2018

Gépi tanulás

KÉPFELISMERÉS GÉPI TANULÁSOS MÓDSZEREK ALKALMAZÁSÁVAL PYTHON NYELVEN

ÍRTA: KÖŐ JÓZSEF TÉMAVEZETŐ: CZÚNI LÁSZLÓ

1. Tartalom

1 Bevezetés 2

2 Projekt kezelés 2

3 Python programnyelv 2

3.1 Python verzió 3

3.2 Fejlesztői környezet 3

4 Képadatbázis 3

4.1 ImageNET 3

4.2 CIFAR-10 4

5 K-legközelebbi szomszéd 6

6 Lineáris osztályozás 7

6.1 Veszteségszámítás 9

6.2 Optimalizációs eljárások 10

7 Neurális hálózatok 11

8 Összegzés 11

1 Bevezetés

Célom a projekt során megismerni, és bemutatni a gépi tanulás néhány módszerét és alkalmazását. Néhány gyakorlati példán keresztül ismertetném az egyes megoldásokat adott problémára.

Forrásként főként a Stanford egyetem egy online elérhető, a CS231n nevű kurzusának anyagát használtam. A kurzus alapvetően a konvolúciós neurális hálókat hivatott bemutatni, de emellett bemutat még más tanulási módszereket is.

A projekt során a fő probléma, amin keresztül bemutatom az egyes tanulási algoritmusokat, egy osztályozás jellegű probléma. Egyszerű képek besorolása néhány kategóriába a tartalmuk alapján. Például egy macskát ábrázoló képet a macskák kategóriába kell besorolnunk.

2 Projekt kezelés

A projektet a könnyed kezelhetőség, és követhetőség érdekében Git verziókezelővel menedzseltem. A forrás elérhető a https://github.com/koojozsef/AI-project-for-university url címen nyilvánosan. Az alap struktúra szerint az *src/* mappában található a részfeladatokat tartalmazó Python szkriptek. A projekt fő könyvtárában található egy *Readme.md* fájl, amely egy rövid összefoglalót tartalmaz a projektről. Jelen dokumentáció a *data/* mappában található.

3 Python programnyelv

Mivel a forrásként megjelölt kurzusban is a Python programnyelvet használják, valamint napjainkban elég előrehaladott, és népszerű nyelv ez, ezért a problémák megoldásait én is Python nyelven prezentálom.

A Python egy általános célú szkript nyelv, amely fordítási időben futtatja a programot. A feladatainkra az teszi igazán alkalmassá, hogy igen erős támogatottsága miatt léteznek olyan könyvtárak hozzá, amelyek igen hatékonyan kezelik a tudományos számítási műveleteket. A numpy könyvtár tartalmazza azokat az eszközöket, amelyekkel igen egyszerű és hatékony kezelni a mátrix műveleteket. Könnyen dolgozhatunk vele többdimenziós adattömbökön, adathalmazokon is.

3.1 Python verzió

A python két fejlesztési ága terjedt el: a 3.x és a 2.7.x verziószámozással jelöltek. A projektem során a 2.7 verziót használtam. A két verzió között néhány függvényben van csupán különbség, valamint kompatibilitási problémák is előfordulhatnak. A *http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html* weboldal rövid áttekintést ad a két irányvonal főbb különbségeiről.

3.2 Fejlesztői környezet

Az Anaconda csomagkezelő asztali grafikus program tartalmaz többféle fejlesztői környezetet. Az Anaconda kezeli a parancssori interfészt, így egyszerűen használhatóvá teszi a fejlesztést, adott fejlesztési környezetekben. Támogatja a JupyterLab, Jupyter Notebook, QTConsole, Spyder, VSCode valamint sok egyéb környezetet alapértelmezetten.

Projektem fejlesztéséhez Spyder környezetet használtam. Számomra kényelmes, és igen hamar tanulható alkalmazás. A felület bal oldalán található alapértelmezetten a szövegbeviteli rész. Itt a készítendő szkript fájlokat lehet látni és szerkeszteni. A jobb oldalon helyet kapott a parancssori kimenet, ahol a program futtatásának eredményét, valamint a hibaüzeneteket figyelhetjük. A jobb oldalon még nyomon követhetőek az aktuális futás során létrehozott változók értékei és szerkezeti felépítésük, valamint típusaik is.

4 Képadatbázis

A gépi tanulásos módszerek alapját a tanítási halmaz képezi. Szükséges a megléte egy olyan adathalmaznak, amely már osztályozott elemeket tartalmaz, és rendelkezik olyan címkével, amivel majd az új adatokat az algoritmusunk fog ellátni.

