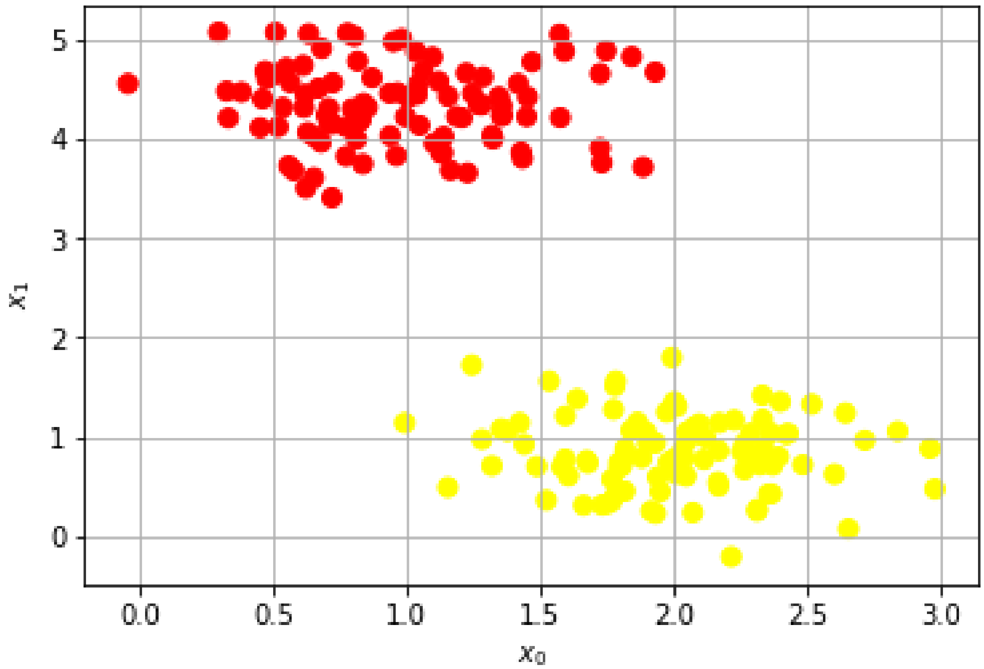
Inf264-Assignment 4

September 20, 2019

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy import stats
        from math import sqrt
        from mpl_toolkits.mplot3d import Axes3D

        # Some code (such as the code to do PCA) will generate useless warnings
        # We will suppress these warning using the code below
        import warnings
        warnings.filterwarnings("ignore", category=DeprecationWarning)
```

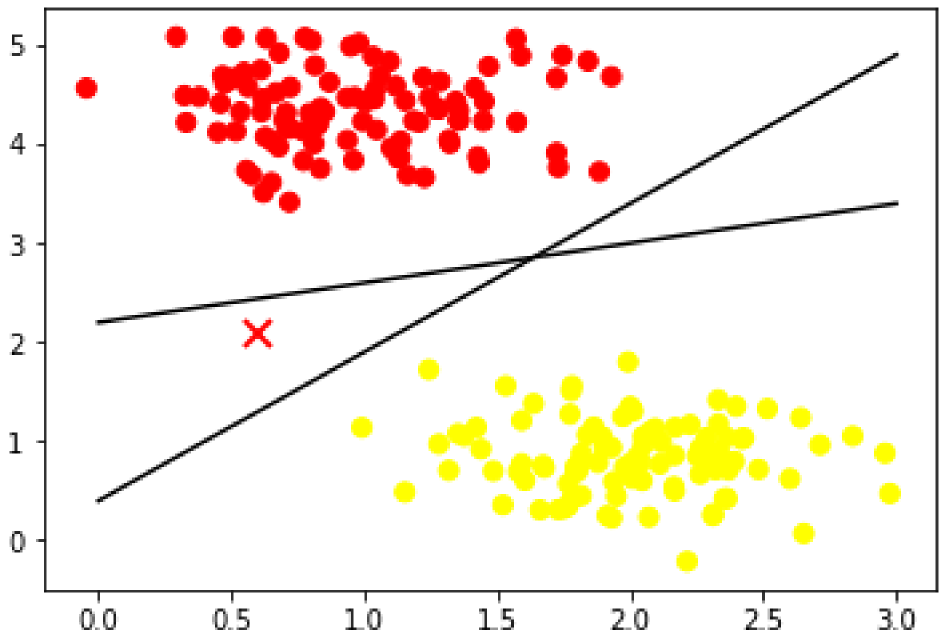
```
In [4]: from sklearn.datasets.samples_generator import make_blobs
        Xy1 = np.genfromtxt('svmlin.csv',delimiter=',')
        X1 = Xy1[:,:2]
        y1 = Xy1[:,2]
        # Each datapoint in X is tuple of values (x0, x1). Lets plot them
        # on x and y axes respectively.
        plt.scatter(X1[:, 0], X1[:, 1], c=y1, s=50, cmap='autumn');
        plt.xlabel(r"$x_0$")
        plt.ylabel(r"$x_1$")
        plt.grid()
```



0.1 Marginal distance from a hyperplane

Find two lines that classify the red X as two different classes without changin the label of any of the points.

