**Egyszerű regresszió**

Egy független és egy függő változó esetén becsüljük meg a kapcsolatukat egy lineáris függvénnyel.

**Példa probléma: házárak**

Adatokkal rendelkezünk eladott házakról: terület és eladási ár. Meg szeretnénk jósolni házunk eladási árát (függő változó). Ismerjük annak területét (független változó). Generáljunk véletlenszerű adatokat.

**Probléma: találjunk egy *egyenest*, amely a legjobban illeszkedik az adatokhoz**

Egyenes ~ lineáris függvény: y = ax + b

Meg kell találnunk az a és b paraméterek értékét.

**Az adatokatól való távolság mérése (azaz a hiba, vagy veszteség)**

Négyzetes hiba (Mean Squared Error, MSE)

A képen Betűtípus, szöveg, fehér, diagram látható

Automatikusan generált leírás

(m az oktatási adatok elemeinek száma, n a bemeneti változók száma, ami most 1)

**Minimalizáljuk a hibát**

Keressünk a és b értékeket, amelyek minimalizálják a hibafüggvényt. Ez egy optimalizációs probléma, ahol a és b a döntési változók, és a cél az MSE(a,b) minimalizálása. A probléma jobb megértéséhez nézzük meg a hibafüggvény grafikonját különböző a,b értékek esetén.

**Használhatunk okosabb keresési módszert, mint a brute-force.**

A LP és MILP nem jöhet szóba, mivel a célfüggvény nem lineáris.

Használhatnánk genetikus algoritmust vagy más kifinomult metaheurisztikát, de ez túlzás lenne. Ez csak egy kvadratikus függvény, csak 2 változóval.

Ha csak egy változó lenne, könnyen megoldhatnánk a derivált egyenletét 0-ra. Több változóval bonyolultabb, de még mindig lehetséges, lásd a [normálegyenlet módszerét](https://www.datacamp.com/tutorial/tutorial-normal-equation-for-linear-regression).

Ehelyett egyszerűbb, iteratív numerikus módszert használunk, a **gradiens módszert**. Ez a parciális deriváltakat használja az irányított kereséshez, így sokkal kevesebb lépést igényel, mint a brute-force keresés. Ez a módszer más gépi tanulási problémák esetén is hasznos lesz, míg a normálegyenlet módszer csak lineáris regresszióra alkalmas. A parciális deriváltak bármely ponton megadják a lejtés meredekségét (gradiens) azon a helyen. Ha nagy, messze vagyunk a minimumtól.

*A képen szöveg, Betűtípus, kézírás, diagram látható

Automatikusan generált leírás*

*A konstans faktor irreleváns az optimalizációhoz. A 2-t gyakran kihagyják, de a jó normalizáláshoz.*

Induljunk bármely (a,b) pontból, és frissítsük a pozíciót a gradiens alapján.

*Ha a gradiens pozitív, csökkentenünk kell a változót, ha negatív, növelnünk kell.*

A képen szöveg, Betűtípus, kézírás, fehér látható

Automatikusan generált leírás

α a tanulási ráta, vagy lépésméret paraméter. Ha túl kicsi, a konvergencia lassú lesz, de ha túl nagy, akár divergens is lehet.

**Regresszió több jellemzővel**

Egyszerű esetben 1 bemeneti változónk volt x , és ezt használtuk a y érték előrejelzéséhez.

Az előrejelző egy egyenes volt az y=ax+b egyenlettel. Meg kellett találnunk az a és b értékeket.

*Több kimeneti változó előrejelzése könnyen elvégezhető, ha külön regressziót végzünk mindegyikre.*

Mi van, ha különböző típusú adatokat (jellemzőket) ismerünk a tárgyakról? Például nemcsak a házak területét ismerjük, hanem a szobák számát, az emeletek számát, a házak korát és a városközponttól való távolságot is. Több jellemző alapján történő előrejelzés azt jelenti, hogy a bemeneti adat nem csak egy szám x , hanem egy számsorozat \mathbf{x} .

A lineáris regresszió megpróbálja a kimeneti változót lineáris függvénnyel becsülni, ami a bemeneti változók lineáris kombinációja:

A képen szöveg, Betűtípus, fehér, tipográfia látható

Automatikusan generált leírás

A 2 szám, a és b helyett most n+1 szám értékét kell meghatároznunk: n együtthatót és a y -tengelymetszetet.

Ettől eltekintve a módszer ugyanaz, csak több parciális deriváltunk van.

A hibafüggvény és a gradiens módszer megvalósítása a többváltozós esetben az olvasóra marad. Itt a scikit-learn könyvtár megoldása kerül bemutatásra.

