Homework 6 - Support Vector Machines

March 8, 2020

Student: Dimitrios Tanoglidis

[1]:
```
#Import stuff
import numpy as np
import pandas as pd
import itertools
from sklearn import linear_model
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import rcParams
#import seaborn as sns
rcParams['font.family'] = 'serif'

# Adjust rc parameters to make plots pretty
def plot_pretty(dpi=200, fontsize=8):

    import matplotlib.pyplot as plt

    plt.rc("savefig", dpi=dpi)
    plt.rc('text', usetex=True)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=10)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=10)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [0.5, 1.1])

    return
plot_pretty()
```

1 Conceptual exercises

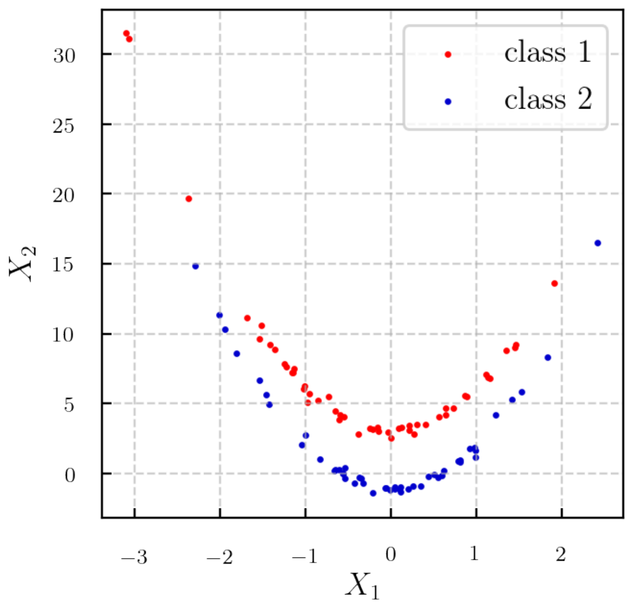
1.1 Non-linear separation

Let's generate a simulated two-class dataset with 100 observations and a non-linear separation between the two classes.

[2]:
```
np.random.seed(seed=2046)

X_1 = np.random.normal(0,1,100) # Draw from a random distribution
X_2 = 3.0*(X_1**2.0)+1. + np.random.normal(0,0.3,100)

X_2[0:50] = X_2[0:50] + 2.0
X_2[50:] = X_2[50:] - 2.0
# Create also a class model
Y = np.zeros(100)
Y[50:] = 1.0
```

Let's plot our dataset to see how it looks like.

[3]:
```
plt.figure(figsize=(3.5,3.5))

plt.scatter(X_1[Y==0.],X_2[Y==0.0], color='red', s=2., label='class 1')
plt.scatter(X_1[Y==1.],X_2[Y==1.0], color='mediumblue', s=2., label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



Let's try a linear support vector classifier first

[4]:
```
# Construct a feature matrix
X_ft = np.zeros([100,2])
X_ft[:,0] = X_1; X_ft[:,1] = X_2
```

Let's create a grid in the $X_1 - X_2$ space as well.

[5]:
```
x1 = np.linspace(X_1.min()-1,X_1.max()+1,100)
x2 = np.linspace(X_2.min()-1,X_2.max()+1,100)
x1g, x2g = np.meshgrid(x1,x2)

# Define feature matrix of the grid space
x_ft_gr = np.zeros([10000,2])
x_ft_gr[:,0] = x1g.ravel();x_ft_gr[:,1] = x2g.ravel()
```

[6]:
```
from sklearn.model_selection import train_test_split
# Split in train/test sets
X_train, X_test, y_train, y_test = train_test_split(X_ft, Y,
                                                        test_size=0.50,␣
 ↪random_state=42)
```

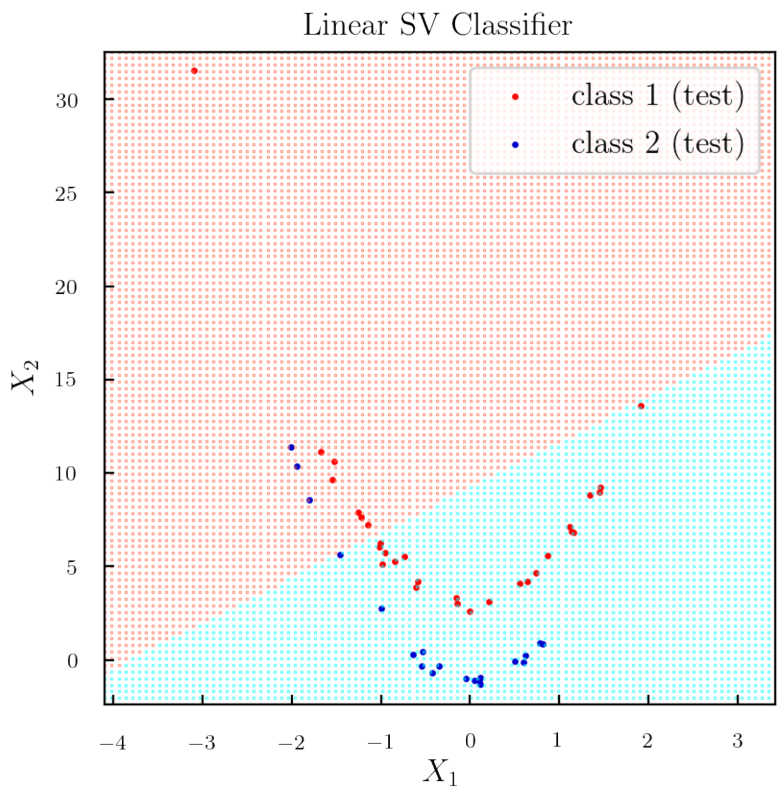
[7]:
```
from sklearn.svm import LinearSVC

Lin_SVM = LinearSVC()
# Fit on the training set
Lin_SVM.fit(X_train,y_train)
```

```
# Predict on the test set and the grid
y_lin_pr_test = Lin_SVM.predict(X_test) # Predict on the test set
y_lin_pr_grid = Lin_SVM.predict(x_ft_gr) # Predict on the grid
#Reshape the grid
y_lin_pr_grid = y_lin_pr_grid.reshape(100,100)
```

```
//anaconda/envs/python2/lib/python2.7/site-packages/sklearn/svm/base.py:931:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Let's plot to see how well the linear SVM can separate the two classes.

