OCR

Feladat: Írjunk egy osztályozó algoritmust, mely kézzel írt számjegyeket ismer fel, a számjegyekről készült beszkennelt képek alapján.

1. Bevezető

- A fenti feladat egy **gépi tanulási** probléma, mely a **felügyelt tanulás** kategóriájába tartozik (rendelkezésükre áll egy adathalmaz, melyben a kézzel írott számjegyek képei fel vannak címkézve, azaz mindegyik képről tudjuk, hogy milyen számjegyet reprezentál).
- A feladat egy **osztályozási probléma** (**classification**), melynek során egy adott képet be kell sorolni a $\{0, 1, 2, \ldots, 9\}$ kategóriák valamelyikébe.

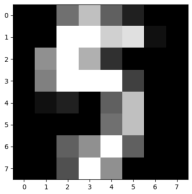
2. Az adathalmazról

A feladat megoldásához rendelkezésünkre áll egy felcímkézett adathalmaz, melyben kézzel írott számjegyek képei találhatók, a következő formában: minden egyes adatot egy $8 \cdot 8 + 1 = 65$ dimenziós vektor formájában tárolunk, ahol a vektor első 64 komponense adja meg a beszkennelt képet, a 65-ik szám pedig a képen található számjegy.

Például, a tanulási adathalmazt tartalmazó *optdigits.tra* állomány 14. sora a

$$\begin{bmatrix} 0 & 0 & 7 & 12 & 6 & 2 & 0 & 0 \\ 0 & 0 & 16 & 16 & 13 & 14 & 1 & 0 \\ 0 & 9 & 16 & 11 & 3 & 0 & 0 & 0 \\ 0 & 8 & 16 & 16 & 16 & 4 & 0 & 0 \\ 0 & 1 & 2 & 0 & 6 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 12 & 0 & 0 \\ 0 & 0 & 6 & 9 & 16 & 6 & 0 & 0 \\ 0 & 0 & 5 & 16 & 9 & 0 & 0 & 0 \end{bmatrix}$$

8×8-as bittérképnek felel meg, mely az 5-ös számot ábrázolja:



A fenti tárolási mód azért ennyire specifikus, mert az adathalmazban levő adatok már átestek egy előfeldolgozáson, így az osztályozó algoritmusunk sokkal jobb eredményeket érhet el, mintha az eredeti, nyers képekkel dolgoznánk. Az előfeldolgozásról pontosabban az *optdigit.names* állományban olvashattok.

3. A feladat

A következőkben, a gépi tanulás két alapvető módszeréről, a **kNN** és a **centroid** módszerekről olvashattok. Ezek - egyszerűségük ellenére - széles körben alkalmazott technikák[1], így ezeket általánosan, tetszőleges adathalmaz esetén mutatjuk be.

Legyen a címkézett adatok halmaza

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in X, y_i \in Y, i \in \{1, 2, \ldots, \ell\}\},$$

ahol $\mathbf{x}_i$ egy tetszőleges adatot, y_i pedig az illető adat címkéjét jelöli, $X \subseteq \mathbb{R}^d$, Y a címkék halmaza, ℓ pedig az adathalmazban levő adatok száma. (A kézzel írt számjegyek esetén $d = 64$, azaz az $\mathbf{x}_i$ adatok 64-dimenziós vektorok, $Y = \{0, 1, \ldots, 9\}$ pedig a lehetséges számjegyek halmaza.)

Ekkor tehát feltételezhetjük, hogy létezik egy

$$f : X \to Y$$

leképezés, mely minden tetszőleges $\mathbf{x}_i$ adathoz hozzárendeli az adat valódi osztályát/címkéjét.

Az osztályozási algoritmusunk célja a lehető legjobban megközelíteni az $f : X \to Y$ függvényt egy $\hat{f}$ leképezéssel. Az f függvény értékeit csak bizonyos pontokban ismerjük, ezeket a pontokat tartalmazza a tanulási adathalmaz (a mi esetünkben az *optdigits.tra*), vagyis tudjuk, hogy

$$f(\mathbf{x}_i) = y_i, \quad \forall i \in \{1, 2, \ldots, m\},$$

ahol m a tanulási adatok számát jelöli.

A továbbiakban az f függvény becslésére alkalmazható módszerekről lesz szó.

