ECSE-551 First Mini-project Report of Group 23: Logistic Regression Analysis of Binary Class Data Sets

Ali Raoofian[a], Asal Zabetian Hosseini[b] and Alaa S. Abdelgawad[c]

[a]*Department of Mechanical Engineering and Centre for Intelligent Machines, McGill University, Montréal, Canada*
[b]*Department of Electrical and Computer Engineering, McGill University, Montréal, QC, Canada*
[c]*Department of Biomedical Engineering, McGill University, Montréal, QC, Canada*

ARTICLE INFO

Keywords:
Logistic regression
Binary classification
Data analysis
Machine learning
Python coding

ABSTRACT

In this study, two sample data sets are analyzed using logistic regression approach. To this end, first and by using logistic regression, the data sets are fitted in 1st to 4th order models. Then, using cross validation method, the best model is picked. To this end, different combination of learning rates and iteration numbers are used and compared to each other. At the end, methods such as, normalization, feature selection and oversampling were used to improve the accuracy of the picked fitting model.

1. Introduction

It is an undeniable fact that industry and research sectors are always facing non-linear problems which need to be modelled and solved. Realistic problems in natural sciences and engineering often consists of hybrid models with different attributes and characteristics. Generally, functionality of these problems cannot be specified with explicit analytical expressions. Hence, the only information in hand are the numerical data sets which are usually given by experience, statistical information or testing. However, predicting and modelling the behaviour of the numerical data sets associated with these problems is found to be a challenging task. This fact is a major driving force for developing efficient *machine learning* techniques to model the behaviour of numerical data sets efficiently, i.e., with minimum computation cost and maximum accuracy. Once the efficient model is obtained, it can enable us to predict unseen inputs in a reliable way.

In this study, two sample data sets are analyzed using logistic regression approach. To this end, first and by using logistic regression, the data sets are fitted in 1st to 4th order models. Then, using cross validation method, the best model is picked. At the end, methods such as, normalization, feature selection and oversampling were used to improve the accuracy of the fitting.

2. Data Sets Description

There are two classification data sets that are going to be studied in this project. The first data set is related to classification of patients diagnosed with hepatitis in which 19 features associated

ali.raoofian@mail.mcgill.ca (A. Raoofian); asal.zabetian-hosseini@mail.mcgill.ca (A. Zabetian Hosseini); alaa.abdelgawad@mail.mcgill.ca (A. S. Abdelgawad)

STUDENT ID(s): 260856711 (A. Raoofian); 260859392 (A. Zabetian Hosseini); 260912527 (A. S. Abdelgawad)

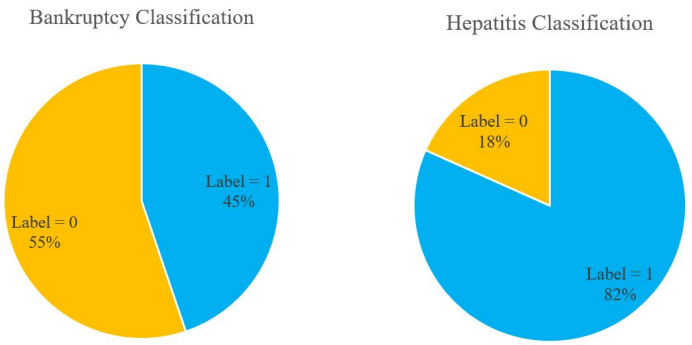


Figure 1: The class label data distribution in (a) Bankruptcy and, (b) Hepatitis patients.

with the patients are measured as attributes and, the target is to discriminate two classes: survivors and patients for whom the hepatitis proved terminal. The second data set is related to bankruptcy status prediction in which various econometric attributes are investigated. The number of features in this data set is 64. The distribution of class labels in each data set is shown in Fig. 1

2.1. First Data Set: Patients Diagnosed with Hepatitis

In this data set, the observation matrix $\mathbf{X}$ has a dimension of 142×19. Hence, there are 142 training instances and 19 attributes associated with this data. Most of the features in this data set are binary. The class label data distribution of this data set is shown in Fig. 1b. As it is shown in the Fig. 1b, %82 of the class labels belong to a single class which is known as *imbalanced class*, referred to data sets with a disproportionate ratio of observations in each class.

