Optimalizálás intro

Legmeredekebb csökkenés módszere

- Feladat: minimalizálni akarjuk az *f* függvényt, azaz keressük $x^* = \underset{x}{argmin}\, f(x)$-ot

Legmeredekebb csökkenés módszere

- Feladat: minimalizálni akarjuk az *f* függvényt, azaz keressük $x^* = \underset{x}{argmin}\, f(x)$-ot
- Ötlet: adott η lépésköz mellett iteratívan próbáljuk meg közelíteni x*-ot

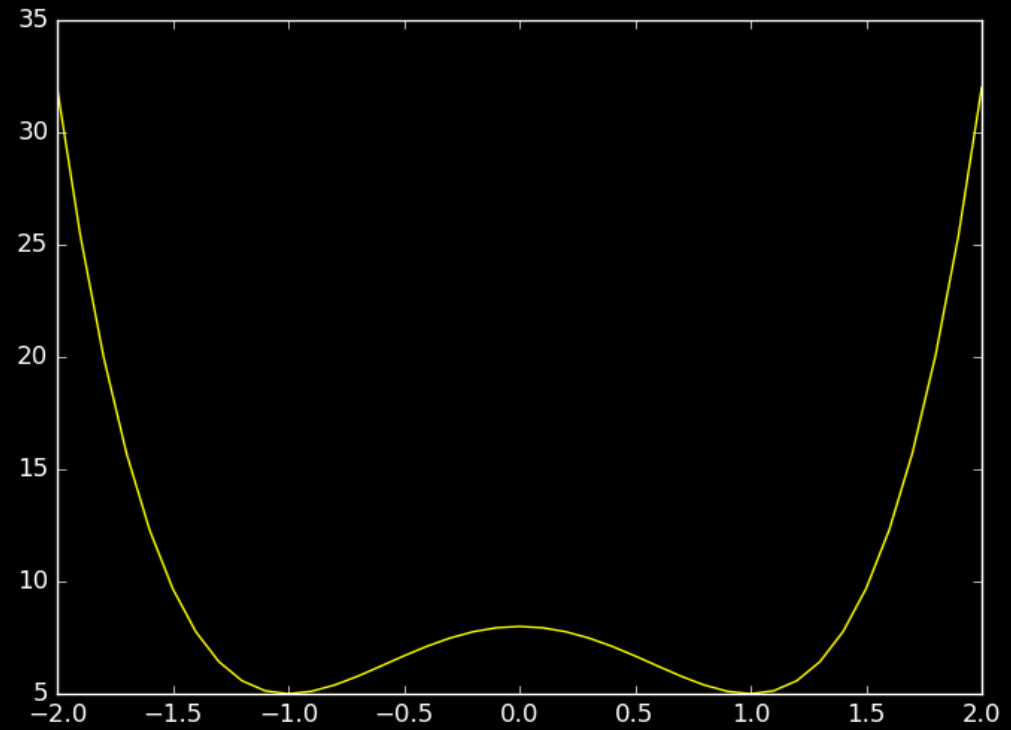
Legmeredekebb csökkenés módszere

- Feladat: minimalizálni akarjuk az *f* függvényt, azaz keressük $x^* = \underset{x}{argmin}\, f(x)$-ot
- Ötlet: adott η lépésköz mellett iteratívan próbáljuk meg közelíteni x*-ot

```
1: x(0) ← 0                  # inicializáljuk x-et
2: For i=0 to max_iter:
3:    x^(i+1) ← x^(i) − η f'(x^(i))
4: return x(max_iter)
```

Gyakorlás

- Python-t használva próbáljuk meg minimalizálni a $3*x^4-6*x^2+8$ függvényt!
- Mit tapasztalunk különböző η értékek használata mellett?
  - Pl. η={0.01, 0.1, 1}



Legmeredekebb csökkenés kiterjesztése többdimenziós esetre

- $x^{(i+1)} \leftarrow x^{(i)} - \eta f'(x^{(i)})$ helyett $x^{(i+1)} \leftarrow x^{(i)} - \eta \nabla f(x^{(i)})$
  - ∇f az f függvény gradiense, azaz a parciális deriváltjait tartalmazó vektor $\nabla f = [\partial f(x)/\partial x_1 \; \partial f(x)/\partial x_2 \; \ldots \; \partial f(x)/\partial x_d]$

