

# Machine Learning: Mini Project II

By Colin Vande Vijvere

## Kernel Perceptron

### Description:

Kernelized perceptron (polynomial kernel) using ECOC to classify the mnist dataset.

### Notes:

With this algorithm, I've noticed that the # of iterations is quite important. Unfortunately, those iterations ask a lot of computing power, which I do not have available at the time. Below you can see How my algorithm increased after raising the #of iterations.

### Training and testing accuracies + confusion matrix:

#### with # of iterations = 10:

```

Training Accuracy = 71.0%
2522 test case predicted.
1651 are correct.
Testing Accuracy = 65.0%
[[194  0  4  0  0  0  0  0 39  0]
 [  0 89  6  3  1  0  0  0 10 177]
 [  1  0 224 14  0  0  6  1  2  1]
 [  1  0  25 211  0  1  1 12  1  2]
 [  7  0 15  1 129  6 22  3 62 18]
 [  8  2 14 14 32 68  4 22 46 26]
 [  4  0 22  0  1  0 208  0  1  0]
 [  0  0  7 65  1  6  9 179  4 14]
 [  0  0 43  8  0  0  1  0 168 10]
 [  0  0  4 20  0  6  0  4  31 181]]

```

#### with # of iterations = 30:

```

Training Accuracy = 83.0%
6000 test case predicted.
4732 are correct.
Testing Accuracy = 79.0%
[[594  0  5  2  1  0  2  0  5  0]
 [  4 615  1 24  0  2  1  1  1 22]
 [  9  0 499 53  0  1 12  2  5  0]
 [  1  6 14 538  0  2  2 20  2 19]
 [ 19 11 14  5 332 111 32 12 40 43]
 [ 22 67  9 47 12 283  6 45  6 27]
 [  8  0 17  3  4  2 563  4  2  0]
 [  7  7 11 72  6 11  9 492  2 13]
 [  9 40  5 23  1 26  0  8 13 469]]

```

Kernel Perceptron Code:

```

1  import numpy as np
2  from numpy import linalg
3  import matplotlib
4  import matplotlib.pyplot as plt
5  from libsvm.svmutil import svm_read_problem
6  from sklearn.model_selection import train_test_split
7  from sklearn.metrics import confusion_matrix, classification_report
8
9
10 def decToBin(array):
11     result = {
12         0: [0, 0, 0, 0],
13         1: [0, 0, 0, 1],
14         2: [0, 0, 1, 0],
15         3: [0, 0, 1, 1],
16         4: [0, 1, 0, 0],
17         5: [0, 1, 0, 1],
18         6: [0, 1, 1, 0],
19         7: [0, 1, 1, 1],
20         8: [1, 0, 0, 0],
21         9: [1, 0, 0, 1]
22     }
23     return np.array([result[number] for number in array])
24
25
26 def binToDec(array):
27     result = np.array([num[0]*8 + num[1] * 4 + num[2]
28         * 2 + num[3] for num in array])
29     if result > 9:
30         return np.array([num[0]*0 + num[1] * 4 + num[2]
31             * 2 + num[3] for num in array])
32     else:
33         return result
34
35
36 def kernel(a, x, z, p, lr):
37     result = np.sum(a*(1+(lr*(np.dot(x, z)))**p))
38     return result

```

```

41 class perceptron:
42     def __init__(self, learning_rate=0.001, n_iters=30):
43         self.lr = learning_rate
44         self.n_iters = n_iters
45         self.weights = None
46         self.bias = None
47         self.alpha = None
48
49     def fit(self, X, y):
50         self.alpha = np.zeros(len(X))
51         pred = []
52         for i in range(self.n_iters):
53             for j in range(len(X)):
54                 val = kernel(self.alpha, X, X[j], p=3, lr=0.001)
55                 result = np.sign(val)
56                 pred.append(result)
57                 if result != y[j]:
58                     self.alpha[j] = self.alpha[j] + self.lr*y[j]
59
60     def predict(self, X, Xi):
61         val = kernel(self.alpha, X, Xi, p=5, lr=0.005)
62         result = np.sign(val)
63         if result <= 0:
64             return 0
65         else:
66             return 1