4. kNN

A **kNN** (k-nearest neighbors, magyarul k legközelebbi szomszéd) azon az egyszerű alapelven működik, hogy ha meg akarjuk becsülni egy ismeretlen adat címkéjét (pl. meg akarjuk mondani, hogy milyen számjegyet ábrázol egy 8×8-as szürkeárnyalatú kép), akkor keressük meg a **hozzá legjobban hasonlító** k darab adatot, majd vizsgáljuk meg ezen adatok címkéit. Ekkor, az ismeretlen adat osztályának azt a címkét választjuk, amelyikből a legtöbb van. Például, ha $k = 5$, és a felcímkézendő képhez a legjobban hasonlító adatok rendre az 1, 1, 7, 1, 7 címkével rendelkeznek, akkor a felcímkézendő adat az 1-es címkét kapja.

A kérdés csak az, hogy hogyan tudjuk eldönteni, hogy két adat mennyire **hasonlít** egymáshoz? Az adatok közti hasonlóságot illetve különbséget különböző **metrikák** segítségével mérhetjük. Mivel a mi esetünkben az adatok vektor formájában vannak eltárolva, többek között az alábbi metrikákat alkalmazhatjuk:

- **euklideszi metrika**: két tetszőleges adat esetén megadja a két vektor csúcspontjának euklideszi távolságát. Képlete:
$$d_2(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - z_i)^2}.$$
- **koszinusz-hasonlóság**: két tetszőleges adat esetén megadja a két vektor térbeli szögének koszinuszát. Képlete:
$$sim(\mathbf{x}, \mathbf{z}) = \cos(\theta(\mathbf{x}, \mathbf{z})) = \frac{\langle \mathbf{x}, \mathbf{z} \rangle}{\|\mathbf{x}\|_2 \cdot \|\mathbf{z}\|_2},$$
ahol $\langle \mathbf{x}, \mathbf{z} \rangle$ az $\mathbf{x}$ és $\mathbf{z}$ vektorok skalárszorzatát jelöli.

Megjegyzés: az euklideszi távolságot és koszinusz-hasonlóságot azért tudjuk alkalmazni, mert az adatok már átestek egy bizonyos normalizációs előfeldolgozáson. Például, az euklideszi távolság nagyon

[1] A kNN felhasználási területeiről itt olvashattok

érzékeny a képek skálázására vagy eltolására. Ezt a problémát az előfeldolgozás kiküszöböli. Ugyanakkor, ebből az is kiderül, hogy a megfelelő távolságfüggvény (metrika) kiválasztása az illető gépi tanulási feladattól függ.

A kNN módszer által meghatározott leképezésünk tehát a következőképpen írható fel:

$$\hat{f}(\mathbf{x}) = \arg\max_{y \in Y} \sum_{\mathbf{z} \in N_k(\mathbf{x})} I(f(\mathbf{z}), y),$$

ahol $N_k(\mathbf{x})$ az $\mathbf{x}$ adat k legközelebbi szomszédjainak halmaza a tanulási adatok közül, I pedig Kronecker-delta függvény, azaz $I(a, b) = 1$, ha $a = b$, különben 0.

5. Centroid módszer

A centroid módszer alapgondolata, hogy minden egyes osztályhoz egy úgynevezett **osztályprototípust** rendelünk, amely az illető osztály egyedeinek átlagát képezi. A mi esetünkben, mivel az adathalmaz elemei vektorok, egy-egy osztály prototípusa az osztály egyedeiből álló **pontfelhő közepe/ centruma /centroidja** lesz, melyet az osztályban levő pontok koordinátáinak átlagolásával határozhatunk meg:

$$\mathbf{c_i} = \frac{1}{|\{(\mathbf{x}, y) \in \mathcal{D} | f(\mathbf{x}) = \mathbf{i}\}|} \sum_{\{(\mathbf{x}, y) \in \mathcal{D} | f(\mathbf{x}) = \mathbf{i}\}} \mathbf{x}.$$

Az $\hat{f}$ leképezést ekkor úgy határozzuk meg, hogy rendelje azt a címkét egy adott felcímkézendő ponthoz, amely osztály prototípusa (centroidja) a legközelebb található:

$$\hat{f}(\mathbf{x}) = \arg\min_{\mathbf{i} \in Y} d(\mathbf{x}, \mathbf{c_i}).$$

Ebben az esetben is, a legközelebbi centrum meghatározásához használhatjuk az euklideszi távolságot.