Legmeredekebb csökkenés kiterjesztése többdimenziós esetre

- $x^{(i+1)} \leftarrow x^{(i)} - \eta f'(x^{(i)})$ helyett $x^{(i+1)} \leftarrow x^{(i)} - \eta \nabla f(x^{(i)})$
  - ∇f az f függvény gradiense, azaz a parciális deriváltjait tartalmazó vektor $\nabla f = [\partial f(x)/\partial x_1 \; \partial f(x)/\partial x_2 \; \ldots \; \partial f(x)/\partial x_d]$
- Gyakorlás
  - Python-t használva próbáljuk meg minimalizálni az $f(x,y)=3*x^4-6*y^2+8$ függvényt! (f most alulról nem korlátos)
  - Mit tapasztalunk különböző η értékek használata mellett? (pl. η={0.01, 0.1, 1})

Másodrendű eljárások

- Egyváltozós eset

$$x^{(i+1)} \leftarrow x^{(i)} - \eta f'\left(x^{(i)}\right) \text{ helyett } x^{(i+1)} \leftarrow x^{(i)} - \frac{f'\left(x^{(i)}\right)}{f''\left(x^{(i)}\right)}$$

- Többváltozós eset

$$x^{(i+1)} \leftarrow x^{(i)} - H^{-1}\left(x^{(i)}\right) \nabla f\left(x^{(i)}\right)$$

  - H az *f* függvény Hesse-mátrixa (*f* másodrendű parciális deriváltjait tartalmazza). Mi az előnye/hátránya?

- Gyakorlás: a korábbi optimalizálási feladatok megoldása másodrendű módszereket alkalmazva

Gyakorlati szempontok

- Érzékenység a paraméterek kezdeti értékeire
- η-ra való érzékenység
- Olcsóság-vs-sebesség
  - Másodrenű eljárások **sokkal** költségesebbek, de gyorsabbak (kevesebb iteráció alatt képesek a tényleges optimum közelébe eljutni)

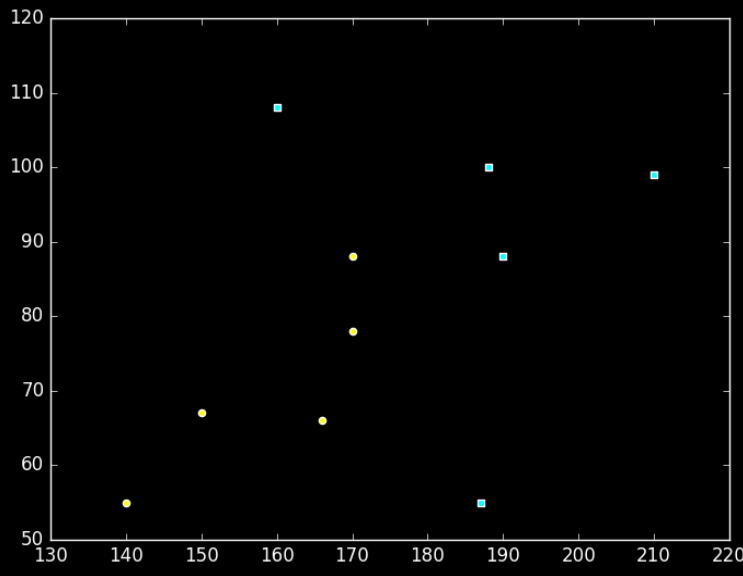
További olvasnivalók

- Interaktív JS vizualizáció
  - http://www.benfrederickson.com/numerical-optimization/
- Gradiens módszer kiterjesztései
  - http://sebastianruder.com/optimizing-gradient-descent/

Modellezés

- Gépi tanulással valós életbeli jelenségek modellezhetők prediktív jegyek/jellemzők alapján
  - Pl. ha ismerjük egy ember magasságát és testsúlyát, modellezhetjük valamely betegségre való hajlamát
- Tanítóadatbázis: $\{(x_i, y_i)\}_{i=1}^{n}$ párosok halmaza

N	N	N	N	N	P	P	P	P	P
140	150	170	166	170	188	190	210	187	160
55	67	78	66	88	100	88	99	55	108



- Olyan modellt keresünk, ami „minél kevesebbet” hibázik a tanítópéldákon
  - **A modell általánosító képessége is fontos!**

Veszteségfüggvények

- A legjobb modell meghatározása fölfogható egy optimalizálási feladatként
- Számos modell,-és hibafüggvény adható ugyanannak a problémának a jellemzésére
  - Egy lehetőség: logisztikus regresszió