```

```

69 # reading in the data
70 y_raw, x_raw = svm_read_problem('mnist.scale')
71
72 y = np.array(y_raw)
73 x = np.zeros((len(y_raw), 780))
74 for i in range(len(y_raw)):
75     line = x_raw[i]
76     for k, v in line.items():
77         x[i][k - 1] = v
78
79 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5)
80
81 y_trainAcc = y_train
82 y_train = decToBin(y_train)
83 y_train = np.where(y_train == 0, -1, 1)
84
85 #splitting digits of binary number
86 y_train1 = np.copy(y_train[:, 0])
87 y_train2 = np.copy(y_train[:, 1])
88 y_train3 = np.copy(y_train[:, 2])
89 y_train4 = np.copy(y_train[:, 3])
90

```



```

97  perceptron1 = perceptron()
98  perceptron1.fit(x_train, y_train1)
99  perceptron2 = perceptron()
100 perceptron2.fit(x_train, y_train2)
101 perceptron3 = perceptron(learning_rate=0.1)
102 perceptron3.fit(x_train, y_train3)
103 perceptron4 = perceptron(learning_rate=0.05)
104 perceptron4.fit(x_train, y_train4)
105
106 # Calculating training accuracy
107 predicted = np.zeros(len(y_trainAcc))
108 for i in range(len(y_trainAcc)):
109     pred = np.zeros(4)
110     pred[0] = perceptron1.predict(x_train, x_train[i])
111     pred[1] = perceptron2.predict(x_train, x_train[i])
112     pred[2] = perceptron3.predict(x_train, x_train[i])
113     pred[3] = perceptron4.predict(x_train, x_train[i])
114     predicted[i] = binToDec([pred])
115 correct_num = np.sum(predicted == y_trainAcc)
116 print('Training Accuracy = ', np.round(
117     correct_num * 100 / len(predicted)), '%', sep='')
118
119
120 # Calculating testing accuracy
121 predicted = np.zeros(len(y_test))
122 for i in range(len(y_test)):
123     pred = np.zeros(4)
124     pred[0] = perceptron1.predict(x_train, x_test[i])
125     pred[1] = perceptron2.predict(x_train, x_test[i])
126     pred[2] = perceptron3.predict(x_train, x_test[i])
127     pred[3] = perceptron4.predict(x_train, x_test[i])
128     predicted[i] = binToDec([pred])
129 correct_num = np.sum(predicted == y_test)
130 print('Testing Accuracy = ', np.round(
131     correct_num * 100 / len(predicted)), '%', sep='')
132
133 # printing confusion matrix:
134 print(confusion_matrix(y_test, predicted))

```

## Pegasos SVM

### Description:

Pegasos SVM written with ECOC to classify the mnist dataset.

### Training and testing accuracies + confusion matrix:

```

Training Accuracy = 74.0%
Testing Accuracy = 69.0%
[[47  4  3  0  3  1  0  1  1  0]
 [ 2 65  0  1  0  2  0  1  1  0]
 [14  1 46  4  1  0  5  4  0  0]
 [ 1 12  2 40  0  1  2  3  2  1]
 [ 3  1  0  0 53 12  2  0  1  1]
 [ 2 13  0  2  6 25  1  2  2  0]
 [ 7  0  4  0  4  0 48  1  0  0]
 [ 0  5  0  6  2 20  0 44  0  0]
 [ 5  1  1  2  3  0  0  0 43  4]
 [ 2  4  0  0  2  9  0  2  2 43]]

```

Note: this is the highest achieved accuracy. Average accuracy of 10 runs was 66.53%.

