Classification Systems

Daniel Cerdán, Fernando Freire

March 25, 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.decomposition import PCA
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,
↪  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

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

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 Δ_{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 then 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)

```python
VERBOSE = False

def get_component_number(df_data, desired_variance=99.0, scaling=False):
    """
    Obtain the number of components that explains a %desired_variance
    Args:
        df_data (dataframe): dataframe of features in cols and samples in rows
        desired_variance (float): desired explained variance
        scaling (boolean): True if pre-scaling is needed prior to compute PCA
    Returns:
        int: number of components to maintain to have a explained variance >=
    ↪ desired_variance
        float: variance explained for the nunber of components returned
        numpy array: cumulative variance by number of components retained
    """
    if scaling:
        df_data_2 = preprocessing.StandardScaler().fit_transform(df_data)
    else:
        df_data_2 = df_data
    # project the data into this new PCA space
    pca = PCA().fit(df_data_2)
    desired_variance = desired_variance/100.0
    explained_variance = np.cumsum(pca.explained_variance_ratio_)
    component_number = 0
    for cumulative_variance in explained_variance:
```

```
        component_number += 1
        if cumulative_variance >= desired_variance:
            break
    return component_number, cumulative_variance, explained_variance


def create_datasets_from_file(data_file, header, random_state, label_pos,
                              label_value, features_ini, features_fin=None,
                              with_dim_red=False, retained_variance=99.0):
    """Create training and test sets from file

        Args:
            data_file (string): Name of the data file (csv) of samples a features
            header (string): None or position of the header (pandas read_csv parameter)
            random_state (int): Seed for the random split (as needed for sklearn
    ↪  train_test_split)
            label_pos (int): Column of the labels in data_file
            label_value (int): Value of the label to asign internal '1' value
            features_ini (int): First column of features in data_file
            features_fin (int): Last column + 1 of features in data_file. If None, last
    ↪  column of file.
            with_dim_red (bool): If True, it performs a dimensionality reduction by PCA
            retained_variance (float): If dimensionality reduction, variance to retain

        Returns:
            (np.array): train set scaled
            (np.array): test set scaled
            (np.array): class labels for the train set
            (np.array): class labels for the test set

    """
    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=random_state)
    # 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"

    if with_dim_red:
```

```
74          desired_variance = retained_variance
75          component_number, _, _ =\
76              get_component_number(x_train_scaled, desired_variance, scaling=None)
77          print("Features reduced to", component_number)
78          pca = PCA(n_components = component_number)
79          pca.fit(x_train_scaled)
80          x_train_scaled = pca.transform(x_train_scaled)
81          x_test_scaled = pca.transform(x_test_scaled)
82
83      return x_train_scaled, x_test_scaled, y_train, y_test
84
85  def prediction_accuracy(x_train, x_test, y_train, y_test, method_func, method_param,
