Gépi Látás

Kézzel írt karakterfelismerés

Látrányi Péter Krisztián | VEC2Ao |

Tartalomjegyzék

Bevezetés

Az írásfelismerés fontossága többek között abban rejlik, hogy az írásunk szinte olyan pontossággal azonosít minket, mint az ujjlenyomatunk vagy a DNS-ünk, sőt, esetenkéntjobban is, hiszen még az egypetéjű ikrekre sem jellemző, hogy tökéletesen azonos lenne azírásképük, míg ez a DNS-szerkezetükre és az ujjlenyomatukra teljesül.

A kézírást már régóta használják azonosítási célokra az informatikában is. Az első aláírás-felismerő rendszert 1965-ben fejlesztették ki, ezt pedig további fejlesztések követték. Az aláírások felismerése azonban még nem okoz akkora nehézséget, mint a karakterfelismerés, mivel az ember aláírásában az azonosságokat vizsgáljuk és nem törődünk azzal, hogy ténylegesen milyen betűkből áll össze a név. Amikor azonban az egyes betűket kell felismernünk, figyelembe kell vennünk, hogy emberenként, sőt, még az egyes embereknél időben is szembetűnő eltérések tapasztalhatók az írásképben. Ezt figyelembe véve pedig nem állíthatjuk, hogy egy betű kinézetét egyetlen vagy akár egy tucat mintapélda alapján megtudjuk határozni, és ezek segítségével egy leírt karaktert valamilyen osztályba be tudunk sorolni.

KARAKTERFELISMERÉS NEHÉZSÉGEI:

A karakterfelismerést nehezíti:

- **„Ugyanazon" karakterek emberenként meglehetősen különbözőek lehetnek méretben, alakban és stílusban,** emellett még ugyanazon ember esetén is megfigyelhetők bizonyos eltérések a különböző időpillanatokban leírt karakterek között.
- Minden más képpel egyetemben a karaktereket, illetve szövegeket tartalmazó képek esetén is számolnunk kell a képalkotás – digitalizálás – folyamán, illetve az egyéb okokból a képre került **zajokkal és egyéb képhibákkal** (pl. elmosódás).
- A karakterek kinézetének nincsenek megszeghetetlen, tudományosan lefektetett alapszabályai. Ennek megfelelően csak minták alapján tudunk valamiféle szabályosságot megállapítani az egyes karaktertípusok kinézetével kapcsolatban.

A fejlesztendő programot Pythonban fogom fejleszteni a
NumPY és a Scikit-Learn csomagok felhasználásával.
A program működési elvét a Lineáris SVM algoritmus határozza meg.

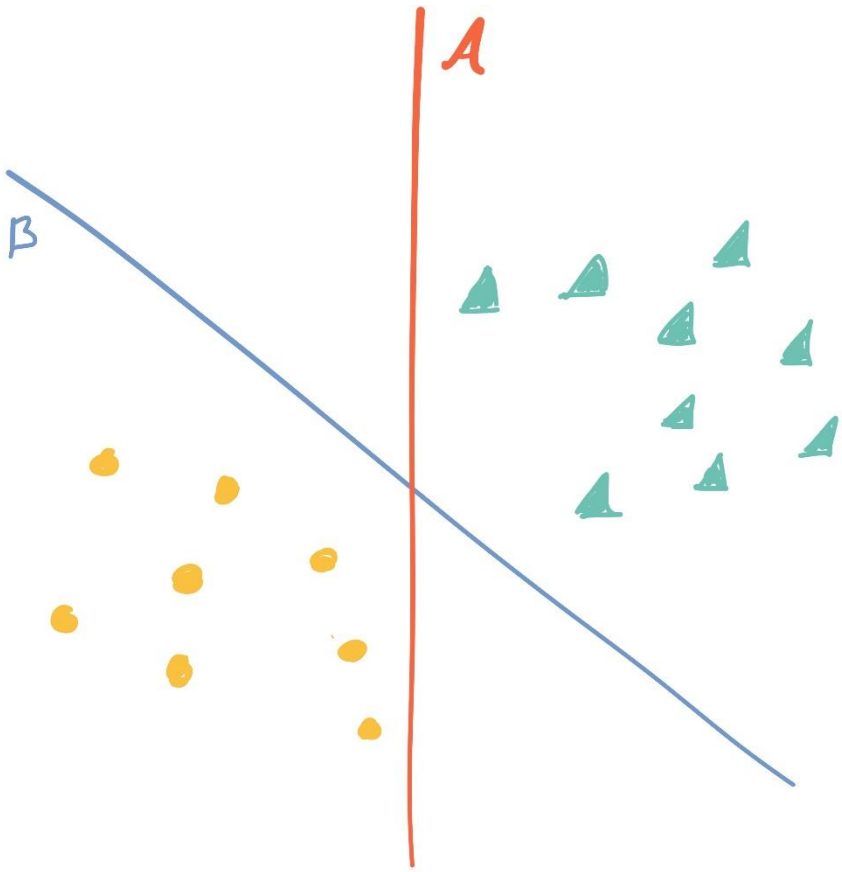
Támaszvektor-gépek (SVM)