2.2. Second Data Set: Bankruptcy Status Prediction

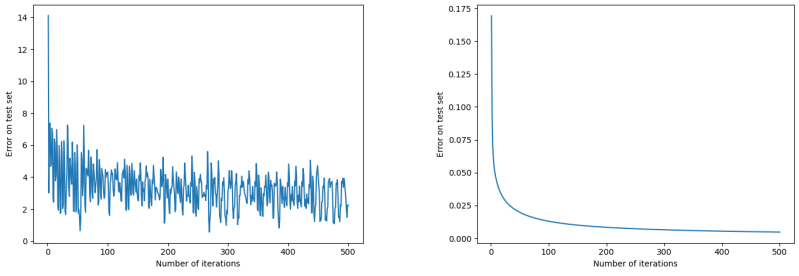
The observation matrix $\mathbf{X}$ in this data set is a 453×64 matrix which means there are 453 training instances and 64 attributes associated with the model. Unlike the first data set, the ratio of observations in each class is near 1 and sample distribution is balanced in this case (Fig. 1a).

3. Results

The two data sets are fitted using logistic regression method and the results are shown here. There are multiple challenges regarding fitting a data set. The first challenge is related to convergence. The equation to find the optimal weighting vector for local minimum is:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \left[\sum_{i=1:n} \mathbf{x}_i \left(\mathbf{y}_i - \sigma(\mathbf{w}_k^{\mathrm{T}} \mathbf{x}_i) \right) \right] \tag{1}$$

where α_k is the learning rate and σ is the sigmoid function. Assigning different values to α_k will change the convergence quality. In current study, two main approaches are considered to define α_k. One can assign a constant value to α_k or define it as a function of k to satisfy the Robbins-Monroe conditions, e.g., define the learning rate as $\alpha_k = 1/(1 + \beta k)$; where β is a constant value and k is the number of iterations. The first approach is referred to as constant learning rate and the latter is adaptive learning rate. $\hat{\mathbf{w}}$ is obtained using iterations over Eq. (1). The iteration stops, when



(a) (b)

Figure 2: Error $\|\mathbf{w}_{k+1} - \mathbf{w}_k\|$, vs number of iterations for two arbitrary data sets.

the error between $\mathbf{w}_{k+1}$ and $\mathbf{w}_k$ reaches a threshold, i.e., $\|\mathbf{w}_{k+1} - \mathbf{w}_k\| < \epsilon$. The key idea about convergence is to monitor the error and make sure that it is constantly decreasing; like showing in Fig. 2b. If the error fluctuates as Fig. 2a, it means that we are missing the local minimum point.

For the bankruptcy data, we first distributed the instances (rows of matrix $\mathbf{X}$) randomly since first 250 class labels are all 0 and the rest are 1. This can impose some errors in cross validation because in that case, there is high probability that all validation fold class labels belong to the same label. The bankruptcy data can be modeled with different fitting orders; here we fitted the data using 1st order to 4th order. In order to find out which model suits the best, a 10-fold cross validation method is used. The cross validation is done using both constant and adaptive learning rates. The errors obtained in each fold are shown in Table. 1. According to Table. 1, the obtained mean error for bankruptcy is the lowest for the 1st order fitting. This holds true for both constant and adaptive learning rates. Hence, the 1st order fitting model is picked as the optimal solution for bankruptcy data. For the adaptive learning, the error threshold and learning rates of $\epsilon = 0.001$ and $\alpha_k = 1/(1 + \beta k)$ are used, respectively. Defining small ϵ for constant learning rates will keep the code in constant loop for some cases. Thus, for constant learning rates, a limit of 8000 iterations is used and the $\hat{\mathbf{w}}$ is given after 8000 iterations. The values used for α and β in both learning approaches are shown in Table. 1. As stated before, it is important to monitor the error $\|\mathbf{w}_{k+1} - \mathbf{w}_k\|$ to make sure it is not fluctuating. For instance, we tested our model using different values of learning rate 0.0001, 0.001, 0.1 and 1 for an arbitrary configuration. As the α increases accuracy was decreasing: %79.48, %80.0, %64.35, %34.10, respectively to the alpha values.

