In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as pltcolors
import scipy.stats as stats
from sklearn import linear_model, svm, discriminant_analysis, metrics
from scipy import optimize
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [2]:
```
x_scaled = pd.read_csv('X_scaled.csv')
```

In [3]:
```
x_scaled.head()
```

Out[3]:

	age	balance	duration	campaign	pdays	previous	default	housing	loan	jo
0	1.606947	0.256416	0.011016	-0.569344	-0.411449	-0.251938	0	1	0	
1	0.288526	-0.437890	-0.416122	-0.569344	-0.411449	-0.251938	0	1	0	
2	-0.747376	-0.446758	-0.707353	-0.569344	-0.411449	-0.251938	0	1	1	
3	0.571045	0.047205	-0.645224	-0.569344	-0.411449	-0.251938	0	1	0	
4	-0.747376	-0.447086	-0.233618	-0.569344	-0.411449	-0.251938	0	0	0	

5 rows × 39 columns

In [4]:
```
def plotLine(ax, xRange, w, x0, label, color='grey', linestyle='-', alpha=1.):
    """ Plot a (separating) line given the normal vector (weights) and point of
    if type(x0) == int or type(x0) == float or type(x0) == np.float64:
        x0 = [0, -x0 / w[1]]
    yy = -(w[0] / w[1]) * (xRange - x0[0]) + x0[1]
    ax.plot(xRange, yy, color=color, label=label, linestyle=linestyle)

def plotSvm(X, y, support=None, w=None, intercept=0., label='Data', separatorLa
            ax=None, bound=[[-1., 1.], [-1., 1.]]):
    """ Plot the SVM separation, and margin """
    if ax is None:
        fig, ax = plt.subplots(1)

    im = ax.scatter(X[:,0], X[:,1], c=y, cmap=cmap, alpha=0.5, label=label)
    if support is not None:
        ax.scatter(support[:,0], support[:,1], label='Support', s=80, facecolo
                   edgecolors='y', color='y')
        print("Number of support vectors = %d" % (len(support)))
    if w is not None:
        xx = np.array(bound[0])
        plotLine(ax, xx, w, intercept, separatorLabel)
        # Plot margin
        if support is not None:
            signedDist = np.matmul(support, w)
            margin = np.max(signedDist) - np.min(signedDist) * np.sqrt(np.dot(w
            supportMaxNeg = support[np.argmin(signedDist)]
            plotLine(ax, xx, w, supportMaxNeg, 'Margin -', linestyle='-.', alph
```

```
            supportMaxPos = support[np.argmax(signedDist)]
            plotLine(ax, xx, w, supportMaxPos, 'Margin +', linestyle='--', alph
            ax.set_title('Margin = %.3f' % (margin))
    ax.legend(loc='upper left')
    ax.grid()
    ax.set_xlim(bound[0])
    ax.set_ylim(bound[1])
    cb = plt.colorbar(im, ax=ax)
    loc = np.arange(-1,1,1)
    cb.set_ticks(loc)
    cb.set_ticklabels(['-1','1'])
```

In [5]:

```
colors = ['blue','red']
cmap = pltcolors.ListedColormap(colors)
nFeatures = 2
N = 100
```

In [6]:

```
class KernelSvmClassifier:

    def __init__(self, C, kernel):#initialized a function here->GRBF
        self.C = C
        self.kernel = kernel          # <---
        self.alpha = None
        self.supportVectors = None

    def fit(self, X, y):
        N = len(y)
        # --->
        # Gram matrix of h(x) y
        hXX = np.apply_along_axis(lambda x1 : np.apply_along_axis(lambda x2:  s
                                  1, X)



        print("1")
        yp = y.reshape(-1, 1)
        GramHXy = hXX * np.matmul(yp, yp.T)
        # <---
        print("2")
        # Lagrange dual problem
        def Ld0(G, alpha):
            return alpha.sum() - 0.5 * alpha.dot(alpha.dot(G))
        print("3")
        # Partial derivate of Ld on alpha
        def Ld0dAlpha(G, alpha):
            return np.ones_like(alpha) - alpha.dot(G)
        print("4")
        # Constraints on alpha of the shape :
        # -  d - C*alpha  = 0
        # -  b - A*alpha >= 0
        A = np.vstack((-np.eye(N), np.eye(N)))             # <---
        b = np.hstack((np.zeros(N), self.C * np.ones(N)))  # <---
        constraints = ({'type': 'eq',   'fun': lambda a: np.dot(a, y),     'jac
                       {'type': 'ineq', 'fun': lambda a: b - np.dot(A, a), 'jac
        print("5")
        # Maximize by minimizing the opposite
        optRes = optimize.minimize(fun=lambda a: -Ld0(GramHXy, a),
                                   x0=np.ones(N),
```

```
                                            method='SLSQP',
                                            jac=lambda a: -Ld0dAlpha(GramHXy, a),
                                            constraints=constraints)
        self.alpha = optRes.x
        print("6")
        # --->
        epsilon = 1e-8
        supportIndices = self.alpha > epsilon
        self.supportVectors = X[supportIndices]
        self.supportAlphaY = y[supportIndices] * self.alpha[supportIndices]
        print("7")
        # <---

    def predict(self, X):
        """ Predict y values in {-1, 1} """
        # --->
        def predict1(x):
            x1 = np.apply_along_axis(lambda s: self.kernel(s, x), 1, self.suppo
            x2 = x1 * self.supportAlphaY
            return np.sum(x2)

        d = np.apply_along_axis(predict1, 1, X)
        return 2 * (d > 0) - 1
        # <---
```