```
In [5]: xfit = np.linspace(0, 3)
        plt.scatter(X1[:, 0], X1[:, 1], c=y1, s=50, cmap='autumn')
        plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)


        # TODO
        # plot two lines that classify X differently

        for m, b in [(.4, 2.2),  (1.5, .4)]:
            plt.plot(xfit, m * xfit + b, '-k')
```



Find margin for both of you lines

```
In [6]: def calculate_margin(m,b,X):
            # TODO
            # find the minimum margin distance to the given line
            margins = []
            for i in X:
                margins.append(np.absolute(i[1]+m*i[0]+b)/(np.sqrt((m**2)+1)))# calculate dist
            return np.min(margins)
```

```
In [7]: # TODO
        # check the values of minimum margin for your lines
        print('margin of the first line: %.3f'%calculate_margin(.4, 2.2,X1))
        print('margin of the second line: %.3f'%calculate_margin(1.5, .4,X1))
```

```
margin of the first line: 2.669
margin of the second line: 1.452
```

Fit a support vector model to the dataset:

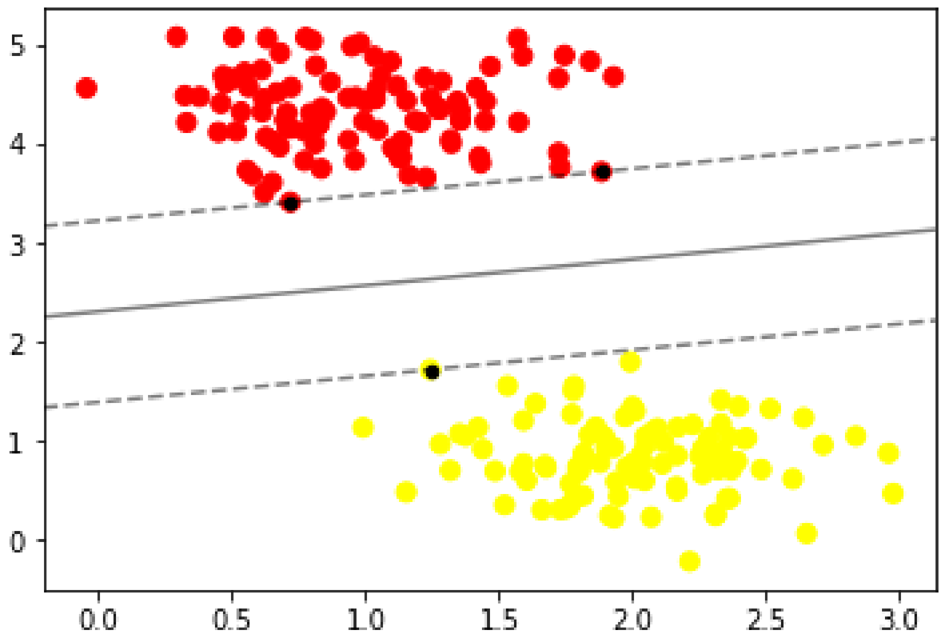
```
In [8]: from sklearn.svm import SVC # "Support vector classifier"
        model = SVC(kernel='linear', C=1E10)
        model.fit(X1, y1)
```

```
Out[8]: SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

Complete the function below for ploting the SVM boundry:

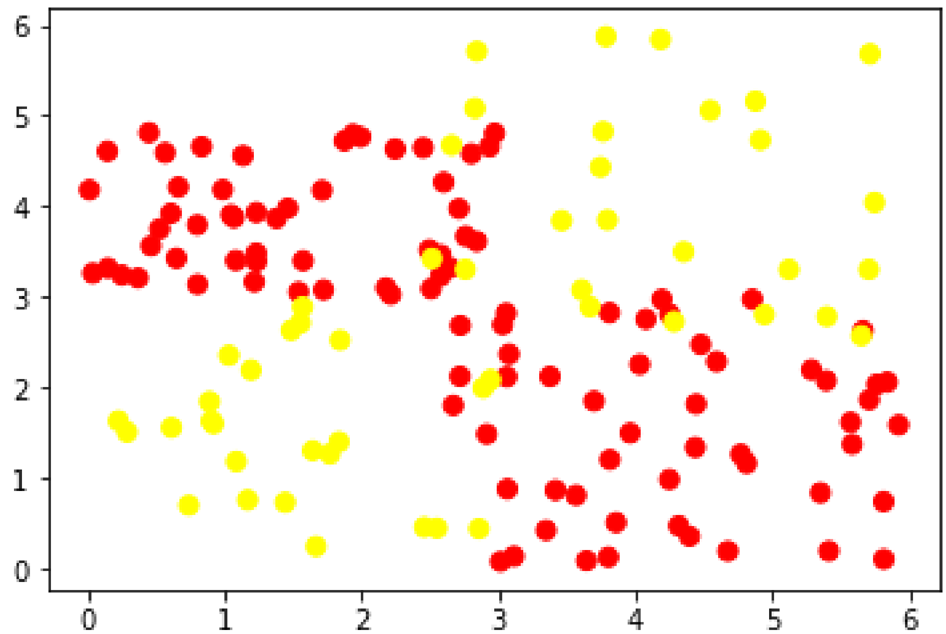
```
In [9]: def plot_svc_decision_function(X, y, model, ax=None, plot_support=True):
            """Plot the decision function for a 2D SVC"""
            if ax is None:
                ax = plt.gca()
            xlim = ax.get_xlim()
            ylim = ax.get_ylim()

            # create grid to evaluate model
            x = np.linspace(xlim[0], xlim[1], 30)
            y = np.linspace(ylim[0], ylim[1], 30)
            Y, X = np.meshgrid(y, x)
            xy = np.vstack([X.ravel(), Y.ravel()]).T
            P = model.decision_function(xy).reshape(X.shape)
            # plot decision boundary and margins
            ax.contour(X, Y, P, colors='k',
                       levels=[-1, 0, 1], alpha=.5,
                       linestyles=['--', '-', '--'])

            # plot a black dot over the support vectors
            if plot_support:
                ax.scatter(model.support_vectors_[:, 0],
                           model.support_vectors_[:, 1],
                           s = 20, facecolor= 'black');
            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
```

```
In [10]: plt.scatter(X1[:, 0], X1[:, 1], c=y1, s=50, cmap='autumn')
         plot_svc_decision_function(X1, y1 ,model, plot_support=True);
```

