

```

1  # -*- coding: utf-8 -*-
2  """Hyperparameter_Testing
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1Tkjs6AN6HaRH09FqtLdYzPa6BQHDaYwC
8
9  <center><h1>Mini Project 1 - Logistic Regression</h1>
10 <h4>This is a testing file aiming to find the best hyperparameters for the
    model.</h4></center>
11
12 <h3>Team Members:</h3>
13 <center>
14 Yi Zhu, 260716006<br>
15 Fei Peng, 260712440<br>
16 Yukai Zhang, 260710915
17 </center>
18 """
19
20 from google.colab import drive
21 drive.mount('/content/drive')
22
23 import numpy as np
24 import pandas as pd
25 import matplotlib.pyplot as plt
26 import seaborn as sns
27 import time
28
29 class LogisticRegression:
30     '''
31         This is the logistic regression class, containing fit, perdict and
32         accu_eval functions,
33         as well as many other useful functions.
34     '''
35     def __init__(self, data, folds, lr=0.01, max_iter=10000, beta=0.99, epsilon
36 =5e-3):
37         self.data = data
38         self.folds = folds
39         self.lr = lr
40         self.max_iter = max_iter
41         self.beta = beta
42         self.epsilon = epsilon
43     def shuffle_data(self):
44         '''
45             This function randomly shuffles the input dataset.
46         '''
47         # Load data from data file.
48         self.data.insert(0, column='Bias', value=1)
49
50         self.data = self.data.sample(frac=1)
51
52     def set_learning_rate(self, lr):
53         self.lr = lr
54
55     def set_max_iter(self, max_iter):
56         self.max_iter = max_iter
57
58     def set_epsilon(self, epsilon):
59         self.epsilon = epsilon
60

```

```

61     def set_beta(self, beta):
62         self.beta = beta
63
64     def partition(self, fold):
65         '''
66             This function divides the dataset into training and validation set
67
68             fold - the current fold
69         '''
70         data = self.data
71         # to exclude last term in previous partition for training data
72         train_add = 1 if fold < self.folds else 0
73         # to exclude last term in previous partition for testing data
74         test_add = 1 if fold > 0 else 0
75
76         # number of data sets
77         n = len(self.data)
78
79         train_set_1 = data.iloc[0:int((fold)/self.folds*n), :]
80         train_set_2 = data.iloc[int((fold+1)/self.folds*n)+train_add:n, :]
81         train_set = pd.concat([train_set_1, train_set_2])
82
83         test_set = data.iloc[int((fold)/self.folds*n+test_add):int((fold+1)/
self.folds*n), :]
84
85         train_X = train_set.iloc[:, :-1].values
86         train_y = train_set.iloc[:, -1].values
87         train_y = np.reshape(train_y, (-1,1))
88
89         test_X = test_set.iloc[:, :-1].values
90         test_y = test_set.iloc[:, -1].values
91         test_y = np.reshape(test_y, (-1,1))
92
93         return train_X, train_y, test_X, test_y
94
95     def normalization(self, X, v_X):
96         '''
97             This function performs the z-score normalization
98
99             X - training data
100            v_X - validation data
101        '''
102        mean = np.mean(X[:,1:], axis = 0)
103        sigma = np.std(X[:,1:], axis = 0)
104        mean = np.reshape(mean, (1,-1))
105        sigma = np.reshape(sigma, (1,-1))
106        X[:,1:] = (X[:,1:] - mean) / sigma
107        v_X[:,1:] = (v_X[:,1:] - mean) / sigma
108        return X, v_X
109
110     def fit(self, X, y, v_X, v_y, normalize=False):
111         '''
112             This function takes the training data X and its corresponding
labels vector y
113             as well as other hyperparameters (such as learning rate) as input,
114             and execute the model training through modifying the model
parameters (i.e. W).
115
116             X - training data
117             y - class of training data
118             v_X - validation data
119             v_y - class of validation data