Similar to what we did for the bankruptcy data, the Hepatitis data is also fitted using 1st order to 4th order modelling. Again, a 10-fold cross validation method is used using both constant and adaptive learning rates. In each CPU run, we monitored error figure and tuned the values of α and β to avoid having error figures shown in Fig. 2a. After a cross fold validation, the errors obtained in each fold alongside the mean error are shown in Table. 1. According to Table. 1, the obtained mean error is the lowest for 1st order fitting. This holds true for both constant and adaptive learning rates. However, the obtained accuracy values are not quite reliable due to the imbalanced data distribution in the Hepatitis data. According to Fig. 1b, %82 of the class labels belong to class 1. This introduce bias in accuracy calculation and as it is evident from Table. 1, most of the obtained accuracy values are near %83. For the adaptive learning, the same error threshold and learning rate function of $\epsilon = 0.001$ and $\alpha_k = 1/(1 + \beta k)$ are used, respectively. Also, the iteration limit and learning rate α are shown in Table. 1 for constant learning approach.

Table 1
Errors obtained in cross-fold validation for the bankruptcy and Hepatitis data sets alongside the number of iterations, run time, accuracy and learning parameters (the values are obtained with constant learning rate and the values in parenthesis are obtained with adaptive learning approach).

	Bankruptcy				**Hepatitis**			
	Order 1 fit	Order 2 fit	Order 3 fit	Order 4 fit	Order 1 fit	Order 2 fit	Order 3 fit	Order 4 fit
Fold 1 err(%)	23.07(30.76)‡	25.64(25.64)	25.64(35.89)	33.33(38.48)	14.28(14.28)	0(14.28)	14.28(28.57)	14.28(14.28)
Fold 2 err(%)	23.07(28.2)	20.51(25.64)	23.07(28.2)	23.07(38.46)	14.28(14.28)	14.28(14.28)	14.28(14.28)	14.28(14.28)
Fold 3 err(%)	25.64(30.76)	23.07(33.33)	25.64(30.76)	30.76(33.33)	14.28(14.28)	14.28(14.28)	14.28(14.28)	14.28(14.28)
Fold 4 err(%)	15.38(17.94)	15.38(25.64)	17.94(20.51)	17.94(41.02)	28.57(28.57)	28.57(28.57)	28.57(28.57)	28.57(28.57)
Fold 5 err(%)	17.94(20.51)	17.94(28.2)	12.82(12.82)	12.82(30.76)	42.85(42.85)	42.85(42.85)	42.85(42.85)	42.85(42.85)
Fold 6 err(%)	25.64(28.2)	23.07(25.64)	25.64(25.64)	23.07(33.33)	14.28(14.28)	14.28(0)	14.28(0)	14.28(14.28)
Fold 7 err(%)	25.64(25.64)	28.2(25.64)	33.33(33.33)	33.33(38.46)	0(0)	14.28(14.28)	14.28(0)	14.28(14.28)
Fold 8 err(%)	23.07(28.2)	23.07(30.78)	23.07(30.76)	23.07(33.33)	28.57(28.57)	28.57(28.57)	28.57(28.57)	28.57(28.57)
Fold 9 err(%)	17.94(17.94)	28.2(25.64)	25.64(38.46)	25.64(33.33)	0(0)	0(14.28)	0(0)	0(0)
Fold 10 err(%)	10.25(17.94)	10.25(15.38)	12.82(12.82)	17.94(25.64)	0(0)	0(28.57)	0(0)	0(14.28)
Mean err(%)	20.77(24.62)	21.54(26.15)	22.56(26.92)	24.10(34.62)	15.71(15.71)	17.14(21.43)	17.14(15.71)	17.14(18.57)
Accuracy(%)	**79.23**(75.38)	78.46(73.85)	77.44(73.08)	75.89(65.38)	**84.29**(84.29)	82.86(78.57)	82.86(84.29)	82.86(81.43)
Run time(s)	18.15(0.58)	21.26(0.34)	23.54(1.28)	26.22(16.04)	0.99(0.14)	1.13(0.15)	2.34(0.16)	16.63(21.5)
# of Iter.	8000(173)	8000(60)	8000(352)	8000(1746)	8000(7)	8000(5)	8000(52)	8000(10482)
$\alpha(\beta)^*$	0.001(1E+2)	0.001(1E+3)	0.00001(1E+2)	0.000001(1E+2)	1E-4(1E+5)	1E-9(1E+8)	1E-14(1E+9)	1E-18(1E+10)