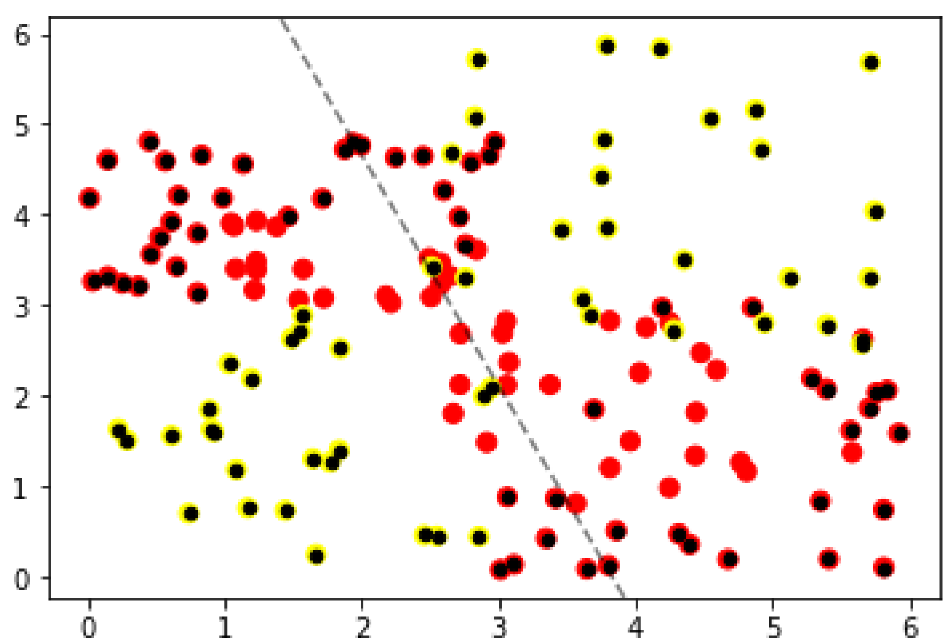


0.2 Kernel SVM

```
In [11]: Xy2 = np.genfromtxt('svmnonlin.csv',delimiter=',')
         X2 = Xy2[:,:2]
         y2 = Xy2[:,2]


         plt.scatter(X2[:, 0], X2[:, 1],c=y2, s=50, cmap='autumn')

Out[11]: <matplotlib.collections.PathCollection at 0x2499af0ec50>
```



In [12]:
```
clf = SVC(kernel='linear', C=1) # fit a linear SVM with C=1
clf.fit(X2, y2)
plt.scatter(X2[:, 0], X2[:, 1], c=y2, s=50, cmap='autumn')
plot_svc_decision_function(X2, y2 ,clf, plot_support=True);
```



```
In [13]: # Computes the weighted sum of kernel values
         def compute_f(training_points , test_point, kernel, weights, gamma=1, intercept=0):
             f = 0
             for i in range(len(training_points)):
                 trainSample = [training_points[i]] # put the training sample into two diemnsi
                 f += weights[i]*kernel(test_point,trainSample,gamma) # Complete the dual form
             return f + intercept
```

```
In [14]: from sklearn.metrics.pairwise import polynomial_kernel, rbf_kernel
         def surface_kernel(X,y,weights,clf,kernel):
             p = np.linspace(0, 6, 10)
             xx1, xx2 = np.meshgrid(p, p)

             z = float('NaN')*np.ones(xx1.shape)

             intercept = clf.intercept_
             for i in range(z.shape[0]):
                 for j in range(z.shape[1]):
                     tst = [[xx1[i, j], xx2[i, j]]]
                     z[i, j] = compute_f(X, tst, kernel, weights, gamma=1, intercept=intercept)
             zz = np.empty(y.shape)

             for i in range(X.shape[0]):
                 zz[i] = compute_f(X, [X[i,:]], kernel, weights, gamma=1, intercept=intercept)

             return xx1,xx2,z,zz
```

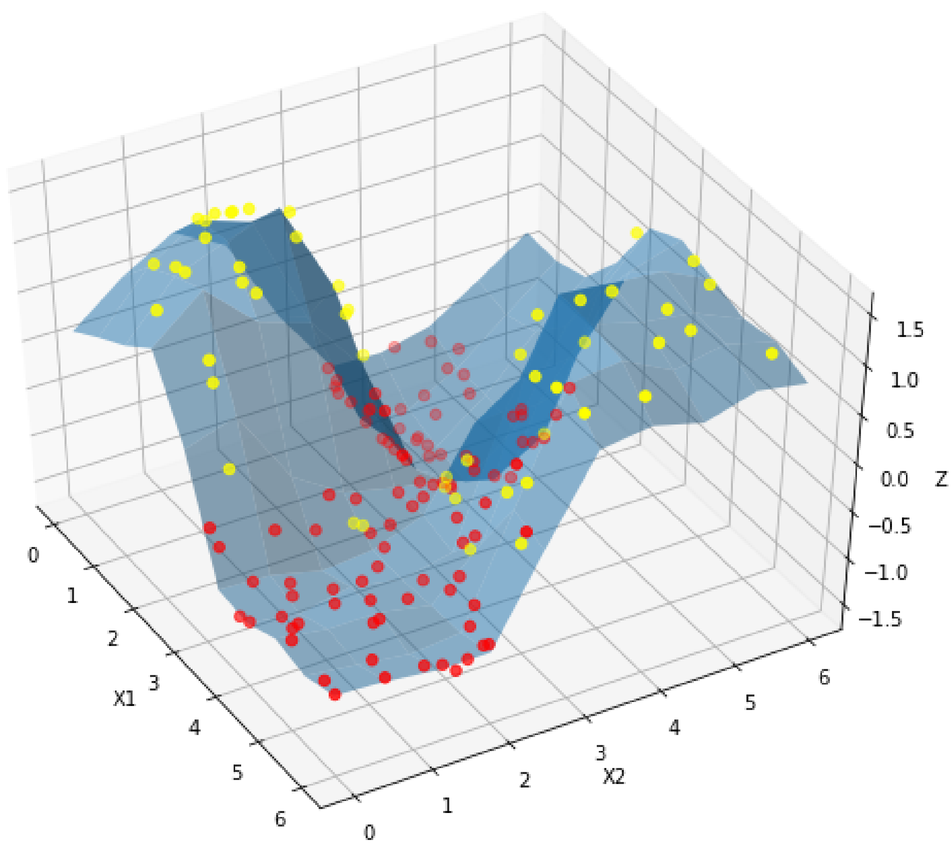
```
In [15]: def twodprojection(X,model):

             x_min = np.min(X[:, 0])
             y_min = np.min(X[:, 1])
             x_max = np.max(X[:, 0])
             y_max = np.max(X[:, 1])
             h = 0.01

             xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                                  np.arange(y_min, y_max, h))
             Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

             Z = Z.reshape(xx.shape)
             return xx,yy,Z
```