```

```

120         epsilon - the threshold value for gradient descent
121         normalize - whether to perform normalization
122         '''
123         gradient_values, t_acc_val, v_acc_val = [], [], []
124
125         if normalize:
126             X, v_X = self.normalization(X, v_X)
127
128         # Retrive the learning rate, maximum iteration, momentum (beta)
129         lr, max_iter, beta, epsilon = self.lr, self.max_iter, self.beta, self.
epsilon
130
131         # initial weight vector
132         w = np.zeros((len(X[0]), 1))
133         # record the best weight vector
134         best_w = w
135         # iteration number, validation accuracy, last validation accuracy
136         # the step to take in gradient descent, maximum validation accuracy
137         iteration, v_acc, step, v_acc_max = 0, 0, 0, 0
138
139         dw = np.inf
140         # if the gradient delta w is smaller than threshold or achieved
maximum iteration, stop
141         while (np.linalg.norm(dw) > epsilon and iteration <= max_iter):
142             dw = self.gradient(X, y, w)
143             gradient_values.append(np.linalg.norm(dw))
144             # if beta = 0, it will be the same as general gradient descent
145             step = beta * step + (1 - beta) * dw # gradient descent with
momentum
146             w = w - lr * step
147
148             # predict once every 10 iterations
149             if iteration % 10 == 0:
150                 t_y_pred = self.predict(X, w)
151                 t_acc = self.accu_eval(t_y_pred, y)
152                 v_y_pred = self.predict(v_X, w)
153                 v_acc = self.accu_eval(v_y_pred, v_y)
154
155             # record the next best value
156             if v_acc >= v_acc_max:
157                 v_acc_max = v_acc
158                 best_w = w
159                 self.marker = iteration # move the iteration marker
160
161             t_acc_val.append(t_acc)
162             v_acc_val.append(v_acc)
163
164             iteration = iteration + 1
165         return gradient_values, t_acc_val, v_acc_val, best_w
166
167     def predict(self, X, w):
168         '''
169         This function takes a set of data as input and outputs predicted
labels for the input points.
170         '''
171         result = self.log_func(np.dot(X, w))
172         # the prediction result converted to binary
173         predict_bin = []
174         for i in result:
175             if i >= 0.5:
176                 predict_bin.append(1)
177             else:
178                 predict_bin.append(0)

```

```

179         return predict_bin
180
181     def accu_eval(self, y_pred, y):
182         '''
183         This function evaluates the models' accuracy.
184         '''
185         count = 0
186         for i in range(len(y_pred)):
187             if y_pred[i] == y[i]:
188                 count = count + 1
189         # return the accuracy ratio: #corret prediction / #data points
190         return count / len(y)
191
192     def log_func(self, alpha):
193         return 1 / (1 + np.exp(-alpha))
194
195     def gradient(self, X, y, w):
196         N = len(X[0])
197         y_hat = self.log_func(np.dot(X, w))
198         delta = np.dot(X.T, y_hat - y) / N
199         return delta
200
201 class KFoldValidation:
202     def __init__(self, folds, path, lr, max_iter, epsilon, beta):
203         self.folds = folds
204         self.data = pd.read_csv(path)
205         self.lr = lr
206         self.max_iter = max_iter
207         self.epsilon = epsilon
208         self.beta = beta
209         self.log_reg = LogisticRegression(data=self.data, folds=folds, lr=lr,
max_iter=max_iter, beta=beta, epsilon=epsilon)
210         self.log_reg.shuffle_data()
211
212     def set_learning_rate(self, lr):
213         self.log_reg.set_learning_rate(lr)
214
215     def set_max_iter(self, max_iter):
216         self.log_reg.set_max_iter(max_iter)
217
218     def set_epsilon(self, epsilon):
219         self.log_reg.set_epsilon(epsilon)
220
221     def set_beta(self, beta):
222         self.log_reg.set_beta(beta)
223
224     def k_fold_validation(self):
225         '''
226         This function performs the k-fold validation
227
228         normalize - whether to perform normalization
229         inc_od - whether to increase the feature order
230         order - the order of the added feature
231         '''
232         folds = self.folds
233         data = self.data
234         log_reg = self.log_reg
235         accuracies = []
236         tic = time.time()
237
238         for fold in range(folds):
239             t_X, t_y, v_X, v_y = log_reg.partition(fold)
240             # t_X --> test value X, v_X --> validation value X