Logisztikus regresszió

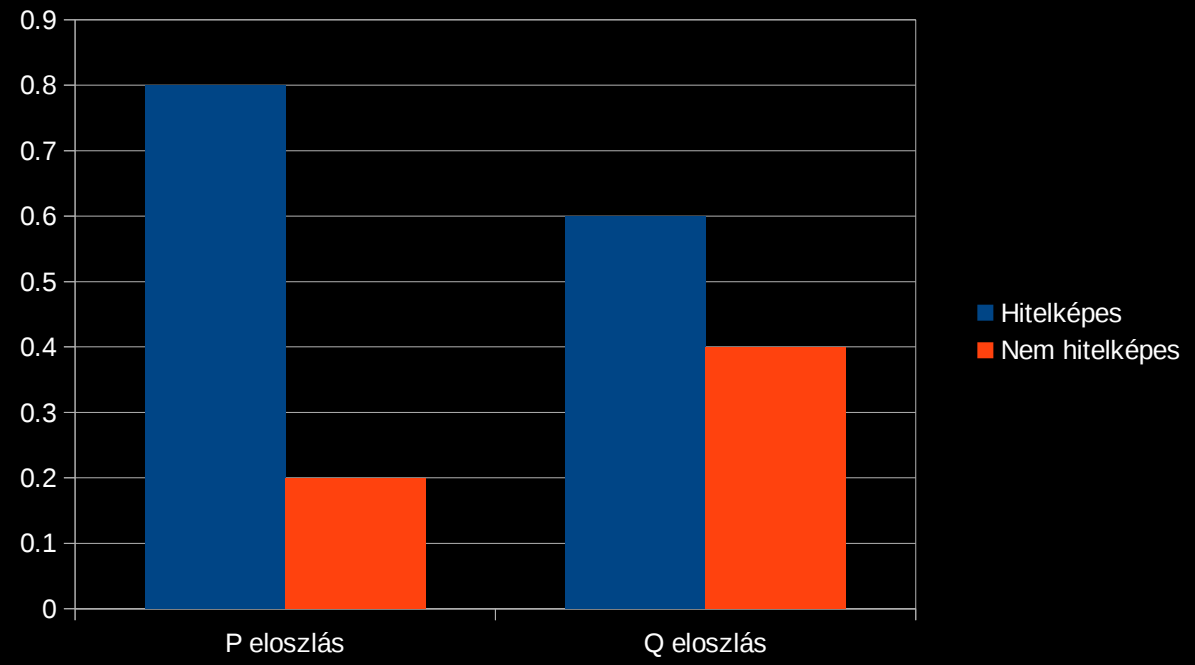
- Kétosztályos eset, azaz $Y \in \{+1, -1\}$
- Modell: $\widetilde{P}(Y_i = 1 | x_i) = \sigma(w^T x_i) = \frac{1}{1 + e^{-w^T x_i}}$
  - Értelemszerűen: $\widetilde{P}(Y_i = -1 | x_i) = \sigma(-w^T x_i) = \frac{1}{1 + e^{w^T x_i}}$
  - Tömörebben: $\widetilde{P}(Y_i = y | x_i) = \frac{1}{1 + e^{-y w^T x_i}}$
- Hogy viselkedik a σ függvény?
- Az egyes tanítópéldák kapcsán az ún. **keresztentrópia** legyen a hibatag

Hogy viselkedik a σ függvény?

- $\sigma(z) = 1/(1+\exp(-z))$
  - $\sigma(z) \rightarrow 1$, ha $z = ?$
  - $\sigma(z) = 0.5$, ha $z = 0$
  - $\sigma(z) \rightarrow 0$, ha $z = ?$
- Gyakorlás: mi lesz $\sigma(z)$ deriváltja?

Keresztentrópia

- Eloszlások különbözőségének számszerűsítésére használható $H(P,Q) = -\sum_{x \in X} P(X = x) \log Q(X = x)$



- H(P, Q) = ?

Keresztentrópia mint hibatag

- Az egyes pontokon elszenvedett veszteség mértéke legyen az adott pontra vonatkozó keresztentrópia
  - P eloszlás legyen a pontra vonatkozó osztályeloszlás
    - $P(Y_i=1|x_i) \in \{1.0, 0.0\}$
  - i. pontra vonatkozó hiba: $l_i(x_i, y_i) = H(P, \widetilde{P}) = \log(1+e^{-y_i w^T x_i})$

$$\nabla_w l_i(\boldsymbol{x_i}, y_i) = \frac{1}{1+e^{-y_i w^T \boldsymbol{x_i}}} e^{-y_i w^T x_i} - y_i \boldsymbol{x_i} = -y_i P(Y_i = -y_i | x_i) \boldsymbol{x_i}$$

    - Értelmezzük a kapott gradienst!
- A tanítóadatbázis hibája a pontonkénti hibák **összege**

Implementáljuk a saját LogRegünket!