Fit a svm with RBF kernel to the dataset: (set the kernel value to 'rbf' and gamma to 1)

```
In [16]: clf = SVC(kernel='rbf', gamma=1) #Fit a SVM model with RBF kernel and gamma = 1
         clf.fit(X2, y2)
         weights = np.zeros((X2.shape[0], 1))
         weights[clf.support_] = clf.dual_coef_.T
```

Plot the 3D projection of the surface of dual function of SVM:

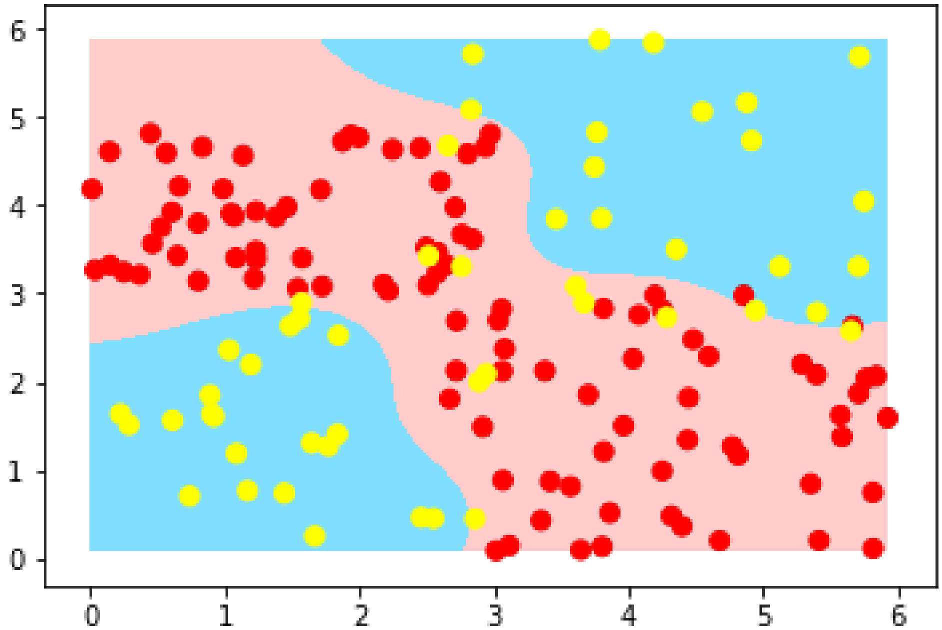
```
In [17]: xx1, xx2, z, zz = surface_kernel(X2,y2,weights,clf,rbf_kernel)
         fig = plt.figure(figsize=(10,8))
         ax = fig.add_subplot(111, projection='3d')
         ax.plot_surface(xx1, xx2, z, alpha=0.5)
         ax.view_init(elev=45, azim=-30)
         ax.scatter(X2[:, 0], X2[:, 1], zz, s=30,c=y2,cmap='autumn')
         ax.set_xlabel('X1')
         ax.set_ylabel('X2')
         ax.set_zlabel('Z')
         plt.show()
```



Plot the decision boundaries for RBF kernel:

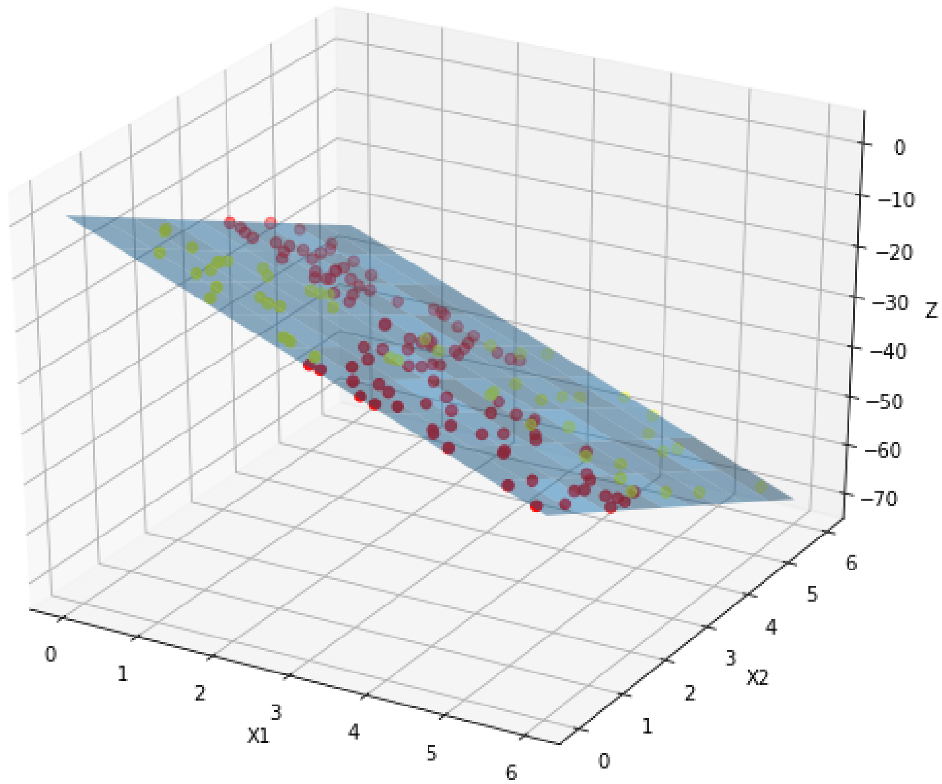
```
In [18]: # Plot decision boundary
         from matplotlib.colors import ListedColormap
         cmap_light = ListedColormap(['#FFCCCC', '#80dfff'])
         xx,yy,Z = twodprojection(X2,clf)
         plt.figure()
         plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
plt.scatter(X2[:, 0], X2[:, 1],c=y2, s=50, cmap='autumn')
plt.show()
```



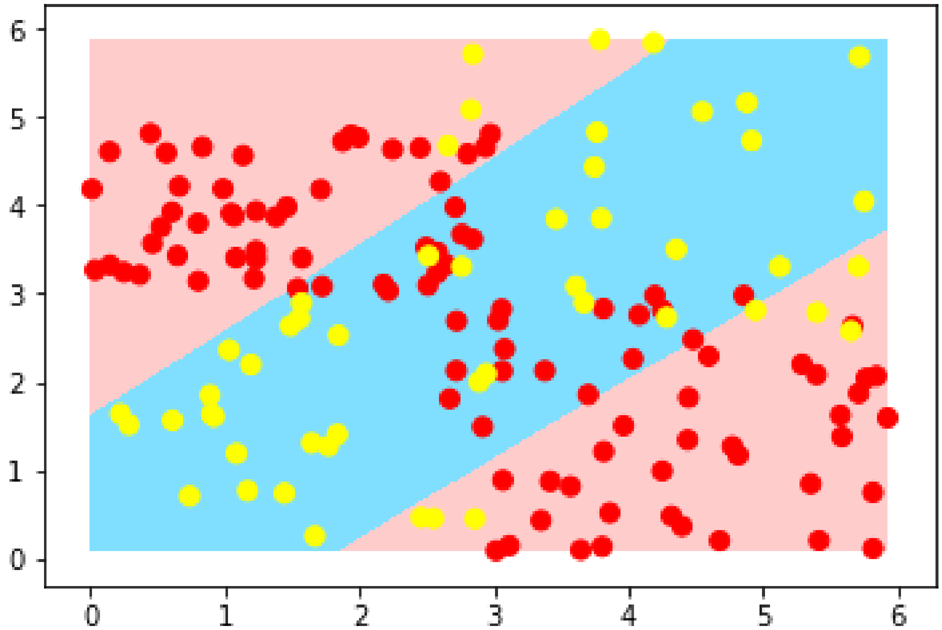
Fit a svm with RBF kernel to the dataset: (set the kernel value to 'poly' and gamma to 1 and degree to 2)

```
In [19]: clf_poly = SVC(kernel='poly', degree=2, gamma=1)# Fit a SVM with second degree polyno
         clf_poly.fit(X2,y2)
         weights = np.zeros((X2.shape[0], 1))
         weights[clf_poly.support_] = clf_poly.dual_coef_.T

In [20]: xx1, xx2, z, zz = surface_kernel(X2,y2,weights,clf_poly,polynomial_kernel)
         fig = plt.figure(figsize=(10,8))
         ax = fig.add_subplot(111, projection='3d')
         ax.plot_surface(xx1, xx2, z, alpha=0.5)
         ax.scatter(X2[:, 0], X2[:, 1], zz, s=30,c=y2,cmap='autumn')
         ax.set_xlabel('X1')
         ax.set_ylabel('X2')
         ax.set_zlabel('Z')
         plt.show()
```



Plot the decision boundaries for Polynomial kernel:

In [21]:
```
# Plot decision boundary
xx,yy,Z = twodprojection(X2,clf_poly)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X2[:, 0], X2[:, 1],c=y2, s=50, cmap='autumn')

plt.show()
```