### Code for pegasos\_SVM:

```

1  import numpy as np
2  from numpy import linalg
3  import math
4  from random import randint
5  import matplotlib
6  import matplotlib.pyplot as plt
7  from libsvm.svmutil import svm_read_problem
8  from sklearn.model_selection import train_test_split
9  from sklearn.metrics import confusion_matrix
10
11
12  # function to change decimal to binary array
13  def decToBin(array):
14      result = []
15      0: [0, 0, 0, 0],
16      1: [0, 0, 0, 1],
17      2: [0, 0, 1, 0],
18      3: [0, 0, 1, 1],
19      4: [0, 1, 0, 0],
20      5: [0, 1, 0, 1],
21      6: [0, 1, 1, 0],
22      7: [0, 1, 1, 1],
23      8: [1, 0, 0, 0],
24      9: [1, 0, 0, 1]
25  ]
26  return np.array([result[number] for number in array])
27

```

```

29 # function to change binary array to decimal
30 def binToDec(array):
31     result = np.array([num[0]*8 + num[1] * 4 + num[2]
32                        * 2 + num[3] for num in array])
33     if result > 9:
34         return np.array([num[0]*0 + num[1] * 4 + num[2]
35                        * 2 + num[3] for num in array])
36     else:
37         return result
38
39
40 # pegasos part of the algorithm
41 def pegasos(x, y, weights=None, iterations=2000, lam=0.001):
42     if type(weights) == type(None):
43         weights = np.zeros(x[0].shape)
44     num_S = len(y)
45     for i in range(iterations):
46         it = randint(0, num_S-1)
47         step = 1/(lam*(i+1))
48         decision = y[it] * weights @ x[it].T
49         if decision < 1:
50             weights = (1 - step*lam) * weights + step*y[it]*x[it]
51         else:
52             weights = (1 - step*lam) * weights
53     return weights

```

```

56 class SVM:
57     def __init__(self, learning_rate=0.0005, lambda_param=0.01, n_iters=100):
58         self.lr = learning_rate
59         self.lam = lambda_param
60         self.n_iters = n_iters
61         self.weights = None
62         self.b = None
63
64     def fit(self, X, y):
65         self.weights = pegasos(x=X, y=y, iterations=self.n_iters)
66
67     def predict(self, X):
68         approx = np.dot(X, self.weights)
69         sign_approx = np.sign(approx)
70         if sign_approx <= 0:
71             return 0
72         else:
73             return 1

```



```

76 # reading in the data
77 y_raw, x_raw = svm_read_problem(['mnist|scale'])
78
79 y = np.array(y_raw)
80 x = np.zeros((len(y_raw), 780))
81 for i in range(len(y_raw)):
82     line = x_raw[i]
83     for k, v in line.items():
84         x[i][k - 1] = v
85
86 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
87
88 # converting label to binary and splitting them up
89 y_trainAcc = y_train
90 y_trainzeroOne = decToBin(y_train)
91 y_train = np.where(y_trainzeroOne == 0, 1, -1)
92
93 y_train1 = np.copy(y_train[:, 0])
94 y_train2 = np.copy(y_train[:, 1])
95 y_train3 = np.copy(y_train[:, 2])
96 y_train4 = np.copy(y_train[:, 3])
97
98 #training 4 algorithms for each binary digit
99 svm1 = SVM()
100 svm1.fit(x_train, y_train1)
101 svm2 = SVM()
102 svm2.fit(x_train, y_train2)
103 svm3 = SVM()
104 svm3.fit(x_train, y_train3)
105 svm4 = SVM()
106 svm4.fit(x_train, y_train4)
107
108
109 # training accuracy
110 predicted = np.zeros(len(y_trainAcc))
111 for i in range(len(y_trainAcc)):
112     pred = np.zeros(4)
113     pred[0] = svm1.predict(x_train[i])
114     pred[1] = svm2.predict(x_train[i])
115     pred[2] = svm3.predict(x_train[i])
116     pred[3] = svm4.predict(x_train[i])
117     predicted[i] = binToDec([pred])
118     # print("binary answer: ", pred, " | decimal prediction: ",
119     #       predicted[i], " | correct prediction: ", y_train[i])
120 print(len(predicted), ' test case predicted.', sep='')
121 correct_num = np.sum(predicted == y_trainAcc)