4.1 ImageNET

A képek forrásaként először az Image-net.org weboldalt használtam. Itt elérhető több mint 14 millió kategorizált kép, és kategóriánként van lehetőség a letöltésükre. Van lehetőség továbbá a képek URL-jének letöltésére is. Ezek szintén kategóriák szerint érhetőek el egy-egy szövegfájlban felsorolásként. Ez utóbbi forrást választottam, amihez szükségeltetett egy feldolgozó program, ami letölti automatizáltan a képeket az URL-ek alapján.

A projektemben a *00_image_download_from_URL.py* fájl tartalmazza a kódot, ami a képek letöltéséért felelős. Az első részletben beolvastam a sorokat a fájlból egy tömbbe.

```
17  #read URL-s from file
18  fname = "..\data\images_URL\imgurl_cat.txt"
19  with open(fname) as f:
20      URL = f.readlines()
```

Így az URL tömb tartalmazza az összes elérési utat, ami macskát ábrázoló képekre mutat.

```
27  for url in URL:
28      try:
29          response = requests.get(url)
30          img = Image.open(BytesIO(response.content))
31          #imgArray.append(img)
32
33          #save image to ../data/images_gen/cat/
34          img.thumbnail(size, Image.ANTIALIAS)
35          name = "%04d.jpg" % i
36          savepath = "..\data\images_gen\cat\cat_" + name
37          img.save(savepath)
38          i=i+1
39      except:
40          print("error")
```

Egy ciklussal végighaladva a tömbön egyesével letöltöttem a képeket egy meghatározott mappába. A képek nevei és a mappa neve reprezentálta a rendszeremben a címkéket.

A képek letöltésekor azonban probléma lépett fel, ugyanis a fájl tartalmazott sok olyan elérési utat ahol már vagy nem létezett a kép, vagy nem volt elérhető. Így a teljes adathalmaznak csak egy részét tudtam ezzel a módszerrel megszerezni. Továbbá a sikeresen letöltött képek is tartalmaztak fals értékeket, ugyanis néhány oldalon a megszűnt képet egy hibát reprezentáló képpel helyettesítették, így ezt a programom már nem tudta kiszűrni, és címkézett adatként le is töltötte.

4.2 CIFAR-10

Egy másik megoldást, a CIFAR-10 adatbázis jelentett. Ez egy speciális adathalmaz, ami 60000 darab 32x32 pixel méretű színes képet tartalmaz. A képek tíz osztályba vannak sorolva:

- repülőgép

- autó
- madár
- macska
- szarvas
- kutya
- béka
- ló
- hajó
- teherautó

Ezeket a képeket speciális formátumban lehet elérni. A letöltést követően 6 kiterjesztés nélküli fájlhoz jutunk hozzá. A 6 fájl összekeverve tartalmazza a képeket, mindegyikhez hozzárendelve egy osztályt jelentő számot. Öt fájlt jelöltek meg tanító halmazként, és egy fájlt tesztelő halmazként. Minden fájl azonos hosszúságú, és a következőképpen tárolja a kép adatokat:

Egy fájl 10000 képet tartalmaz. Egy képet egy „sor” reprezentál. Az első bájt az osztályt jelölő címkét tartalmazza (0-tól 9-ig), az utána következő 1024 bájt jelenti a kép vörös pixelkomponenseit, majd 1024 bájt a zöld, és újabb 1024 bájt a kék komponenseket tartalmazza.

```
21  #read base images
22  def unpickle(file):
23      import cPickle
24      with open(file, 'rb') as fo:
25          dict = cPickle.load(fo)
26      return dict
27
28  dictionary = unpickle('../info/cifar-10-batches-py/data_batch_1') #loading first batch
29  data = dictionary['data']
30  labels = dictionary['labels']
```

A batch fájlok tartalmát a fenti módon olvastam be. Definiáltam egy *unpicle* nevű függvényt, amely bemeneti paraméterként a fájl nevét és elérési útvonalát kapja meg, majd a beolvasott fájlt eltárolja egy programváltozóban és ezt adja vissza függvényhíváskor. Az adathalmazt tartalmazó *dictionary* változó egy speciális adatstruktúra, amely elemeire kulcsszavakkal is lehet hivatkozni. Ez a típus automatikusan, a fájl beolvasásakor generálódik.