**Adatok beolvasása**

Adatforrás: [Kaggle](https://www.kaggle.com/datasets/harlfoxem/housesalesprediction)

**Jellemzők mérnöki munkája**

Új jellemzők létrehozása a bemeneti adatok alapján, meglévő jellemzők kombinálásával vagy átalakításával.

Használható polinomiális regresszióhoz is, ha egy jellemzőt bizonyos hatványra emelünk.

De vigyázat, túl sok jellemző, vagy alacsony korrelációjú jellemzők túltanuláshoz vezethetnek, valamint elégtelen oktatási példákhoz.

**Osztályozási probléma**

Mi van, ha az előrejelzési cél változó nem folyamatos, hanem bináris? Igen vagy nem?

Vagy értéket vehet egy véges, előre meghatározott kategóriákból álló halmazból? Ez az általánosabb eset átalakítható bináris osztályozási problémák sorozatává. Erről később bővebben.

Az osztályok lehetnek ordinálisak vagy kategóriák:

* Ordinális értékek: A kategóriáknak van sorrendje, mint például A, B, C osztályzat; 5 csillagos skála; méretosztályok. Közel áll a regressziós problémához, de ha az osztályok száma kicsi, általában jobb osztályozásként kezelni.
* Kategóriális értékek: A kategóriáknak nincs természetes sorrendjük, például színek, vércsoport.

**Próbáljuk használni a lineáris regressziót**

Próbálhatjuk használni a korábban tanult lineáris regressziót. Kit érdekel, ha az y értékek mind 0 vagy 1!?

A folyamatosan előrejelzett érték helyett 1 -et adunk ki, ha , és 0 -át, ha y<0.5 .

**A lineáris illeszkedés nem ideális**

Azok a részek, ahol y<0 vagy 1<y téves hibákat tartalmaznak.

A kiugró értékek rossz pozícióba torzíthatják a lineáris függvényt.

**Logisztikus regresszió**

Egy lineáris függvény helyett olyan függvényre van szükségünk, amely jobban illeszkedik ezekhez a 0-1 értékekhez.

Használhatnánk a Heaviside lépésfüggvényt, amely 0, ha x<0 és 1, ha x≥ 0 , de ez nem differenciálható, így nem tudnánk gradient descent-et használni az illesztéshez.

Itt jön a képbe a logisztikus (vagy szigmoid) függvény.

A képen sor, diagram, képernyőkép, Diagram látható

Automatikusan generált leírás

Játssz a paraméterekkel [a Desmoson](https://www.desmos.com/calculator/mju5rfiyju)

**Más hibafüggvényre is szükség van**

A szigmoid függvény előnye, hogy jobban illeszkedik, mint egy egyenes. De van egy probléma.

Ha a hibát a MSE (Mean Square Error) függvénnyel mérjük, akkor egy ráncos, nem konvex felületet kapunk sok lokális minimummal. A gradient descent könnyen beragadhat egy lokális optimumba.

A képen diagram, sor, Diagram látható

Automatikusan generált leírás

Ezért a Maximum Likelihood Estimation (MLE) módszert használjuk helyette. A matematikai magyarázata túlmutat a mi kereteinken, statisztikán és valószínűségszámításon alapul, és a minimalizálandó függvény így néz ki, ahol p a szigmoid függvény által előrejelzett érték:

A képen Betűtípus, szöveg, fehér, kézírás látható

Automatikusan generált leírás

Mivel yi vagy 0 vagy 1, az i -edik elem hibája vagy -\log(1-p) , vagy -\log(p) . Tehát a hiba nagyon kicsi, ha az előrejelzett érték közel van a tényleges értékhez, és nagyon nagy, ha messze van tőle.

A képen diagram, sor, Diagram, szöveg látható

Automatikusan generált leírás

Ez a hibafüggvény konvex, így sokkal könnyebb optimalizálni. Ez gradient descent-tel elvégezhető, hasonlóan a lineáris regresszióhoz. Itt az scikit-learn könyvtárban már megvalósított módszereket nézzük meg.

**Példa: Vásárol-e az ember egy SUV-t?**

**Jellemzők kiválasztása**

Használjuk-e a nemet bemeneti változóként? Segít-e előrejelezni, hogy egy személy vásárol-e egy SUV-t vagy sem?

Hogyan határozhatjuk meg?

**Support Vector Machines (SVMs)**

Egy másik módszer az osztályozásra (vagy regresszióra).