    ↪  param_value):
86      """Estimate parameter given training and test sets:
87          Args:
88              x_train (np.array): train set
89              x_test (np.array): test set
90              y_train (np.array): class labels for the train set
91              y_test (np.array): class labels for the test set
92              method_func (string) : name of the learning method
93              param (string): name of learning method parameter
94              param_value (float): value of parameter to try
95          Returns:
96              list of float: train_accuracy, test_accuracy as (TP + TN) / (TN + TP + FP + FN)
97
98      """
99      if method_param != "" :
100         params = {method_param : param_value}
101     else:
102         params ={}
103     method = globals()[method_func](**params)
104
105     # Training
106     method.fit(x_train, y_train)
107
108     # Prediction of test
109     y_pred = method.predict(x_test)
110     conf = confusion_matrix(y_test, y_pred, labels=[1,0])
111     # With this order of labels:
112     #   TP in 0,0
113     #   FN in 0,1
114     #   TN in 1,1
115     #   FP in 1,0
116     TP = conf[0][0]
117     TN = conf[1][1]
118     FN = conf[0][1]
119     FP = conf[1][0]
120     #print(conf)
121     test_accuracy = (TP + TN) / (TN + TP + FP + FN)
122
123     if VERBOSE: print("Score test,", method.score(x_test, y_test, sample_weight=None))
124     #print('True positive rate is: %f' % (TP / (TP + FN)))
```

```
    #print('True negative rate is: %f\n' % (TN / (TN + FP)))

    # Prediction of train
    y_pred = method.predict(x_train)
    conf = confusion_matrix(y_train, y_pred, labels=[1,0])
    if VERBOSE: print("Train set conf matrix.",conf)
    if VERBOSE: print("Score train,", method.score(x_train, y_train, sample_weight=None))
    TP = conf[0][0]
    TN = conf[1][1]
    FN = conf[0][1]
    FP = conf[1][0]
    #print(conf)
    train_accuracy = (TP + TN) * 1.0 / (TN + TP + FP + FN)
    if VERBOSE: print('True positive rate of train set is: %f' % (TP / (TP + FN)))
    if VERBOSE: print('True negative rate of train set is: %f\n' % (TN / (TN + FP)))
    return [train_accuracy, test_accuracy]

def estimate_parameter(x_train, x_test, y_train, y_test,
                       method_func, param, param_values,
                       best_param_value_method="max_in_test"):
    """Estimate parameter given training and test sets:
        Args:
            x_train (np.array): train set
            x_test (np.array): test set
            y_train (np.array): class labels for the train set
            y_test (np.array): class labels for the test set
            method_func (string) : name of the learning method
            param (string): name of learning method parameter
            param_values (list of float): list of parameter values to try
            best_param_value_method: if "max_in_test" gives the value with the maximum
    ↪   accuracy
                                      in test data.
        Returns:
            (float): best parameter value to use in prediction

    """
    # Pipeline for estimate the regularization parameter
    pipeline = Pipeline([ ('method', globals()[method_func]()) ])

    # Construct the grid the hyperparameter candidate shronk theshold
    param_grid = { 'method__' + param : param_values }

    # Evaluating
    skfold = RepeatedStratifiedKFold(n_splits=10, n_repeats=1, random_state=0)
    gridcv = GridSearchCV(pipeline, cv=skfold, n_jobs=1, param_grid=param_grid,\
            scoring=make_scorer(accuracy_score))
    result = gridcv.fit(x_train, y_train)

    # Accuracies
    accuracies = gridcv.cv_results_['mean_test_score']
    std_accuracies = gridcv.cv_results_['std_test_score']

