

Source Codes

codes/Regression.py

```

1  import numpy as np
2  import random
3
4  random.seed(10)
5  np.random.seed(10)
6
7  class Regression(object):
8      def __init__(self, m=1, reg_param=0):
9          """
10         Inputs:
11             - m Polynomial degree
12             - regularization parameter reg_param
13         Goal:
14             - Initialize the weight vector self.theta
15             - Initialize the polynomial degree self.m
16             - Initialize the regularization parameter self.reg
17         """
18         self.m = m
19         self.reg = reg_param
20         self.dim = [m+1, 1]
21         ### These two lines set the random seeds... you can ignore. #####
22         random.seed(10)
23         np.random.seed(10)
24         #####
25         self.theta = np.random.standard_normal(self.dim)
26     def get_poly_features(self, X):
27         """
28         Inputs:
29             - X: A numpy array of shape (n,1) containing the data.
30         Returns:
31             - X_out: an augmented training data as an mth degree feature vector
32               e.g. [1, x, x^2, ..., x^m], x \in X.
33         """
34         n,d = X.shape
35         m = self.m
36         X_out= np.zeros((n,m+1))
37         if m==1:
38             # ===== #
39             # YOUR CODE HERE:
40             # IMPLEMENT THE MATRIX X_out with each entry = [1, x]
41             # ===== #
42             for i in range(0, n):
43                 X_out[i, :] = np.array([1, X[i, 0]])
44             pass
45             # ===== #
46             # END YOUR CODE HERE
47             # ===== #
48         else:
49             # ===== #
50             # YOUR CODE HERE:
51             # IMPLEMENT THE MATRIX X_out with each entry = [1, x, x^2, ..., x^m]

```

```

52         # ===== #
53         for i in range(0, n):
54             for j in range(0, m+1):
55                 X_out[i, j] = pow(X[i, 0], j)
56         pass
57         # ===== #
58         # END YOUR CODE HERE
59         # ===== #
60         pass
61     return X_out
62
63     def loss_and_grad(self, X, y):
64         """
65         Inputs:
66         - X: n x d array of training data.
67         - y: n x 1 targets
68         Returns:
69         - loss: a real number represents the loss
70         - grad: a vector of the same dimensions as self.theta containing the
71           gradient of the loss with respect to self.theta
72         """
73         loss = 0.0
74         grad = np.zeros_like(self.theta)
75         m = self.m
76         n, d = X.shape
77         if m==1:
78             # =====
79             # YOUR CODE HERE:
80             # Calculate the loss function of the linear regression
81             # and save loss function in loss.
82             # Calculate the gradient and save it as grad.
83             # =====
84
85             errorMatrix = self.predict(X) - y
86             loss = 1 / (2 * n) * np.dot(errorMatrix.T, errorMatrix) +
np.ndarray.item(self.reg / 2 * (np.dot(self.theta.T, self.theta) -
pow(self.theta[0], 2)))
87             grad = (1 / n * np.dot(self.get_poly_features(X).T, errorMatrix)) +
(self.reg * self.theta).reshape(2, )
88
89             # =====
90             # END YOUR CODE HERE
91             # =====
92         else:
93             # ===== # YOUR CODE
94             HERE:
95             # Calculate the loss and gradient of the polynomial regre # with order
96             m
97             #
98             # =====
99
100             errorMatrix = self.predict(X) - y
101             loss = 1 / (2 * n) * np.dot(errorMatrix.T, errorMatrix) +
np.ndarray.item(self.reg / 2 * (np.dot(self.theta.T, self.theta)-

