# Classification Systems

Daniel Cerdán, Fernando Freire

March 21, 2019

## Contents

# 1 Classification Systems

In this practical, you are asked to compare the prediction error of:

1. The Naive Bayes Classifier
2. LDA
3. QDA
4. Nearest Shrunken Centroids Classifier

On the Breast Cancer dataset provided in the previous notebooks, and the Prostate cancer dataset attached. The details about this last dataset are found in the reference:

Singh, D., Febbo, P., Ross, K., Jackson, D., Manola, J., Ladd, C., Tamayo, P., Renshaw, A., D'Amico, A., Richie, J., Lander, E., Loda, M., Kantoff, P., Golub, T., & Sellers, W. (2002). Gene expression correlates of clinical prostate cancer behavior. Cancer Cell, 1, 203–209.

This dataset is in CSV format and the last column contains the class label. The task of interest is to discriminate between normal and tumor tissue samples.

Importantly:

Use a random split of 2 / 3 of the data for training and 1 / 3 for testing each classifier. Any hyper-parameter of each method should be tuned using a grid-search guided by an inner cross-validation procedure that uses only training data. To reduce the variance of the estimates, report average error results over 20 different partitions of the data into training and testing as described above. Submit a notebook showing the code and the results obtained. Give some comments about the results and respond to these questions:

What method performs best on each dataset? What method is more flexible? What method is more robust to over-fitting?

Script 1.0.1 (python)

```
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib as mpl
from matplotlib import colors
import seaborn as sns; sns.set()
import scipy.stats as stats
import scipy as sp
from scipy import linalg
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import NearestCentroid
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold, GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, make_scorer, confusion_matrix
```

## 1.1 Methods

Script 1.1.1 (python)

```python
def create_datasets_from_file(data_file, header, label_pos, label_value, features_ini, features_fin=None):
    """
    """
    data = pd.read_csv(data_file, header = header)
    if features_fin == None:
        X = data.values[ :, features_ini:].astype(np.float)
    else:
        X = data.values[ :, features_ini:features_fin].astype(np.float)
    y = (data.values[ :, label_pos ] == label_value).astype(np.int)
    # Split dataset between training and test
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1.0/3, random_state=1)
    # Data standardization
    scaler = preprocessing.StandardScaler().fit(x_train)
    x_train_scaled = scaler.transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    # Check standardization
    for i in range (1, np.size(x_train_scaled,1)):
        assert round(np.var(x_train_scaled[:,0]),3) == round(np.var(x_train_scaled[:,i]),3),\
        "Warning: revise data standardization"

    return x_train_scaled, x_test_scaled, y_train, y_test

def prediction_accuracy(x_train, x_test, y_train, y_test, method_func, method_param, param_value):
    """
    """
    if method_param != "" :
        params = {method_param : param_value}
    else:
        params ={}
    method = globals()[method_func](**params)

    # Training
    method.fit(x_train, y_train)

    # Prediction
    y_pred = method.predict(x_test)
    conf = confusion_matrix(y_test, y_pred)
    TN = conf[0][0]
    TP = conf[1][1]
    FP = conf[0][1]
    FN = conf[1][0]
    print(conf)
    print('Predicion accuracy is: %f' % ((TP + TN) / (TN + TP + FP + FN)))
    print('True postive rate is: %f' % (TP / (TP + FN)))
    print('True negative rate is: %f\n' % (TN / (TN + FP)))
```

```
47
48  def estimate_parameter(x_train, x_test, y_train, y_test, method_func, param, param_values):
49      """
50      """
51      # Pipeline for estimate the regularization parameter
52      pipeline = Pipeline([ ('method', globals()[method_func]()) ])
53
54      # Construct the grid the hyperparameter candidate shronk theshold
55      param_grid = { 'method__' + param : param_values }
56
57      # Evaluating
58      skfold = RepeatedStratifiedKFold(n_splits=10, n_repeats=1, random_state=0)
59      gridcv = GridSearchCV(pipeline, cv=skfold, n_jobs=1, param_grid=param_grid, \
60          scoring=make_scorer(accuracy_score))
61      result = gridcv.fit(x_train, y_train)
62
63      # Accuracies
64      accuracies = gridcv.cv_results_['mean_test_score']
65      std_accuracies = gridcv.cv_results_['std_test_score']
66
67      test_accuracies = np.ones(len(param_values))
68
69      for i in range(len(param_values)):
70          method_params = {param : param_values[ i ]}
71          method = globals()[method_func](**method_params)
72          method.fit(x_train, y_train)
73          test_accuracies[ i ] = accuracy_score(method.predict(x_test), y_test)
74
75      # Plot
76      plt.figure(figsize=(15, 10))
77      line1, = plt.plot(param_values, accuracies, 'o-', color="g")
78      line2, = plt.plot(param_values, test_accuracies, 'x-', color="r")
79      plt.fill_between(param_values, accuracies - std_accuracies / np.sqrt(10), \
80          accuracies + std_accuracies / np.sqrt(10), alpha=0.1, color="g")
81      plt.grid()
82      plt.title("Different regularization parameter values for " + method_func)
83      plt.xlabel('Regularization Parameter')
84      plt.xticks(np.round(np.array(param_values), 2))
85      plt.ylabel('Classification Accuracy')
86      plt.ylim((min(accuracies) - 0.1, max(accuracies) + 0.1))
87
88      plt.xlim((min(param_values), max(param_values)))
89      legend_handles = [ mlines.Line2D([], [], color='g', marker='o', \
90                                markersize=15, label='CV-estimate'), \
91                      mlines.Line2D([], [], color='r', marker='x', \
92                                markersize=15, label='Test set estimate')]
93      plt.legend(handles=legend_handles, loc = 3)
94      plt.show()
95
96  def learn_dataset(data_file, header, label_pos, label_value, features_ini, features_fin=None
    ):
97      """
```

```
    """

    X_train_scaled, X_test_scaled, y_train, y_test = \
        create_datasets_from_file(data_file, header, label_pos, label_value, features_ini,
        ↪ features_fin=None)
    print(X_train_scaled.shape)

    # Naive Bayes accuracy
    prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test, "GaussianNB", "", "")

    # LDA accuracy
    prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test,
    ↪ "LinearDiscriminantAnalysis", "", "")

    # QDA estimate reg parameter
    param_values = np.linspace(0, 1, 10).tolist()
    estimate_parameter(X_train_scaled, X_test_scaled, y_train, y_test,\
                       "QuadraticDiscriminantAnalysis", "reg_param", param_values)
    # QDA accuracy
    # Best parameter reg value according CV estimate
    best_param_value = 0.444 #to compute automagically
    prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test, \
                        "QuadraticDiscriminantAnalysis", "reg_param", best_param_value)

    # Centroids
    # Best parameter shrink_threshold value according CV estimate
    param_values = np.linspace(0, 8, 20).tolist()
    estimate_parameter(X_train_scaled, X_test_scaled, y_train, y_test,\
                       "NearestCentroid", "shrink_threshold", param_values)
    best_param_value = 3.4 #to compute automagically
    # Centroids accuracy
    prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test, "NearestCentroid",
    ↪ "shrink_threshold", best_param_value)
```

## 1.2 Breast cancer

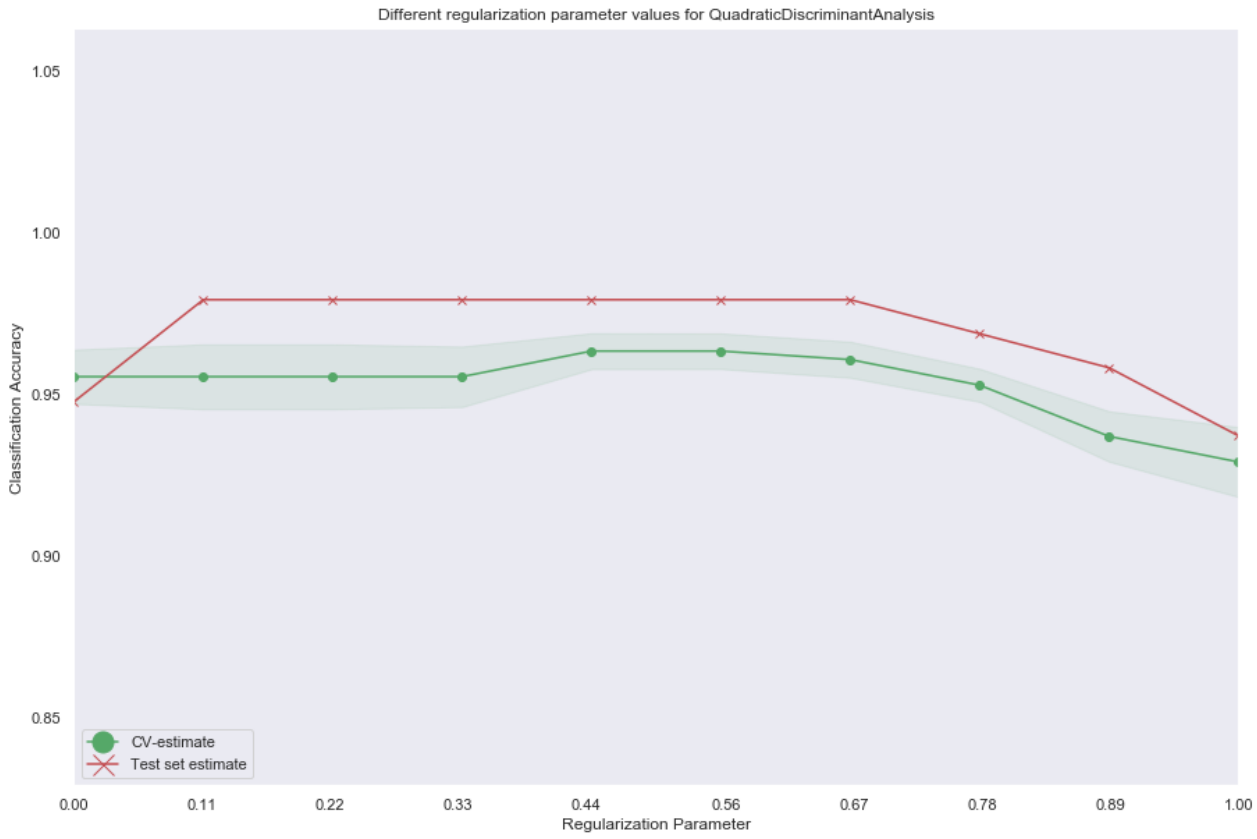
Script 1.2.1 (python)

```
# Breast Cancer
data_file = './data/wdbc.csv'
learn_dataset(data_file , None, 1, "B", 2)
```

Output

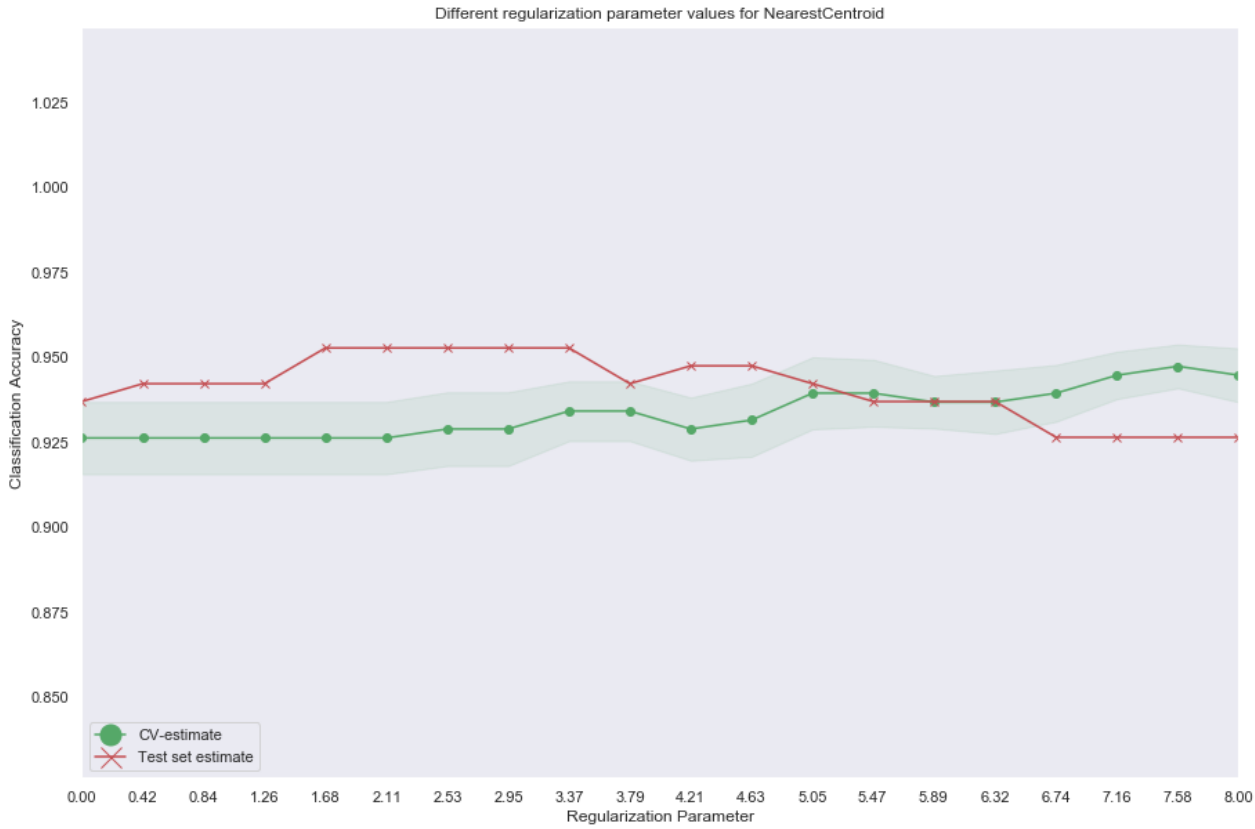
```
(379, 30)
[[ 61   5]
 [  7 117]]
Predicion accuracy is: 0.936842
True postive rate is: 0.943548
True negative rate is: 0.924242
```

```
[[ 60   6]
 [  1 123]]
Predicion accuracy is: 0.963158
True postive rate is: 0.991935
True negative rate is: 0.909091
```




## Output

```
[[ 62   4]
 [  0 124]]
Predicion accuracy is: 0.978947
True postive rate is: 1.000000
True negative rate is: 0.939394
```




Output

```
[[ 60   6]
 [  3 121]]
Predicion accuracy is: 0.952632
True postive rate is: 0.975806
True negative rate is: 0.909091
```

## 1.3 Prostate cancer

Script 1.3.1 (python)

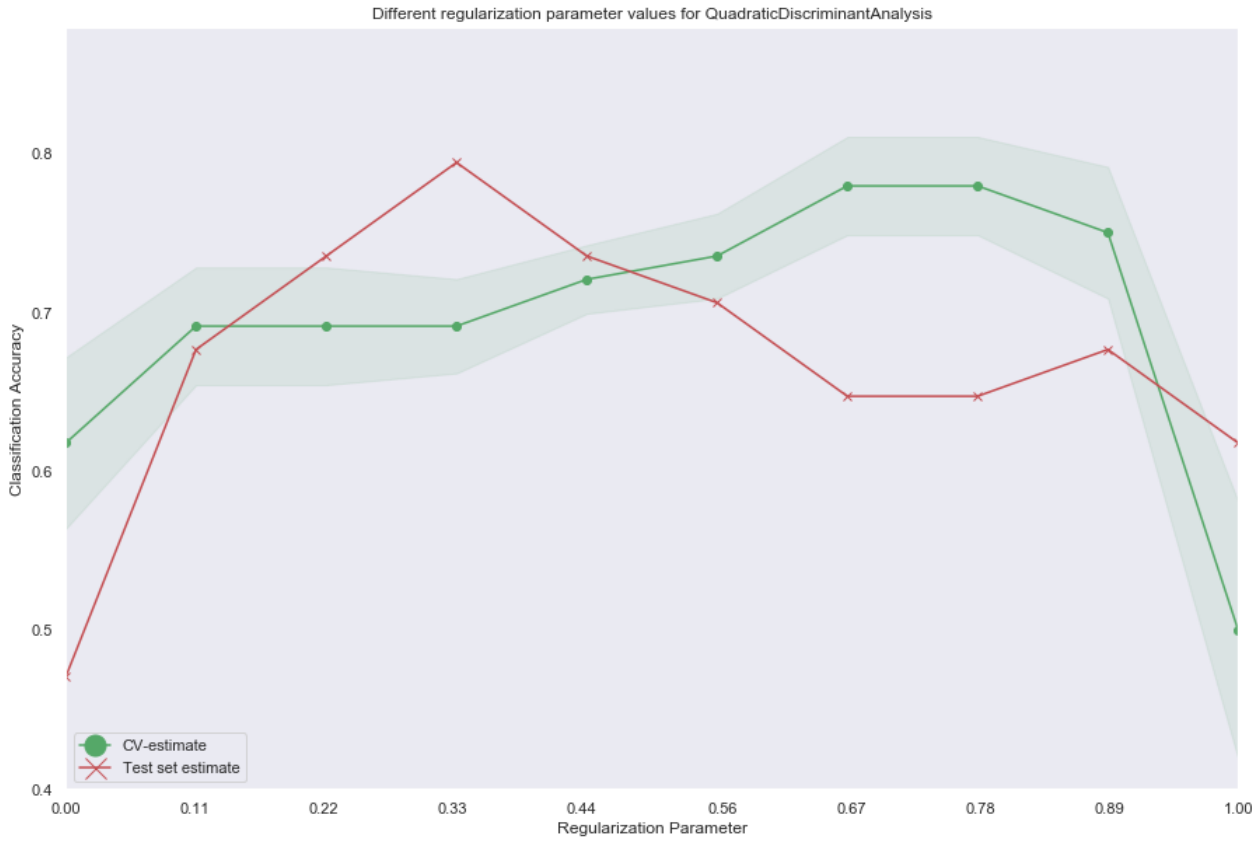
```python
# Prostate Cancer
data_file = './data/prostate.csv'
learn_dataset(data_file, 0, -1, 1, 0, -1)
```

Output

```
(68, 12626)
[[16  0]
 [ 0 18]]
```

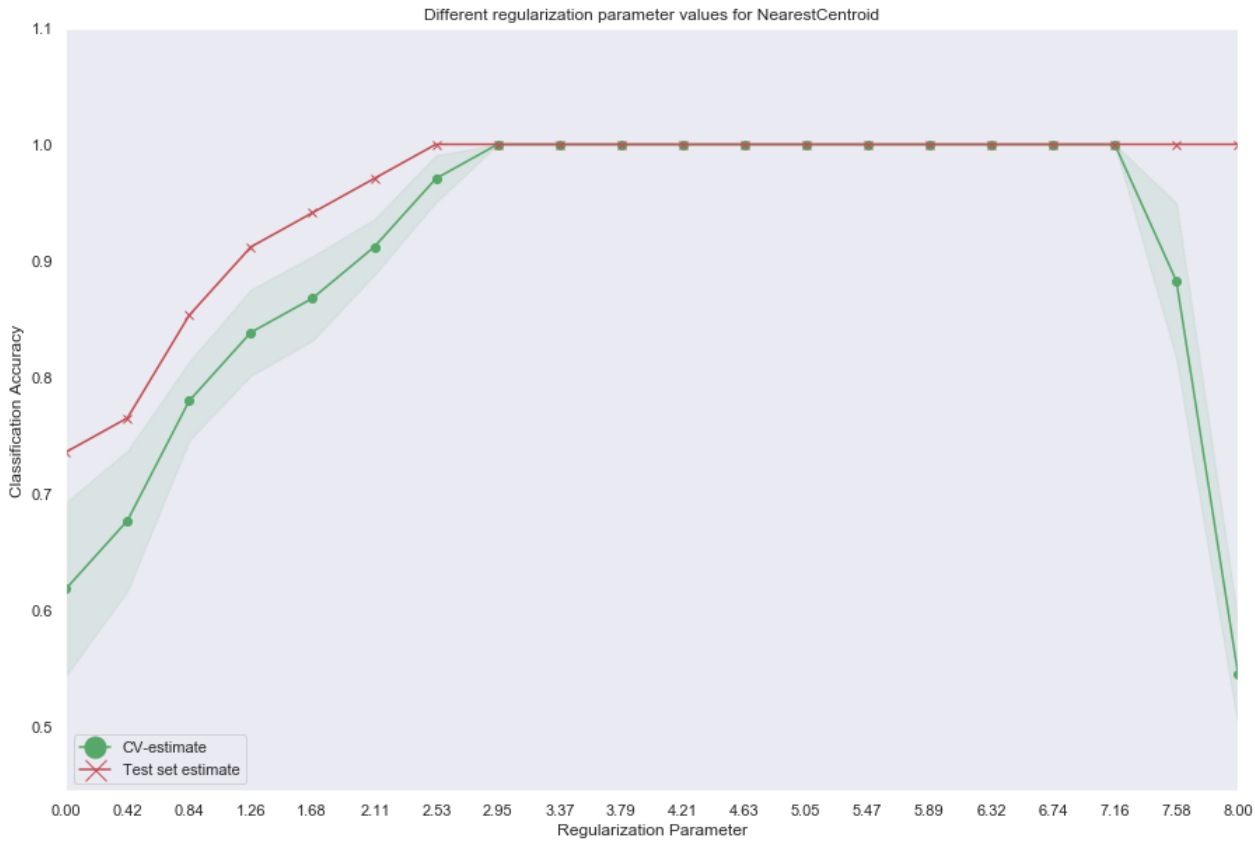
```
Predicion accuracy is: 1.000000
True postive rate is: 1.000000
True negative rate is: 1.000000

[[15  1]
 [ 4 14]]
Predicion accuracy is: 0.852941
True postive rate is: 0.777778
True negative rate is: 0.937500
```




Output

```
[[13  3]
 [ 6 12]]
Predicion accuracy is: 0.735294
True postive rate is: 0.666667
True negative rate is: 0.812500
```




## Output

```
[[16  0]
 [ 0 18]]
Predicion accuracy is: 1.000000
True postive rate is: 1.000000
True negative rate is: 1.000000
```