```

```
    test_accuracies = np.ones(len(param_values))

    for i in range(len(param_values)):
        method_params = {param : param_values[ i ]}
        method = globals()[method_func](**method_params)
        method.fit(x_train, y_train)
        test_accuracies[ i ] = accuracy_score(method.predict(x_test), y_test)

    max_test_accuracy = max(test_accuracies)

    # Obtain best_param_value as max
    if best_param_value_method == "max_in_test":
        best_param_value = 0
        best_train_accuracy = 0
        for i in range(len(param_values)):
            if test_accuracies[ i ] == max_test_accuracy:
                if accuracies[i] > best_train_accuracy:
                    best_train_accuracy = accuracies[i]
                    best_param_value = param_values[i]
    else:
        best_param_value = param_values[ np.argmax(accuracies) ]
    # Plot
    if not DISABLE_PLOTS:
        plt.figure(figsize=(9, 9))
        line1, = plt.plot(param_values, accuracies, 'o-', color="g")
        line2, = plt.plot(param_values, test_accuracies, 'x-', color="r")
        plt.fill_between(param_values, accuracies - std_accuracies / np.sqrt(10), \
            accuracies + std_accuracies / np.sqrt(10), alpha=0.1, color="g")
        plt.grid()
        plt.title("Different hyper-parameter " + param + " values for " + method_func)
        plt.xlabel('Hyper-parameter')
        plt.xticks(np.round(np.array(param_values), 2))
        plt.ylabel('Classification Accuracy')
        plt.ylim((min(min(accuracies), min(test_accuracies)) - 0.1,
                  min(1.02, max(max(accuracies), max(test_accuracies))  + 0.1)))

        plt.xlim((min(param_values), max(param_values)))
        legend_handles = [ mlines.Line2D([], [], color='g', marker='o', \
                                 markersize=15, label='CV-estimate'), \
                           mlines.Line2D([], [], color='r', marker='x', \
                                 markersize=15, label='Test set estimate')]
        plt.legend(handles=legend_handles, loc = 3)
        plt.show()

    print("Best param value %s Method %s: %s" % (method_func, best_param_value_method,
    ↪  best_param_value))
    return best_param_value

def print_accuracies(accuracy_NBC, accuracy_LDA, accuracy_QDA, accuracy_NSC):
    print("")
    print("Accuracies")
    d = {'NBC': accuracy_NBC, 'LDA': accuracy_LDA, 'QDA': accuracy_QDA,'NSC': accuracy_NSC}
```

```
227     df = pd.DataFrame(data = d, index = ['Train', 'Test'])
228     display(df)
229     print("")
230 
231 def learn_dataset(data_file, header, random_state, label_pos,
232                   label_value, features_ini, features_fin=None,
233                   best_param_value_method="max_in_test",
234                   with_dim_red=False, retained_variance=99.0):
235     """Learn data sets from file, methods:
236             1. The Naive Bayes Classifier
237             2. LDA
238             3. QDA
239             4. Nearest Shrunken Centroids Classifier
240         Args:
241             data_file (string): Name of the data file (csv) of samples a features
242             header (string): None or position of the header (pandas read_csv parameter)
243             random_state (int): Seed for the random split of sets (as needed for sklearn
    ↪  train_test_split)
244             label_pos (int): Column of the labels in data_file
245             label_value (int): Value of the label to asign internal '1' value. We consider
    ↪  this label as
246             the positive label in prediction validation. We asign malign or cancer status to
    ↪  this label.
247             features_ini (int): First column of features in data_file
248             features_fin (int): Last column + 1 of features in data_file. If None, last
    ↪  column of file
249             best_param_value_method (str): if "max_in_test" gives the value with the maximum
    ↪  accuracy
250                                     in test data
251             with_dim_red (bool): If True, it performs a dimensionality reduction by PCA
252             retained_variance (float): If dimensionality reduction, variance to retain
253 
254     """
255     X_train_scaled, X_test_scaled, y_train, y_test = \
256         create_datasets_from_file(data_file, header, random_state,
257                                   label_pos, label_value, features_ini,
                                      ↪  features_fin=features_fin,
258                                   with_dim_red=with_dim_red,
                                      ↪  retained_variance=retained_variance)
259     if VERBOSE: print(X_train_scaled.shape)
260 
261     if VERBOSE: print("NBC")
262     # Naive Bayes accuracy
263     accuracy_NBC = prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test,
        ↪  "GaussianNB", "", "")
264 
265     # LDA accuracy
266     if VERBOSE: print("LDA")
267     accuracy_LDA = prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test,
        ↪  "LinearDiscriminantAnalysis", "", "")
268 
269     # QDA estimate reg parameter
```

```
    if VERBOSE: print("QDA")
    param_values = np.linspace(0, 1, 10).tolist()
    best_param_value = estimate_parameter(X_train_scaled, X_test_scaled, y_train, y_test,\
                       "QuadraticDiscriminantAnalysis", "reg_param", param_values,\
                        best_param_value_method)
    # QDA accuracy
    # Best parameter reg value according CV estimate
    accuracy_QDA = prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test, \
                        "QuadraticDiscriminantAnalysis", "reg_param", best_param_value)

    # Centroids
    if VERBOSE: print("NSC")
    # Best parameter shrink_threshold value according CV estimate
    param_values = np.linspace(0, 8, 20).tolist()
    best_param_value = estimate_parameter(X_train_scaled, X_test_scaled, y_train, y_test,\
                       "NearestCentroid", "shrink_threshold", param_values,\
                        best_param_value_method)
    # Centroids accuracy
    accuracy_NSC = prediction_accuracy(X_train_scaled, X_test_scaled, y_train, y_test,\
                                       "NearestCentroid", "shrink_threshold",
                                       ↪  best_param_value)
    print_accuracies(accuracy_NBC, accuracy_LDA, accuracy_QDA, accuracy_NSC)
```