6. Az osztályozó tanítása

Gépi tanulás során egy gyakran alkalmazott eljárás a rendelkezésre álló adatokat felbontani tanulási- és teszt-adathalmazokra. Az általunk kiválasztott modellt a tanulási adathalmazon tanítjuk, majd a teljesítményét a teszt adathalmazon (és néha a tanulási halmazon is) leellenőrizzük.

Egy gépi tanulásos osztályozó algoritmus tanítása abból áll, hogy a tanulási halmazból kinyerjük azokat az adatokat, amelyekre szükségünk van a modellünk felépítéséhez és a későbbi, felcímkézetlen adatok kiértékeléséhez; ezeket az információkat lementjük a modellünkbe (ezt a folyamatot nevezik a **modell betanításának**). Mivel a **knn** és a **centroid** nagyon egyszerű módszerek, az osztályozó tanítási fázisa is triviális lesz (ellentétben például a Spamszűrés feladattal, ahol a tanítási folyamat során a szavak paramétereit kellett megbecsülni).

A **knn** esetében, az osztályozó tanítása pusztán a bemeneti adatok eltárolását jelenti (hiszen a kNN modell direkt ezeket az adatokat használja). A **centroid** esetén, a tanítási folyamatot a centrumok meghatározása képezi, így a különböző osztályok jellemzőit egy általánosított, absztrakt formában tároljuk.

7. Az osztályozó kiértékelése

Az osztályozó algoritmus helyességét a tanulási hiba és teszthiba kiszámításának segítségével mérhetjük. Jelöljük $\mathcal{D}_{\text{tr}}$-rel a tanulási adathalmazt, illetve $\mathcal{D}_{\text{te}}$-vel a teszt halmazt.

7.1. Tanulási hiba

A tanulási hiba azt méri, hogy az osztályozó mennyire képes „megtanulni” a tanulási adatok jellemzőit. Ebben az esetben a tanítási fázis, majd a tesztelési fázis is a $\mathcal{D}_{\text{tr}}$ halmazon történik.

A hibát úgy mérjük, hogy megnézzük, az osztályozó hány bemenetre nem a helyes címkét prediktálja. Az eddigi jelölést követve, $f(\mathbf{x}_i) = y_i$ az $\mathbf{x}_i$ tényleges címkéje, és jelölje $\hat{f}(\mathbf{x}_i)$ az osztályozó módszer által prediktált címkét. Ekkor a tanulási hiba a következőképpen írható fel:

$$\frac{1}{m} \sum_{\mathbf{x}_i \in \mathcal{D}_{\text{tr}}} [f(\mathbf{x}_i) \neq \hat{f}(\mathbf{x}_i)],$$

ahol m a tanuló adatok számossága, $[\cdot]$ pedig 1 (igaz), ha a benne levő feltétel teljesül, különben 0 (hamis).

7.2. Teszt hiba

A teszt hiba számolása esetén a tanulási és teszt adathalmazok különböznek: a tanulási halmazon betanítjuk a rendszert, majd a teszt halmazon ellenőrizzük a kapott modell helyességét. Ha t jelöli a teszt halmaz méretét, $t = |\mathcal{D}_{\text{te}}|$, akkor a teszt hiba a következőképpen írható fel:

$$\frac{1}{t} \sum_{\mathbf{x}_i \in \mathcal{D}_{\text{te}}} [f(\mathbf{x}_i) \neq \hat{f}(\mathbf{x}_i)].$$

8. Legkisebb négyzetek módszere (lineáris regresszió)

A lineáris regresszió, vagy legkisebb négyzetek módszere alapértelmezetten egy bináris osztályozási feladat, vagyis két osztály (2 különböző címke) esetén működik. Természetesen kiterjeszthető többosztályos osztályozási feladatra is, de ez a kiterjesztés nem triviális, ezért itt csak a bináris esettel foglalkozunk.

Legyen most Y összesen 2 címkét tartalmazó halmaz, $Y = \{-1, +1\}$; a $+1$ címkéjű adatokat pozitív példáknak/adatoknak, a -1 címkével rendelkezőket pedig negatív példáknak nevezzük. A döntési függvényünk egy lineáris leképezés lesz, egy hipersík (= az egyenes általánosítása n-dimenziós térben; azaz az n-dimenziós tér egy $(n-1)$-dimenziós hipersíkja, amely két részre osztja a teret), amely szét fogja választani a pozitív példákat a negatívaktól,

$$\hat{f}(\mathbf{x}) = sgn(\mathbf{w}^T\mathbf{x} + b),$$

ahol $\mathbf{w}^T\mathbf{x}$ a $\mathbf{w}$ és $\mathbf{x}$ vektorok skalárszorzatát jelöli, $\mathbf{w}^T$ a $\mathbf{w}$ vektor transzponáltja, b pedig egy valós skalár; a sgn a szignum, azaz az előjelfüggvény.[2]