[8]:
```
X_test_1 = X_test[:,0]
X_test_2 = X_test[:,1]
plt.figure(figsize=(4.5,4.5))

#print the test data
plt.scatter(X_test_1[y_test==0.],X_test_2[y_test==0.0], color='red', s=2.,␣
 ↪label='class 1 (test)')
plt.scatter(X_test_1[y_test==1.],X_test_2[y_test==1.0], color='mediumblue', s=2.
 ↪, label='class 2 (test)')

# Print grid
plt.scatter(x1g[y_lin_pr_grid==0.],x2g[y_lin_pr_grid==0.], s=0.05,␣
 ↪color='tomato')
plt.scatter(x1g[y_lin_pr_grid==1.],x2g[y_lin_pr_grid==1.], s=0.05, color='cyan')

plt.xlim(X_1.min()-1,X_1.max()+1)
plt.ylim(X_2.min()-1,X_2.max()+1)
plt.title('Linear SV Classifier', fontsize=12)
plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



We can see that the linear classifier does not perform very well in the test data. Let's get the error rates at the trainin and test sets.

[9]:
```
from sklearn.metrics import accuracy_score as ACC
# Predict on the training set
y_lin_pr_train = Lin_SVM.predict(X_train)

# Let's get the error rates - these are 1 - accuracy score
Error_train = 1.0 - ACC(y_train,y_lin_pr_train )
Error_test = 1.0 - ACC(y_test,y_lin_pr_test)

print('Training error rate:', Error_train)
print('Testing error rate:', Error_test)
```

```
('Training error rate:', 0.33999999999999997)
('Testing error rate:', 0.52)
```

Now let's try a polynomial kernel

I will leave all the other parameters in their default values.

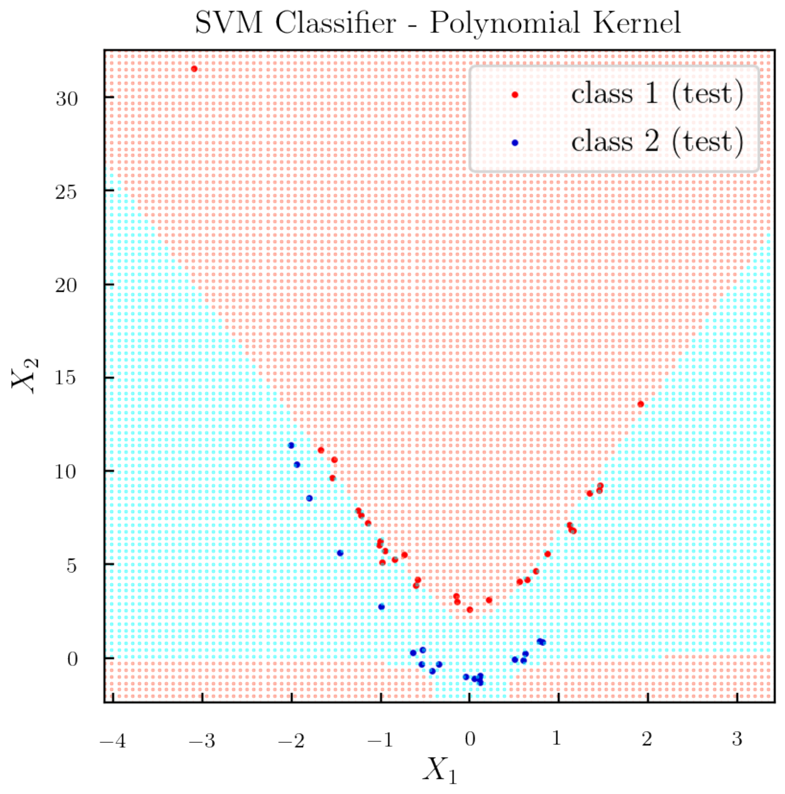
[10]:
```
from sklearn.svm import SVC
```

```
SVC_poly = SVC(kernel='poly', C=100)

# Fit on the trainin data, and predict on the training, test and grid
SVC_poly.fit(X_train,y_train) #Fit
y_poly_pr_train = SVC_poly.predict(X_train) # Predict on the training set
y_poly_pr_test = SVC_poly.predict(X_test) # Predict on the test set
y_poly_pr_grid = SVC_poly.predict(x_ft_gr)
#Reshape the grid
y_poly_pr_grid = y_poly_pr_grid.reshape(100,100)
```

```
//anaconda/envs/python2/lib/python2.7/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Let's plot to see the decision boundaries and the classification results, on the test set.

[11]:
```
plt.figure(figsize=(4.5,4.5))

#print the test data
plt.scatter(X_test_1[y_test==0.],X_test_2[y_test==0.0], color='red', s=2.,␣
 ↪label='class 1 (test)')
plt.scatter(X_test_1[y_test==1.],X_test_2[y_test==1.0], color='mediumblue', s=2.
 ↪, label='class 2 (test)')

# Print grid
plt.scatter(x1g[y_poly_pr_grid==0.],x2g[y_poly_pr_grid==0.], s=0.05,␣
 ↪color='tomato')
plt.scatter(x1g[y_poly_pr_grid==1.],x2g[y_poly_pr_grid==1.], s=0.05,␣
 ↪color='cyan')

plt.xlim(X_1.min()-1,X_1.max()+1)
plt.ylim(X_2.min()-1,X_2.max()+1)
plt.title('SVM Classifier - Polynomial Kernel', fontsize=12)
plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



This seems to perform much better, let's calculate and present the error rates.

[12]:
```python
# Let's get the error rates - these are 1 - accuracy score
Error_train = 1.0 - ACC(y_train,y_poly_pr_train )
Error_test = 1.0 - ACC(y_test,y_poly_pr_test)

print('Training error rate:', Error_train)
print('Testing error rate:', Error_test)
```

```
('Training error rate:', 0.09999999999999998)
('Testing error rate:', 0.38)
```

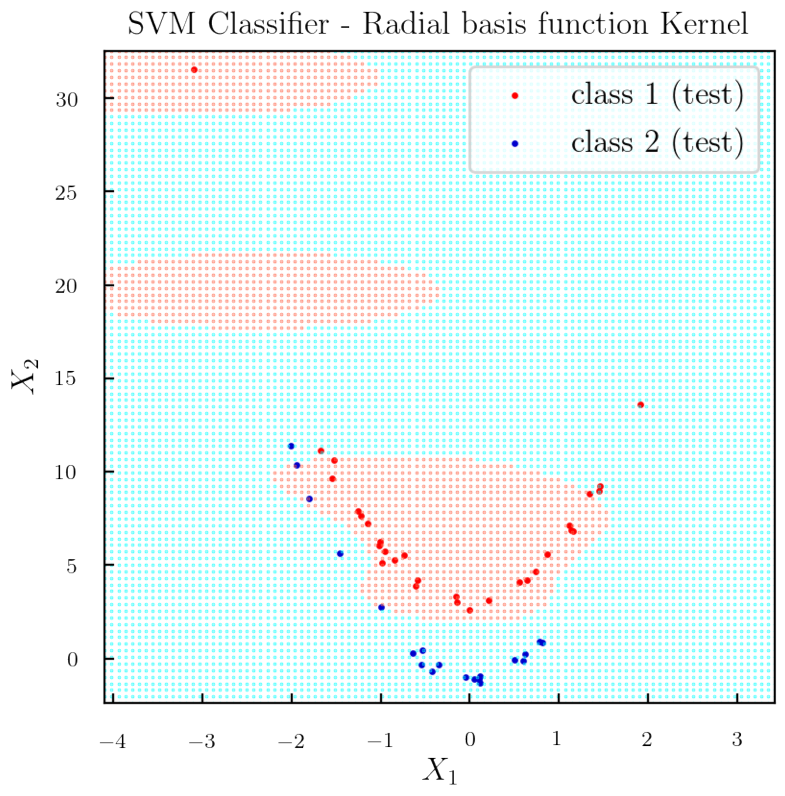
Radial Basis Function Kernel

Let's finally check a radial kernel

[13]:
```python
SVC_rbf = SVC(kernel='rbf',gamma=0.1, C=100)

# Fit on the trainin data, and predict on the training, test and grid
SVC_rbf.fit(X_train,y_train) #Fit
y_rbf_pr_train = SVC_rbf.predict(X_train) # Predict on the training set
y_rbf_pr_test = SVC_rbf.predict(X_test) # Predict on the test set
y_rbf_pr_grid = SVC_rbf.predict(x_ft_gr)
```