Ebben a fejezetben a support vector machine (SVM) algoritmust mutatom be, amely klasszifikációra, regresszióra, de még akár klaszterezésre is alkalmazható. SVM-et már a '60-as években használtak, népszerűsége azonban a modern SVM kialakulása után kezdett növekedni. Azóta ezzel a módszerrel jelentős eredményeket értek el. Többek között használható arc- és kézírásfelismerésre, képek klasszifikációjára

Fontos megkötés, hogy csak numerikus prediktorokat tud kezelni, valamint a kimeneti változó értéke csak -1 vagy 1 lehet, de előnye, hogy nem használ semmilyen véletlent. Az ebben a fejezetben bemutatásra kerülő SVM-ek alapkérdése megegyezik: hogyan lehet a prediktorok terét egy hipersíkkal elválasztani.

LINEÁRIS SVM

A kiinduló feladatban a változók $\mathbb{R}p$ terében egy p-1 dimenziós hipersíkot keres, mely tökéletesen elválasztja a kimeneti két kategóriáját egymástól. [4] Valahogy így:



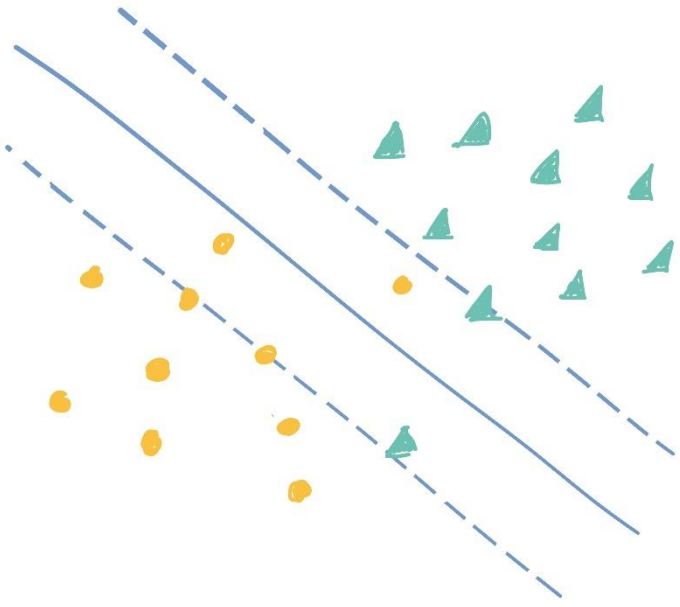
$\mathbf{w} \cdot \mathbf{x} + b = 0$ Hipersík egyenlete

A fenti ábrán a két csoportot ugye könnyű elválasztani, de vegyük észre, hogy végtelen számú lehetséges elválasztási lehetőség van. Fel is rajzoltam kettőt (A, B)