```



```
128 # Testing accuracy
129 predicted = np.zeros(len(y_test))
130 for i in range(len(y_test)):
131     pred = np.zeros(4)
132     pred[0] = svm1.predict(x_test[i])
133     pred[1] = svm2.predict(x_test[i])
134     pred[2] = svm3.predict(x_test[i])
135     pred[3] = svm4.predict(x_test[i])
136     predicted[i] = binToDec([pred])
137     print("binary answer: ", pred, " | decimal prediction: ",
138         predicted[i], " | correct prediction: ", y_test[i])
139 print(len(predicted), ' test case predicted.', sep='')
140 correct_num = np.sum(predicted == y_test)
141 print(correct_num, ' are correct.', sep='')
142 print('Accuracy = ', np.round(correct_num * 100 / len(predicted)), '%', sep='')
```

## Neural Network:

### Description:

Neural Net to classify the mnist dataset.

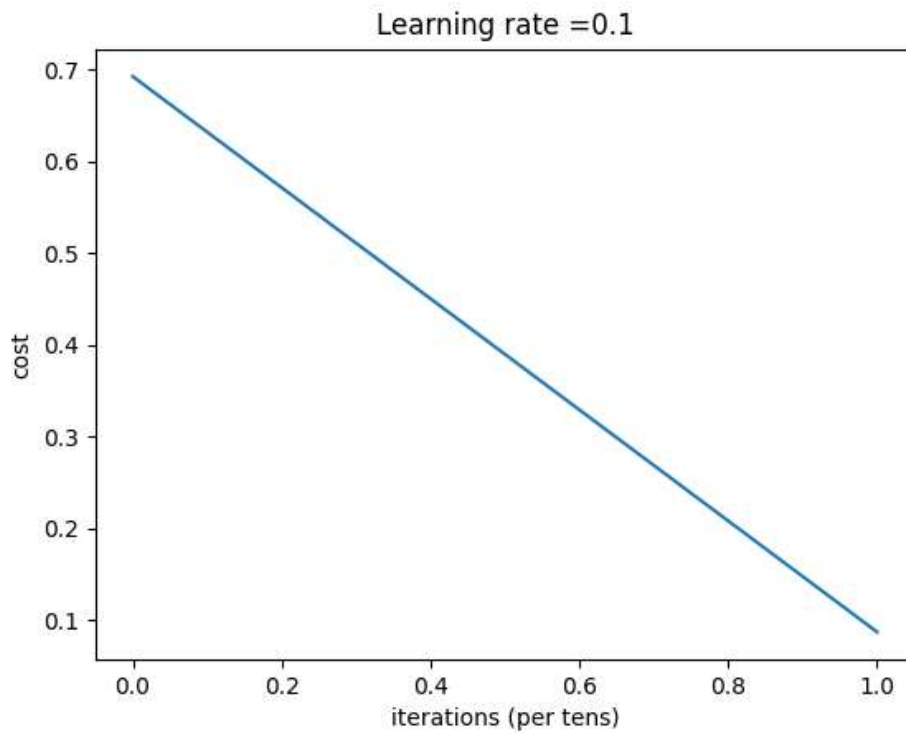
### Training and testing accuracies + confusion matrix:

```

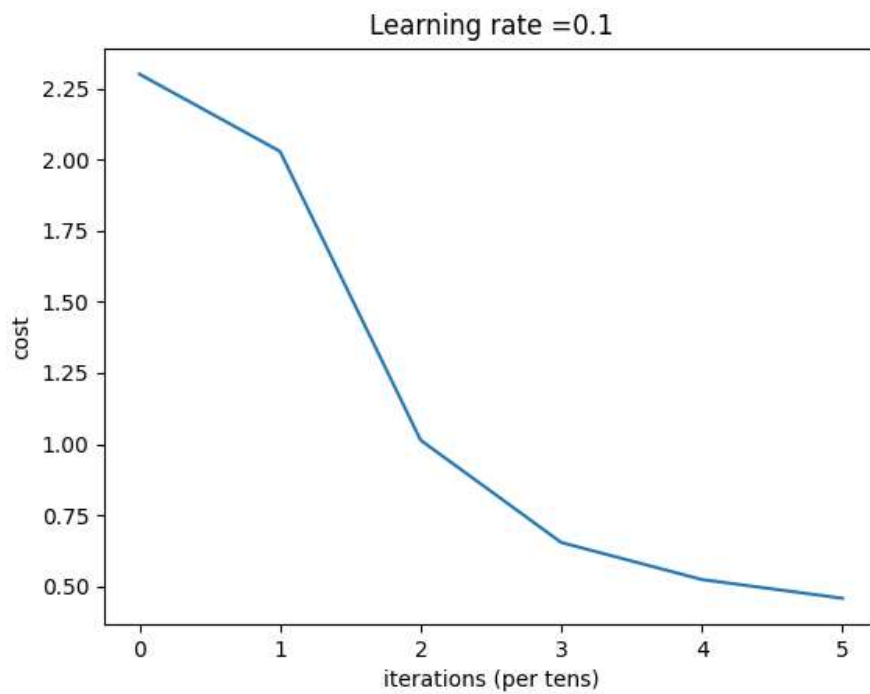
===== Binary (Digit 1 and 2) =====
Cost after iteration 0: 0.6926057785748851
Cost after iteration 50: 0.08490423769633489
Training Accuracy:
8890 test case predicted.
8890 are correct.
Accuracy = 1.0
Testing Accuracy:
3810 test case predicted.
3810 are correct.
Accuracy = 1.0
===== Multi-class =====
Cost after iteration 0: 2.302454060157147
Cost after iteration 50: 2.0319194764807524
Cost after iteration 100: 1.0125986456112632
Cost after iteration 150: 0.6517215378108929
Cost after iteration 200: 0.5212031076333735
Cost after iteration 250: 0.45508607556340214
Training Accuracy:
42000 Test case predicted.
41383 are correct.
Accuracy = 0.98530952381
Testing Accuracy:
18000 test case predicted.
15961 are correct.
Accuracy = 0.8867222222222222
[[1690  0  26  9  4  39  15  13  7  9]
 [  0 1881  41  14  6  15  7  34  36  13]
 [  9  9 1553  44  11  14  38  34  29  13]
 [ 10  9  43 1549  0  69  1  5  68  30]
 [  5  2  36  2 1563  16  24  16  8  99]
 [ 31  8  3  91  3 1358  28  2  63  8]
 [ 13  4  54  4  26  37 1674  0  18  2]
 [  2  6  21  26  1  13  0 1676  3  80]
 [ 12 29  54  50  12  64  11  7 1445  22]
 [  1  1  7  22 114  27  0  75  52 1572]]