5 K-legközelebbi szomszéd

A k-legközelebbi szomszéd osztályozó algoritmus egy elég egyszerűen használható módszer az osztályozásra. Az alapvető logikája, hogy tanulás során felveszünk bizonyos pontokat a paramétertérben. A betanított algoritmus ezután a beérkező új adatot elhelyezi a paramétertérben, és megnézi a hozzá legközelebb eső k darab pontot. Ezen pontok, mivel címkézett adatokat használtunk a betanítás során, meghatározzák az új pont osztályát.

Az algoritmus implementálására pythonban egy osztályt hoztunk létre, amely magába foglalja a módszer alkalmazásának állomásait. Az első megoldandó feladat a betanítás. A probléma prezentálására a CIFER-10 által szolgáltatott címkézett képadatbázist használtam.

```
51      def train(self, X, y):
52        """ X is N x D where each row is an example. Y is 1-dimension of size N """
53        # the nearest neighbor classifier simply remembers all the training data
54        self.Xtr = X
55        self.ytr = y
```

A fenti kódrészlet a betanító függvényt mutatja. Bemenő paraméterként megkapja a tanításul szolgáló képadat halmazt (K), valamint a példákhoz tartozó címkéket (y). A „self” paraméter az osztályra hivatkozik, hogy ezen keresztül érhessük el az ott tárolt változókat.

A tanítás folyamata egyszerű, mivel csak eltároljuk az összes bemeneti pontot a hozzájuk tartozó címkével. Az egyes képek vektorként vannak reprezentálva, így felfogható egy D dimenziós térben lévő pont ábrázolásaként. Az D jelen esetben a 32x32 pixeles kép 3 szín komponensével együtt értelmezett kiterítése, tehát 3072.

Az algoritmus osztályozó része már nem ennyire primitív.

```
57    def predict(self, X):
58      """ X is N x D where each row is an example we wish to predict label for """
59      num_test = X.shape[0]
60      print X.shape[0]
61      # lets make sure that the output type matches the input type
62      Ypred = np.zeros(num_test)
63
64      # loop over all test rows
65      for i in xrange(num_test):
66        # find the nearest training image to the i'th test image
67        # using the L1 distance (sum of absolute value differences)
68        distances = np.sum(np.abs(self.Xtr - X[i]), axis = 1)
69        min_index = np.argmin(distances) # get the index with smallest distance
70        Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
```

Az 59-62 sorban csupán egy ellenőrzés van, hogy a bemenő X képek számának megfelelő kimenetet kapjunk. Pl.: 3 bemenő képre, 3 címkét kapjunk eredményül.

A 65. sorban látható ciklus biztosítja, hogy ellenőrizzük az összes bemenetet. A „distances” vektorban tároljuk az adott kép pontként való reprezentációjának távolságát a betanított ponthalmaztól. Ezután már csak a „k” legkisebb távolságokat kell meghatározni, és megkapjuk a legközelebb eső „k” darab pontot a paramétertérben. Ezen pontok határozzák meg az új ponthoz (képhez) rendelt címkét.

Könnyen belátható, hogy ez az eljárás nem alkalmas a képek által ábrázolt objektumok szerinti osztályozásra, ugyanis nem vesszük figyelembe az egyes pixelek összetartozását, vagy közvetlen környezetével való kapcsolatot. Csak a pixelszinten hasonló képeket csoportosítjuk.

6 Lineáris osztályozás

Egy másik eljárás a lineáris osztályozás, amely módszerek közül az SVM, azaz Support Vector Machine módszert szeretném bemutatni. Ez annyiban hasonlít az előbbihez, hogy itt is egy paramétertérben helyezzük el az képeinket. Ám itt az osztályozás nem a környező pontok alapján történik, hanem megpróbálunk létrehozni és optimalizálni szeparátorokat, amelyek az adott csoportba tartozó pontokat (képeket) válassza el a többitől.