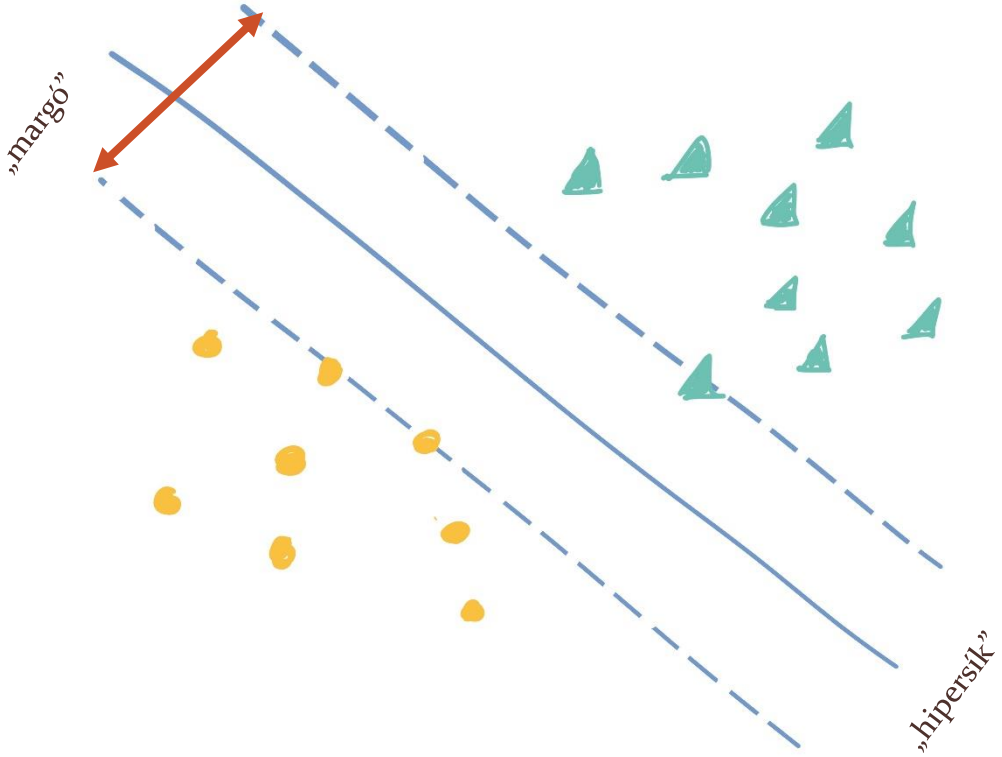
A tesztadatok alapján mindkettő jó eredményt ad, de melyiket választanánk a gyakorlati életben? Én a B-t tenném. Azon egyszerű oknál, hogy az nagyjából egyforma távolságra helyezkedik el mindkét csoportól. A lineáris SVM lényegében ezt a fajta gondolkozást próbálja megvalósítani. Vagyis azt a határvonalat keresi, ami egyforma távolságra van mindkét osztálytól. Ehhez pedig segéd "margókat" használ. Lényegében kijelöl két párhuzamos egyenest, nevezzük margóknak őket, a hipersík körül, és arra törekszik, hogy ezek a margók minél távolabb kerüljenek a hipersíktól.

Amennyiben a két osztályunk lineárisan jól elválasztható egymástól, akkor az úgynevezett **szigorú margókat (hard margin)** tudjuk alkalmazni, mint a lenti példában is.