A kérdés tehát az, hogy hogyan találjunk egy olyan hipersíkot ($\mathbf{w}$ és b paramétereket), amely szétválasztja az adatainkat? Itt jön be a legkisebb négyzetek módszere: próbáljuk meg **minimalizálni a prediktált és a tényleges címkék közötti különbség négyzetét**. Ekkor az optimalizálási feladatunk a következő lesz:

$$\min_{\mathbf{w},b} \sum_{i=1}^{\ell} (\mathbf{w}^T\mathbf{x}_i + b - y_i)^2.$$

Ahhoz, hogy jelöléseiben egyszerűsítsük a feladatot, a következő módosítást végezzük: a $\mathbf{w}$ vektort kiegészítjük a b paraméterrel, azaz d-dimenziós helyett $(d+1)$-dimenziós vektort kapunk, melynek utolsó komponense a b érték lesz. Hasonlóképpen, az $\mathbf{x}$ adatainkat kiegészítjük egy új, $(d+1)$-edik értékkel, egy konstans 1-essel.[3] Emiatt a továbbiakban a $\mathbf{w}$ és $\mathbf{x}$ alatt az új, $(d+1)$-dimenziós vektorokat értjük, így az optimalizálási feladatunk a következőképpen alakul:

$$\min_{\mathbf{w}} \sum_{i=1}^{\ell} (\mathbf{w}^T\mathbf{x}_i - y_i)^2.$$

A feladat tehát egy többváltozós függvény globális minimumpontjának meghatározása, mely pontosan meg fogja adni az optimális $\mathbf{w}$ vektort, azaz annak a hipersíknak a paramétereit, amit mi keresünk.

[2] Látható, hogy a ± 1 címkék nem véletlenül lettek így kiválasztva. Tudjuk, hogy a hipersík két részre osztja a teret; ha behelyettesítünk egy tetszőleges térbeli pontot a hipersík egyenletébe, akkor az negatív értéket térít vissza, ha a pont a sík egyik oldalán levő féltérben helyezkedik el, illetve pozitívat, ha a másikban (és nullát, ha a pont a hipersík része). Így, ha sikerül találnunk egy hipersíkot, amely szétválasztja a pozitív adatokat a negatívaktól, akkor egyszerűen behelyettesítve egy pontot a hipersík egyenletébe és megvizsgálva a visszakapott érték előjelét, meg tudjuk mondani a pont címkéjét, azaz, hogy melyik osztályba tartozik.

[3] Belátható, hogy $\mathbf{w}_{\text{új}}^T\mathbf{x}_{\text{új}} = \mathbf{w}^T\mathbf{x} + b$ lesz.

8.1. Lineáris regresszió egzakt módszerrel

Az egzakt módszer lényege, hogy az

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{\ell} (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

hibafüggvény globális minimumpontját a lokális minimumpontok között keressük. A lokális minimumpontok meghatározására pedig egy egyszerű feltételünk van: ezekben a pontokban a derivált (pontosabban a gradiensvektor) értéke 0.

Ezzel az információval tehát a lokális minimumpontokat egyértelműen, egy explicit képlettel meghatározhatjuk, egy pici számolás segítségével. Az alábbbiakban levezetjük ezt a képletet:

Helyezzük az $\mathbf{x}_i$ vektorokat egy $\mathbf{X}$ mátrix soraiba ($\mathbf{X}$ egy $\ell \times (d+1)$ méretű mátrix lesz), illetve az y_i címkéket az $\mathbf{y}$ oszlopvektorba. Ekkor a fenti minimalizálandó kifejezés felírható a következőképpen:

$$\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}.$$

Ezt $\mathbf{w}^T$ szerint deriválva, majd egyenlővé téve zéróval kapjuk, hogy

$$2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} = 0$$

$$\Leftrightarrow$$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Ezzel mindössze az a baj, hogy nem tudjuk garantálni, hogy az $\mathbf{X}^T\mathbf{X}$ mindig invertálható lesz. Egy ún. *regularizációs paraméterrel* viszont megoldhatjuk a problémát: ha $\mathbf{A}$ egy négyzetes mátrix, akkor $\lambda \in (0, 1)$ esetén $(\mathbf{A} + \lambda\mathbf{I})^{-1} \approx \mathbf{A}^{-1}$, és az előbbi mátrix mindig invertálható. Ennek ismeretében az optimális $\mathbf{w}$-re kapott összefüggésünket a következő módon alakíthatjuk át:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