Geometriai intuíció alapján, nem statisztikai elemzés alapján.

Ötlet: Keressünk egy határt az osztályok között, amely a *legmesszebb* van mindkét (vagy több) kategóriától. (Maximalizálja a margót.)

A távolságot a kategória legközelebbi pontjához mérjük. Az adatok, amelyek jól a kategória régiójában vannak, nem befolyásolják a határ helyzetét. (És mivel nem használják őket a számításban, a módszer is hatékonyabb.)

Lehet, hogy nincs egyértelmű határ az osztályok között (mint az előző példában), ilyenkor bevezetünk egy toleranciát, amely lehetővé teszi, hogy a pontok a rossz oldalon legyenek.

További információ az SVM-ekről: [Magyar cikk](https://sajozsattila.home.blog/2019/07/02/support-vector-machine/) az SVM-ek matematikai hátteréről. Az SVM-ek általában jobbak, mint a logisztikus regresszió a strukturálatlan adatok esetében, mint például képek, szövegek. Nehéz előre megmondani, melyik módszer jobb egy adott probléma esetén, ezért érdemes mindkettőt kipróbálni.

A pontosabb osztályok közötti elválasztás érdekében a határnak egy görbének kell lennie, nem pedig egy egyenesnek. (Egy görbült hipersík helyett lineáris hipersík.)

Ez könnyebben elérhető, ha a bemeneti adatokat nemlineáris függvénnyel (kernel függvény) transzformáljuk, majd az átalakított értékeken lineáris határt illesztünk.

Valójában nem is szükségesek az átalakított értékek, csak a távolságuk. Ezt használja ki az úgynevezett Kernel Trick a hatékonyabb számításhoz. A módszert jól szemlélteti ez a [videó a Visually Explained-től](https://www.youtube.com/embed/Q7vT0--5VII).

**Többosztályos (multinomiális) osztályozás**

Lehet több mint 2 osztály, amelybe az elemeket be kell sorolni.

Néhány osztályozási módszer ezzel is megbirkózik, néhány csak bináris osztályozásra képes. De a többosztályos problémák sorozatos bináris problémákon keresztül is megoldhatók.

**Átalakítás binárissá**

**One-vs-one**

Minden osztály-párhoz egy osztályozót tanítunk.

Az előrejelzés a klasszifikátorok eredményeinek szavazatai alapján történik, és a megjósolt osztály az, amelyik a legtöbb szavazatot kapja.

Az sklearn.svm.SVC esetén állítsuk be a decision\_function\_shape = "ovo" értéket.

**One-vs-rest**

Minden osztályhoz c egy osztályozót tanítunk, ahol az eredeti osztályérték y\_c egy bináris függő változóvá van konvertálva: 

Az előrejelzés azzal történik, hogy kiválasztjuk annak a klasszifikátornak az osztályát, amelyiknek a legmagasabb az  előrejelzési valószínűsége.

Az sklearn.svm.SVC esetén állítsuk be a decision\_function\_shape = "ovr" értéket (ez az alapértelmezett). [Megjegyzés](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html): belsőleg még mindig "ovo" módszert használ a bináris SVM modellek illesztéséhez, és az "ovr" mátrix ezekből számítódik ki.

**Klaszterezés**

A klaszterezésben az a cél, hogy az adatokat diszjunkt csoportokba (klaszterekbe) osszuk.

Ez hasonló az osztályozási problémához, de nincs előzetes információnk a csoportok jelentéséről. Csak azt szeretnénk, ha a hasonló adatokat egy csoportba sorolnánk.

Ezért ez a probléma felügyelt tanulást igényel, mivel nincs címkézett tanító adatunk. (Vagy talán van, de azt szeretnénk megtudni, hogy van-e jobb csoportosítás.)

**k-means klaszterezés**

Kezdjük egy egyszerű esettel: 2 dimenziós numerikus adataink vannak, és tudjuk, hogy 3 klasztert szeretnénk.

**Intuíció**

Hogyan közelítsük meg? Az osztályozásban vonalakat használtunk a csoportok elválasztására, és a vonalak helyzetét úgy állítottuk be, hogy a csoportok jobban illeszkedjenek a címkéikhez.

Itt nincsenek címkéink. Bárhol rajzolhatunk vonalakat, de honnan tudnánk, melyik csoportosítás jobb?

Ötlet: Az elválasztók helyett határozzuk meg a csoportok középpontjait (centroidok), és rendeljük az adatokat a legközelebbi csoporthoz.

Következő lépés: Állítsuk be a centroidok helyzetét.