```

**Cost/Iteration for the Binary:**



**Cost/Iteration for the multiclass:**



Code for the Neural Net:

```

1  import numpy as np
2  from numpy import linalg
3  import matplotlib
4  import matplotlib.pyplot as plt
5  from libsvm.svmutil import svm_read_problem
6  from sklearn.model_selection import train_test_split
7  from sklearn.metrics import confusion_matrix
8
9  def oneHot(array):
10     result = np.zeros((len(array),10))
11     for number in range(len(array)):
12         result[number][int(array[number])] = 1
13     return result
14
15  def softmax(Z):
16     Z_shift = Z - np.max(Z, axis=0)
17     A = np.exp(Z_shift) / np.sum(np.exp(Z_shift), axis=0)
18
19     cache = Z_shift
20
21     return A, cache
22
23  def relu(Z):
24     A = np.maximum(0, Z)
25     assert (A.shape == Z.shape)
26     cache = Z
27     return A, cache
28
29  def initialize_parameters(n_x, n_h, n_y):
30     np.random.seed(1)
31
32     W1 = np.random.randn(n_h, n_x) * 0.01
33     b1 = np.zeros((n_h, 1))
34     W2 = np.random.randn(n_y, n_h) * 0.01
35     b2 = np.zeros((n_y, 1))
36
37     assert (W1.shape == (n_h, n_x))
38     assert (b1.shape == (n_h, 1))
39     assert (W2.shape == (n_y, n_h))
40     assert (b2.shape == (n_y, 1))
41
42     parameters = [{"W1": W1,
43                   "b1": b1,
44                   "W2": W2,
45                   "b2": b2}]
46     return parameters

