4. feladat

Tanulni és/vagy bulizni

A hallgatók a tanulás mellett még sok időt töltenek bulizással is. Vizsgáljuk meg, hogy az ezekkel töltött idő hogyan befolyásolja a Halnum tárgy sikeres elvégzésének esélyét! Ehhez megkérdeztük 1001 korábbi hallgatónkat visszamenőleg 42 évig arról, hogy mennyi idő töltöttek a második éves programozás jellegű tantárgyak tanulásával és közben bulizással, továbbá, hogy milyen eredményt értek el. Ha valaki átment, akkor azt "1"-gyel jelöltük, ha nem ment át, akkor "0"-val.

A feladat célja két dolog megtanulása (természetesen a tanulással töltött időbe beszámítható módon): Igen/nem kimenetelű kísérletek (mérések, vagy próbálkozások) eredményeihez valószínűségi eloszlásfüggvény illesztése (logisztikus regresszió) és többdimenziós minimumkeresés (azon belül a gradiens módszer).

Az elméleti alapokat az elméleti weboldalról kell elsajátítani. A feladat szövegében csak a legfontosabb pontokat foglaljuk össze.

Logisztikus regresszió

Kiindulásul feltesszük, hogy a siker valószínűsége az őt befolyásoló x_1, x_2 változóktól a következő logisztikus függvényalak szerint függ:

$$g(\mathbf{ax}) = \frac{1}{1+e^{-\mathbf{ax}}} \ ,$$

ahol egy konstans $x_0 = 1$ változót bevezetve az exponensben $\mathbf{ax} = a_0 + a_1 x_1 + a_2 x_2$, ami az általános lineáris kifejezés. Az adathalmazunkat az egyes, i-vel sorszámozott kísérletekhez tartozó $\mathbf{x}^{(i)}$ változóvektorok és $y^{(i)}$ kimenetelek összessége alkotja ($i = 1 \ldots N$). Az illesztés jóságát a maximum likelihood elv alapján a következő költségfüggvény jellemzi:

$$J(\mathbf{a}, \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \mathrm{cost}(y^{(i)}, g(\mathbf{ax}^{(i)})) \ ,$$

ahol a tagonkénti cost függvény alakja

$$\mathrm{cost}(y^{(i)}, g(\mathbf{ax}^{(i)})) = \begin{cases} -\log(g(\mathbf{ax}^{(i)})) \ , & \text{ha } y^{(i)} = 1 \\ -\log(1 - g(\mathbf{ax}^{(i)})) \ , & \text{ha } y^{(i)} = 0 \end{cases}$$

Ennek a J függvénynek az $a_j, j = 0, 1, 2$ paraméterek szerinti minimumát keressük.

Minimum keresés

A minimum megkeresésére a gradiens módszert (legmeredekebb ereszkedés) fogjuk használni. Kicsit tovább számolva könnyen megkaphatjuk a $J(\mathbf{a}, \mathbf{x}^{(i)}, y^{(i)})$ költségfüggvény gradiensének komponenseit:

$$\frac{\partial}{\partial a_j} J(\mathbf{a}, \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial a_j} \mathrm{cost}(y^{(i)}, g(\mathbf{a}\mathbf{x}^{(i)})) \ ,$$

ahol a tagonkénti derivált

$$\frac{\partial}{\partial a_j} \mathrm{cost}(y^{(i)}, g(\mathbf{a}\mathbf{x}^{(i)})) = (g(\mathbf{a}\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

Figyeljünk arra, hogy az egyes i értékekehez tartozó $\mathbf{a}\mathbf{x}^{(i)}$ szorzatot és a rajta felvett g értéket csak egyszer számoljuk ki, ne minden j értékhez külön! Az így kapott gradienssel ellentétes irányba léptetjük a paraméterek vektorát, azaz

$$a_j := a_j - \alpha \frac{\partial}{\partial a_j} J(\mathbf{a}, \mathbf{x}^{(i)}, y^{(i)}) \ ,$$

amihez persze megfelelő α értéket kell találni.

A programnak a következő parancsargumentumokat kell használnia ebben a sorrendben:

adatfájl_neve α

A programot a következő adatfájllal kell kipróbálni:

- tanul.dat

Ennek első két oszlopában az x_1, x_2 értékek, harmadik oszlopában a hozzájuk tartozó y kimenetel található.

Beadni a C++ programot és az alábbiak szerint elkészített Jupyter notebookot kell. A C++ program neve main-a.cpp, main-b.cpp ... ill. main-f.cpp legyen attól függően, hogy melyik szinten működik! A program legyen úgy felépítve, hogy a b,c,d,f részek egymás után fussanak le, így egy programot elég beadni. A Jupyter notebook neve legyen results.ipynb !

a)

Hogy áttekintő képet kapjunk az adathalmazról, először ábrázoljuk azt pythonban a következő utasításokkal:

```
%pylab inline
xy=loadtxt("tanul.dat")
n=len(xy)
x=ones((n,3))
x[:,1:]=xy[:,:2]
y=xy[:,2]
figsize(6,6)
```

```
plot(x[y==1,1],x[y==1,2],"r.",ms=4)
plot(x[y==0,1],x[y==0,2],"k.",ms=4)
```

Ezután olvassuk be az adatokat C++ programmal, és írjunk függvényt, ami ebből ki tudja számolni a költségfüggvényt az a_0, a_1 és a_2 paraméterek függvényében! (A)

b)