1.2 Breast cancer

Script 1.2.1 (python)

```
# Breast Cancer
DISABLE_PLOTS = True
VERBOSE = False
DIM_RED = False
#learn_dataset(data_file, None, 1, 1, "M", 2, features_fin = None, with_dim_red = False)
learn_dataset(data_file = './data/wdbc.csv', header = None, random_state=1,
              label_pos=1, label_value="M", features_ini = 2, features_fin = None,
              best_param_value_method="max_in_test",
              with_dim_red = DIM_RED, retained_variance = 99.0)

learn_dataset(data_file = './data/wdbc.csv', header = None, random_state=1,
              label_pos=1, label_value="M", features_ini = 2, features_fin = None,
              best_param_value_method="max_in_cv",
              with_dim_red = DIM_RED, retained_variance = 99.0)
```

Output

```
Best param value QuadraticDiscriminantAnalysis Method max_in_test: 0.5555555555555556
Best param value NearestCentroid Method max_in_test: 3.3684210526315788

Accuracies

          NBC        LDA        QDA        NSC
```

```
Train  0.936675  0.973615  0.970976  0.931398
Test   0.936842  0.963158  0.978947  0.952632
```

Output

```
Best param value QuadraticDiscriminantAnalysis Method max_in_cv: 0.5555555555555556
Best param value NearestCentroid Method max_in_cv: 7.578947368421052

Accuracies
```

```
            NBC       LDA       QDA       NSC
Train  0.936675  0.973615  0.970976  0.944591
Test   0.936842  0.963158  0.978947  0.926316
```

Output

1.3 Prostate cancer

Script 1.3.1 (python)

```python
# Prostate Cancer
#learn_dataset(data_file, 0, 1, -1, 1, 0, -1, with_dim_red = True)
DISABLE_PLOTS = True
VERBOSE = False
DIM_RED = True
learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
              label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
              best_param_value_method = "max_in_test",
              with_dim_red = DIM_RED, retained_variance = 99.0)

learn_dataset(data_file = './data/prostate.csv', header = 0, random_state = 1,
              label_pos = -1, label_value = 1, features_ini = 0, features_fin = -1,
              best_param_value_method = "max_in_cv",
              with_dim_red = DIM_RED, retained_variance = 99.0)
```

Output

```
Features reduced to 63
Best param value QuadraticDiscriminantAnalysis Method max_in_test: 0.7777777777777777
Best param value NearestCentroid Method max_in_test: 0.42105263157894735

Accuracies
```

```
            NBC       LDA       QDA       NSC
Train  0.941176  1.000000  0.838235  0.882353
Test   0.823529  0.911765  0.852941  0.852941
```

Output

```
Features reduced to 63
Best param value QuadraticDiscriminantAnalysis Method max_in_cv: 0.0
Best param value NearestCentroid Method max_in_cv: 0.8421052631578947

Accuracies
```

```
            NBC       LDA       QDA       NSC
Train  0.941176  1.000000  1.000000  0.852941
Test   0.823529  0.911765  0.764706  0.794118
```

Output

1.4 Conclusions

We observe that **QDA** performs very poorly in the prostate dataset, given the high dimensionality of this dataset, which do not ease the accurate computation of the covariance matrices. Perhaps if we perform previously a dimensionality reduction by PCA, we'll 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).