```



```

48 def linear_forward(A, W, b):
49     Z = np.dot(W, A) + b
50
51     assert (Z.shape == (W.shape[0], A.shape[1]))
52     cache = (A, W, b)
53
54     return Z, cache
55
56 def linear_activation_forward(A_prev, W, b, activation):
57     if activation == "softmax":
58         # Inputs: "A_prev, W, b". Outputs: "A, activation_cache".
59         Z, linear_cache = linear_forward(A_prev, W, b)
60         A, activation_cache = softmax(Z)
61
62     elif activation == "relu":
63         # Inputs: "A_prev, W, b". Outputs: "A, activation_cache".
64         Z, linear_cache = linear_forward(A_prev, W, b)
65         A, activation_cache = relu(Z)
66
67     assert (A.shape == (W.shape[0], A_prev.shape[1]))
68     cache = (linear_cache, activation_cache)
69
70     return A, cache
71
72 def compute_cost(AL, Y):
73     m = Y.shape[1]
74     cost = -(np.sum(Y * np.log(AL))) / float(m)
75     # cost = np.squeeze(cost)
76     assert (cost.shape == ())
77
78     return cost
79
80 def linear_backward(dZ, cache):
81     A_prev, W, b = cache
82     m = A_prev.shape[1]
83
84     dW = np.dot(dZ, A_prev.T) / float(m)
85     db = np.sum(dZ, axis=1, keepdims=True) / float(m)
86     dA_prev = np.dot(W.T, dZ)
87
88     assert (dA_prev.shape == A_prev.shape)
89     assert (dW.shape == W.shape)
90     assert (db.shape == b.shape)
91
92     return dA_prev, dW, db

```

```

94  ✓ def relu_backward(dA, cache):
95      Z = cache
96      dZ = np.array(dA, copy=True)
97      dZ[Z <= 0] = 0
98
99      assert (dZ.shape == Z.shape)
100
101      return dZ
102
103  ✓ def softmax_backward(Y, cache):
104      Z = cache
105
106      s = np.exp(Z) / np.sum(np.exp(Z), axis=0)
107      dZ = s - Y
108
109      assert (dZ.shape == Z.shape)
110
111      return dZ
112
113  ✓ def linear_activation_backward(dA, cache, activation):
114      linear_cache, activation_cache = cache
115
116  ✓   if activation == "relu":
117       dZ = relu_backward(dA, activation_cache)
118       dA_prev, dW, db = linear_backward(dZ, linear_cache)
119
120  ✓   elif activation == "softmax":
121       dZ = softmax_backward(dA, activation_cache)
122       dA_prev, dW, db = linear_backward(dZ, linear_cache)
123
124      return dA_prev, dW, db

```

```

127  def update_parameters(parameters, grads, learning_rate):
128      L = len(parameters) // 2 # number of layers in the neural network
129
130      # Update rule for each parameter. Use a for loop.
131      for l in range(1, L + 1):
132          parameters['W' + str(l)] -= learning_rate * grads['dW' + str(l)]
133          parameters['b' + str(l)] -= learning_rate * grads['db' + str(l)]
134
135      return parameters