(Ha pedig nem tudjuk az osztályokat egyértelműen elválasztani egymástól, mert van néhány pontunk, amik megakadályozzák, hogy egyértelműen elválaszthassuk a két osztályt lineárisan, ekkor a **Hinge loss**-t hívjuk segítségül. Ebben az esteben **lágy margókról (soft margin)** beszélünk. Ez lényegében egy olyan függvény, ami o értéket vesz fel, ha a pont a margó jó oldalán helyezkedik el, és egyenletesen növekszik ahogyan egyre távolodunk a rossz irányba. A Hinge loss segítségével most már tudjuk mérni a pontok rosszaságát)
Ezek az esetek így néznek ki:



Szigorú margós eset:



Vegyük észre, hogy mindkét margón van legalább egy megfigyelési pont. Ezek a “**support vektorok**”. Lényegében ezek azok a pontok, amik számítanak a hipersík meghatározásában.

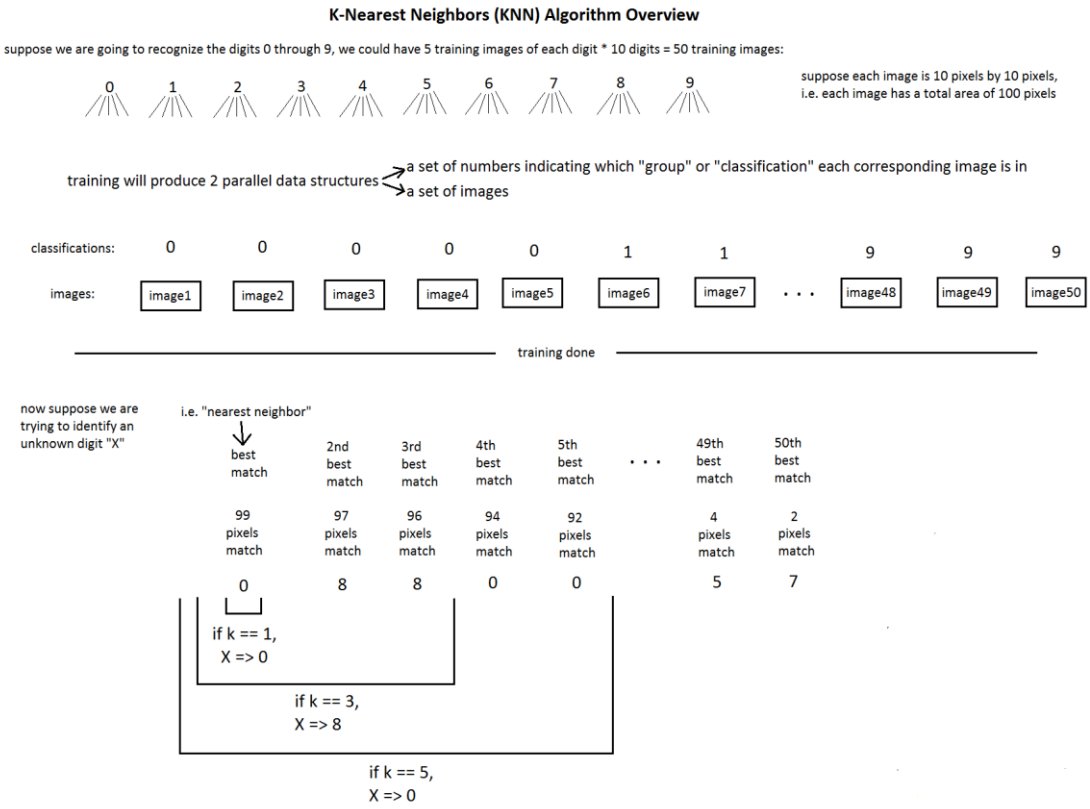
Support vektor egyenletei:

$$w * x + b = +1$$
$$w * x + b = -1$$

A feladat a hipersík és a támaszvektor közötti távolság („margó”) maximalizálása.

