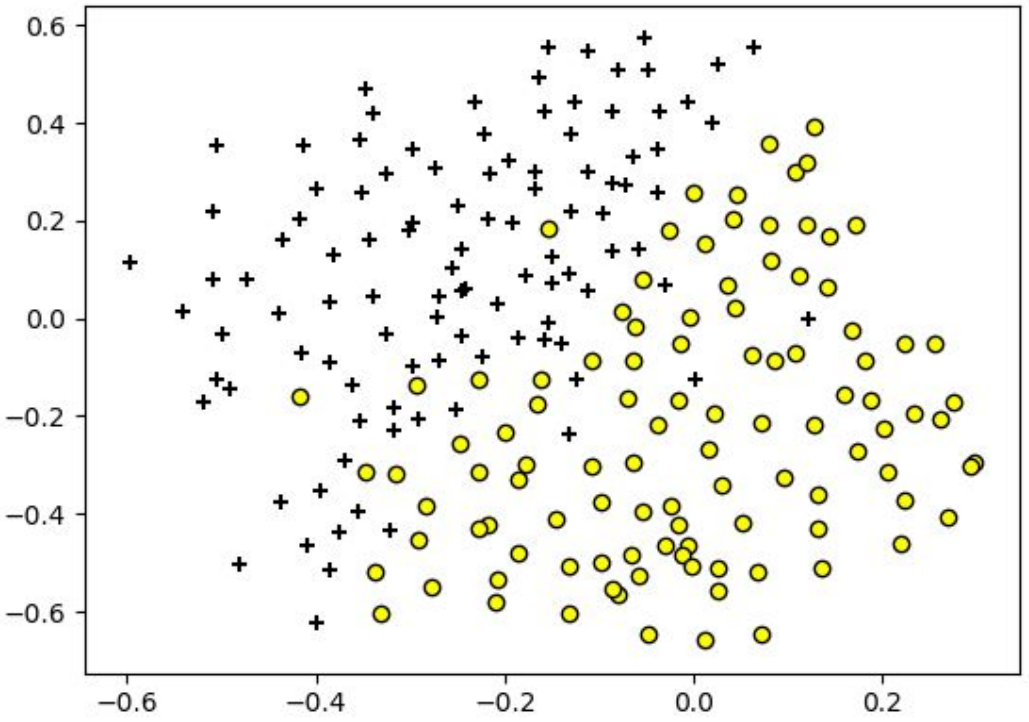
1.3. Choosing the parameters **C** and **σ**

Our code:

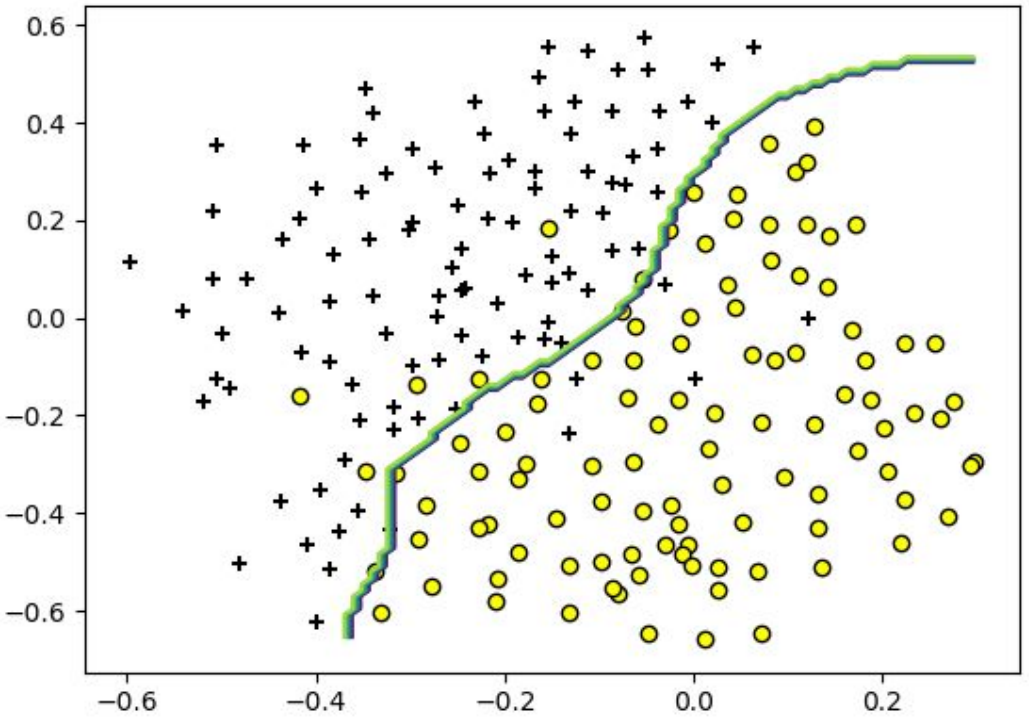
```
def choose_params(X, y, X_val, y_val):
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))
    best_score = [0, 0]

    for c in range(len(C_vec)):
        for s in range(len(sigma_vec)):
            clf = svm.SVC(kernel='rbf', C=C_vec[c], gamma= 1 / ( 2 *
sigma_vec[s] ** 2))
            clf.fit(X, y)
            scores[c][s] = clf.score(X_val, y_val)

            if scores[c][s] > scores[best_score[0]][best_score[1]]:
                best_score[0] = c
                best_score[1] = s
    print(f"Best accuracy: {scores[best_score[0]][best_score[1]] * 100}
```

```
%")
    return (C_vec[best_score[0]], sigma_vec[best_score[1]])

draw_graphic(X3, y3)
c, sigma = choose_params(X3, y3, X_val, y_val)
gaussian_kernel(X3, y3, c, sigma)

plt.show()
```