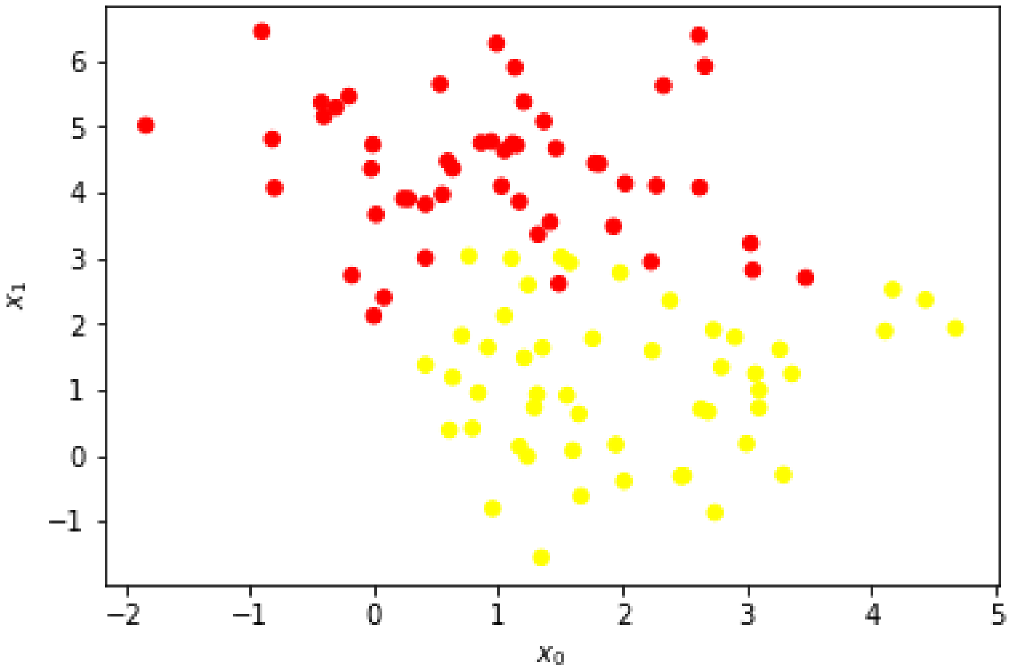


0.2.1 Tuning the SVM: Softening Margins

Generate the dataset by running the code below:

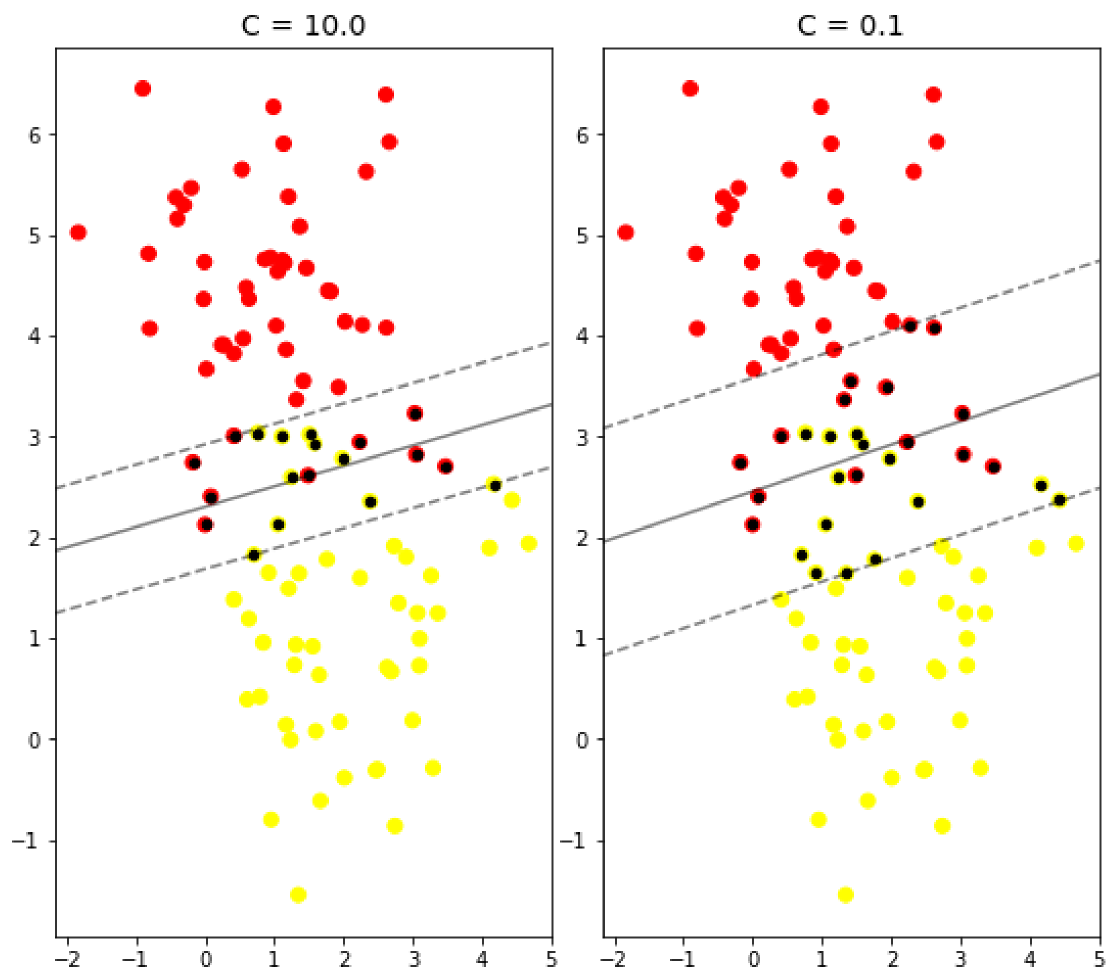
```
In [22]: from sklearn.datasets.samples_generator import make_blobs

         Xy3 = np.genfromtxt('svmmargin.csv',delimiter=',')
         X3 = Xy3[:,:2]
         y3 = Xy3[:,2]
         plt.scatter(X3[:, 0], X3[:, 1], c=y3, s=30, cmap='autumn');
         plt.xlabel(r"$x_0$")
         plt.ylabel(r"$x_1$")
```

```
Out[22]: Text(0,0.5,'$x_1$')
```