8.2. Lineáris regresszió iteratív módszerrel

Az előző fejezetben a modell paramétereként szolgáló $\mathbf{w}$ vektort egy explicit képlettel határoztuk meg. Láttuk viszont ennek hátrányát is: konkrétan szingularitási problémákba ütközhetünk, azaz megtörténhet, hogy a képletben kapott mátrix determinánsa 0. Így az optimális $\mathbf{w}$ vektor kiszámítására az egzakt megoldás helyett alkalmazhatunk egy más, iteratív algoritmust is (mint például gradiens módszert vagy Newton–Raphson-módszert).

A továbbiakban legyen $\mathbf{w}^*$ az optimális súlyvektor, azaz az a vektor, amelyikre a modellünk a legjobban teljesít. A modellünk "jóságát" a hibafüggvény segítségével számszerűsítjük. Továbbra is a négyzetes hibafüggvényt használjuk, így az optimális súlyvektor a következőképpen írható fel:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left(\sum_{i=1}^{\ell} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 \right).$$

8.2.1. Gradiens módszer

A gradiens módszer (gradient descent) egy lokális minimumkeresési algoritmus. Ez azt jelenti, hogy kiindulunk egy kezdeti $\mathbf{w}$ vektorból, majd minden egyes iterációs lépésben úgy frissítjük az aktuális $\mathbf{w}$ vektort, hogy az újonnan kapott súlyvektorra minél kisebb legyen a hibafüggvény értéke. Egy másik fontos objektív, hogy lehető leghamarabb, a lehető legkevesebb lépésben határozzuk meg a hibafüggvény minimumát. Ennek érdekében, minden egyes frissítés során arra törekedünk, hogy a lehető legnagyobb csökkenés irányába haladjunk. A megoldás kulcsa a deriváltfüggvény, amely megadja, hogy milyen irányba mekkora a változás.[4].

[4] Részletesebben: itt

Jelölje $\mathbf{w}_i$ az i. lépésbeli súlyvektort. Ekkor az i. lépésben a következőképpen írható fel a bináris osztályozó átlagos négyzetes hibája:

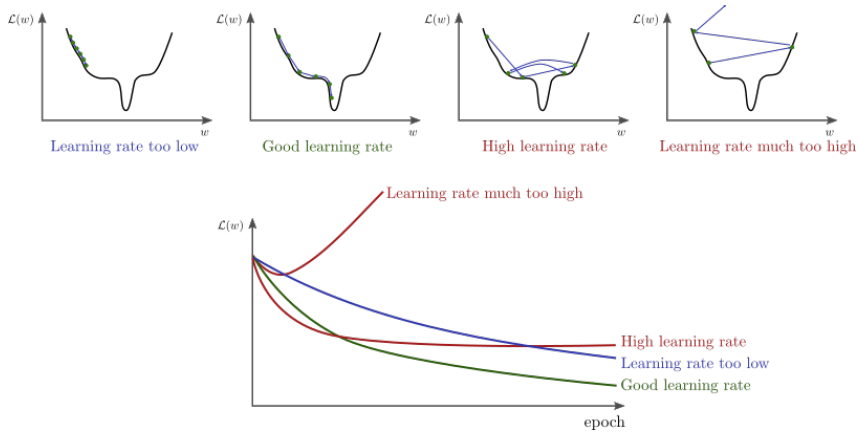
$$\mathcal{L}(\mathbf{w}_i) = \sum_{i=1}^{\ell} (\mathbf{w}_i^T \mathbf{x}_i - y_i)^2 = \|\mathbf{X}\,\mathbf{w}_i - \mathbf{y}\|_2^2,$$

ahol a $\mathbf{w}_i$ vektor $(d+1)$-dimenziós oszlopvektor, mely tartalmazza a b szabadtagot is. Ekkor az $\mathbf{X}\,\mathbf{w}_i$ szorzat eredménye egy $\ell \times 1$ dimenziós mátrix, ahol ℓ a tanulási adathalmaz elemszáma.