*In adaptive learning, the error threshold $\epsilon = 0.001$ is considered.
‡The values in parenthesis are obtained with adaptive learning approach

4. Additional remarks

Ridge regression and normalization: There are multiple techniques to pre-process and tune the data. Here, we employed 1) data normalization and 2) Regularization methods. According to the results obtained from Sec.3, in the best case, the total accuracy values of %79.23 (with constant learning rate) and %75.28(with adaptive learning) were obtained for 1st order bankruptcy data set. Here, we fitted the bankruptcy data set again, by introducing Ridge regression into our problem. Different L2-penalty values were used. The whole procedure was also repeated for normalized data sets. The results are shown in Table. 2. According to Table. 2, the total accuracy of the 1st order fitting model of bankruptcy can rise to %80 by using normalized training data alongside L2-penalty value of 0.5. This holds true for both constant and adaptive learning approaches.

This is also done for the Hepatitis data set and the results are shown in Table. 3. According to Table. 3, the total accuracy of the 1st order fitting model of Hepatitis can rise to %88.57 by using normalized training data alongside L2-penalty value of 0.1 and using constant learning rate approach. By noting to Table. 3, it is evident that using different L2-penalty values does not change the output.

Feature selection: We also tried to improve the fitting by using feature selection techniques. Here, we used the random forest method form Scikit learn library. After calculation the correlations between the features and scoring them, only the ones with the score greater than the mean of all features are selected to keep. This leaded us to drop 9 and 18 features from Hepatitis and bankruptcy data sets, receptively. This imposed no change in maximum accuracy of Hepatitis data (%80) but decreased the bankruptcy maximum accuracy from %80 to %76.15.

Over sampling: Due to the imbalanced class labels in Hepatitis data set, we tried oversampling it by adding random training data to class 0. This led to increase of one percent in the maximum accuracy value (%88.57).

Table 2
Obtained accuracy values for 1-order bankruptcy data set using Ridge regression and normalized training data.

		Classic Reg.	Ridge Reg. L2 Pen. = 0.1	L2 Pen. = 0.2	L2 Pen. = 0.5	L2 Pen. = 0.7	L2 Pen. = 1	L2 Pen. = 5
Constant learning rate approach	Accuracy α	79.23(79.75)‡	79.74(79.48) 0.001	78.97(79.74) 0.001	78.72(**80**) 0.001	79.23(79.23) 0.001	79.23(79.49) 0.001	76.92(79.23) 0.001
Adaptive learning approach	Accuracy β	75.38(77.18)	77.95(78.97) 1E+4	79.23(79.74) 1E+4	78.72(**80**) 1E+4	79.23(79.23) 1E+4	79.23(79.48) 1E+4	76.92(79.23) 1E+4

‡The values in parenthesis are the obtained accuracy with **normalized** training data

Table 3
Obtained accuracy values for 1-order Hepatitis data set using Ridge regression and normalized training data.