```

```

241     gradient_val, t_acc_val, v_acc_val, best_w = log_reg.fit(t_X, t_y
    , v_X, v_y)
242
243     accuracies.append(np.mean(v_acc_val))
244
245     ## Uncomment this block to display the accuracy diagram
246     # plt.figure()
247     # plt.plot(t_acc_val, label = 'Training accuracy')
248     # plt.plot(v_acc_val, label='Validation accuracy')
249     # plt.axvline(log_reg.marker, color='r', label='Best Weights')
250     # plt.xlabel('Iteration Number')
251     # plt.ylabel('Accuracy')
252     # plt.legend()
253     # plt.show()
254     # print("Learning Rate: " + str(log_reg.lr))
255     # print("Average Accuracy: "+str(np.mean(accuracies)))
256
257     ## Uncomment this block to display the gradient diagram
258     # plt.figure()
259     # plt.plot(gradient_val)
260     # plt.xlabel('Iteration Number')
261     # plt.ylabel('Gradient')
262     # plt.show()
263     # print("-----")
264
265     mean_acc = np.max(accuracies)
266     toc = time.time()
267     return mean_acc, toc-tic
268
269     def rise_order(self, data, order=3):
270         ret_val = data
271         for i in range(2, order + 1):
272             data_powered = data.pow(i)
273             ret_val = ret_val.iloc[:, :-1]
274             ret_val = pd.concat([ret_val, data_powered],axis=1)
275         return ret_val
276
277     def plot_fig(x, y, xlabel, ylabel, log_x=False, plt_name="Untitled_Figure.png"
    , download=False):
278         plt.plot(x, y)
279         if log_x:
280             plt.xscale("log")
281             plt.xlabel(xlabel)
282             plt.ylabel(ylabel)
283         if download:
284             plt.savefig(plt_name, dpi = 1200)
285             files.download(plt_name)
286         plt.show()
287
288     """### Default Values for Hepatitis Analysis"""
289
290     path = "/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_01/hepatitis.csv"
291     dataset_name = "Hepatitis"
292     default_lr = 0.01
293     default_max_iter = 10000
294     default_epsilon = 5e-3
295     default_beta = 0.99
296
297     """### Learning rate testing"""
298
299     lr_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
    default_max_iter, epsilon=default_epsilon, beta=default_beta)
300

```

```

301 Learning_rates = [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]
302
303 mean_acc = []
304 proc_time = []
305
306 for lr in Learning_rates:
307     lr_testing.set_learning_rate(lr)
308     mean_acc_temp, time_temp = lr_testing.k_fold_validation()
309     mean_acc.append(mean_acc_temp)
310     proc_time.append(time_temp)
311
312 plot_fig(Learning_rates, mean_acc, "log(Learning Rate)", "Mean Accuracy",
313         log_x=True, plt_name="Validation_Accuracy_vs_Learning_Rate_for_{}.png".format(
314             dataset_name))
315 plot_fig(Learning_rates, proc_time, "log(Learning Rate)", "Processing Time",
316         log_x=True, plt_name="Validation_Accuracy_vs_Learning_Rate_for_{}.png".format(
317             dataset_name))
318
319 """### Maximum Iteration Test"""
320
321 max_iter_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
322     default_max_iter, epsilon=1e-3, beta=default_beta)
323
324 max_iters = [500, 1000, 5000, 10000, 25000]
325
326 mean_acc = []
327 proc_time = []
328
329 for max_iter in max_iters:
330     max_iter_testing.set_max_iter(max_iter)
331     mean_acc_temp, time_temp = max_iter_testing.k_fold_validation()
332     mean_acc.append(mean_acc_temp)
333     proc_time.append(time_temp)
334
335 plot_fig(max_iters, mean_acc, "log(Maximum Iterations)", "Mean Accuracy",
336     log_x=True, plt_name="Validation_Accuracy_vs_Maximum_Iterations_for_{}.png".
337     format(dataset_name))
338 plot_fig(max_iters, proc_time, "log(Maximum Iterations)", "Processing Time",
339     log_x=True, plt_name="Processing_Time_vs_Maximum_Iterations_for_{}.png".format(
340         dataset_name))
341
342 """### Epsilon Test"""
343
344 epsilon_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
345     default_max_iter, epsilon=default_epsilon, beta=default_beta)
346
347 epsilons = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 0.5]
348
349 mean_acc = []
350 proc_time = []
351
352 for epsilon in epsilons:
353     epsilon_testing.set_epsilon(epsilon)
354     mean_acc_temp, time_temp = epsilon_testing.k_fold_validation()
355     mean_acc.append(mean_acc_temp)
356     proc_time.append(time_temp)
357
358 plot_fig(epsilons, mean_acc, "log(Epsilons)", "Mean Accuracy", log_x=True,
359     plt_name="Validation_Accuracy_vs_Epsilons_for_{}.png".format(dataset_name))
360 plot_fig(epsilons, proc_time, "log(Epsilons)", "Processing Time", log_x=True,
361     plt_name="Processing_Time_vs_Epsilons_for_{}.png".format(dataset_name))
362
363 """### Momentum Gradient Descent Constant - Beta Testing"""