Számoljuk ki a csoportok tényleges középpontjait, és tegyük őket az új centroidokká. Ezek valójában a csoportok átlaga, innen ered a k-means algoritmus neve.

A centroidok az optimális helyzetükre konvergálnak, amely minimalizálja az átlagos távolságot a csoporttagoktól. Ez azonban csak egy lokális optimum, amely az induló centroid értékektől függ.

Egy megoldás az, hogy a folyamatot különböző kezdőpozíciókkal ismételjük meg.

De hogyan válasszuk ki az induló centroidokat? Korábban kézzel választottuk ki őket, de ha a dimenziók száma magas, nehéz jó értékeket kiválasztani.

Ötlet: Véletlenszerűen válasszunk ki k adatpontot induló centroidként.

Szóval, mi van, ha nem tudjuk, hány klaszternek kell lennie?

Próbálkozzunk különböző k-értékekkel, és figyeljük meg az inercia csökkenését. Ezt nevezik *könyök módszernek*.

**Mesterséges Neurális Hálózatok (ANNs)**

Az emberek és állatok agyában található neurális hálózatok inspirálták. Ezek neuronokból állnak, amelyek elektromos jeleket továbbítanak.

A képen szöveg, diagram, térkép látható

Automatikusan generált leírás

Bár a módszer természet inspirálta, sok mindent még mindig nem tudunk az agy működéséről. Az ANNs hasznos matematikai eszközök, de nem az a céljuk, hogy pontosan utánozzák a természetet.

Egyetlen mesterséges neuron (vagy perceptron) hasonló modellje van, mint amit a logisztikus regresszióban használtunk.

A képen diagram, sor, szöveg, képernyőkép látható

Automatikusan generált leírás

Több neuron egyidejű használatával különböző kombinációkat taníthatunk a bemeneti jellemzőkből. Ahelyett, hogy manuálisan próbálnánk kombinálni a jellemzőket, az ANN megtanulja a jellemzők közötti kapcsolatokat. A súlyok megváltoztatásával egy neuron érzékenyebb lesz bizonyos jellemzőkre.

A neuronok rétegekbe csoportosulnak az ANN-ben. Van egy bemeneti réteg, 1 vagy több rejtett (vagy középső) réteg, és egy kimeneti réteg. A hagyományos előrecsatolt (vagy szekvenciális) hálózatokban az egyik réteg kimenete a következő réteg bemenete. (Vannak más lehetséges struktúrák is, később megnézzük őket.)

A képen képernyőkép, szöveg, minta, tervezés látható

Automatikusan generált leírás

Több réteg használatával az ANN egyre bonyolultabb jellemzőket tud építeni egyszerűek kombinálásával. Ez látható a fenti ábrán egy speciális típusú ANN-nél, egy konvolúciós neurális hálózatnál (CNN).

**Előrecsatolás (inferencia)**

A legtöbb ANN-ben minden réteg minden csomópontja kapcsolódik a szomszédos rétegek minden csomópontjához. Ezeket sűrű rétegeknek nevezik. Így minden neuron bemeneti vektora ugyanaz, csak a súlyuk különbözhet.

A súlyozott összeg és az aktivációs függvény kiszámításával minden neuronra rétegenként, kiszámíthatjuk egy (már betanított) ANN kimenetét. Ezt nevezzük előrecsatolásnak vagy inferenciának.

A regressziós modellekben láttuk, hogyan számolhatjuk ki az előrejelzett kimenetet a vektorok skalárszorzatával. Itt a bemenet is egy 1D vektor, de több súlyvektorunk van - minden neuronhoz egy vektor egy rétegen. A súlyvektorok mátrixba szervezésével a neuronok skalárszorzata egy rétegen egyetlen mátrixszorzással kiszámítható. A GPU-k nagyon jók ebben, mivel a 3D grafikus transzformációkat is mátrixszorzásokkal végzik.

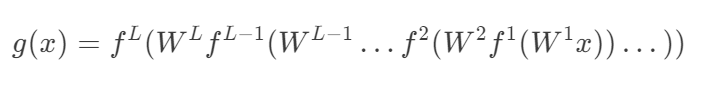
De hogyan tanítjuk meg a hálózatot?

Hogyan számoljuk ki a hibát? Honnan tudjuk, melyik réteg okozza a hibát? Ha több rétegünk van, hogyan frissítjük a súlyokat?

**Hátracsatolás (backpropagation)**

Egy lehetséges tanítási algoritmus a hátracsatolás.