		Classic Reg.	Ridge Reg. L2 Pen. = 0.1	L2 Pen. = 0.2	L2 Pen. = 0.5	L2 Pen. = 0.7	L2 Pen. = 1	L2 Pen. = 5
Constant learning rate approach	Accuracy α	84.28(85.71)‡	85.71(88.57) 0.001	85.71(88.57) 0.001	85.71(88.57) 0.001	85.71(88.57) 0.001	85.71(88.57) 0.001	85.71(88.57) 0.001
Adaptive learning approach	Accuracy β	84.29(84.29)	84.29(84.29) 1	84.29(84.29) 1	84.29(84.29) 1	84.29(84.29) 1	84.29(84.29) 1	84.29(84.29) 1

‡The values in parenthesis are the obtained accuracy with **normalized** training data

5. Discussion and Conclusion

In this work, we modeled two different data sets, bankruptcy and Hepatitis, using logistic regression algorithms. First, the training sets were randomly ordered to distribute class labels evenly to each fold. Then, using constant and adaptive (function of iteration count) learning rates, the data sets were fitted from order 1 to 4. In order to pick the best model, a 10-fold cross validation was employed and the corresponding errors and accuracy values were reported. Also, the number of convergence iterations and run times were brought up for each model. In most cases, both the convergence iterations and run times have lower values for adaptive learning since it changes accordingly to reach a certain error threshold faster. By comparing the results obtained from cross validation, a 1st order model was selected for both bankruptcy and Hepatitis as its corresponding error was minimum for both data sets. In bankruptcy, the accuracy of modeling in the 1st order fitting are %79.23 and %75.38 using constant and adaptive learning rates. If normalized data set and Ridge regression are employed, both accuracy values will rise uo to %80. Also, by using feature selection, 18 features were dropped from bankruptcy data set and the maximum accuracy dropped to %76.15. For the Hepatitis data, since the data is imbalanced (ratio of class labels is 9/41) the obtained results are not completely reliable. In this case, the maximum accuracy which is associated with 1st order model, is %84.29 for both constant and adaptive learning methods. If normalized data set and Ridge regression are used, the maximum accuracy will rise to %88.57. Next, by using feature selection and dropping 9 features from Hepatitis data, the maximum accuracy did not change. However, by employing oversampling, the maximum accuracy (%88.57) will increase by one percent.

6. Statement of Contributions

All the group members were involved equally in developing the code and writing the report.

