DLM LABOR: Lineare Klassifikation in NumPy

Wintersemester 2017

Aufgaben:

3_1:

Kopieren Sie sich die py-Files und versuchen Sie die Elemente aus der Vorlesung wieder zu erkennen.

3_2:

Binden Sie anschließend Ihre im letzten Labor erstellte Funktion „myGrid()“ ein. Diese wird benötigt um die graphische Ausgabe des Trainings zu realisieren.

```
100
101 def plot_predict_grid(X_train, y_train, X_val, y_val, W, i):
102     xmin, xmax = X_train.min(), X_train.max()
103     #
104     # Aufruf ihrer Funktion aus der vorherigen Laborübung!
105     #
106     meshgrid = myGrid(xmin, xmax, 100)
107     #
108     #
109     #
110     meshgrid = np.hstack((meshgrid, np.ones((meshgrid.shape[0], 1))))
111     plt.figure(1)
112     plt.cla()
113     grid_pred = predict(meshgrid, W)
114     colors = ['red' if p==0 else 'blue' for p in grid_pred]
115     plt.scatter(meshgrid[:,0], meshgrid[:,1], c=colors, alpha=0.1, edgecolors=None)
116     colors = ['red' if y_i==0 else 'blue' for y_i in y_val]
117     plt.scatter(X_val[:,0], X_val[:,1], c=colors)
118     plt.savefig('results/%d_result.png' % i)
119
120
121
```

Lassen Sie anschließend den Klassifikator trainieren. Wenn sie den Aufruf von „plot_predict_grid()“ in der Trainingsfunktion zulassen wird pro Trainingsepoche ein Bild erzeugt.

```
82 def train(X_train, y_train, X_val, y_val, W, iterations=100):
83
84
85     for i in range(iterations):
86
87         grad = eval_gradient(X_train, y_train, W)
88         W = update_W(grad, W)
89
90         y_pred = predict(X_val, W)
91
92         accuracy = np.mean(np.array(y_pred == y_val, dtype=np.uint))
93         print 'Accuracy: ', accuracy
94
95         #
96         #
97         #plot_predict_grid(X_train, y_train, X_val, y_val, W, i)  #Hier einkommentieren!
98         #
99         #
100     return W
```

Bewundern Sie ihre Bilder! (Achten Sie darauf, dass Sie Schreibrechte auf das Verzeichnis haben, in dem die Bilder gespeichert werden sollen).

3_3:

Die Loss Funktion des Klassifikators ist momentan die Softmax Funktion. Implementieren Sie eine Funktion „l_i = L_i_hinge(x_i, y_i, W)“ die den Hinge Loss / Multiclass SVM Loss berechnet. Verwenden Ihre neue Loss Funktion und Trainieren Sie erneut.

```
19 def L(X, y, W):
20     loss = 0
21     for x_i, y_i in zip(X[:,], y[:]):
22         l_i = L_i_softmax(x_i, y_i, W)  # tauschen Sie diesen Aufruf...
23         #
24         #
25         #
26         #  l_i = L_i_hinge(x_i, y_i, W) # ...gegen diesen
27         #
28         #
29         #
30         loss += l_i
31     loss = loss / X.shape[0]
32
33     # regularizer
34     loss += 0.1 * np.linalg.norm(W, ord=2)
35
36     return loss
37
38
39 def L_i_softmax(x_i, y_i, W):
40     s_i = score(x_i, W)
41     e_i = np.exp(s_i)
42     e_i_norm = e_i / np.sum(e_i)
43     loss_i = -1.0 * np.log(e_i_norm[y_i])
44
45     return loss_i
```

Stellen Sie Unterschiede fest?