- Töltsük be az Iris adatbázist
  - Az első 100 megfigyelés első 2 jellemzőjét használjuk!
  - Alkalmazzunk 80:20-as tanító:teszt vágást

```
import numpy as np
from sklearn import linear_model, datasets

iris = datasets.load_iris()
X_train = np.concatenate((iris.data[0:40, :2],
                          iris.data[50:90, :2]))
X_test = np.concatenate((iris.data[40:50, :2],
                         iris.data[90:100, :2]))
Y_train = np.array(40*[1] + 40*[-1])
Y_test = np.array(10*[1] + 10*[-1])
```

Egy minimalista LogReg implementáció

```
def cost(x, y, w):
    return sum(np.log(1+np.exp(-y*x.dot(w))))

def gradients(x, y, w):
    misclass_probs = 1/(1+np.exp(y*x.dot(w)))
    return x*(-y*misclass_probs)[:, np.newaxis]
```

Egy másik minimalista implementáció

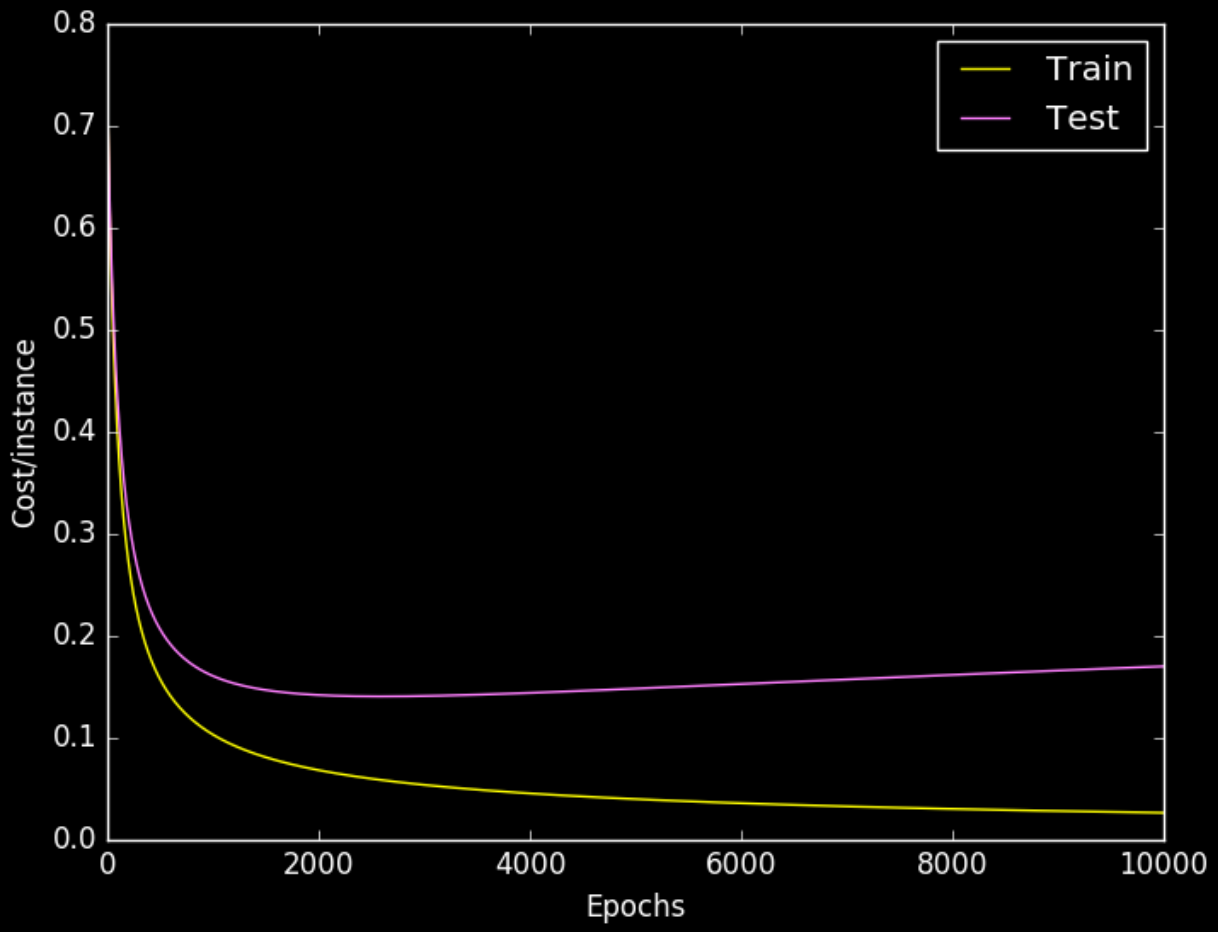
```
P = lambda x,y,w: 1/(1+np.exp(-y*x.dot(w)))

cost = lambda x,y,w: sum(np.log(1+np.exp(-y*x.dot(w))))

gradients = lambda x,y,w: np.sum(x*(-y*(1-
P(x,y,w))).reshape(x.shape[0], 1), axis=0)
```