A. Appendix: Codes

```
# Importing the libraries
import numpy as np
import pandas as pd
import random
import math
import matplotlib.pyplot as plt
import pickle
import timeit


start = timeit.default_timer() # start to calculate run time
# This function randomly distributes the rows of training data
def data_rnd_distribute(x, y, rnd_rows):
# # creat a random order for rows
# rnd_rows = random.sample(range(x.shape[0]), x.shape[0])
# Initialize new X and y matrices
x_new = np.empty((0, x.shape[1]), float)
y_new = np.empty((0, 1), float)
# put each row in sequel
for j in range(len(rnd_rows)):
row_ = x[rnd_rows[j], :]  # jth random row
class_ = y[rnd_rows[j]] # jth random class
row_ = np.c_[row_] # transform into numpy array
class_ = np.c_[class_] # transform into numpy array
# Add the new row and class
x_new = np.append(x_new, row_.T, axis=0)
y_new = np.append(y_new, class_, axis=0)
return x_new, y_new


def normalize_data(in_data):
x = in_data[:, 0:-1] # first remove the dummy features
features_mean = []
features_std_deviation = []
for i in range(x.shape[1]):
feature_i_mean = x[:, i].mean() # calculate mean of each attribute set
# calculate standard deviation of each attribute set
feature_i_std_deviation = np.std(x[:, i])
# feature_i_std_deviation = x[:, i].max() - x[:, i].min()
features_mean.append(feature_i_mean)
features_std_deviation.append(feature_i_std_deviation)
```

```
# converting list to numpy array
features_mean = np.asarray(features_mean)
features_std_deviation = np.asarray(features_std_deviation)

x_norm_ = (x - features_mean) / features_std_deviation
# Adding the dummy features back to the data set
x_norm = np.c_[x_norm_, np.ones((x_norm_.shape[0], 1))]
return x_norm


def set_order(in_data, order):
data = in_data[:, 0:-1] # first remove the dummy features
if order <= 1:
data_new = data
else:
data_new = data
i = 2
while i <= order:
data_new = np.hstack((data ** i, data_new))
i = i + 1
# Adding the dummy features back to the data set
data_new = np.c_[data_new, np.ones((data_new.shape[0], 1))]
return data_new


# Import data
def get_data(data_path):
data = pd.read_csv(data_path)
return data


def sigmoid(variable):
# The Sigmoid function
return 1 / (1 + np.exp(-variable))


def gradient_vector(x, y, w, rig_reg, l2_pen):
# Computes the gradient of the cost function at the point theta
z = np.dot(x, w)
if rig_reg:
dw = 2 * np.dot(x.T, sigmoid(z) - y) + 2 * l2_pen * w
else:
dw = np.dot(x.T, sigmoid(z) - y)
```

```
return dw


class LogisticReg:
def __init__(self, adaptive_iteration, epsilon, alpha, num_it, k, x, y, rig_reg,
l2_pen):
self.alpha = alpha
self.epsilon = epsilon
self.x = x
self.y = y
self.w = np.zeros((self.x.shape[1], 1)) # Initializing vector of weights
self.it_list = []
self.err_list = []
self.k = k
self.num_it = num_it
self.adaptive_iteration = adaptive_iteration
self.rig_reg = rig_reg
self.l2_pen = l2_pen


def fit(self):
it_list = []
err_list = []
it = 1 # initializing the iterations

if self.adaptive_iteration:

# Initializing err
err = self.epsilon + 0.5
while err >= self.epsilon:
self.alpha = 1 / (1 + 1 * it)
delta = gradient_vector(self.x, self.y, self.w, self.rig_reg, self.l2_pen)
w_old = self.w # store w for comparison
self.w = self.w - self.alpha * delta
err = np.linalg.norm(self.w - w_old) # Calculate the norm-2 of the difference
it_list.append(it)
err_list.append(err)
it += 1

else:

for i in range(self.num_it):
delta = gradient_vector(self.x, self.y, self.w, self.rig_reg, self.l2_pen)
w_old = self.w # store w for comparison
```

```
self.w = self.w - self.alpha * delta
err = np.linalg.norm(self.w - w_old) # Calculate the norm-2 of the difference
it_list.append(it)
err_list.append(err)
it += 1
self.it_list = it_list
self.err_list = err_list
return self.w, self.it_list, self.err_list


def predict(self, x):
z = np.dot(x, self.w)
return sigmoid(z)


def cross_validate(plotting, adaptive_iteration, rig_reg, l2_pen, epsilon, alpha,
num_it, k, x, y):
total_err = []
total_acc = []
data = np.c_[x, y]
fold_size = math.ceil(data.shape[0] / k) # round up the fold size
training_set_size = (k - 1) * fold_size