```
#Reshape the grid
y_rbf_pr_grid = y_rbf_pr_grid.reshape(100,100)
```

[14]:
```
plt.figure(figsize=(4.5,4.5))

#print the test data
plt.scatter(X_test_1[y_test==0.],X_test_2[y_test==0.0], color='red', s=2.,␣
 ↪label='class 1 (test)')
plt.scatter(X_test_1[y_test==1.],X_test_2[y_test==1.0], color='mediumblue', s=2.
 ↪, label='class 2 (test)')

# Print grid
plt.scatter(x1g[y_rbf_pr_grid==0.],x2g[y_rbf_pr_grid==0.], s=0.05,␣
 ↪color='tomato')
plt.scatter(x1g[y_rbf_pr_grid==1.],x2g[y_rbf_pr_grid==1.], s=0.05, color='cyan')

plt.xlim(X_1.min()-1,X_1.max()+1)
plt.ylim(X_2.min()-1,X_2.max()+1)
plt.title('SVM Classifier - Radial basis function Kernel', fontsize=12)
plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



[15]:
```python
# Let's get the error rates - these are 1 - accuracy score
Error_train = 1.0 - ACC(y_train,y_rbf_pr_train )
Error_test = 1.0 - ACC(y_test,y_rbf_pr_test)

print('Training error rate:', Error_train)
print('Test error rate:', Error_test)
```

```
('Training error rate:', 0.0)
('Test error rate:', 0.14)
```

The Radial Basis function gives the best test error rate of $\sim 14\%$.

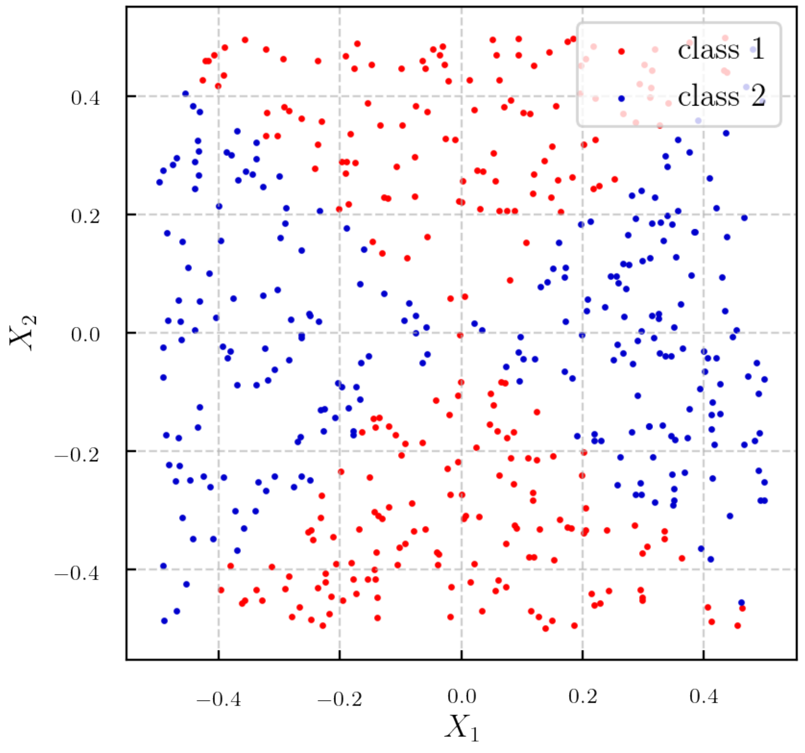
1.2 SVM vs. Logistic Regression

1.2.1 Data Generation

[16]:
```python
n_size = 500
X_1 = np.random.random(n_size) - 0.5
X_2 = np.random.random(n_size) - 0.5

Y = np.zeros(500)
# Non-linear separation boundary, following the suggestion of the book
# "statistical learning with applications "
Y[(X_1**2.0-X_2**2.0)>0.0] = 1.0
```

1.2.2 Color-coded plot

[17]:
```python
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[Y==0.],X_2[Y==0.0], color='red', s=2., label='class 1')
plt.scatter(X_1[Y==1.],X_2[Y==1.0], color='mediumblue', s=2., label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



As we can see there is an overlapping, non-linear boundary between the two classes.

1.2.3 Logistic Regression model

```
[18]: from sklearn.linear_model import LogisticRegression as LR

      # Create a feature matrix consisted of X_1, X_2
      X_ft = np.zeros([500,2])
      X_ft[:,0] = X_1;X_ft[:,1] = X_2

      LR_model = LR()
      # Fit and predict on the train data
      LR_model.fit(X_ft,Y)#Fit
      y_pr_LR = LR_model.predict(X_ft)

      ERR = 1.0 - ACC(y_pr_LR, Y)
      print('Error rate for the Logistic Regression model:', ERR)
```

```
('Error rate for the Logistic Regression model:', 0.382)
```

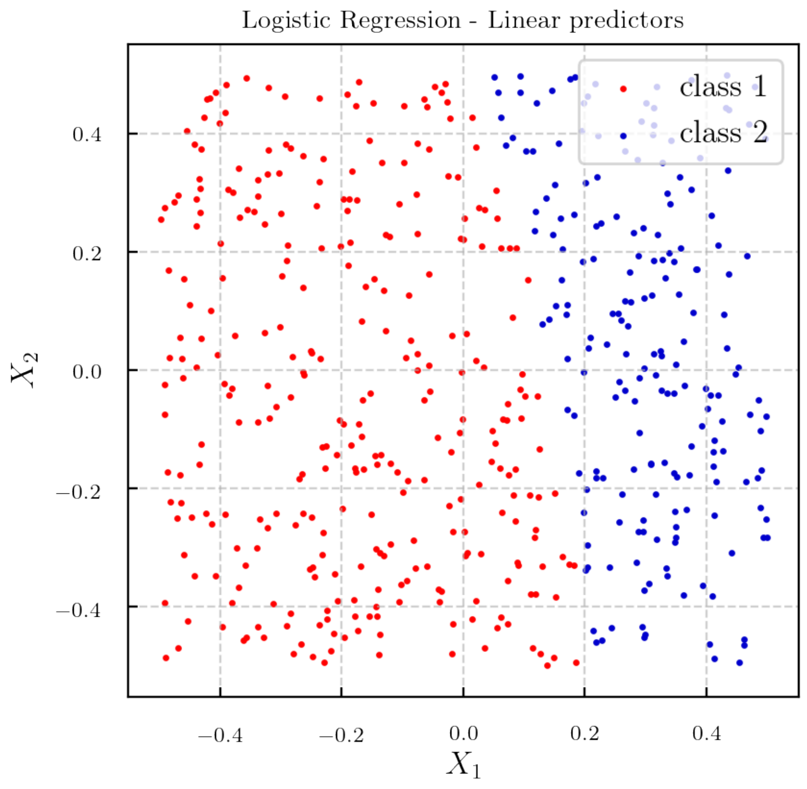
```
//anaconda/envs/python2/lib/python2.7/site-
packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver
```

```
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

We see that the Logistic Regression model gives a very high error rate.

1.2.4 Plot predictions

```
[19]: plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[y_pr_LR==0.],X_2[y_pr_LR==0.0], color='red', s=2., label='class␣
 ↪1')
plt.scatter(X_1[y_pr_LR==1.],X_2[y_pr_LR==1.0], color='mediumblue', s=2.,␣
 ↪label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Logistic Regression - Linear predictors')
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



Almost all of them are predicted to belong to class 1.

1.2.5 Logistic Regression with non-linear predictors

Here I will use as predictors the following combinations: $X_1^2, X_1 \times X_2, X_2^2$.

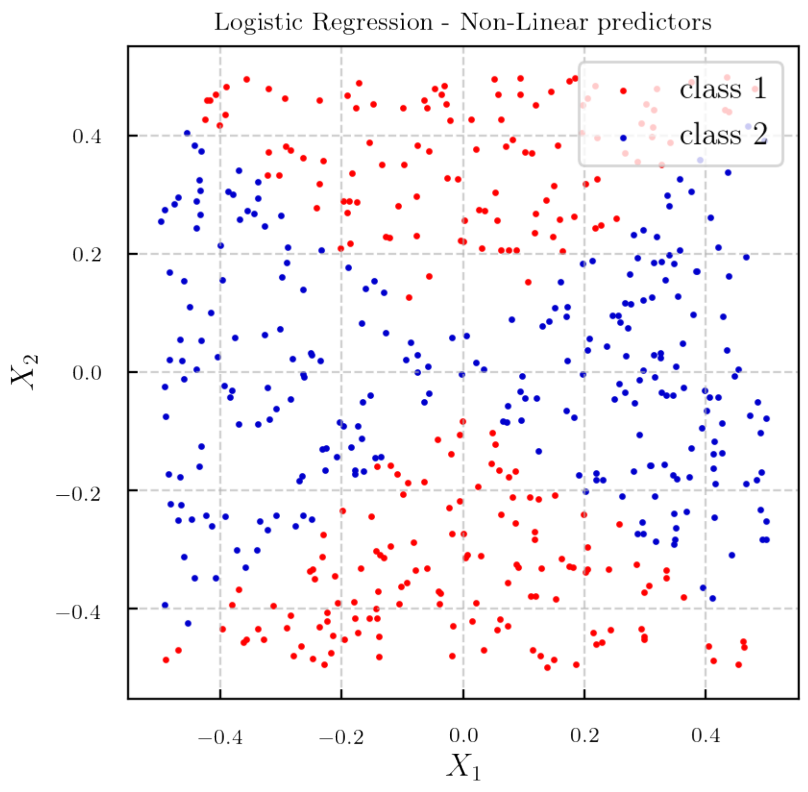
[20]:
```
X_1_sq = X_1**2.0
prod = X_1*X_2
X_2_sq = X_2**2.0

# Create a feature matrix of non-linear combinations
X_ft_nl = np.zeros([500,3])
X_ft_nl[:,0] = X_1_sq;X_ft_nl[:,1] = prod;X_ft_nl[:,2] = X_2_sq

# Fit and predict
LR_model = LR()
# Fit and predict on the train data
LR_model.fit(X_ft_nl,Y)#Fit
y_pr_LR_nl = LR_model.predict(X_ft_nl)

ERR = 1.0 - ACC(y_pr_LR_nl, Y)
print('Error rate for the Logistic Regression model (non-linear predictors):',␣
 ↪ERR)
```

```
('Error rate for the Logistic Regression model (non-linear predictors):',
0.038000000000000034)
```

Now the error rate is very small now.

1.2.6 Plot the predictions

[21]:
```
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[y_pr_LR_nl==0.],X_2[y_pr_LR_nl==0.0], color='red', s=2.,␣
 ↪label='class 1')
plt.scatter(X_1[y_pr_LR_nl==1.],X_2[y_pr_LR_nl==1.0], color='mediumblue', s=2.,␣
 ↪label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Logistic Regression - Non-Linear predictors')
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



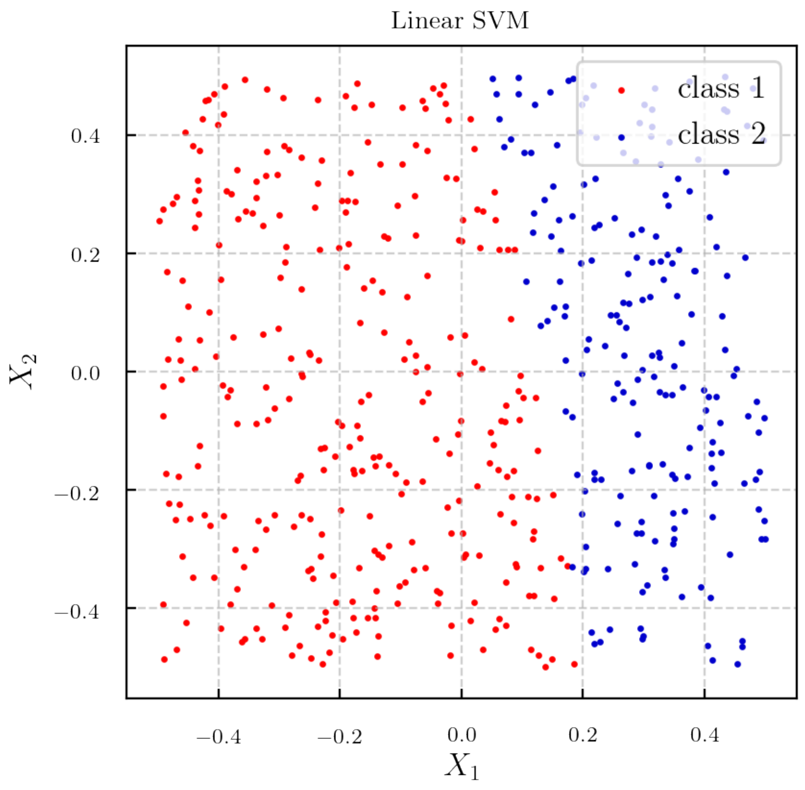
1.2.7 Support Vector Classifier - Linear Kernel

[23]:
```
Lin_SVC = LinearSVC() #Linear Support Vector Classifier

# Fit
Lin_SVC.fit(X_ft,Y)
# Predict
y_lin_SVC_pr = Lin_SVC.predict(X_ft)

# Calculate and report error
ERR = 1.0 - ACC(y_lin_SVC_pr, Y)

print('Linear SVM error:', ERR)
```

```
('Linear SVM error:', 0.386)
```

[24]:
```
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[y_lin_SVC_pr==0.],X_2[y_lin_SVC_pr==0.0], color='red', s=2.,␣
 ↪label='class 1')
```

```
plt.scatter(X_1[y_lin_SVC_pr==1.],X_2[y_lin_SVC_pr==1.0], color='mediumblue',␣
↪s=2., label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Linear SVM')
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



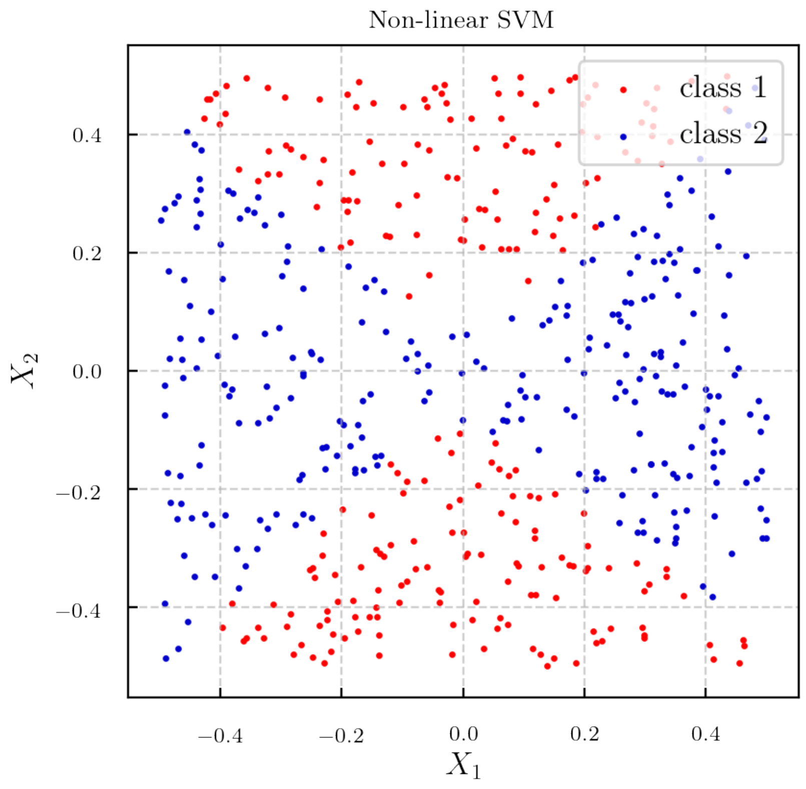
1.2.8 Non-Linear SVM (rbf)

[25]:
```
SVC_rbf = SVC(kernel='rbf',gamma=0.1, C=100)

# Fit
SVC_rbf.fit(X_ft,Y)
# Predict
y_rbf_SVC_pr = SVC_rbf.predict(X_ft)

# Calculate and report error
```

```
ERR = 1.0 - ACC(y_rbf_SVC_pr, Y)

print('Non-Linear SVM error:', ERR)
```

```
('Non-Linear SVM error:', 0.04400000000000004)
```

[27]:
```
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[y_rbf_SVC_pr==0.],X_2[y_rbf_SVC_pr==0.0], color='red', s=2.,␣
↪label='class 1')
plt.scatter(X_1[y_rbf_SVC_pr==1.],X_2[y_rbf_SVC_pr==1.0], color='mediumblue',␣
↪s=2., label='class 2')

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Non-linear SVM')
plt.legend(loc='upper right', fontsize=12)
plt.show()
```



1.2.9 Discussion

We see that SVM with a radial basis function kernel is very good in finding a non-linear decision boundary, without requiring to engineer non-linear features (as in the case of Logistic Regression) to achieve that.

1.3 Tuning Cost

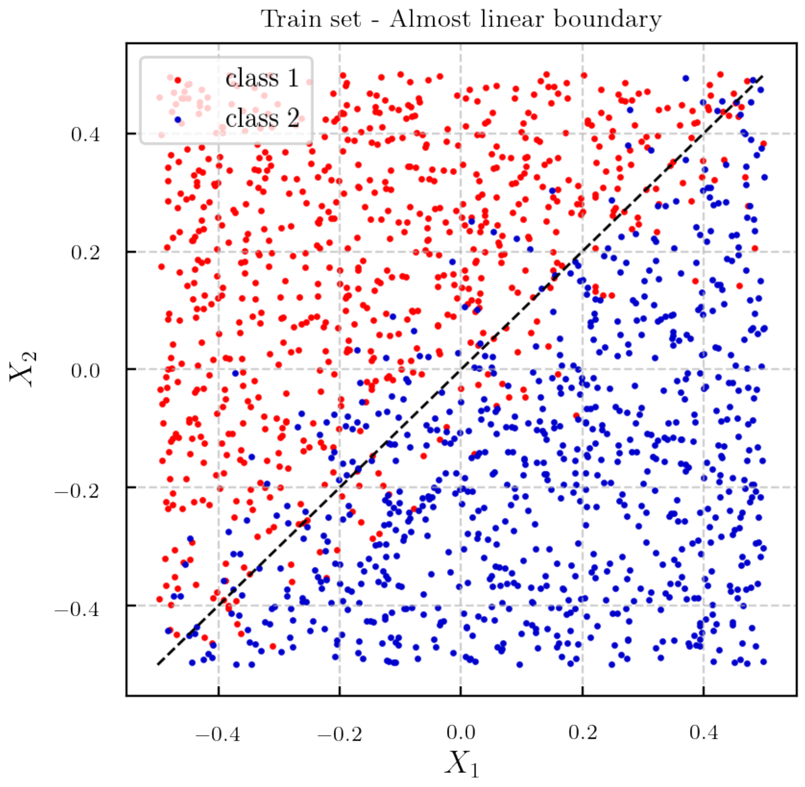
1.3.1 Dataset creation

Here I create synthetic two-class data with $p = 2$ that is barely linearly separable.

The boundary will be $y = 0.9x + err$

[92]:
```
n_size = 1500
X_1 = np.random.random(n_size) - 0.5
X_2 = np.random.random(n_size) - 0.5

Y = np.zeros(1500)
diff = 0.9*X_1 - X_2 + 0.4*np.random.normal(0,0.3,1500)
Y[diff>0] = 1.0
```

[93]:
```
x_lin = np.linspace(-0.5,0.5,100)
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1[Y==0.],X_2[Y==0.0], color='red', s=2., label='class 1')
plt.scatter(X_1[Y==1.],X_2[Y==1.0], color='mediumblue', s=2., label='class 2')
plt.plot(x_lin,x_lin,ls='--',c='k',linewidth=1.0)

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Train set - Almost linear boundary')
plt.legend(loc='upper left', fontsize=10)
plt.show()
```

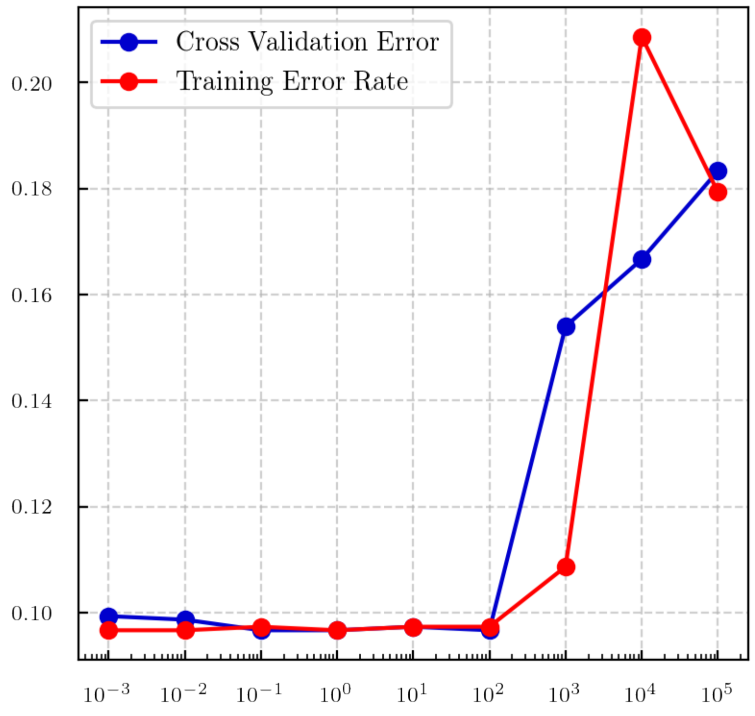


1.3.2 Cross Validation error rates

[94]:
```
from sklearn.metrics import zero_one_loss as Error_Rate
from sklearn.model_selection import cross_val_score as CV_score
```

[95]:
```
# Create feature matrix
X_ft = np.zeros([1500,2])
X_ft[:,0] = X_1;X_ft[:,1] = X_2

# Let's create an array of costs
Costs = [0.001,0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Cross_val_errors = np.zeros(9)
train_err = np.zeros(9)


for i in range(9):
    # Define the classifier
    Lin_SVC_c = LinearSVC(C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(Lin_SVC_c, X_ft, Y, cv=10, scoring='accuracy')
    # Cross-Validation error rate
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    Cross_val_errors[i] = CV_err_SVC
    # ================================
    # ================================
    # Calculate training errors
    # Fit on the model
    Lin_SVC_c.fit(X_ft,Y)
    # Predict
    y_pred_c = Lin_SVC_c.predict(X_ft)
    train_err[i] = Error_Rate(Y,y_pred_c)
```

[96]:
```
plt.figure(figsize=(4.5,4.5))

plt.plot(Costs,Cross_val_errors,c='mediumblue',marker='o',label='Cross␣
 ↪Validation Error')
plt.plot(Costs,train_err,c='red',marker='o', label='Training Error Rate')


plt.xscale('log')
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='upper left', fontsize=10)
plt.show()
```



For the number of errors, I can just multiply the error rate with the total number of samples.

[97]:
```
Err_numb = 1500*train_err
print(Err_numb)
```

```
[145. 145. 146. 145. 146. 146. 163. 313. 269.]
```

The minimum number of errors is achieved for $C = 0.01$. The minumum cross-validation error rate is achieved for $C = 0.1$.

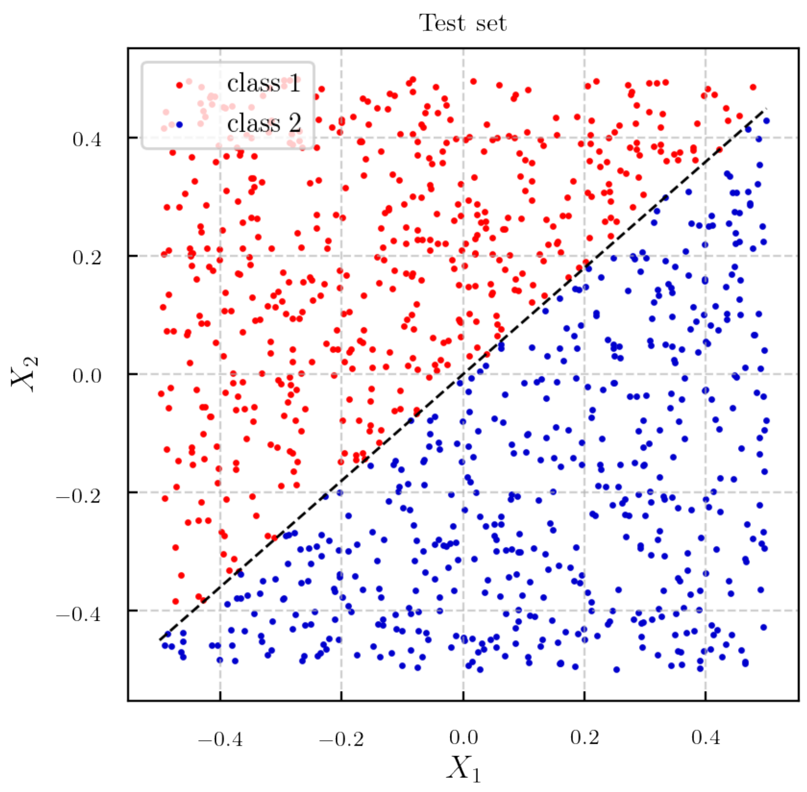
1.3.3 Errors of a test set

For the test set I will not have any error; the boundary will be $y = 0.9x$.

[98]:
```
n_test = 1000
X_1_test = np.random.random(n_test) - 0.5
X_2_test = np.random.random(n_test) - 0.5

Y_test = np.zeros(1000)
diff_test = 0.9*X_1_test - X_2_test
Y_test[diff_test>0] = 1.0
```

Let's plot it.

[99]:
```
plt.figure(figsize=(4.5,4.5))

plt.scatter(X_1_test[Y_test==0.],X_2_test[Y_test==0.0], color='red', s=2.,␣
 ↪label='class 1')
plt.scatter(X_1_test[Y_test==1.],X_2_test[Y_test==1.0], color='mediumblue', s=2.
 ↪, label='class 2')
plt.plot(x_lin,0.9*x_lin,ls='--',c='k',linewidth=1.0)

plt.xlabel('$X_1$',fontsize=12)
plt.ylabel('$X_2$',fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.title('Test set')
plt.legend(loc='upper left', fontsize=10)
plt.show()
```

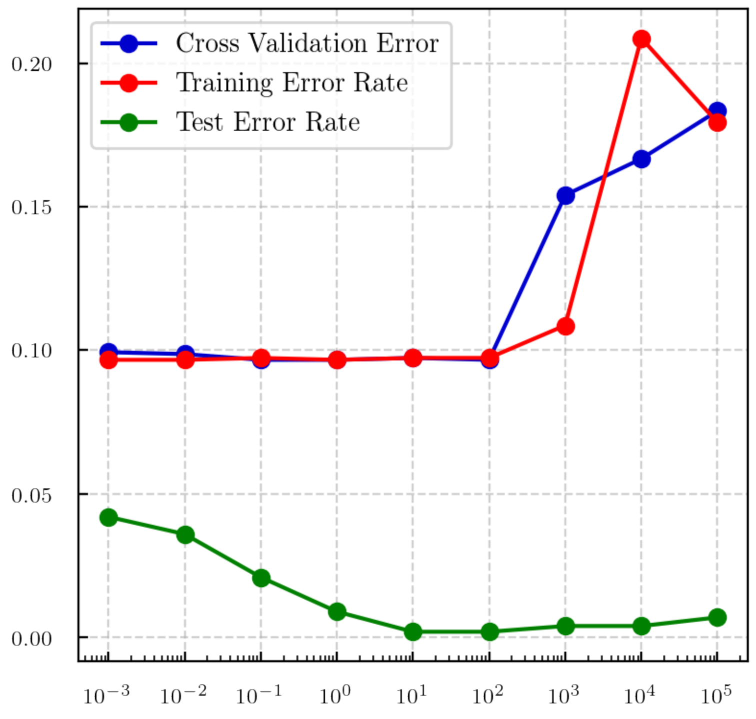


Now the separation is perfect. Let's predict

```
[101]: Test_error_rate = np.zeros(9)
       X_ft_test = np.zeros([1000,2])
       X_ft_test[:,0] = X_1_test;X_ft_test[:,1] = X_2_test
       for i in range(9):
           # Define the classifier
           Lin_SVC_c = LinearSVC(C=Costs[i])
           # ================================
           # Calculate test errors
           # Fit on the model
           Lin_SVC_c.fit(X_ft_test,Y_test)
           # Predict
           y_pred_c = Lin_SVC_c.predict(X_ft_test)
           Test_error_rate[i] = Error_Rate(Y_test,y_pred_c)
```

```
[102]: plt.figure(figsize=(4.5,4.5))

       plt.plot(Costs,Cross_val_errors,c='mediumblue',marker='o',label='Cross␣
        ↪Validation Error')
       plt.plot(Costs,train_err,c='red',marker='o', label='Training Error Rate')
       plt.plot(Costs,Test_error_rate,c='green',marker='o', label='Test Error Rate')
```

```python
plt.xscale('log')
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='upper left', fontsize=10)
plt.show()
```



Test error rate is minimized for $C = 10$.

1.3.4 Discussion

Generally we see that the liner Support Vector classifier performs well in separating data taht are barely linearly separable. We also noticed a case of overfitting of the cost value; the best results for the test set are do not correspond to the same value of C as those for the training set.

2 Application: Predicting attitudes towards racist college proffessors

Let's read the data first

[103]:
```
df_train = pd.read_csv('gss_train.csv') #Train data
df_test = pd.read_csv('gss_test.csv') #Test data

#df_train.head()
```

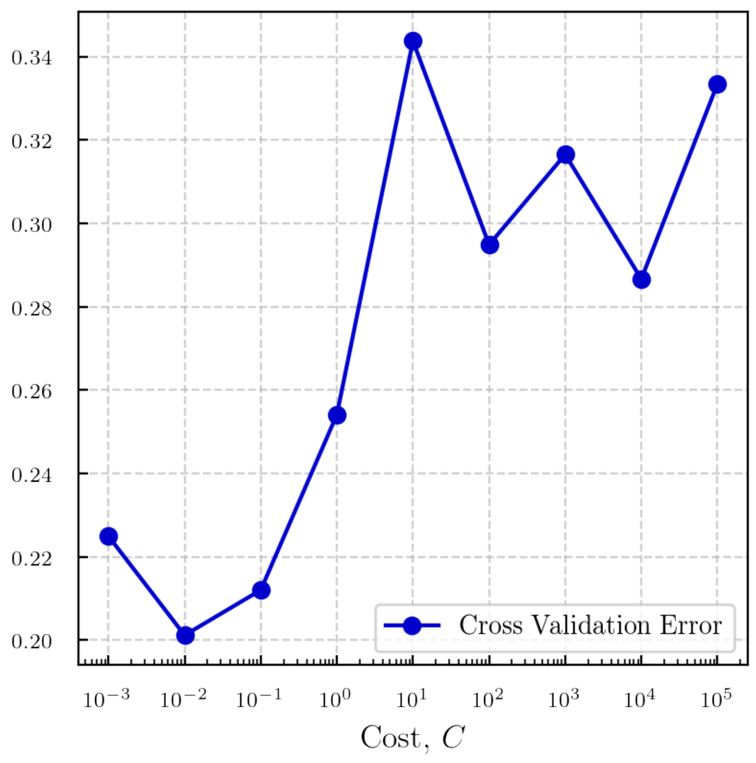
Get the labels, corresponding to the variable `colrac` and the fatures, corresponding to all the other columns of the dataframes.

[104]:
```
# Get the train/test target values
y_train = df_train['colrac'].values
y_test = df_test['colrac'].values
# Get the train/test features
X_train = df_train.loc[:, df_train.columns != 'colrac'].values
X_test = df_test.loc[:, df_test.columns != 'colrac'].values
```

2.1 Linear Support Vector Classifier

[105]:
```
# Let's create an array of costs
Costs = [0.001,0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
Cross_val_errors = np.zeros(9)

for i in range(9):
    # Define the classifier
    Lin_SVC_c = LinearSVC(C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(Lin_SVC_c, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    Cross_val_errors[i] = CV_err_SVC
```

[113]:
```
plt.figure(figsize=(4.5,4.5))

plt.plot(Costs,Cross_val_errors,c='mediumblue',marker='o',label='Cross␣
↪Validation Error')

plt.xscale('log')
plt.xlabel('Cost, $C$', fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='lower right', fontsize=10)
plt.show()
```



The Cross Validation error is minimized for $C = 0.01$. Let's use this to calculate and predict on the test set.

[111]:
```
Lin_SVC_best = LinearSVC(C=0.01)
# Fit
Lin_SVC_best.fit(X_train,y_train)
# Predict on the test set
y_pr_test = Lin_SVC_best.predict(X_test)

# Calculate error rate on the test set
Test_error_rate = Error_Rate(y_test,y_pr_test)

print('Test error rate is:',Test_error_rate)
```

```
('Test error rate is:', 0.2190669371196755)
```

We have a $\sim 22\%$ error rate on the test set.

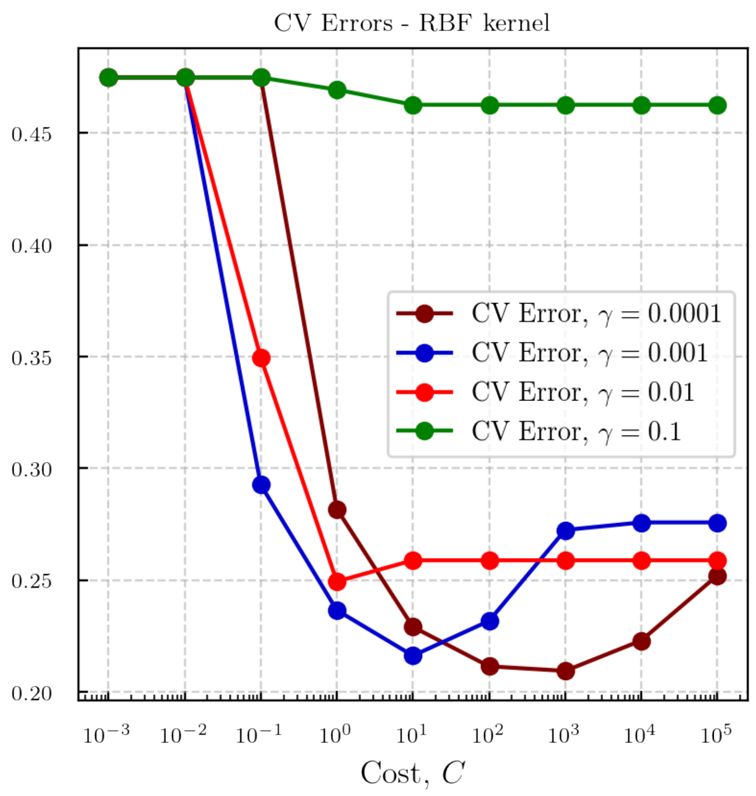
2.2 Radial and Polynomial basis kernels

2.2.1 Radial Basis Kernel

Let's start with radial basis kernel. I'll make plots of the Cross Validation error for the same range of cost values C, as before, and three values of $\gamma : 0.0001, 0.001, 0.01, 0.1$.

[126]:
```
# Let's create an array of costs
Costs = [0.001,0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
CV_error_0001 = np.zeros(9) # CV errors for gamma = 0.001
CV_error_001 = np.zeros(9) # CV errors for gamma = 0.001
CV_error_01  = np.zeros(9) # CV errors for gamma = 0.01
CV_error_1 = np.zeros(9) # CV errors for gamma = 0.1
# =======================================================
# =======================================================
# For gamma = 0.0001
for i in range(9):
    # Define the classifier
    SVC_rbf = SVC(kernel='rbf',gamma=0.0001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_rbf, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_0001[i] = CV_err_SVC
# =======================================================
# =======================================================
# For gamma = 0.001
for i in range(9):
    # Define the classifier
    SVC_rbf = SVC(kernel='rbf',gamma=0.001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_rbf, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_001[i] = CV_err_SVC
# ==========================================================
# ==========================================================
# For gamma = 0.01
for i in range(9):
    # Define the classifier
    SVC_rbf = SVC(kernel='rbf',gamma=0.01,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_rbf, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_01[i] = CV_err_SVC
    # ==========================================================
# ===========================================================
# For gamma = 0.1
for i in range(9):
    # Define the classifier
    SVC_rbf = SVC(kernel='rbf',gamma=0.1,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_rbf, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_1[i] = CV_err_SVC
```

[128]:
```
plt.figure(figsize=(4.5,4.5))

plt.plot(Costs,CV_error_0001,c='maroon',marker='o',label=r'CV Error, $\gamma=0.
 ↪0001$')
plt.plot(Costs,CV_error_001,c='mediumblue',marker='o',label=r'CV Error,␣
 ↪$\gamma=0.001$')
plt.plot(Costs,CV_error_01,c='red',marker='o',label=r'CV Error, $\gamma=0.01$')
plt.plot(Costs,CV_error_1,c='green',marker='o',label=r'CV Error, $\gamma=0.1$')

plt.xscale('log')
plt.xlabel('Cost, $C$', fontsize=12)
plt.title('CV Errors - RBF kernel')
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='center right', fontsize=10)
plt.show()
```



The minimun CV error is achieved for $C = 1000$ and $\gamma = 0.0001$.

Let's predict and calculate the error rate for this best classifier on the test set.

[132]:
```
SVC_rbf_best = SVC(kernel='rbf',gamma=0.0001,C=1000)
SVC_rbf_best.fit(X_train,y_train)# Fit
y_rbf_best_pr  = SVC_rbf_best.predict(X_test) # Predict on the test set

# Calculate error rate
Test_error_rate = Error_Rate(y_test,y_rbf_best_pr)

print('Test error rate is:',Test_error_rate)
```

```
('Test error rate is:', 0.1947261663286004)
```

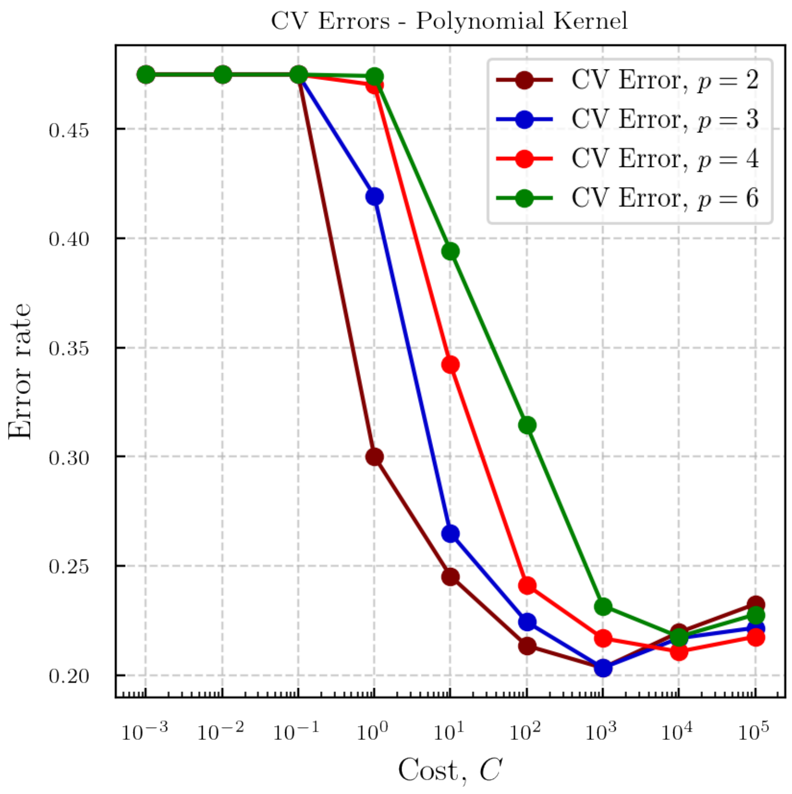
2.2.2 Polynomial Kernel

For the polynomial kernel I will fix $\gamma = 0.0001$ and check for different levels of cost, C, and polynomial orders $p = 2, 3, 4, 5$.

[133]:
```
# Let's create an array of costs
Costs = [0.001,0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
```

```
CV_error_p2 = np.zeros(9) # CV errors for p=2
CV_error_p3 = np.zeros(9) # CV errors for p=3
CV_error_p4  = np.zeros(9) # CV errors for p=4
CV_error_p5 = np.zeros(9) # CV errors for p=5
# =======================================================
# =======================================================
# For p=2
for i in range(9):
    # Define the classifier
    SVC_poly = SVC(kernel='poly',degree=2,gamma=0.0001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_poly, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_p2[i] = CV_err_SVC
# =======================================================
# =======================================================
# For p=3
for i in range(9):
    # Define the classifier
    SVC_poly = SVC(kernel='poly',degree=3,gamma=0.0001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_poly, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_p3[i] = CV_err_SVC
# =======================================================
# =======================================================
# For p=4
for i in range(9):
    # Define the classifier
    SVC_poly = SVC(kernel='poly',degree=4,gamma=0.0001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_poly, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_p4[i] = CV_err_SVC
# =======================================================
# =======================================================
# For p=5
for i in range(9):
    # Define the classifier
    SVC_poly = SVC(kernel='poly',degree=5,gamma=0.0001,C=Costs[i])
    # Get cross-validation scores
    scores_SVC = CV_score(SVC_poly, X_train, y_train, cv=10, scoring='accuracy')
    # ====================================
    CV_err_SVC = 1.0 - np.mean(scores_SVC)
    CV_error_p5[i] = CV_err_SVC
```

[138]:
```
plt.figure(figsize=(4.5,4.5))

plt.plot(Costs,CV_error_p2,c='maroon',marker='o',label='CV Error, $p=2$')
plt.plot(Costs,CV_error_p3,c='mediumblue',marker='o',label='CV Error, $p=3$')
plt.plot(Costs,CV_error_p4,c='red',marker='o',label='CV Error, $p=4$')
plt.plot(Costs,CV_error_p5,c='green',marker='o',label='CV Error, $p=6$')

plt.xscale('log')
plt.xlabel('Cost, $C$', fontsize=12)
plt.ylabel('Error rate', fontsize=12)
plt.title('CV Errors - Polynomial Kernel')
plt.grid(ls='--', alpha=0.6)
plt.legend(loc='upper right', fontsize=10)
plt.show()
```



We get the minimum error rate for a polynomial order $p = 3$ and $C = 1000$.

Let's check the error rate on the test set for this best configuration.

[140]:
```
SVC_poly_best = SVC(kernel='poly',degree=3,gamma=0.0001,C=1000)
# Fit
SVC_poly_best.fit(X_train,y_train)# Fit
y_poly_best_pr  = SVC_poly_best.predict(X_test) # Predict on the test set

# Calculate error rate
Test_error_rate = Error_Rate(y_test,y_poly_best_pr)

print('Test error rate is:',Test_error_rate)
```

```
('Test error rate is:', 0.19066937119675453)
```

The test error rate is $\sim 19\%$.

The best results on the test set come from the polynomial kernel, reaching a test error rate of 19%; the radial basis function kernel performs slightly worse with an error rate on the test set of the order of $\sim 19.5\%$

[ ]:
```

```