Fit two different SVMs with C=10 and C=0.1:

```
In [23]: fig, ax = plt.subplots(1, 2, figsize=(8, 8))
         fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

         for axi, C in zip(ax, [10.0, 0.1]):
             model = SVC(kernel='linear', C=C).fit(X3,y3)  # fit a linear SVM with C=C
             axi.scatter(X3[:, 0], X3[:, 1], c=y3, s=50, cmap='autumn')
             plot_svc_decision_function(X3,y3,model, axi)
             axi.set_title('C = {0:.1f}'.format(C), size=14)
```



Load the iris dataset and split it into training and test sets and perform cross validation on the training set: ##### Hints: Use `train_test_split` and `cross_val_score` from `sklearn.model_selection` to split the dataset and perform cross validation for best value of C, respectively.

```
In [24]: from sklearn.model_selection import cross_val_score,train_test_split
         from sklearn import datasets
         iris = datasets.load_iris()
         # TODO
         # Split the dataset into training and test set. (75% to 25%)
         X_train, X_test, y_train, y_test =  train_test_split( iris.data,  iris.target, test_s
         score_list = []
         C_list = np.arange(0.1,20,0.05)
         # TODO
         # perform cross validation on the training set
         for C in C_list:
             clf = SVC(kernel='linear', C=C)
             score = cross_val_score(clf,X_train,y_train)
```

```
            score_list.append(score.mean())
        best_C = C_list[np.argmax(score_list)]
        print('The best value for C is :%.3f'%best_C)
```

```
The best value for C is :1.200
```

Measure the accuracy score on the test set

```
In [25]: from sklearn.metrics import accuracy_score
         clf = SVC(kernel='linear', C=best_C)
         clf.fit(X_train,y_train)
         print(accuracy_score(y_test,clf.predict(X_test)))
```

```
0.9736842105263158
```

0.3 Kernel Trick

Complete the implementation of phi:

```
In [26]: def phi(x):
             # TODO
             # Complete the implementation
             transformed_x = np.empty(0)
             for i in range(len(x)):
                 transformed_x = np.concatenate((transformed_x,x[i]**2))# Power of two terms
             for i in range(len(x)):
                 for j in range(len(x)):
                     if (i!=j):
                         transformed_x = np.concatenate((transformed_x,np.sqrt(2)*x[i]*x[j]))
             for i in range(len(x)):
                 transformed_x = np.concatenate((transformed_x,np.sqrt(2)*x[i])) #single Terms
             transformed_x = np.concatenate((transformed_x,[1]))
             return transformed_x
```

Measure the run time in normal way:

```
In [27]: import timeit
         np.random.seed(3)
         run_time = []
         kernel_values = []
         for d in range(1,500,100):
             start = timeit.default_timer()

             x = np.random.uniform(0,100,[d,1])
             z = np.random.uniform(0,100,[d,1])
             x_k = phi(x)
             z_k = phi(z)
```

```
        kernel_values.append(np.transpose(x_k)*z_k) # the value of K(x,z) using phi

        stop = timeit.default_timer()
        run_time.append(stop - start)
```

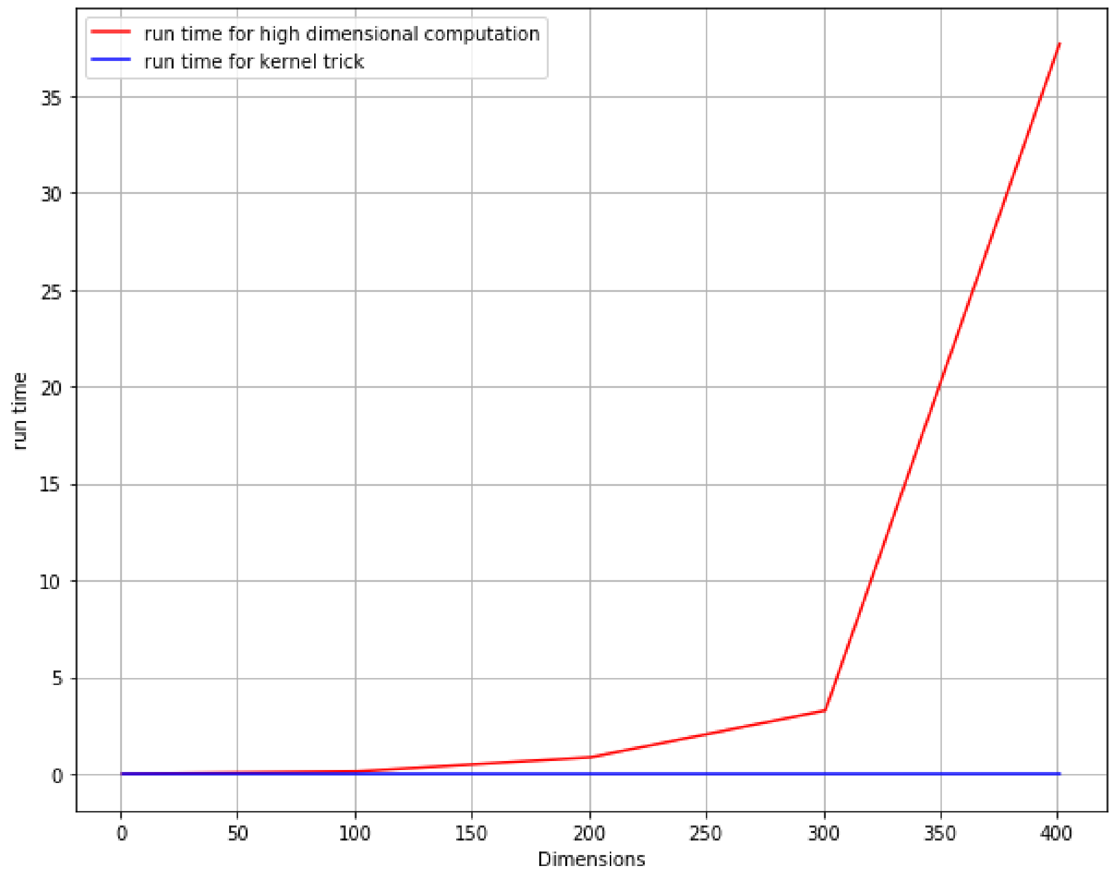
Measure the run time for Kernel Trick:

```
In [34]: np.random.seed(3)
         run_time_trick = []
         kernel_values_trick = []
         for d in range(1,500,100):
             start = timeit.default_timer()

             x = np.random.uniform(0,100,[d,1])
             z = np.random.uniform(0,100,[d,1])
             # TODO
             kernel_values_trick.append(((1+np.transpose(x)*z)**2)) # the value of K(x,z) usin

             stop = timeit.default_timer()
             run_time_trick.append(stop - start)
```