validation_set_size = data.shape[0] - training_set_size
for i in range(k):
data_validation = data[fold_size * i:  fold_size * i + validation_set_size, :]
if i == 0:
data_training_upper = np.empty((0, data.shape[1]), float)
else:
data_training_upper = data[0:  fold_size * i, :]
if i == k - 1:
data_training_lower = np.empty((0, data.shape[1]), float)
else:
data_training_lower = data[fold_size * i + validation_set_size:  data.shape[0],
:]
data_training = np.concatenate((data_training_upper, data_training_lower), axis=0)
x_training = data_training[:, 0:  x.shape[1]] # separate the features and classes
y_training = data_training[:, -1]
y_training = np.c_[y_training] # make y_training a (n,1) vector instead of (n,)
x_validation = data_validation[:, 0:  x.shape[1]] # separate the features and
classes
y_validation = data_validation[:, -1]
y_validation = np.c_[y_validation] # make y_training a (n,1) vector instead of
(n,)
```

```
model = LogisticReg(adaptive_iteration, epsilon, alpha, num_it, k, x_training,
y_training, rig_reg, l2_pen)
w_, it_list, err_list = model.fit()
y_guessed = model.predict(x_validation)

for j in range(len(y_guessed)):
if y_guessed[j] < 0.5:
y_guessed[j] = 0
else:
y_guessed[j] = 1

# Calculate the norm-2 of the difference
# fold_err = np.linalg.norm(y_guessed - y_validation)
# Calculate the norm-2 of the difference
fold_err = (y_guessed != y_validation).mean() * 100
fold_acc = (y_guessed == y_validation).mean() * 100
print('The', i + 1, 'th fold error and accuracy are', fold_err, 'and', fold_acc,
'respectively')
total_err.append(fold_err)
total_acc.append(fold_acc)

if i == 0 & plotting:
plt.xlabel('Number of iterations')
plt.ylabel('Error on test set')
plt.plot(it_list, err_list)
plt.show(block=False)

mean_err = sum(total_err) / len(total_err)
mean_acc = sum(total_acc) / len(total_acc)
print('The total error and accuracy of this model are', mean_err, 'and', mean_acc,
'respectively')
return len(it_list)


# ////////////////////////////////////////////////////////////////////////////
# ////////////////////////////////// START HERE //////////////////////////////
# ////////////////////////////////////////////////////////////////////////////

# ============================================================ Define problem
variables
path = 'data/hepatitis.csv'
# ============================================================

data_raw = get_data(path)
```

```
# X = feature values, all the columns except the last column
X_raw = data_raw.iloc[:, 0:-1]
# y = target values, last column of the data set
y_raw = data_raw.iloc[:, -1]
# making sure all the data is float
y_raw = y_raw.astype(float)
# Adding the dummy features to the data set
X_raw = np.c_[X_raw, np.ones((X_raw.shape[0], 1))]
# Converting y to a numpy array
y_raw = np.c_[y_raw]
print('Dimension of X is', X_raw.shape, '; subsequently the dimension of y is',
y_raw.shape)
# ============================================================
# rnd_rows = random.sample(range(X_raw.shape[0]), X_raw.shape[0])
# print('first data', rnd_rows)
# with open('rows.data', 'wb') as filehandle:
# # store the data as binary data stream
# pickle.dump(rnd_rows, filehandle)
with open('rows.data', 'rb') as filehandle:
# read the data as binary data stream
rnd_rows = pickle.load(filehandle)
print('second data', rnd_rows)
# ============================================================

k_ = 10
order_ = 2
normalization = False
adaptive_iteration_ = True
Ridge_reg_ = False
plotting_ = True
# ------------ Adaptive iteration method
epsilon_ = 0.01
# ------------ fixed number of iteration method
alpha_ = 0.0001
num_it_ = 8000
# ------------ Ridge Regression L2-Penalty
l2_pen_ = 0.5
# ============================================================

if normalization:
X_raw = normalize_data(X_raw)
X_raw = set_order(X_raw, order_)
X_, y_ = data_rnd_distribute(X_raw, y_raw, rnd_rows)
num_iter = cross_validate(plotting_, adaptive_iteration_, Ridge_reg_, l2_pen_,
```

```
epsilon_, alpha_, num_it_, k_, X_, y_)
# ---------------------------- End
stop = timeit.default_timer()
print('Time:  ', stop - start)
print('# of Iterations:', num_iter)
```