Executing the above code we obtained 96.5% best accuracy and the following graphs.



Graph of the third dataset-*ex6data3.mat.*



Calculated decision boundary for this set of data.

Second part

Our code:

```
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm as svm
import codecs
import get_vocab_dict as vocabDic
import process_email as procEm
import random
from p6_1 import choose_params

emails = [500, 2551, 250]
config = [10, 10, 10]

def read_data_dir(X_train, y_train, X_val, y_val, dic, dir, num_files,
porc, spam):
    set_ = set(np.arange(1, num_files + 1 , 1))
    sec = random.sample(set_, k = int(porc))
```

```
    for i in range(1, num_files + 1) :
        email_contents = codecs.open('{0}/{1:04d}.txt'.format(dir, i),
'r', encoding='utf-8', errors='ignore').read()
        email = procEm.email2TokenList(email_contents)
        email = set(email)

        res = [1 if elem in email else 0 for elem in dic.keys()]

        if(i in sec):
            X_val = np.vstack((X_val, res))
            y_val = np.vstack((y_val, spam))
        else:
            X_train = np.vstack((X_train, res))
            y_train = np.vstack((y_train, spam))

    return X_train, y_train, X_val, y_val




def process_emails(config):

    dic = vocabDic.getVocabDict()

    X_train = np.empty((0, len(dic)))
    y_train = np.empty((0, 1))
    X_val = np.empty((0, len(dic)))
    y_val = np.empty((0, 1))


    X_train, y_train, X_val, y_val = read_data_dir(X_train, y_train,
X_val, y_val, dic, 'spam', emails[0], (config[0] * emails[0]) / 100, 1)

    X_train, y_train, X_val, y_val = read_data_dir(X_train, y_train,
X_val, y_val, dic, 'easy_ham', emails[1], (config[1] * emails[1]) /
100, 0)

    X_train, y_train, X_val, y_val = read_data_dir(X_train, y_train,
X_val, y_val, dic, 'hard_ham', emails[2], (config[2] * emails[2]) /
```

```
100, 0)

    return (X_train, y_train, X_val, y_val)

X_train, y_train, X_val, y_val = process_emails(config)

c, sigma = choose_params(X_train, y_train.ravel(), X_val,
y_val.ravel())

clf = svm.SVC(kernel='rbf', C=c, gamma = 1 / (2* sigma ** 2))
clf.fit(X_train, y_train.ravel())
print(f"Accuracy: {clf.score(X_val, y_val)}")
```

Accuracy: 98.78787878787879%

Conclusion:

In this practice, we have implemented decision boundaries for several different datasets. Additionally we have created an email spam classifier capable of detecting a spam message with accuracy over 98%.

Gasan Nazer and Veronika Yankova