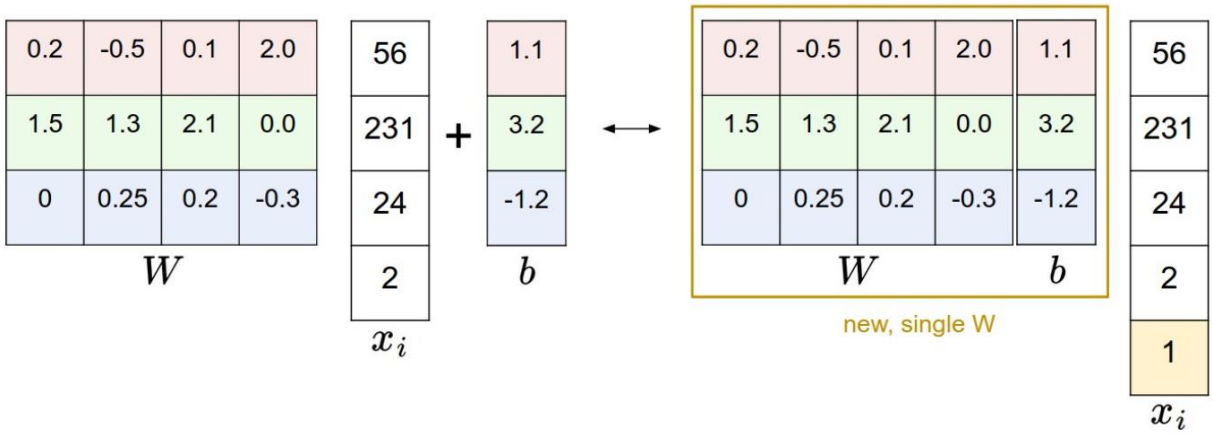
$$f(x_i, W, b) = Wx_i + b$$

A képlet írja le, hogy hogyan valósítsuk meg ezt a szeparátort. Kétdimenziós síkon tekintve ez egy egyenes egyenlete, ahol a W a meredekséget, a b az egyenes eltolását, az Xi pedig az adott értéket jelöli. Mivel a bemenetünk jelen esetben egy vektorformában meghatározott kép, ezért egy 3072 dimenziós térről beszélünk, és így a W paraméter is 3072 dimenziós vektorrá kell, hogy váljon. Továbbá több osztályba szeretnénk besorolni a képeket, így mivel egy hipersíkkal csak egy osztályt tudunk elvágni a többi példától, ezért annyi hipersík meghatározása szükséges amennyi az osztályok száma. Tehát W egy [10x3072] elemű mátrix, Xi egy [3072] elemű vektor, és b egy [10] elemű vektor (a hipersíkok számának megfelelően).

```
46  def SVM_Scores(x,W,b):
47      #calculate scores
48      scores = W.dot(x) + b
49      return scores
```

Alkalmazva az adott képekre az egyenletet megkapjuk egy „scores” nevű [10] elemű vektorba az egyes hipersíkoktól való távolságát a paramétertérben.

A számítások egyszerűsítése érdekében integrálhatjuk a „b” eltolás értékeket a W mátrixba a képen látható módon:



Kódszinten ez a következőképpen valósul meg:

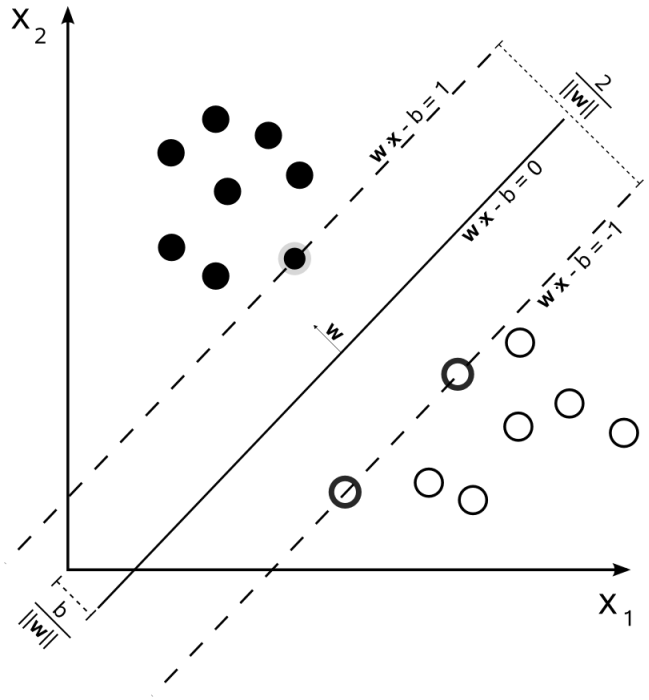
```
59  #init variables
60  X = np.append(data[0],[1])
61  B = np.reshape(B,(1,K))
62  W = np.append(W,B.T,axis=1)
63
64  def SVM_Scores_bias(x,W):
65      #calculate scores
66      scores = W.dot(x)
67      return scores
```

6.1 Veszteségszámítás

Hogy mérhetővé tegyük az egyes osztályokba való beletartozás helyességét, létre kell hoznunk egy veszteségfüggvényt. Ez a függvény fogja meghatározni a címkével jelölt adatok alapján, hogy mennyit téved az algoritmus a besorolás folyamán.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

A fenti képletet használtuk a veszteség meghatározására. Az S a pontra számított, hipersíkoktól vett távolságot mutatja. Megnézzük az összes osztályhoz tartozó síktól való távolságokat, és egyenként kivonjuk a valódi osztályhoz tartozó síktól való távolságból. Természetesen a valódi osztályra nem alkalmazzuk (j nem egyenlő Yi-vel), hogy ne kapjunk megtévesztő értékeket. Ezekhez még kapcsolhatunk egy Δ határértéket, ami szélesíti a hipersík által elválasztott terület határvonalát.