Lényegében az előrecsatolás számítása egy sor mátrixszorzás és függvénykompozíció:



A hiba/költség/veszteség a tényleges y értékek különbsége az előrejelzett p=g(x) értékektől a tanítókészletben. Ez nem újdonság, használhatunk MSE-t, MLE-t vagy más veszteségfüggvényeket, az output típusától függően.

Ahhoz, hogy megtudjuk, egy adott súlyt w hogyan kellene megváltoztatni, szükségünk van a részleges deriváltra A képen szöveg, Betűtípus, tipográfia, kalligráfia látható

Automatikusan generált leírás

Röviden, kiszámíthatjuk a deriváltakat minden réteghez visszafelé haladva, és a deriváltakat egy rétegnél használjuk a megelőző réteg deriváltjának kiszámításához. Ez is egy sor mátrixszorzás és elemi szorzás.

A részletes magyarázathoz olvassa el [Michael Nielsen Neural Networks and Deep Learning című könyvének 2. fejezetét](http://neuralnetworksanddeeplearning.com/chap2.html). [Játéktér!](http://playground.tensorflow.org/)

**TensorFlow és Keras használata**

A [TensorFlow](https://www.tensorflow.org/) a Google által létrehozott nyílt forráskódú gépi tanulási könyvtár.

*A Tensor a vektorok és mátrixok általánosítása. Egy skalár (egy szám) egy 0. rendű tensor, egy 1D vektor egy 1. rendű tensor, egy 2D mátrix egy 2. rendű tensor, stb.*

A [Keras](https://keras.io/) egy magas szintű Python API az ANN-ekkel való munkához. Ez a TensorFlow Python csomag része lett.

Manapság a szigmoid függvény helyett egy másik aktivációs függvényt preferálnak.

A ReLU (Rectified Linear Unit) jobban teljesít a legtöbb esetben, és könnyebben kiszámítható:

A képen Betűtípus, kézírás, kalligráfia, fehér látható

Automatikusan generált leírás

Ez nem differenciálható 0-nál, de a derivált lehet 0 vagy 1 (mivel 0 x<0 esetén, és 1 x>0 esetén).

Van néhány módosított változata is:

A képen szöveg, diagram, Betűtípus, szám látható

Automatikusan generált leírás

A modell 10 számot ad ki 0 és 1 között. De ezek nem valódi valószínűségek, mivel nem adódnak össze 1-re. (Ha ez címkézési probléma lenne, ez megfelelő modell lenne.)

Eloszthatnánk minden értéket az összeggel, akkor az összeg 1 lenne. Ez egy lineáris normalizáció a kimenetre. yi→yi∑j=1nyjy\_i \rightarrow \frac{y\_i}{\sum\_{j=1}^n y\_j}yi​→∑j=1n​yj​yi​​

A képen Betűtípus, diagram, vázlat, fehér látható

Automatikusan generált leírás

Ehelyett egy nem-lineáris normalizációs módszert használnak a legtöbb esetben, amelyet softmaxnak neveznek: yi→eyi∑j=1neyjy\_i \rightarrow \frac{e^{y\_i}}{\sum\_{j=1}^n e^{y\_j}}yi​→∑j=1n​eyj​eyi​​

A képen Betűtípus, kézírás, vázlat, diagram látható

Automatikusan generált leírás

Ez is 1-re adódó összeget eredményez, de fő előnye, hogy szélesebb értéktartományon használható, nem csak 0-1 értékeken.

Tehát használhatjuk a softmax függvényt az aktivációs függvényként a kimeneti rétegen a szigmoid függvény helyett.

Bár a fenti modell jól működik, ugyanez elérhető egy másik módszerrel is, amely hatékonyabb és kevesebb numerikus hibát okoz.

Ahelyett, hogy a softmaxot használnánk az értékeken a kimeneti rétegen, mondjuk a veszteségfüggvénynek, hogy használjon logiteket a valószínűségek helyett.

*Logitek a z értékek a szigmoid függvényben \frac{1}{1+e^{-z}}*

A képen szöveg, Betűtípus, szimbólum, képernyőkép látható

Automatikusan generált leírás

**PyTorch használata**

A [PyTorch](https://pytorch.org) a Facebook (most Meta) által készített nyílt forráskódú gépi tanulási keretrendszer, amely a [Torch](http://torch.ch/) könyvtáron alapul. A Torch egy Lua script nyelven alapuló API-t használt, míg a PyTorch Python és C++ API-t kínál.

A TensorFlow népszerűbb, különösen a képfelismerési alkalmazásoknál, míg a PyTorch gyakran használt NLP - Természetes Nyelvfeldolgozási feladatokhoz.