```

```

352
353 beta_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
    default_max_iter, epsilon=default_epsilon, beta=defulat_beta)
354
355 betas = [0, 0.5, 0.9, 0.99, 0.999]
356
357 mean_acc = []
358 proc_time = []
359
360 for beta in betas:
361     beta_testing.set_beta(beta)
362     mean_acc_temp, time_temp = beta_testing.k_fold_validation()
363     mean_acc.append(mean_acc_temp)
364     proc_time.append(time_temp)
365
366 plot_fig(betas, mean_acc, "betas", "Mean Accuracy", log_x=False, plt_name="
    Validation_Accuracy_vs_betas_for_{}.png".format(dataset_name))
367 plot_fig(betas, proc_time, "betas", "Processing Time", log_x=False, plt_name="
    Processing_Time_vs_betas_for_{}.png".format(dataset_name))
368
369 """<hr>
370
371 ### Default Values for Bankruptcy Analysis
372 """
373
374 path = "/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_01/bankrupcy.csv"
375 dataset_name = "Bankruptcy"
376 default_lr = 0.1
377 default_max_iter = 25000
378 default_epsilon = 1e-3
379 defulat_beta = 0.99
380
381 """### Learning rate testing"""
382
383 lr_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
    default_max_iter, epsilon=default_epsilon, beta=defulat_beta)
384
385 Learning_rates = [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]
386
387 mean_acc = []
388 proc_time = []
389
390 for lr in Learning_rates:
391     lr_testing.set_learning_rate(lr)
392     mean_acc_temp, time_temp = lr_testing.k_fold_validation()
393     mean_acc.append(mean_acc_temp)
394     proc_time.append(time_temp)
395
396 plot_fig(Learning_rates, mean_acc, "log(Learning Rate)", "Mean Accuracy",
    log_x=True, plt_name="Validation_Accuracy_vs_Learning_Rate_for_{}.png".format(
    dataset_name))
397 plot_fig(Learning_rates, proc_time, "log(Learning Rate)", "Processing Time",
    log_x=True, plt_name="Validation_Accuracy_vs_Learning_Rate_for_{}.png".format(
    dataset_name))
398
399 """### Maximum Iteration Test"""
400
401 max_iter_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter
    =default_max_iter, epsilon=1e-3, beta=defulat_beta)
402
403 max_iters = [500, 1000, 5000, 10000, 25000]
404
405 mean_acc = []

```

```

406 proc_time = []
407
408 for max_iter in max_iters:
409     max_iter_testing.set_max_iter(max_iter)
410     mean_acc_temp, time_temp = max_iter_testing.k_fold_validation()
411     mean_acc.append(mean_acc_temp)
412     proc_time.append(time_temp)
413
414 plot_fig(max_iters, mean_acc, "log(Maximum Iterations)", "Mean Accuracy",
log_x=True, plt_name="Validation_Accuracy_vs_Maximum_Iterations_for_{}.png".
format(dataset_name))
415 plot_fig(max_iters, proc_time, "log(Maximum Iterations)", "Processing Time",
log_x=True, plt_name="Processing_Time_vs_Maximum_Iterations_for_{}.png".format
(dataset_name))
416
417 """### Epsilon Test"""
418
419 epsilon_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
default_max_iter, epsilon=default_epsilon, beta=default_beta)
420
421 epsilons = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 0.5]
422
423 mean_acc = []
424 proc_time = []
425
426 for epsilon in epsilons:
427     epsilon_testing.set_epsilon(epsilon)
428     mean_acc_temp, time_temp = epsilon_testing.k_fold_validation()
429     mean_acc.append(mean_acc_temp)
430     proc_time.append(time_temp)
431
432 plot_fig(epsilons, mean_acc, "log(Epsilons)", "Mean Accuracy", log_x=True,
plt_name="Validation_Accuracy_vs_Epsilons_for_{}.png".format(dataset_name))
433 plot_fig(epsilons, proc_time, "log(Epsilons)", "Processing Time", log_x=True,
plt_name="Processing_Time_vs_Epsilons_for_{}.png".format(dataset_name))
434
435 """### Momentum Gradient Descent Constant - Beta Testing"""
436
437 beta_testing = KFoldValidation(folds=10, path=path, lr=default_lr, max_iter=
default_max_iter, epsilon=default_epsilon, beta=default_beta)
438
439 betas = [0, 0.5, 0.9, 0.99, 0.999]
440
441 mean_acc = []
442 proc_time = []
443
444 for beta in betas:
445     beta_testing.set_beta(beta)
446     mean_acc_temp, time_temp = beta_testing.k_fold_validation()
447     mean_acc.append(mean_acc_temp)
448     proc_time.append(time_temp)
449
450 plot_fig(betas, mean_acc, "betas", "Mean Accuracy", log_x=False, plt_name="
Validation_Accuracy_vs_betas_for_{}.png".format(dataset_name))
451 plot_fig(betas, proc_time, "betas", "Processing Time", log_x=False, plt_name="
Processing_Time_vs_betas_for_{}.png".format(dataset_name))

```