Egyéb algoritmusok karakterfelismerésre

KNN K legközelebbi szomszéd algoritmus[2]:



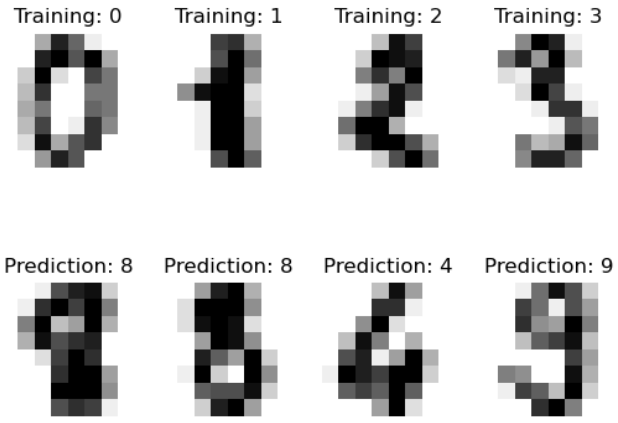
Lépései:

1. Euklideszi távolság számítása a test adatpont és a betanított adatok között.
2. A kiszámított távolság rendezése növekvő sorrendben
3. A legközelebbi k szomszédos elemek megtalálása a rendezett tömbben
4. Leggyakoribb osztály keresés a k szomszédos elemek között
5. A prediktált osztály visszaadása

6. Megegyezőség vizsgálata, pontossági szám alapján, hogy meggyőződjünk arról, hogy a predikció jó-e, avagy sem

Scikit-Learn

Az SK Learn digits adatacsomagja 180 mintát 8x8 pixel felbontásban tartalmaz minden egyes számjegyre (0-9). A minták előállításához 43 ember kézírását használták fel.
A példákban **fekete háttérrel, fehérrel írt számok** találhatóak, ezért ugyanilyen tulajdonságú képekre működik a legjobban az algoritmus.



Programfejlesztés Lépései

- Importálandó könyvtárak importálása
- A tanuló algoritmus betanítása.
- A teszt kép beolvasása, átméretezése.
- Teszt kép normalizálása (intenzitás csökkentése)
- Predikció
- Eredmény kiíratása
- Tesztelés
- Hibakezelés

PROGRAM FUTTATÁSÁHOZ SZÜKSÉGES ESZKÖZÖK:

- ❑ Java program futtatására alkalmas fordító
- ❑ A program futtatásához szükséges csomagok telepítése
- ❑ Tesztelendő kép megléte
- ❑ config.py és main.py forrásfájlok

PROGRAM FUTTATÁSÁHOZ SZÜKSÉGES CSOMAGOK:

- ➢ scikit-image
- ➢ scikit-learn

PROGRAM FUTTATÁSÁNAK LÉPÉSEI:

- ➔ Digit Recognition.py megnyitása a fordítóban
- ➔ A forrásfájl futattása (F5)
- ➔ **A tesztelendő képnek az elérési útját a parancssorban kell megadni(elérési út, fájl neve, kiterjesztése)**, majd Entert kell ütni
  Pl: C:\Users\latranyi\Documents\Képek\2_b.jpg

Tesztelés eredményei

A programot minden egyes számra 10 képpel teszteltem. Így összességében 100 tesztesetet végeztem

Leginkább a 2-es, 3-as karakterek felismerése jelentette a problémát.
A 2-es karakterek esetében 40%-os a hiba arány, míg a 3-as karakterek esetében pedig 70%-os.

Az 1-es, 4-es, 5-ös, 7-es karaktereket minden esetben hiba nélkül felismerte.
A 6-os, 8-as, 9-es esetekben 1x,2x hibázott az algoritmus.

Összességében az elvégzett tesztesetekre a program 84%-ban helyes eredményt adott.

A mintául szolgáló adatcsomag tesztelésekor a Scikit Learn 97%-os pontosságot állapított meg.

Felhasznált irodalom

[1] Brown, Erik W. Applying Neural Networks to CharacterRecognition,1992

[2] https://hu.wikipedia.org/wiki/Python_(programoz%C3%A1si_nyelv)

[3]https://scikitlearn.org/stable/auto_examples/classification/plot_digits_classification.html

[4] Virtás Dávid: Prediktív módszerek és gépi tanulás alkalmazásai a biztosításban Szakdolgozat, 2019