Legyen $\nabla\mathcal{L}$ az $\mathcal{L}$ függvény gradiensfüggvénye, és $\nabla\mathcal{L}(\mathbf{w}_i)$ a gradiensfüggvény értéke a $\mathbf{w}_i$ vektorban. Ebben az esetben, az i. iterációban a $\mathbf{w}_i$ vektort a következőképpen frissítjük:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \gamma\,\nabla\mathcal{L}(\mathbf{w}_i)$$

A fenti képletben szereplő γ együtthatót nevezik tanulási rátának (angolul learning rate). Ennek szerepe, hogy befolyásolja, mennyire gyorsan/hamar konvergál az algoritmusunk a minimumértékhez. A minimum keresés egyik érzékeny pontja a γ paraméter. Hogyha a tanulási ráta túl kicsi, akkor egy iterációban keveset változik a $\mathbf{w}_i$ súlyvektor, és az algoritmus csak hosszú idő után kezd konvergálni az optimumhoz. Ellenben, ha a tanulási ráta túl nagy, akkor a keresés nem eléggé kifinomult, cikk-cakkban fog szökdösni a minimumérték körül, de nem fogja tudni meghatározni azt:



1. ábra. A hibafüggvény változása különböző tanulási ráták esetén a gradiens módszer egymást követő iterációiban.

Az intuíció amögött, hogy miért a gradiens vektorral ellenétes irányban változtatjuk a $\mathbf{w}_{i+1}$ értéket, egyváltozós függvény esetén a következő: amikor a gradiens (a derivált) pozitív, akkor a hibafüggény pozitív irányban növekvő, ezért a $\mathbf{w}$ értékét csökkenteni kell, hogy közeledjünk a minimumponthoz. Amikor a gradiens negatív, akkor a hibafüggvényünk csökkenő, ezért a $\mathbf{w}$ paramétert növelni kell annak érdekében, hogy közeledjünk az optimális értékhez (lásd 2. ábra).

Az $\mathcal{L}$ hibafüggvény gradiense a következő módon írható fel[5]:

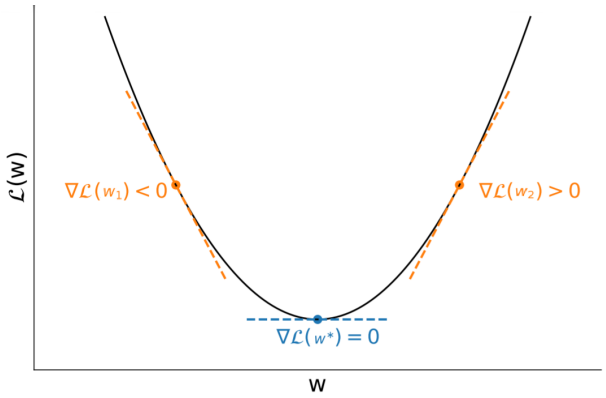
$$\nabla\mathcal{L}(\mathbf{w}) = \nabla\left((\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})\right) = 2\,\mathbf{X}^T(\mathbf{X}\,\mathbf{w} - \mathbf{y})$$

A $(\mathbf{X}\,\mathbf{w} - \mathbf{y})$ egy $\ell \times 1$ dimenziós mátrix, az $\mathbf{X}^T$ dimenziója $(d+1) \times \ell$, ezért a gradiens egy $(d+1) \times 1$-es mátrix, akárcsak az argumentuma, a $\mathbf{w}$ vektor. A deriváltat behelyettesítve kapjuk, hogy:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - 2\,\gamma\,\mathbf{X}^T(\mathbf{X}\,\mathbf{w}_i - \mathbf{y})$$

E kifejezés segítségével meg tudjuk határozni az optimális $\mathbf{w}^*$ paraméter-vektort – vagy annak egy elég jó közelítését. Megállási feltételként megadhatunk előre egy maximális iteráció számot, vagy beállíthatunk egy küszöbértéket, aminél ha kisebb lesz a változás, akkor leállítjuk az optimum keresését.

[5] A használt mátrix deriválási szabályok megtalálhatóak: itt [(69) és (77)-es képletek]



2. ábra. Gradiens módszer alkalmazásakor azt az optimális $\mathbf{w}^*$ vektort (paramétert) akarjuk meghatározni, amelyre az $\mathcal{L}$ hibafüggvény értéke minimális. A $\nabla\mathcal{L}(\mathbf{w})$ jelöli a hibafüggvény gradiensét (deriváltját) egy rögzített $\mathbf{w}$ esetén.