Tanulás

```
weights = np.array([0.1, -0.1])
costs = []
for it in range(10000):
    train_cost = cost(X_train, Y_train, weights)
    test_cost = cost(X_test, Y_test, weights)
    costs.append([train_cost, test_cost])
    train_g = gradients(X_train, Y_train, weights)
    weights -= 0.001*np.sum(train_g, axis=0)
```

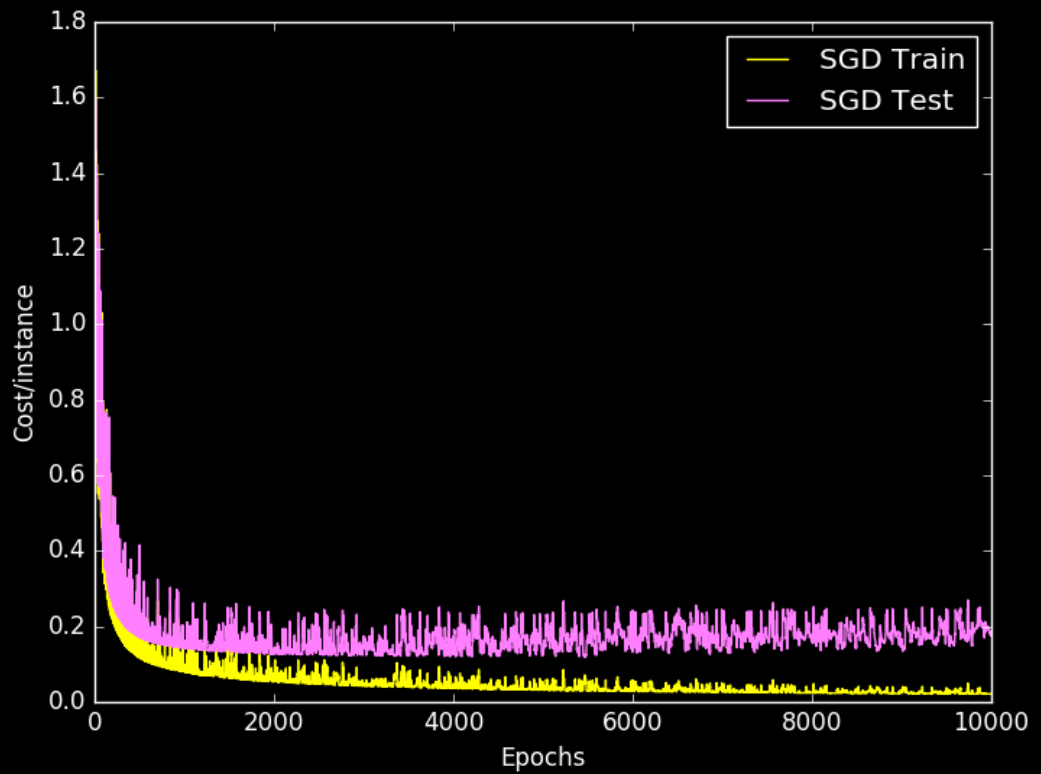
A hibafüggvény változása

- Túltanulás/alultanulás
  - Regularizáció fontossága



Sztochasztikus gradiens módszer

- Gradiensek összegzése helyett véletlenszerűen válasszunk egy pontot, amely szerint frissítünk
  - Azaz `np.sum(train_g, axis=0)` helyett `train_g[np.random.randint(X_train.shape[0])]` alapján frissítsünk
  - Most egy epoch kicsit mást jelent, mint az előbb
    - Pontosan mit is?



Fontos!

- A tesztadatokat soha, **soha**, **SOHA** ne használd a modell tanítása során
  - A tesztadatokon a modell kiértékelését végezzük, a modell paramétereinek hangolására nem használható

Fontos!

- A tesztadatokat soha, **soha**, **SOHA** ne használd a modell tanítása során
  - A tesztadatokon a modell kiértékelését végezzük, a modell paramétereinek hangolására nem használható
- Dedikált tanító/tesztelő adatbázisok helyett alkalmazhatunk keresztvalidációt is
  - Osszuk a teljes adatbázist *k* részre, amelyből *(k-1)* egységet használjunk tanításra, a fennmaradó 1/k hányadot pedig tesztelésre