Plot the run times:

```
In [35]: plt.figure(figsize=(10,8))
         plt.plot(range(1,500,100),run_time,'R',label='run time for high dimensional computatio
         plt.plot(range(1,500,100),run_time_trick,'B',label='run time for kernel trick')
         plt.grid()
         plt.xlabel('Dimensions')
         plt.ylabel('run time')
         plt.legend()
```

```
Out[35]: <matplotlib.legend.Legend at 0x2499cadc208>
```



Compare the values of the kernel:

```
In [31]: kernel_values

Out[31]: [array([1.52136144e+07, 7.80092671e+03, 1.00000000e+00]),
          array([7.99690646e+06, 1.88963821e+07, 2.35614032e+07, ...,
                 7.81970931e+03, 1.18629689e+03, 1.00000000e+00]),
          array([7.02100153e+05, 1.45283717e+07, 3.26126033e+06, ...,
                 1.42011353e+04, 6.71088133e+03, 1.00000000e+00]),
          array([2.73471144e+07, 2.00567314e+07, 1.84968327e+06, ...,
                 9.34249607e+03, 3.22712038e+02, 1.00000000e+00]),
          array([1.44058818e+07, 1.25365426e+06, 4.67468606e+06, ...,
                 3.34306691e+03, 4.26562867e+03, 1.00000000e+00])]

In [32]: kernel_values_trick

Out[32]: [array([[3901.46335411]]),
          array([[2828.88020685, 4966.74679137, 8681.32273178, ..., 1868.13165033,
                  7055.22254   , 7631.35685001],
                 [2476.51617897, 4347.99691059, 7599.72335084, ..., 1635.48034236,
                  6176.24108182, 6680.58698807],
```

```
        [1582.34165653, 2777.82988054, 4855.00898334, ..., 1045.09410616,
         3945.69890553, 4267.87138722],
        ...,
        [ 223.53131035,  391.76412702,  684.07122313, ...,  147.9281661 ,
          556.1102842 ,  601.44739666],
        [1568.37337505, 2753.30163186, 4812.13262989, ..., 1035.87143103,
         3910.85465511, 4230.18132806],
        [ 220.82625173,  387.01405447,  675.76790754, ...,  146.14212844,
          549.36244339,  594.14844442]]),
 array([[ 838.91416791, 1990.58029721, 2270.65242188, ..., 3399.51294593,
         3329.33229323, 1934.76633396],
        [1606.26424837, 3812.61011796, 4349.16837884, ..., 6511.82358872,
         6377.37247546, 3705.68250748],
        [ 667.70377478, 1584.05080064, 1806.89599157, ..., 2705.09726492,
         2649.25657401, 1539.64126395],
        ...,
        [ 484.67022861, 1149.44788888, 1311.11426684, ..., 1962.72781945,
         1922.21735474, 1117.23032603],
        [1788.58179749, 4245.5129348 , 4843.01068722, ..., 7251.28900734,
         7101.56763719, 4126.44104347],
        [1454.93537186, 3453.28816992, 3939.26487771, ..., 5898.04575131,
         5776.26939607, 3356.4406663 ]]),
 array([[5230.44685194, 4189.10250652, 2035.8269159 , ..., 5832.04543233,
         5840.6963634 , 4702.65919317],
        [5593.01760575, 4479.47422762, 2176.90660361, ..., 6236.32653466,
         6245.57725663, 5028.63709601],
        [3496.23969912, 2800.22954745, 1361.03061483, ..., 3898.33408896,
         3904.11616851, 3143.47879919],
        ...,
        [7022.16267105, 5624.03238063, 2733.00037947, ..., 7829.88127211,
         7841.49618971, 6313.54412822],
        [4184.10162106, 3351.11692062, 1628.68415312, ..., 4665.32804292,
         4672.24803366, 3761.91750226],
        [ 180.46913862,  144.7312914 ,   70.83312847, ...,  201.11537179,
          201.41226268,  162.35601898]]),
 array([[3796.50810202, 1444.10741239, 2984.89164979, ...,  957.23165485,
         1675.57334411, 2373.99119918],
        [2945.82950752, 1120.66703174, 2316.11880395, ...,  742.91362989,
         1300.25503086, 1842.13808128],
        [2751.19253487, 1046.66322239, 2163.10223231, ...,  693.87723491,
         1214.38144624, 1720.44901865],
        ...,
        [4589.76558654, 1745.71545144, 3608.52208887, ..., 1157.08313633,
         2025.55753671, 2869.94404103],
        [3789.61803244, 1441.48770767, 2979.47492545, ...,  955.49578644,
         1672.53345418, 2369.68345591],
        [3412.35445283, 1298.04660479, 2682.88389872, ...,  860.44886061,
         1506.08524299, 2133.81433639]])]
```