In [7]:

```
x_scaled['y']=x_scaled['y'].map({1:1,0:-1})
X_svm=x_scaled.copy()
X_svm.head()

# minority_class_len=len(X_svm[X_svm['y']==1])
# majority_class_indices=X_svm[X_svm['y']==-1].index
# print(minority_class_len)
# print(majority_class_indices)
# random_majority_indices=np.random.choice(majority_class_indices,minority_clas
```

Out[7]:

	age	balance	duration	campaign	pdays	previous	default	housing	loan	jo
0	1.606947	0.256416	0.011016	-0.569344	-0.411449	-0.251938	0	1	0	
1	0.288526	-0.437890	-0.416122	-0.569344	-0.411449	-0.251938	0	1	0	
2	-0.747376	-0.446758	-0.707353	-0.569344	-0.411449	-0.251938	0	1	1	
3	0.571045	0.047205	-0.645224	-0.569344	-0.411449	-0.251938	0	1	0	
4	-0.747376	-0.447086	-0.233618	-0.569344	-0.411449	-0.251938	0	0	0	

5 rows × 39 columns

In [8]:

```
# X_svm=x_scaled.copy()
# X_svm.head()
minority_class_len=len(X_svm[X_svm['y']==1])
majority_class_indices=X_svm[X_svm['y']==-1].index
random_majority_indices=np.random.choice(majority_class_indices,minority_class_
minority_class_indices=X_svm[X_svm['y']==1].index
under_sample_indices=np.concatenate([minority_class_indices,random_majority_inc
under_sample=X_svm.loc[under_sample_indices]
```

```
under_sample['y'].value_counts()
under_sample.head() #under_sample data for svm
```

Out[8]:

	age	balance	duration	campaign	pdays	previous	default	housing	loan	j
83	1.701120	0.322103	3.043698	-0.569344	-0.411449	-0.251938	0	1	0	
86	1.418601	-0.432635	4.694005	-0.569344	-0.411449	-0.251938	0	0	0	
87	0.006007	-0.030305	4.391125	-0.569344	-0.411449	-0.251938	0	1	0	
129	1.324428	0.365784	1.245834	-0.569344	-0.411449	-0.251938	0	1	0	
168	1.230255	-0.386983	1.610843	-0.246558	-0.411449	-0.251938	0	0	0	

5 rows × 39 columns

In [9]:
```
#Drop 'y' column and split the dataset
X = under_sample.drop(['y'], axis = 'columns')
y = under_sample.y
train, test, ytrain, ytest = train_test_split(X, y, test_size= 0.2)
```

In [10]:
```
#RBF Kernel function,will be called for each data point
def GRBF(x1, x2):
    diff = x1 - x2
    return np.exp(-np.dot(diff, diff) * len(x1) / 2)
```

In [11]:
```
SVM_RGB = KernelSvmClassifier(C=70, kernel=GRBF)#we can call different kernels
```

In [12]:
```
SVM_RGB.fit(np.array(train), np.array(ytrain))
```

```
1
2
3
4
5
6
7
```

In [13]:
```
SVM_RGB.supportVectors
```

Out[13]:
```
array([[-0.93572209, -0.40044855, -0.14430705, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.19435314,  0.42063266,  0.28283128, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.51277425, -0.36399254, -0.19867011, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [ 0.19435314, -0.59422371, -0.67240535, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.57104489,  0.12635715,  0.25953282, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.8535637 , -0.31078648,  0.91577262, ...,  0.        ,
         0.        ,  0.        ]])
```

In [14]:
```
predicted=SVM_RGB.predict(test)
print(predicted)
```

```
[-1 -1  1 ...  1 -1  1]
```

In [15]:
```
print(ytest)
```

```
20111   -1
37299   -1
44020    1
38659   -1
24830    1
        ..
44149    1
3065    -1
44555    1
2711    -1
19134    1
Name: y, Length: 2116, dtype: int64
```

In [16]:
```
#Getting perfomance metrics,accuracy

metrics.accuracy_score(ytest,pd.DataFrame(data=predicted))
```

Out[16]:
```
0.7641776937618148
```

In [22]:
```
metrics.precision_score(ytest,pd.DataFrame(data=predicted))
```

Out[22]:
```
0.7728971962616823
```

In [19]:
```
metrics.recall_score(ytest,pd.DataFrame(data=predicted))
```

Out[19]:
```
0.7636195752539243
```

In [21]:
```
metrics.f1_score(ytest,pd.DataFrame(data=predicted))
```

Out[21]:
```
0.7682303762192291
```

In [17]:
```
metrics.confusion_matrix(ytest,pd.DataFrame(data=predicted))
```

Out[17]:
```
array([[790, 243],
       [256, 827]])
```

In [ ]: