

# Classification Systems

Daniel Cerdán, Fernando Freire

March 27, 2019

## Contents

<b>1</b>	<b>Classification Systems</b>	<b>2</b>
1.1	Methods	3
1.1.1	Implementation details	3
1.2	Breast cancer	13
1.2.1	Hyper-parameter guessing plots	16
1.3	Prostate cancer	20
1.3.1	Hyper-parameter guessing plots	24
1.4	Conclusions	28

# 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 Shrunk 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)

```
1 import warnings
2 warnings.filterwarnings("ignore")
3 %matplotlib inline
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import matplotlib.lines as mlines
8 import matplotlib as mpl
9 from matplotlib import colors
10 import seaborn as sns; sns.set()
11 import scipy.stats as stats
12 import scipy as sp
13 from scipy import linalg
14 from sklearn.naive_bayes import GaussianNB
15 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
16 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
17 from sklearn.naive_bayes import GaussianNB
18 from sklearn.neighbors import NearestCentroid
19 from sklearn.decomposition import PCA
20 from sklearn.pipeline import Pipeline
21 from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold, GridSearchCV
22 from sklearn import preprocessing
23 from sklearn.metrics import accuracy_score, make_scorer, confusion_matrix,
   → classification_report, precision_score
```

## 1.1 Methods

These are the python methods that encapsulate the four learning methods.

### 1.1.1 Implementation details

#### Quadratic Discriminant Analysis

Before training the classifier we have chosen a good value for the corresponding regularization hyper-parameter with a grid-search guided by cross-validation.

The regularization parameter regularizes the covariance matrix estimate as

$$(1 - \lambda) \cdot \Sigma + \lambda \cdot \mathbf{I}$$

#### Nearest Centroids

Before training the classifier we have chosen a good value for the shrinkage threshold hyper-parameter with a grid-search guided by cross-validation.

This procedure leads to a reduction in the number of features, by zeroing all deltas that exceed the threshold.

They take the form:

$$\mu_{kj} = m_j + \Delta_{kj},$$

where  $\Delta_{kj}$  is the shrunken component

#### Selecting the best parameter value

To do so we compute the set of values with the maximum test data accuracy, and between them we choose the set of values that have the maximum train data accuracy. From this set we choose the lowest value.

#### Script 1.1.1 (python)

```
1 # Global parameters
2 # Verbose flag
3 gp_verbose = False
4 # Show progress flag
5 gp_show_progress = True
6 # Disable plots
7 gp_disable_plots = True
8 # Activate QDA with hyper-parameter reg_param
9 gp_qda_hyper = True
10 # Activate NSC with hyper-parameter shrink_threshold
11 gp_nsc_hyper = True
12 # Dimensionality reduction(PCA)
13 gp_dim_red = False
14 # Retained variance (PCA)
15 gp_retained_variance = 99
16 # Number of iterations
17 gp_iterations = 1
18 # Test size = number of samples / gp_test_size
19 gp_test_size = 3
20 # Best_hyper_param_method
21 gp_best_hyper_method = "max_in_cv"
22 # Skfold splits for hyperparameter guessing
23 gp_skfold_splits = 10
24
25 # Global execution parameters
```

```

26 # New dimensions after PCA
27 ge_features_reduction = 0
28
29 #Number of params of learning methods
30 ge_complexity = []
31
32 # Methods
33 def get_component_number(df_data, desired_variance=99.0, scaling=False):
34     """
35     Obtain the number of components that explains a %desired_variance
36     Args:
37         df_data (dataframe): dataframe of features in cols and samples in rows
38         desired_variance (float): desired explained variance
39         scaling (boolean): True if pre-scaling is needed prior to compute PCA
40     Returns:
41         int: number of components to maintain to have a explained variance >=
→ desired_variance
42         float: variance explained for the number of components returned
43         numpy array: cumulative variance by number of components retained
44     """
45     if scaling:
46         df_data_2 = preprocessing.StandardScaler().fit_transform(df_data)
47     else:
48         df_data_2 = df_data
49     # project the data into this new PCA space
50     pca = PCA().fit(df_data_2)
51     desired_variance = desired_variance/100.0
52     explained_variance = np.cumsum(pca.explained_variance_ratio_)
53     component_number = 0
54     for cumulative_variance in explained_variance:
55         component_number += 1
56         if cumulative_variance >= desired_variance:
57             break
58     return component_number, cumulative_variance, explained_variance
59
60
61 def create_datasets_from_file(data_file, header, random_state, label_pos,
62                               label_value, features_ini, features_fin=None,
63                               with_dim_red=False, retained_variance=99.0,
64                               reuse=False, dataset=None, labels=None):
65     """Create training and test sets from file
66
67     Args:
68         data_file (string): Name of the data file (csv) of samples a features
69         header (string): None or position of the header (pandas read_csv parameter)
70         random_state (int): Seed for the random split (as needed for sklearn
→ train_test_split)
71         label_pos (int): Column of the labels in data_file
72         label_value (int): Value of the label to assign internal '1' value
73         features_ini (int): First column of features in data_file
74         features_fin (int): Last column + 1 of features in data_file. If None, last
→ column of file.

```

```

75         with_dim_red (bool): If True, it performs a dimensionality reduction by PCA
76         retained_variance (float): If dimensionality reduction, variance to retain
77         reuse (bool): Reuse previous dataset
78         dataset: Dataset to reuse
79         labels: Labels to reuse
80
81     Returns:
82         (np.array): train set scaled
83         (np.array): test set scaled
84         (np.array): class labels for the train set
85         (np.array): class labels for the test set
86         (np.array): dataset
87         (np.array): labels
88
89     """
90     global ge_features_reduction
91     if not reuse:
92         data = pd.read_csv(data_file, header = header)
93         if features_fin == None:
94             X = data.values[ :, features_ini:].astype(np.float)
95         else:
96             X = data.values[ :, features_ini:features_fin].astype(np.float)
97         y = (data.values[ :, label_pos ] == label_value).astype(np.int)
98     else: #reuse previous dataset
99         X = dataset
100         y = labels
101
102     # Split dataset between training and test
103     x_train, x_test, y_train, y_test = train_test_split(X, y,
104                                                         test_size=1.0/gp_test_size,
105                                                         ↪ random_state=random_state)
106
107     # Data standardization
108     scaler = preprocessing.StandardScaler().fit(x_train)
109     x_train_scaled = scaler.transform(x_train)
110     x_test_scaled = scaler.transform(x_test)
111     # Check standardization
112     for i in range (1, np.size(x_train_scaled,1)):
113         assert round(np.var(x_train_scaled[:,0]),3) == round(np.var(x_train_scaled[:,i]),3),\
114             "Warning: revise data standardization"
115
116     if with_dim_red:
117         desired_variance = retained_variance
118         component_number, _, _ =\
119             get_component_number(x_train_scaled, desired_variance, scaling=None)
120         if gp_verbose: print("Features reduced to", component_number)
121         ge_features_reduction = component_number
122         pca = PCA(n_components = component_number)
123         pca.fit(x_train_scaled)
124         x_train_scaled = pca.transform(x_train_scaled)
125         x_test_scaled = pca.transform(x_test_scaled)
126
127     return x_train_scaled, x_test_scaled, y_train, y_test, X, y

```

```

126
127 def prediction_accuracy(x_train, x_test, y_train, y_test, method_func, method_param="",
    ↪ param_value=""):
128     """Estimate parameter given training and test sets:
129         Args:
130             x_train (np.array): train set
131             x_test (np.array): test set
132             y_train (np.array): class labels for the train set
133             y_test (np.array): class labels for the test set
134             method_func (string) : name of the learning method
135             method_param (string): name of learning method parameter
136             param_value (float): value of parameter to try
137         Returns:
138             float: best parameter value to use in prediction
139
140     """
141     if method_param != "" :
142         params = {method_param : param_value}
143     else:
144         params = {}
145     method = globals()[method_func](**params)
146
147     # Training
148     method.fit(x_train, y_train)
149
150     ge_complexity.append([method_func, len(method.get_params())])
151     # Prediction of test
152     y_pred = method.predict(x_test)
153     conf_test = confusion_matrix(y_test, y_pred, labels=[1,0])
154
155     # Prediction of train
156     y_pred = method.predict(x_train)
157     conf_train = confusion_matrix(y_train, y_pred, labels=[1,0])
158
159     return conf_train, conf_test
160
161 def estimate_parameter(x_train, x_test, y_train, y_test,
162                       method_func, param, param_values,
163                       best_param_value_method="max_in_test"):
164     """Estimate parameter given training and test sets:
165         Args:
166             x_train (np.array): train set
167             x_test (np.array): test set
168             y_train (np.array): class labels for the train set
169             y_test (np.array): class labels for the test set
170             method_func (string) : name of the learning method
171             param (string): name of learning method parameter
172             param_values (list of float): list of parameter values to try
173             best_param_value_method: if "max_in_test" gives the value with the maximum
    ↪ accuracy
174
175         Returns:
            float: best parameter value to use in prediction
            in test data.

```

```

176         (float): best parameter value to use in prediction
177
178     """
179     # Pipeline for estimate the regularization parameter
180     pipeline = Pipeline([ ('method', globals()[method_func]() )])
181
182     # Construct the grid the hyperparameter candidate shronk theshold
183     param_grid = { 'method__' + param : param_values }
184
185     # Evaluating
186     skfold = RepeatedStratifiedKfold(n_splits=gp_skfold_splits, n_repeats=1, random_state=0)
187     gridcv = GridSearchCV(pipeline, cv=skfold, n_jobs=1, param_grid=param_grid,\
188         scoring=make_scorer(accuracy_score))
189     result = gridcv.fit(x_train, y_train)
190
191     # Accuracies
192     accuracies = gridcv.cv_results_['mean_test_score']
193     std_accuracies = gridcv.cv_results_['std_test_score']
194
195     test_accuracies = np.ones(len(param_values))
196
197     for i in range(len(param_values)):
198         method_params = {param : param_values[ i ]}
199         method = globals()[method_func](**method_params)
200         method.fit(x_train, y_train)
201         test_accuracies[ i ] = accuracy_score(method.predict(x_test), y_test)
202
203     # Obtain best_param_value as max
204     max_test_accuracy = max(test_accuracies)
205     if best_param_value_method == "max_in_test":
206         best_param_value = 0
207         best_train_accuracy = 0
208         for i in range(len(param_values)):
209             if test_accuracies[ i ] == max_test_accuracy:
210                 if accuracies[i] > best_train_accuracy:
211                     best_train_accuracy = accuracies[i]
212                     best_param_value = param_values[i]
213     else:
214         best_param_value = param_values[ np.argmax(accuracies) ]
215
216     # Plot
217     if not gp_disable_plots:
218         plt.figure(figsize=(9, 9))
219         line1, = plt.plot(param_values, accuracies, 'o-', color="g")
220         line2, = plt.plot(param_values, test_accuracies, 'x-', color="r")
221         plt.fill_between(param_values, accuracies - std_accuracies / np.sqrt(10), \
222             accuracies + std_accuracies / np.sqrt(10), alpha=0.1, color="g")
223         plt.grid()
224         plt.title("Different hyper-parameter " + param + " values for " + method_func)
225         plt.xlabel('Hyper-parameter')
226         plt.xticks(np.round(np.array(param_values), 2))
227         plt.ylabel('Classification Accuracy')
228         plt.ylim((min(min(accuracies), min(test_accuracies)) - 0.1,

```

```

228         min(1.02, max(max(accuracies), max(test_accuracies)) + 0.1)))
229
230     plt.xlim((min(param_values), max(param_values)))
231     legend_handles = [ mlines.Line2D([], [], color='g', marker='o', \
232                         markersize=15, label='CV-estimate'), \
233                       mlines.Line2D([], [], color='r', marker='x', \
234                         markersize=15, label='Test set estimate')]
235     plt.legend(handles=legend_handles, loc = 3)
236     plt.show()
237 if gp_verbose:
238     print("Best param value %s Method %s: %s" % (method_func, best_param_value_method,
239         ↪ best_param_value))
239 return best_param_value
240
241 def calculate_avg_results(train_results, test_results, classifier_name):
242     """Calculate the average accuracy, TPN and TNR for the n=gp_iterations of a classifier
243     Args:
244     ↪ train_results (array nx2x2): Each of the n confusions matrix generated for the
245     ↪ train set
246     ↪ test_results (array nx2x2): Each of the n confusions matrix generated for the
247     ↪ test set
248     ↪ classifier_name (string): Classifier name gp_iterations print the results
249     Returns:
250     ↪ (np.array 3): Average accuracy, TPR and TNR of the n iterations of the train set
251     ↪ (np.array 3): Average accuracy, TPR and TNR of the n iterations of the test set
252     ↪ (np.array 3): Stdev accuracy, TPR and TNR of the n iterations of the train set
253     ↪ (np.array 3): Stdev accuracy, TPR and TNR of the n iterations of the test set
254     """
255
256     measures_train = np.zeros(shape = (gp_iterations,3)) # Each row is an ex. and each
257     ↪ column is the accuracy, TPR and TNR
258     measures_test = np.zeros(shape =(gp_iterations,3))
259
260     avg_results_train = np.zeros(3) # Each element is the average accuracy, the TPR and the
261     ↪ TNR
262     avg_results_test = np.zeros(3)
263     # TP in 0,0
264     # FN in 0,1
265     # TN in 1,1
266     # FP in 1,0
267     for i in range(gp_iterations):
268         # For train set
269         TN = train_results[i,1,1]
270         TP = train_results[i,0,0]
271         FP = train_results[i,1,0]
272         FN = train_results[i,0,1]
273         measures_train[i][0] = (TP + TN) / (TN + TP + FP + FN)
274         measures_train[i][1] = (TP / (TP + FN))
275         measures_train[i][2] = (TN / (TN + FP))

```



```

275     TN = test_results[i,1,1]
276     TP = test_results[i,0,0]
277     FP = test_results[i,1,0]
278     FN = test_results[i,0,1]
279     measures_test[i][0] = (TP + TN) / (TN + TP + FP + FN)
280     measures_test[i][1] = (TP / (TP + FN))
281     measures_test[i][2] = (TN / (TN + FP))
282
283     avg_results_train = measures_train.mean(axis = 0)
284     avg_results_test = measures_test.mean(axis = 0)
285     std_results_train = measures_train.std(axis = 0)
286     std_results_test = measures_test.std(axis = 0)
287     if gp_verbose:
288         print("\nResults for the ", classifier_name, " classifier\n")
289
290         print('For the train set:')
291         print('Prediction accuracy of train set is: %f' % avg_results_train[0])
292         print('True positive rate of train set is: %f' % avg_results_train[1])
293         print('True negative rate of train set is: %f\n' % avg_results_train[2])
294
295         print('For the test set:')
296         print('Prediction accuracy of train set is: %f' % avg_results_test[0])
297         print('True positive rate of train set is: %f' % avg_results_test[1])
298         print('True negative rate of train set is: %f\n' % avg_results_test[2])
299
300     return avg_results_train, avg_results_test, std_results_train, std_results_test
301
302 def print_parameters():
303     print("Parameters")
304     pdata = {'Parameters' : [gp_qda_hyper, gp_nsc_hyper,
305                             gp_dim_red, gp_retained_variance,
306                             gp_iterations, 100.0/gp_test_size,
307                             gp_best_hyper_method]}
308     pdataf = pd.DataFrame(data = pdata, index = ['QDA hyper-parameter reg_param',
309                                                'NSC Hyper-parameter shrink_threshold',
310                                                'Dimensionality Reduction',
311                                                'Retained variance',
312                                                'Number of iterations',
313                                                'Test set size',
314                                                'Best hyperparameter select method'])
315
316     display(pdataf)
317     print("")
318
319 def print_execution_data():
320     print("Execution data")
321     data = [ge_samples, ge_features, ge_features_reduction]
322     index = ['Number of samples', 'Number of features', 'Features reduction']
323     for [method, complexity] in ge_complexity:
324         data.append(complexity)
325         index.append("Params " + method)
326     pdata = {'Execution Data' : data}
327     pdataf = pd.DataFrame(data = pdata, index = index)

```

```

327     display(pdataf)
328     print("")
329
330 def print_accuracies(accuracy_NBC, accuracy_LDA, accuracy_QDA, accuracy_NSC, metric):
331     print(metric)
332     d = {}
333     for m in [accuracy_NBC, accuracy_LDA, accuracy_QDA, accuracy_NSC]:
334         overfit_indicator = 1000 * round(abs(m[1] - m[0]) / (m[0] + 0.0000001), 4)
335         #overfit_indicator = round(abs(m[1] - m[0]),3)
336         m[0] = (round(m[0],2), round(m[2],2))
337         m[1] = (round(m[1],2), round(m[3],2))
338         m[2] = overfit_indicator
339     d = {'NBC': accuracy_NBC[:3],
340         'LDA': accuracy_LDA[:3],
341         'QDA': accuracy_QDA[:3],
342         'NSC': accuracy_NSC[:3]}
343     df = pd.DataFrame(data = d, index = ['Train', 'Test', 'Overfit degree'])
344     display(df)
345     print("")
346
347 def learn_dataset(data_file, header, random_state, label_pos,
348                  label_value, features_ini, features_fin=None,
349                  best_param_value_method="max_in_test",
350                  with_dim_red=False, retained_variance=99.0):
351     """Learn data sets from file, methods:
352     1. The Naive Bayes Classifier
353     2. LDA
354     3. QDA
355     4. Nearest Shrunken Centroids Classifier
356     Args:
357     data_file (string): Name of the data file (csv) of samples a features
358     header (string): None or position of the header (pandas read_csv parameter)
359     random_state (int): Seed for the random split of sets (as needed for sklearn
360     train_test_split)
361     label_pos (int): Column of the labels in data_file
362     label_value (int): Value of the label to assign internal '1' value. We consider
363     this label as
364     the positive label in prediction validation. We assign malign or cancer status to
365     this label.
366     features_ini (int): First column of features in data_file
367     features_fin (int): Last column + 1 of features in data_file. If None, last
368     column of file
369     best_param_value_method (str): if "max_in_test" gives the value with the maximum
370     accuracy
371     in test data
372     with_dim_red (bool): If True, it performs a dimensionality reduction by PCA
373     retained_variance (float): If dimensionality reduction, variance to retain
374
375     """
376     global ge_features_reduction
377     global ge_features
378     global ge_samples

```

```

374 global ge_complexity
375 ge_features_reduction = 0
376 nbc_train = np.zeros(shape=(gp_iterations,2,2))
377 nbc_test = np.zeros(shape=(gp_iterations,2,2))
378 lda_train = np.zeros(shape=(gp_iterations,2,2))
379 lda_test = np.zeros(shape=(gp_iterations,2,2))
380 qda_train = np.zeros(shape=(gp_iterations,2,2))
381 qda_test = np.zeros(shape=(gp_iterations,2,2))
382 nsc_train = np.zeros(shape=(gp_iterations,2,2))
383 nsc_test = np.zeros(shape=(gp_iterations,2,2))
384
385
386 for i in range(gp_iterations):
387     ge_complexity = []
388     if gp_show_progress: print("\nIteration: ",i)
389     if i == 0:
390         reuse = False
391         X = None
392         y = None
393     else:
394         reuse = True
395     X_train_scaled, X_test_scaled, y_train, y_test, X, y = \
396         create_datasets_from_file(data_file, header, random_state + i,
397                                 label_pos, label_value, features_ini, features_fin =
398                                     ↪ features_fin,
399                                     with_dim_red = with_dim_red, retained_variance =
400                                         ↪ retained_variance,
401                                     reuse = reuse, dataset = X, labels = y)
402
403     ge_samples = X.shape[0]
404     ge_features = X.shape[1]
405     if gp_verbose: print(X_train_scaled.shape)
406
407     if gp_verbose: print("NBC")
408     # Naive Bayes accuracy
409     nbc_train[i], nbc_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
410                                                     ↪ y_train, y_test, "GaussianNB")
411
412     # LDA accuracy
413     if gp_verbose: print("LDA")
414     lda_train[i], lda_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
415                                                     ↪ y_train, y_test, "LinearDiscriminantAnalysis")
416
417     # QDA estimate reg parameter
418     if gp_verbose: print("QDA")
419     if gp_qda_hyper:
420         param_values = np.linspace(0, 1, 10).tolist()
421         best_param_value = estimate_parameter(X_train_scaled, X_test_scaled, y_train,
422                                             ↪ y_test,\
423                                             "QuadraticDiscriminantAnalysis", "reg_param", param_values,\
424                                             best_param_value_method)
425
426     # QDA accuracy
427     # Best parameter reg value according CV estimate

```

```

421     qda_train[i], qda_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
422     ↪ y_train, y_test,\
423         "QuadraticDiscriminantAnalysis", "reg_param",
424         ↪ best_param_value)
425 else:
426     qda_train[i], qda_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
427     ↪ y_train, y_test,\
428         "QuadraticDiscriminantAnalysis")
429 # Centroids
430 if gp_verbose: print("NSC")
431 if gp_nsc_hyper:
432     # Best parameter shrink_threshold value according CV estimate
433     param_values = np.linspace(0, 8, 20).tolist()
434     best_param_value = estimate_parameter(X_train_scaled, X_test_scaled, y_train,
435     ↪ y_test,\
436         "NearestCentroid", "shrink_threshold", param_values,\
437         best_param_value_method)
438 # Centroids accuracy
439 nsc_train[i], nsc_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
440     ↪ y_train, y_test,\
441         "NearestCentroid", "shrink_threshold",
442         ↪ best_param_value)
443 else:
444     nsc_train[i], nsc_test[i] = prediction_accuracy(X_train_scaled, X_test_scaled,
445     ↪ y_train, y_test,\
446         "NearestCentroid")
447 # Calculate and print the average results
448 avg_results_train_NBC, avg_results_test_NBC, std_results_train_NBC, std_results_test_NBC
449 ↪ = calculate_avg_results(nbc_train, nbc_test, "NBC")
450 avg_results_train_LDA, avg_results_test_LDA, std_results_train_LDA, std_results_test_LDA
451 ↪ = calculate_avg_results(lda_train, lda_test, "LDA")
452 avg_results_train_QDA, avg_results_test_QDA, std_results_train_QDA, std_results_test_QDA
453 ↪ = calculate_avg_results(qda_train, qda_test, "QDA")
454 avg_results_train_NSC, avg_results_test_NSC, std_results_train_NSC, std_results_test_NSC
455 ↪ = calculate_avg_results(nsc_train, nsc_test, "NSC")
456
457 print_parameters()
458 i = 0
459 for metric in ["Accuracy", "TPR", "TNR"]:
460     print_accuracies([avg_results_train_NBC[i], avg_results_test_NBC[i],
461     ↪ std_results_train_NBC[i], std_results_test_NBC[i]],
462         [avg_results_train_LDA[i], avg_results_test_LDA[i],
463         ↪ std_results_train_LDA[i], std_results_test_LDA[i]],
464         [avg_results_train_QDA[i], avg_results_test_QDA[i],
465         ↪ std_results_train_QDA[i], std_results_test_QDA[i]],
466         [avg_results_train_NSC[i], avg_results_test_NSC[i],
467         ↪ std_results_train_NSC[i], std_results_test_NSC[i]], metric)
468
469     i += 1
470 print_execution_data()

```

## 1.2 Breast cancer

### Script 1.2.1 (python)

```
1 gp_disable_plots = True
2 gp_verbose = False
3 gp_dim_red = False
4 gp_retained_variance = 99
5 gp_qda_hyper = True
6 gp_nsc_hyper = True
7 gp_iterations = 20
8 gp_show_progress = False
9 gp_best_hyper_method = "max_in_cv"
10 gp_skfold_splits = 10
11
12 learn_dataset(data_file = './data/wdbc.csv', header = None, random_state=0,
13               label_pos=1, label_value="M", features_ini = 2, features_fin = None,
14               best_param_value_method = gp_best_hyper_method,
15               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
16
17 gp_dim_red = True
18 learn_dataset(data_file = './data/wdbc.csv', header = None, random_state=0,
19               label_pos=1, label_value="M", features_ini = 2, features_fin = None,
20               best_param_value_method = gp_best_hyper_method,
21               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
```

### Output

#### Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True
Dimensionality Reduction	False
Retained variance	99
Number of iterations	20
Test set size	33.3333
Best hyperparameter select method	max_in_cv

### Output

#### Accuracy

	NBC	LDA	QDA	NSC
Train	(0.94, 0.01)	(0.97, 0.01)	(0.97, 0.01)	(0.94, 0.01)
Test	(0.93, 0.01)	(0.96, 0.01)	(0.97, 0.01)	(0.94, 0.01)
Overfit degree	5.4	7.5	9.4	7

## Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.9, 0.01)	(0.92, 0.01)	(0.95, 0.01)	(0.9, 0.02)
Test	(0.9, 0.03)	(0.89, 0.03)	(0.93, 0.03)	(0.9, 0.04)
Overfit degree	10.1	23.3	14.7	3.6

## Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.97, 0.01)	(1.0, 0.0)	(0.99, 0.0)	(0.97, 0.01)
Test	(0.95, 0.02)	(1.0, 0.0)	(0.98, 0.02)	(0.96, 0.02)
Overfit degree	14.3	0	6.8	9.2

## Output

Execution data

	Execution Data
Number of samples	569
Number of features	30
Features reduction	0
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True
Dimensionality Reduction	True
Retained variance	99
Number of iterations	20
Test set size	33.3333
Best hyperparameter select method	max_in_cv

## Output

### Accuracy

	NBC	LDA	QDA	NSC
Train	(0.9, 0.01)	(0.96, 0.0)	(0.98, 0.01)	(0.93, 0.01)
Test	(0.89, 0.02)	(0.96, 0.01)	(0.97, 0.01)	(0.93, 0.02)
Overfit degree	10.2	7.4	8.8	5.7

## Output

### TPR

	NBC	LDA	QDA	NSC
Train	(0.84, 0.02)	(0.91, 0.01)	(0.95, 0.02)	(0.87, 0.01)
Test	(0.83, 0.04)	(0.89, 0.02)	(0.94, 0.03)	(0.87, 0.03)
Overfit degree	5.4	18.6	17.2	4

## Output

### TNR

	NBC	LDA	QDA	NSC
Train	(0.94, 0.01)	(1.0, 0.0)	(0.99, 0.0)	(0.97, 0.01)
Test	(0.93, 0.03)	(0.99, 0.01)	(0.99, 0.01)	(0.96, 0.02)
Overfit degree	13.1	2.1	4.6	7.6

## Output

### Execution data

	Execution Data
Number of samples	569
Number of features	30
Features reduction	17
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

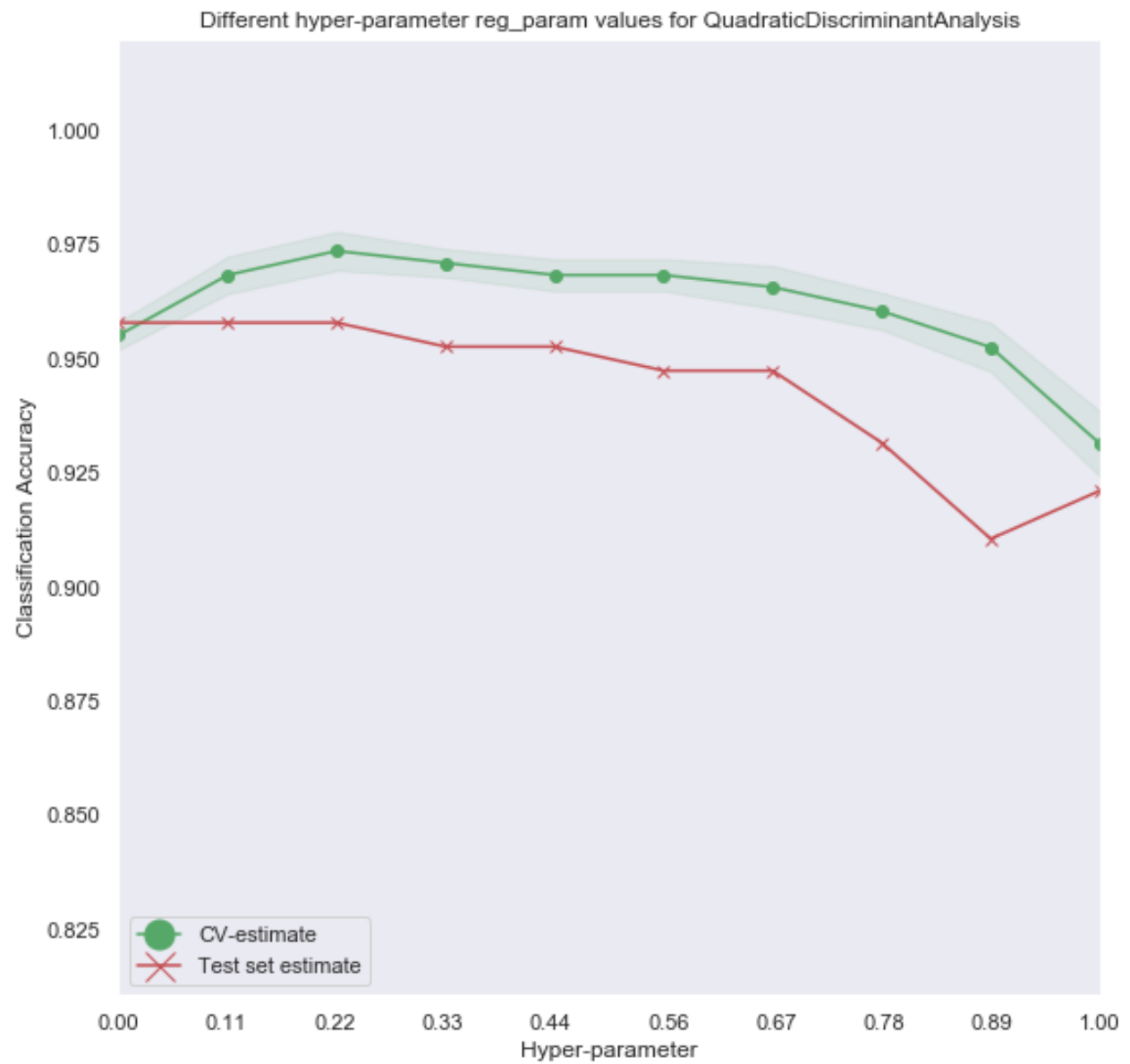
## Output

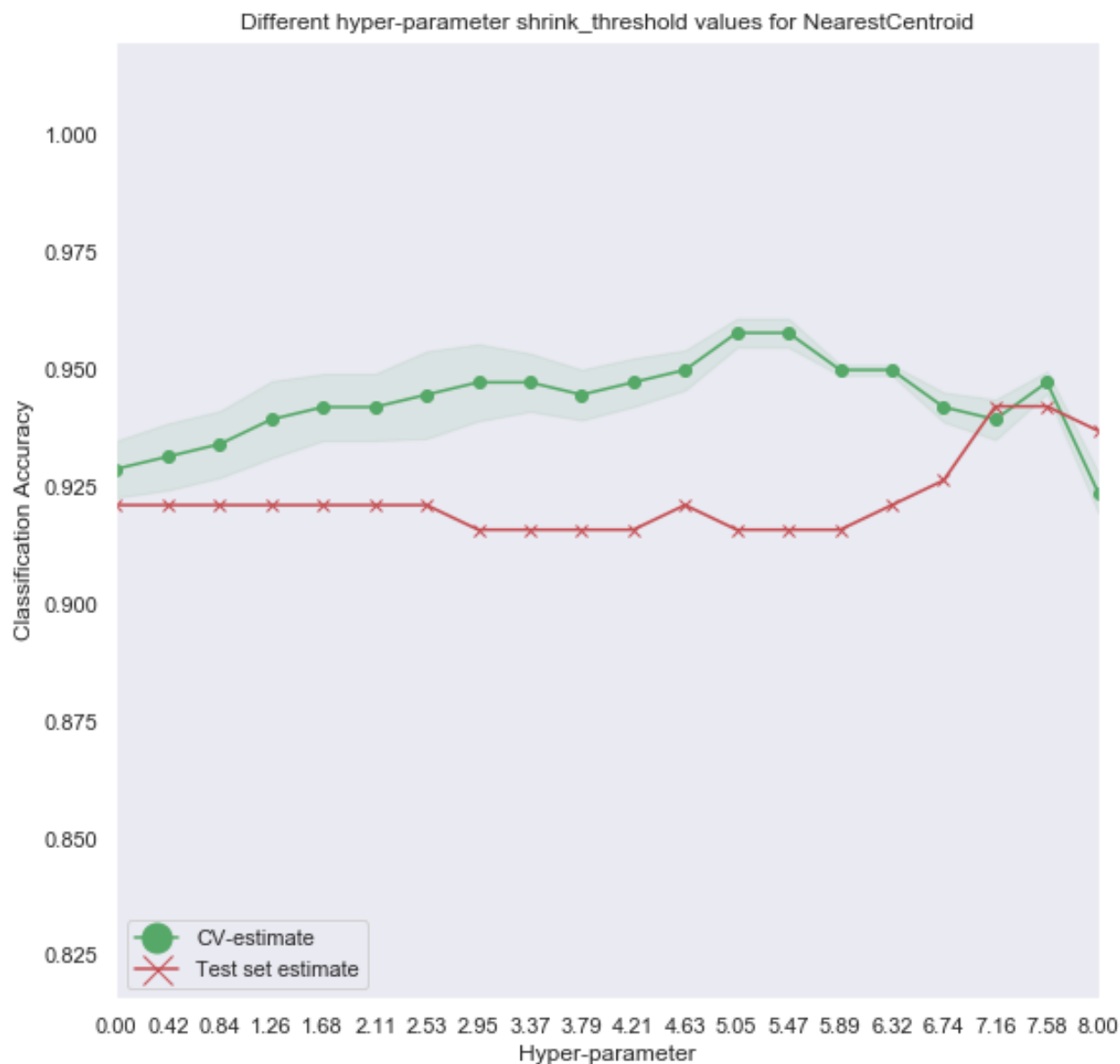
### 1.2.1 Hyper-parameter guessing plots

#### Script 1.2.2 (python)

```
1 gp_disable_plots = False
2 gp_verbose = False
3 gp_dim_red = False
4 gp_retained_variance = 99
5 gp_qda_hyper = True
6 gp_nsc_hyper = True
7 gp_iterations = 1
8 gp_show_progress = False
9 gp_best_hyper_method = "max_in_cv"
10 ge_features_reduction = 0
11
12 learn_dataset(data_file = './data/wdbc.csv', header = None, random_state=0,
13               label_pos=1, label_value="M", features_ini = 2, features_fin = None,
14               best_param_value_method = gp_best_hyper_method,
15               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
```







## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True
Dimensionality Reduction	False
Retained variance	99
Number of iterations	1
Test set size	33.3333

Best hyperparameter select method      max\_in\_cv

### Output

Accuracy

	NBC	LDA	QDA	NSC
Train	(0.95, 0.0)	(0.96, 0.0)	(0.98, 0.0)	(0.94, 0.0)
Test	(0.9, 0.0)	(0.97, 0.0)	(0.96, 0.0)	(0.92, 0.0)
Overfit degree	49.9	11	18.8	30.5

### Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.9, 0.0)	(0.91, 0.0)	(0.94, 0.0)	(0.91, 0.0)
Test	(0.85, 0.0)	(0.93, 0.0)	(0.91, 0.0)	(0.87, 0.0)
Overfit degree	55.2	18.4	34.6	46.3

### Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.97, 0.0)	(1.0, 0.0)	(1.0, 0.0)	(0.97, 0.0)
Test	(0.93, 0.0)	(1.0, 0.0)	(0.98, 0.0)	(0.94, 0.0)
Overfit degree	49.5	4.3	12.2	24.2

### Output

Execution data

	Execution Data
Number of samples	569
Number of features	30
Features reduction	0
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

### 1.3 Prostate cancer

#### Script 1.3.1 (python)

```
1 # Prostate Cancer
2 gp_disable_plots = True
3 gp_verbose = False
4 gp_dim_red = False
5 gp_retained_variance = 99
6 gp_qda_hyper = True
7 gp_nsc_hyper = True
8 gp_iterations = 20
9 gp_best_hyper_method = "max_in_cv"
10 ge_features_reduction = 0
11 gp_skfold_splits = 5
12
13 learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
14               label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
15               best_param_value_method = gp_best_hyper_method,
16               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
17
18 gp_best_hyper_method = "max_in_cv"
19 gp_dim_red = True
20 gp_retained_variance = 99
21 ge_features_reduction = 0
22 learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
23               label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
24               best_param_value_method = gp_best_hyper_method,
25               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
26
27 gp_best_hyper_method = "max_in_cv"
28 gp_dim_red = True
29 gp_retained_variance = 80
30 ge_features_reduction = 0
31 learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
32               label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
33               best_param_value_method = gp_best_hyper_method,
34               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)
```

## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True

Dimensionality Reduction	False
Retained variance	99
Number of iterations	20
Test set size	33.3333
Best hyperparameter select method	max_in_cv

### Output

Accuracy

	NBC	LDA	QDA	NSC
Train	(0.72, 0.06)	(0.86, 0.02)	(0.14, 0.25)	(0.91, 0.02)
Test	(0.64, 0.12)	(0.85, 0.04)	(0.64, 0.1)	(0.89, 0.05)
Overfit degree	106.3	17.9	3557.9	26.5

### Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.68, 0.07)	(0.83, 0.03)	(0.12, 0.24)	(0.88, 0.03)
Test	(0.66, 0.12)	(0.8, 0.07)	(0.62, 0.18)	(0.85, 0.07)
Overfit degree	29.6	36.8	4243.1	26.7

### Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.76, 0.08)	(0.89, 0.03)	(0.16, 0.27)	(0.95, 0.03)
Test	(0.62, 0.18)	(0.9, 0.08)	(0.63, 0.23)	(0.93, 0.06)
Overfit degree	178.1	16.7	2863.6	19.8

### Output

Execution data

	Execution Data
Number of samples	102
Number of features	12625

Features reduction	0
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True
Dimensionality Reduction	True
Retained variance	99
Number of iterations	20
Test set size	33.3333
Best hyperparameter select method	max_in_cv

## Output

Accuracy

	NBC	LDA	QDA	NSC
Train	(0.92, 0.04)	(1.0, 0.0)	(0.87, 0.15)	(0.75, 0.08)
Test	(0.74, 0.12)	(0.9, 0.04)	(0.71, 0.09)	(0.72, 0.11)
Overfit degree	200.6	100	181.1	41.3

## Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.92, 0.05)	(1.0, 0.0)	(0.83, 0.19)	(0.75, 0.08)
Test	(0.95, 0.05)	(0.87, 0.07)	(0.71, 0.14)	(0.73, 0.12)
Overfit degree	40.9	128.7	140.4	22.2

## Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.93, 0.05)	(1.0, 0.0)	(0.92, 0.13)	(0.74, 0.1)
Test	(0.48, 0.29)	(0.93, 0.07)	(0.7, 0.19)	(0.69, 0.17)
Overfit degree	483.8	66.9	235.8	62.9

## Output

Execution data

	Execution Data
Number of samples	102
Number of features	12625
Features reduction	63
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True
Dimensionality Reduction	True
Retained variance	80
Number of iterations	20
Test set size	33.3333
Best hyperparameter select method	max_in_cv

## Output

Accuracy

	NBC	LDA	QDA	NSC
Train	(0.89, 0.06)	(0.96, 0.02)	(0.98, 0.01)	(0.72, 0.08)
Test	(0.78, 0.08)	(0.85, 0.05)	(0.87, 0.05)	(0.67, 0.12)
Overfit degree	116.8	110.6	116.8	76.3

## Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.83, 0.1)	(0.93, 0.04)	(0.97, 0.02)	(0.7, 0.09)
Test	(0.66, 0.13)	(0.82, 0.07)	(0.88, 0.07)	(0.69, 0.12)
Overfit degree	200.4	122.7	91.3	11.2

## Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.94, 0.03)	(0.98, 0.02)	(0.99, 0.01)	(0.75, 0.08)
Test	(0.93, 0.07)	(0.9, 0.1)	(0.86, 0.11)	(0.64, 0.2)
Overfit degree	15.2	82.4	133.7	137.2

## Output

Execution data

	Execution Data
Number of samples	102
Number of features	12625
Features reduction	19
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

### 1.3.1 Hyper-parameter guessing plots

#### Script 1.3.2 (python)

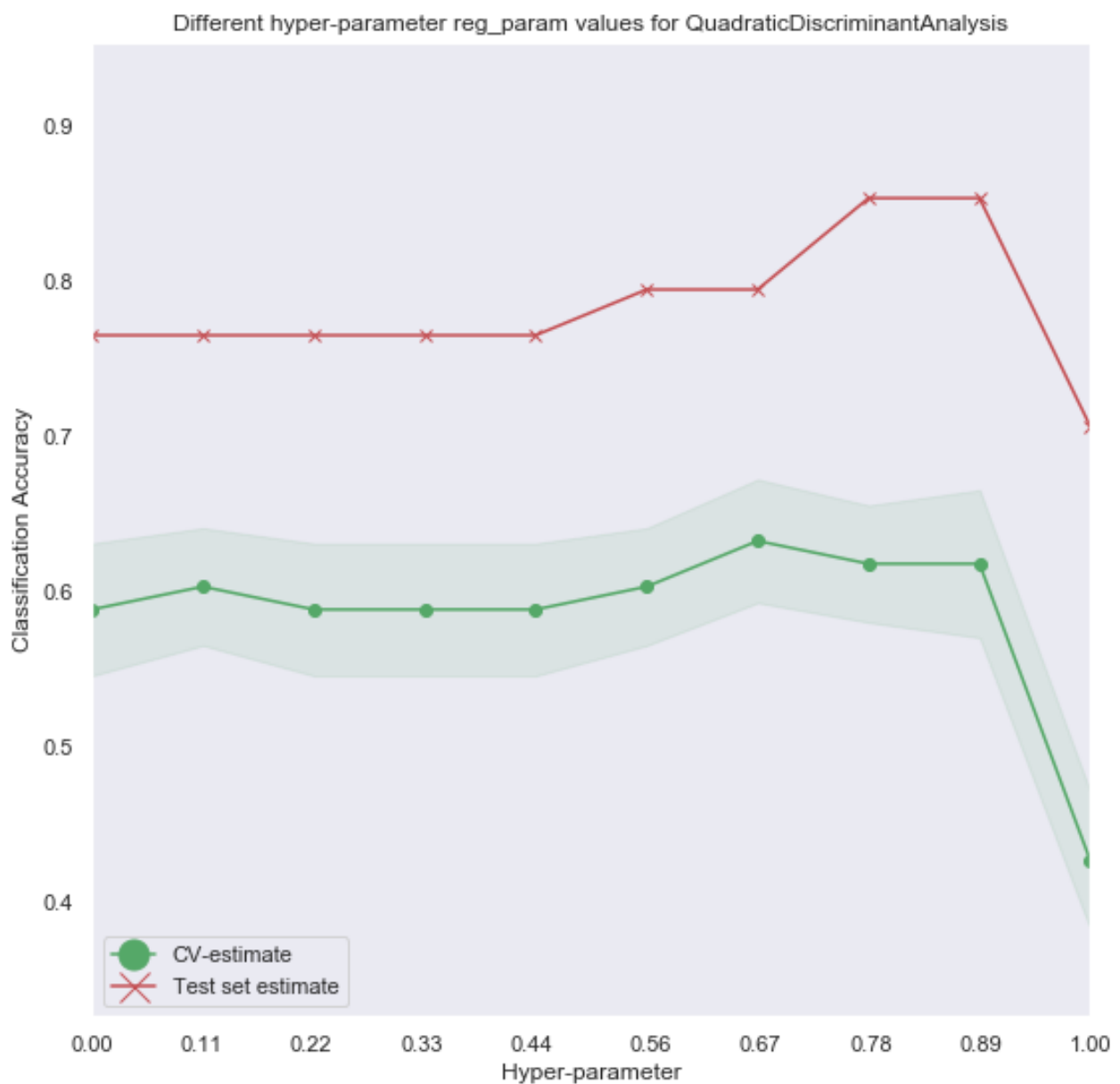
```
1 gp_disable_plots = False
2 gp_verbose = False
```

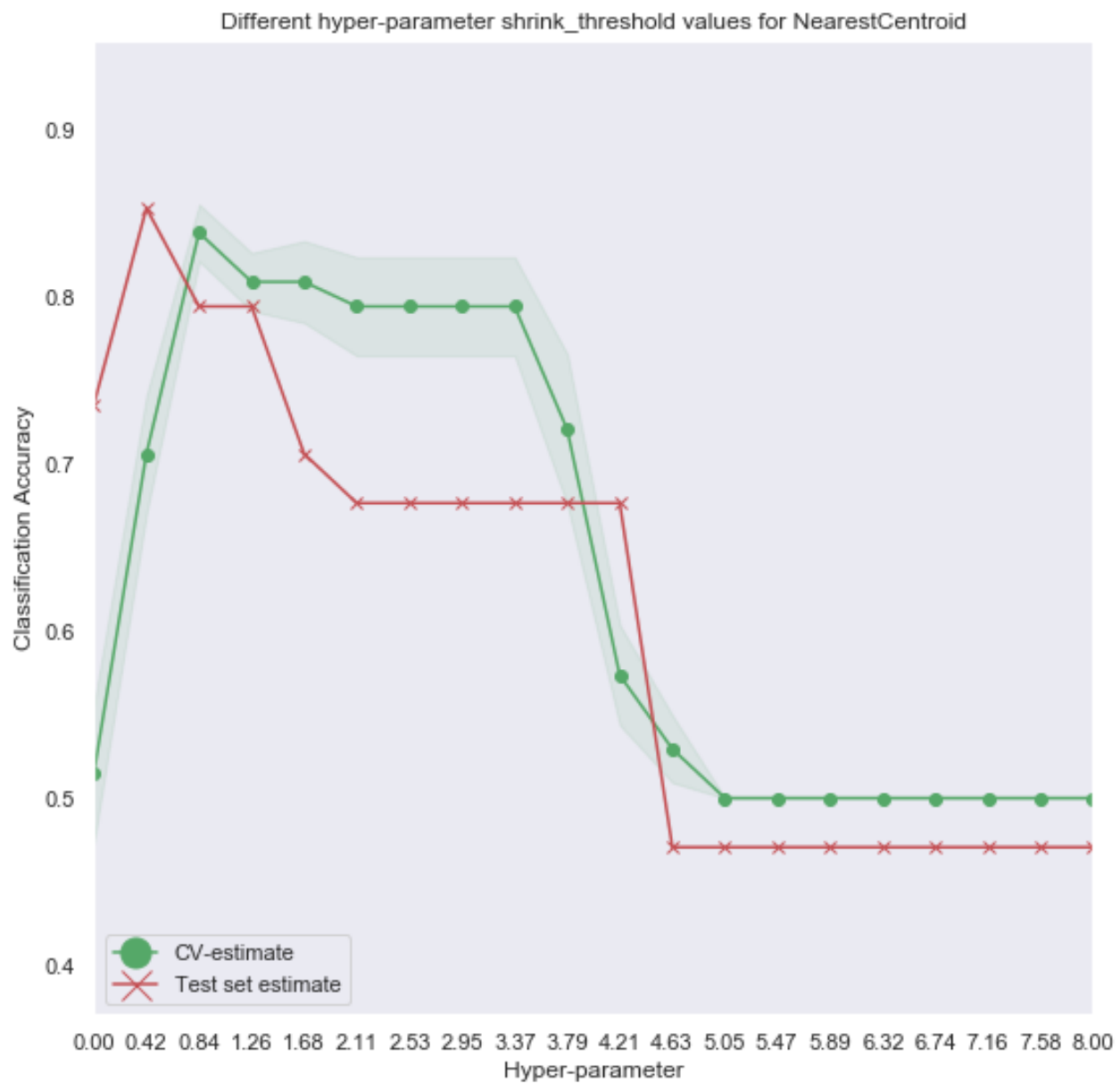


```

3 gp_dim_red = True
4 gp_retained_variance = 99
5 gp_qda_hyper = True
6 gp_nsc_hyper = True
7 gp_iterations = 1
8 gp_best_hyper_method = "max_in_cv"
9 ge_features_reduction = 0
10 gp_skfold_splits = 5
11 learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
12               label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
13               best_param_value_method = gp_best_hyper_method,
14               with_dim_red = gp_dim_red, retained_variance = gp_retained_variance)

```





## Output

Parameters

	Parameters
QDA hyper-parameter reg_param	True
NSC Hyper-parameter shrink_threshold	True

Dimensionality Reduction	True
Retained variance	99
Number of iterations	1
Test set size	33.3333
Best hyperparameter select method	max_in_cv

### Output

Accuracy

	NBC	LDA	QDA	NSC
Train	(0.94, 0.0)	(1.0, 0.0)	(0.93, 0.0)	(0.85, 0.0)
Test	(0.82, 0.0)	(0.91, 0.0)	(0.79, 0.0)	(0.79, 0.0)
Overfit degree	125	88.2	142.9	69

### Output

TPR

	NBC	LDA	QDA	NSC
Train	(0.97, 0.0)	(1.0, 0.0)	(0.91, 0.0)	(0.85, 0.0)
Test	(0.94, 0.0)	(0.83, 0.0)	(0.94, 0.0)	(0.72, 0.0)
Overfit degree	26.9	166.7	35.8	153.3

### Output

TNR

	NBC	LDA	QDA	NSC
Train	(0.91, 0.0)	(1.0, 0.0)	(0.94, 0.0)	(0.85, 0.0)
Test	(0.69, 0.0)	(1.0, 0.0)	(0.62, 0.0)	(0.88, 0.0)
Overfit degree	246	0	335.9	25.9

### Output

Execution data

	Execution Data
Number of samples	102
Number of features	12625

Features reduction	63
Params GaussianNB	2
Params LinearDiscriminantAnalysis	6
Params QuadraticDiscriminantAnalysis	5
Params NearestCentroid	2

## Output

### 1.4 Conclusions

We observe that **QDA** performs very poorly in the prostate dataset, given the high dimensionality of this dataset (more than 12625) and the relatively low number of samples (around 100), which do not ease the accurate computation of the covariance matrices.

We perform previously a dimensionality reduction by PCA, were we see that we improve this result.

**NSC** performs in this case much better due to the reduced number of parameters and the feature selection properties of this classifier and more consistently between both cases (prostate and breast).