4. feladat

Tanulni és/vagy bulizni

A hallgatók alapesetben (Covid-19 előtt/után) a tanulás mellett még sok időt töltenek bulizással is. Vizsgáljuk meg, hogy az ezekkel töltött idő hogyan befolyásolja e tantárgy sikeres elvégzésének esélyét! Ehhez megkérdeztük 1001 korábbi hallgatónkat visszamenőleg 42 évig, hogy a 2. éves programozás jellegű tantárgyak tanulásával és bulizással mennyi időt töltöttek, és milyen eredménnyel. 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. Az elméleti anyaghoz képest kicsit tovább számolva a fenti függvényből könnyen megkaphatjuk a gradiense 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

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

Vegyük észre, hogy ezeket az összegzéseket úgy a legegyszerűbb elvégezni, hogy minden i-re kiszámoljuk a g értékét (index nélküli változóban), majd azt felhasználjuk az összegzésekhez, így nem szükséges a g értékeket eltárolni. 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:

$$\mathrm{Az\_adatfájl\_neve} \ \ N$$

ahol N az adatok száma az adatfájlban.

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.

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(a0,a1,a2=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 keressük a minimumot ennek segítségével! Vegyük α értékét olyan kicsire, hogy ne jelentkezzen oszcilláció, vagy legalább lecsengő legyen! Pl. ha egy kezdetben választott α értékkel oszcillál, akkor ismétlődően vegyük alpha értékét felére-ötödére, amíg jó nem lesz. Ha pedig konvergál, akkor próbáljuk ki 1.5, v. 2-szer nagyobb értékkel is! Vizsgáljuk, hogy a lépések során a paraméterértékeknél felvett költségfüggvényérték hogyan változik! Ábrázoljuk ezt a lefutást különböző α értékek mellett (adatfájlba kiírva a szükséges adatokat)!

Vigyázzunk, mert ha úgy tűnik, hogy egy idő után konstans J értéke, meg kell nézni az adatsort az első néhány lépés lehagyásával. Akkor kiderülhet, hogy csak a minimum körüli völgy keresztirányában jutottunk le és a másik irányban tovább haladva J értéke tovább csökken. 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). Aki ezt tapasztalja, olvassa el a d) részt! (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. Ezen úgy tudunk javítani, hogy az x_1, x_2, y értékeit tartalmazó adatkészletből létrehozunk egy újat, amelyben x_1 és x_2 értékei az átlagukkal eltolva szerepelnek. Ismételjük meg a b) részbeli ábrázolást, 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, a lefutást különböző alfa értékek mellett ábrázolva! Közben persze a notebookban tartsuk meg a korábbi ábrázolásokat is! 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: Elvileg nem csak a feladat elején megadott költségfüggvény alakot használhatjuk. Írjuk fel a minimumkereséshez szükséges képleteket legkisebb négyzetek módszere és a fenti g logisztikus függvény használatával, és futtassuk le a minimumkeresést a gradiens módszer segítségével! Más eredményt kapunk, mint a d) részben?