```

```

100     pow(self.theta[0], 2)))
101     grad = (1 / n * np.dot(self.get_poly_features(X).T, errorMatrix)) +
102     (self.reg * self.theta).reshape(m + 1, )
103     pass
104     # =====
105     # END YOUR CODE HERE
106     # =====
107     return loss, grad
108
109 def train_LR(self, X, y, alpha=1e-2, B=30, num_iters=10000) :
110     """
111     Finds the coefficients of a {d-1}^th degree polynomial
112     that fits the data using least squares mini-batch gradient descent.
113
114     Inputs:
115     - X          -- numpy array of shape (n,d), features
116     - y          -- numpy array of shape (n,), targets
117     - alpha      -- float, learning rate
118     - B          -- integer, batch size
119     - num_iters  -- integer, maximum number of iterations
120
121     Returns:
122     - loss_history: vector containing the loss at each training iteration.
123     - self.theta: optimal weights
124     """
125     ### These two lines set the random seeds... you can ignore. #####
126     random.seed(10)
127     np.random.seed(10)
128     #####
129     self.theta = np.random.standard_normal(self.dim)
130     loss_history = []
131     n,d = X.shape
132     for t in np.arange(num_iters):
133         X_batch = None
134         y_batch = None
135         # ===== #
136         # YOUR CODE HERE:
137         # Shuffle X, y along the batch axis with np.random.shuffle.
138         # Get the first batch_size elements X_batch from X, y_batch from Y.
139         # X_batch should have shape: (B,1), y_batch should have shape: (B,).
140         # ===== #
141         indices = np.arange(n)
142         np.random.shuffle(indices)
143         X = X[indices]
144         y = y[indices]
145         X_batch = X[0:B]
146         y_batch = y[0:B]
147         pass
148         # ===== #
149         # END YOUR CODE HERE
150         # ===== #
151         loss = 0.0
152         grad = np.zeros_like(self.theta)

```

```

152         # ===== #
153         # YOUR CODE HERE:
154         # evaluate loss and gradient for batch data
155         # save loss as loss and gradient as grad
156         # update the weights self.theta
157         # ===== #
158         loss, grad = self.loss_and_grad(X_batch, y_batch)
159         self.theta -= alpha * grad.reshape(-1, 1)
160         pass
161         # ===== #
162         # END YOUR CODE HERE
163         # ===== #
164         loss_history.append(loss)
165     return loss_history, self.theta
166
167
168
169     def closed_form(self, X, y):
170         """
171         Inputs:
172         - X: n x 1 array of training data.
173         - y: n x 1 array of targets
174         Returns:
175         - self.theta: optimal weights
176         """
177         m = self.m
178         n,d = X.shape
179         loss = 0
180         if m==1:
181             # ===== #
182             # YOUR CODE HERE:
183             # obtain the optimal weights from the closed form solution
184             # ===== #
185             polyX = self.get_poly_features(X)
186             self.theta = (np.linalg.inv(polyX.T.dot(polyX))).dot(polyX.T).dot(y)
187             self.theta = self.theta.reshape(-1, 1)
188             loss, grad = self.loss_and_grad(X, y)
189             pass
190             # ===== #
191             # END YOUR CODE HERE
192             # ===== #
193         else:
194             # ===== #
195             # YOUR CODE HERE:
196             # Extend X with get_poly_features().
197             # Predict the targets of X.
198             # ===== #
199             polyX = self.get_poly_features(X)
200             self.theta = (np.linalg.inv(polyX.T.dot(polyX))).dot(polyX.T).dot(y)
201             self.theta = self.theta.reshape(-1, 1)
202             loss, grad = self.loss_and_grad(X, y)
203             pass
204             # ===== #
205             # END YOUR CODE HERE

```

```
206         # ===== #
207     return loss, self.theta
208
209
210 def predict(self, X):
211     """
212     Inputs:
213     - X: n x 1 array of training data.
214     Returns:
215     - y_pred: Predicted targets for the data in X. y_pred is a 1-dimensional
216       array of length n.
217     """
218     y_pred = np.zeros(X.shape[0])
219     m = self.m
220     theta = self.theta
221     if m==1:
222         # ===== #
223         # YOUR CODE HERE:
224         # PREDICT THE TARGETS OF X
225         # ===== #
226         for i in range(0, X.shape[0]):
227             y_pred[i] = theta[0] + theta[1] * X[i]
228         pass
229         # ===== #
230         # END YOUR CODE HERE
231         # ===== #
232     else:
233         # ===== #
234         # YOUR CODE HERE:
235         # Extend X with get_poly_features().
236         # Predict the target of X.
237         # ===== #
238         polyX = self.get_poly_features(X)
239         for i in range(0, polyX.shape[0]):
240             for j in range(0, polyX.shape[1]):
241                 y_pred[i] += theta[j] * polyX[i, j]
242         pass
243         # ===== #
244         # END YOUR CODE HERE
245         # ===== #
246     return y_pred
```