Az implementált függvény a következő:

```
104  def L_i_vectorized(x, y, W):
105    """
106    A faster half-vectorized implementation. half-vectorized
107    refers to the fact that for a single example the implementation contains
108    no for loops, but there is still one loop over the examples (outside this function)
109    """
110    delta = 1.0
111    # compute the margins for all classes in one vector operation
112    margins = np.maximum(0, s2 - s2[y] + delta)
113    # on y-th position scores[y] - scores[y] canceled and gave delta. We want
114    # to ignore the y-th position and only consider margin on max wrong class
115    margins[y] = 0
116    loss_i = np.sum(margins)
117    return loss_i
```

6.2 Optimalizációs eljárások

A tanítás abban valósul meg, hogy a már említett W mátrix értékeit, a címkézett adatok segítségével számított veszteségi értékek alapján változtatjuk. Ezzel optimalizáljuk az eljárást a megfelelő adatok szétválasztására. A veszteségfüggvény minimalizálása oldja meg a problémát, mivel minél kisebb az

egyéb osztályokba tartozó pontokra kapott érték, annál biztosabban zárhatjuk ki a nem oda való pontokat.

A minimalizálásra több példát is bemutat a kurzus. Az egyik egy véletlenszerű keresés, ahol adott számú esetben számolja ki a veszteséget, véletlen számokból generált W mátrixokra. Azt a W mátrixot fogja ideálisnak tekinteni az iteráció végén, amelyikre kiszámolva a veszteségfüggvény a legkisebb értéket adja. Ez a módszer kevésbé hatékony, és igencsak labilis a többszöri alkalmazása során szolgáltatott eredményeket tekintve.

Egy másik optimalizációs eljárás a gradiens alapú minimumkeresés. Tekintsünk egy véletlen értékekkel meghatározott W mátrixot, és határozzuk meg rá a veszteséget. Módosítsuk a W mátrixot. Ez azt jelenti, hogy meghatározhatunk rengeteg W mátrixot, amelyekre kiszámolt veszteségérték létezik. A W mátrixik által meghatározott térben, a veszteségek által kirajzolt függvényalak minimumát kell megkeresnünk. Ennek egyszerű módja, ha egy kiszámolt veszteségű W közvetlen H mértékű környezetét vizsgálva meghatározzuk, hogy merre haladva csökken a függvényérték. Ezt az analógiát követve jutunk el a függvény minimumához ideális esetben.

Az SVM módszer szintén nem a legmegfelelőbb a képek osztályozására, ugyanis ez még mindig csak pixelszintű elemzést valósít meg. Nem képes értelmezni az osztályokra jellemző sajátosságok meglétét a képeken.

7 Neurális hálózatok

A kurzus az eddig feldolgozott módszerek bemutatásával vezeti be a hallgatót a Neurális Hálozatok alapvető felépítésébe. A CIFER-10 adatbázis képeinek osztályozására bemutatja a konvolúciós neurális hálókat, amelyek már az előző módszerektől eltérően nem az egyes pixelek alapján osztályoznak, hanem a képrészleteken felfedezett pixelösszetartozások alapján. A konvolúciós eljárás lényege, hogy egy kernelt futtasson végig a kép részletein és így képes legyen kiemelni sajátosságokat amivel, a kép rendelkezhet, például ferde vonal, vagy sarokpont. A tanítás során változtatandó paraméterek így maguk a kernelek elemei.

8 Összegzés

A kurzus feldolgozása során megismerkedtem többféle osztályozásra alkalmas gépi tanulási módszerrel, a k-legközelebbi szomszéddal, az SVM algoritmussal. Python nyelven lehetőségem volt ezeket a

módszereket kipróbálni és elemezni. A Stanford egyetem CS231n kurzusa úgy állította össze a tananyagot, hogy e tanulási módszerek előfutárai legyenek a neurális hálóknak, és elméleti alapként szolgáljanak hozzá. A továbbiakban szeretném folytatni a kutatást a neurális hálók irányába, és behatóan megismerkedni a tanitási eljárásokkal, valamint az alkalmazásukkal.