A költségfüggvény elég bonyolult viselkedést mutathat az a_0, a_1, a_2 függvényében. Azért, hogy egy kis intuíciót szerezzünk, első lépésként az a) pontban megírt költségfüggvény számoló függvényt használjuk arra, hogy $a_2 = 0$ fixen tartva és a_0, a_1 értékeit egy téglalap rácson futtatva kiírja J(a_0,a_1,a_2=0)-t egy Jgrid.dat nevű fájlba! (Ez annak felel meg, hogy a bulizás alig befolyásolja az eredményt adott tanulási idő mellett). Soronként szerepeljenek az összetartozó a_0, a_1, J értékek! A Jgrid.dat fájlt pedig pythonban a már megkezdett notebookban olvassuk be és készítsünk róla szintvonalas ábrát! Segítségül: az $a_0 \in [-15, -5]$, $a_1 \in [0, 0.6]$ intervallumokat érdemes használni, azon belül kell lennie a minimumnak. Az intervallumokon legalább 50-50 pontot vegyünk! Pythonban így tudjuk elvégezni az ábrázolást, az m értékébe beírva *a ténylegesen használt pontok számát*:

```
Jgrid = loadtxt("Jgrid.dat")
m = 50
a0mesh = resize(Jgrid[:,0],(m,m))
a1mesh = resize(Jgrid[:,1],(m,m))
Jmesh = resize(Jgrid[:,2],(m,m))
figsize(16,4)
cs = contour(a0mesh,a1mesh,Jmesh,levels=66)
clabel(cs);
```

Mit tapasztalunk? Írjuk be a notebookba! (A)

c)

Írjuk meg a gradienst kiszámoló függvényt és ennek segítségével keressük a költségfüggény minimumát! Induljunk el az $a_0 = -12, a_1 = 0.3, a_2 = 0$ értékekből! Első futtatásnál használjunk $\alpha = 0.01$, utána vegyük α értékét olyan kicsire, hogy ne jelentkezzen oszcilláció, vagy legalább gyorsan lecsengő legyen a költségfüggvény értéke. Ha pedig konvergál, akkor próbáljuk ki egy közbenső α értékkel is! Ábrázoljuk ezt a három lefutást, adatfájlba kiírva a szükséges adatokat!

Célszerű megoldás parancssori futtatás esetén a > jellel mindig újabb fájlba irányítani az adatokat. Ha kezdetben Codeblocksban fejlesztjük a programot, akkor pedig írjon fájlba a program, amit utána más-más névre átnevezhetünk.

Figyeljünk oda: ha úgy tűnik, hogy egy idő után már konstans J értéke, mert konvergál, akkor meg kell nézni az adatsort az első néhány lépés lehagyásával, ezáltal ránagyítva a görbe lassan változó végére. Kiderülhet ugyanis, hogy csak a minimum körüli völgy keresztirányában jutottunk le és a másik irányban tovább haladva J értékében még további lényeges csökkenés történhet. Ha a további csökkenés is megállni látszik (akár többszáz

lépés alatt), akkor újra növeljük meg az ábrázolás kezdőpontját, hátha további lassú csökkenést tapasztalunk (mivel van még egy keresztirányú dimenzió). (A)

d)

Azt láthattuk a b) rész megoldása során, hogy nagyon hosszúkás a költségfüggvény minimuma körüli völgy. A két tengely egyforma léptéke esetén pedig még hosszúkásabb lenne. A gradiens módszer pedig azonos léptékben nézve léptet a szintvonalakra merőlegesen, ez okozhatott nagyon lassú konvergenciát.

Mint ki fog derülni, a következő trükköt alkalmazhatjuk a lassú konvergencia javítására. Az x_1, x_2, y értékeit tartalmazó adatkészletből létrehozunk egy újat, amelyben x_1 és x_2 értékei az átlagukkal eltolva szerepelnek. Azaz az $x_1^{(i)}$ értékeket is csökkentjük az átlagukkal, és az $x_2^{(i)}$ értékeket is az átlagukkal. Ismételjük meg a b) részbeli ábrázolást így, egy Jgrid2.dat nevű adatfájlt létrehozva! Ehhez az a_0 paraméter intervallumát állítsuk be úgy, hogy a minimum jól látsszon! A c) részbeli minimum keresést is végezzük el az új adatkészlettel. Ellenőrizzük, hogy a korábbiaknál nagyobb α értékekkel is kapunk-e konvergenciát! Szemléltessük két ábrával, hogy milyen α érték környékén romlik el a konvergencia! Mit tapasztalunk a b), c) részekhez képest? (T)

e) Szorgalmi feladat:

A d) részbeli minimumkeresést írjuk meg az ábrákat készítő notebookban pythonban is! Ábrázoljuk a költségfüggvényt a lépésszám függvényében! Hasonlítsuk össze a C++ és Python változat futásidejét! A Python változatban törekedjünk arra, hogy a költségfüggvény és a gradiens értékeit for ciklusok nélkül, vektorosan számoljuk ki!

f) Szorgalmi feladat:

Írjuk meg az elméleti ismertetőben szereplő Adam módszert felhasználó minimumkeresést is! Használjuk itt is a c) részben használt α értékeket és adathalmazt! A Jupyter notebookban ábrázoljuk együtt a költségfüggvény értékének alakulását a lépésszám függvényében az eredeti és az Adam módszer használata esetére.

g) Szorgalmi feladat

Írjuk meg a minimum keresést a stochastic gradient descent segítségével! Próbáljunk olyan α paraméterértéket használni, amelyről az előző feladatrészekben már láttuk, hogy batch gradient descent használata esetén az eljárás konvergál! Ábrázoljuk a költségfüggvényt az iterációs lépésszám függvényében az előadáson tárgyaltak szerint!