```

```

138 def two_layer_model(X, Y, layers_dims, learning_rate=0.1, num_iterations=3000, print_cost=False):
139
140     np.random.seed(1)
141     grads = {}
142     costs = [] # to keep track of the cost
143     m = X.shape[1] # number of examples
144     (n_x, n_h, n_y) = layers_dims
145
146     # Initialize parameters dictionary, by calling one of the functions you'd previously implemented
147     parameters = initialize_parameters(n_x, n_h, n_y)
148
149     # Get W1, b1, W2 and b2 from the dictionary parameters.
150     W1 = parameters["W1"]
151     b1 = parameters["b1"]
152     W2 = parameters["W2"]
153     b2 = parameters["b2"]
154
155     # Loop (gradient descent)
156
157     for i in range(0, num_iterations):
158
159         # Forward propagation
160         A1, cache1 = linear_activation_forward(X, W1, b1, activation='relu')
161         A2, cache2 = linear_activation_forward(A1, W2, b2, activation='softmax')
162
163         # Compute cost
164         cost = compute_cost(A2, Y)
165
166         # Backward propagation
167         dA1, dW2, db2 = linear_activation_backward(Y, cache2, activation='softmax')
168         dA0, dW1, db1 = linear_activation_backward(dA1, cache1, activation='relu')
169
170         grads['dW1'] = dW1
171         grads['db1'] = db1
172         grads['dW2'] = dW2
173         grads['db2'] = db2
174
175         # Update parameters.
176         parameters = update_parameters(parameters, grads, learning_rate)
177
178         W1 = parameters["W1"]
179         b1 = parameters["b1"]
180         W2 = parameters["W2"]
181         b2 = parameters["b2"]
182
183     # Print the cost every 100 training example
184     if i % 50 == 0:
185         print("Cost after iteration {}: {}".format(i, np.squeeze(cost)))
186         costs.append(cost)
187
188
189
190     plt.plot(np.squeeze(costs))
191     plt.ylabel('cost')
192     plt.xlabel('iterations (per tens)')
193     plt.title("Learning rate =" + str(learning_rate))
194     plt.show()
195
196     return parameters

```



```

198 def predict(X, y, parameters):
199     m = X.shape[1]
200
201     W1 = parameters["W1"]
202     b1 = parameters["b1"]
203     W2 = parameters["W2"]
204     b2 = parameters["b2"]
205
206     A1, _ = linear_activation_forward(X, W1, b1, activation='relu')
207     probs, _ = linear_activation_forward(A1, W2, b2, activation='softmax')
208
209     predicted = np.argmax(probs, axis=0)
210
211     # print ("predictions: " + str(p))
212     # print ("true labels: " + str(y))
213     print(m, ' test case predicted.', sep='')
214     correct_num = np.sum(predicted == y)
215     print(correct_num, ' are correct.', sep='')
216     #print('Accuracy = ', np.round(correct_num * 100 / len(predicted)), '%', sep='')
217     print("Accuracy = " + str(correct_num / float(m)), sep='')
218
219     return predicted
220
221 y_raw, x_raw = svm_read_problem('mnist.scale')
222
223 y = np.array(y_raw)
224 x = np.zeros((len(y_raw), 780))
225 for i in range(len(y_raw)):
226     line = x_raw[i]
227     for k, v in line.items():
228         x[i][k - 1] = v
229
230 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
231
232 y_train = oneHot(y_train)
233
234 print('===== Binary (Digit 1 and 2) =====')
235
236 y_bin = np.concatenate((y[y==1],[0]*sum(y==2)))
237 temp = np.zeros((len(y_bin), 2))
238 for number in range(len(y_bin)):
239     temp[number][int(y_bin[number])] = 1
240 y_bin = temp
241
242 x_bin = np.concatenate((x[y==1],x[y==2]))
243
244 x_train_bin, x_test_bin, y_train_bin, y_test_bin = train_test_split(x_bin, y_bin, test_size=0.3)
245
246 parameters_bin = two_layer_model(x_train_bin.T, y_train_bin.T, (780, 100, 2), 0.1, 100, True)
247 print("Training Accuracy:")
248 training_prediction = predict(x_train_bin.T, y_train_bin.T, parameters_bin)
249 print ("Testing Accuracy:")
250 prediction_bin = predict(x_test_bin.T, y_test_bin.T, parameters_bin)
251
252 print('===== Multi-class =====')
253 parameters = two_layer_model(x_train.T, y_train.T, (780, 100, 10), 0.1, 300, True)
254
255 print("Training Accuracy:")
256 training_pred = predict(x_train.T, y_train.T, parameters)
257
258 print("Testing Accuracy:")
259 prediction = predict(x_test.T, y_test.T, parameters)
260
261 print(confusion_matrix(prediction, y_test))

```