Mesterséges Intelligencia

Házi Feladat

Kozaróczy Zsolt

Medvey Fanni Zsófia

Konzulens: Orbán Gergely

Általános leírás

Az algoritmus

A megvalósítandó algoritmusunk a knn-algoritmus. Ez egy osztályozó algoritmus, ami egy adott távolságfüggvény alapján meghatározza az osztályozandó elem legközelebbi k szomszédját és ezek közül egyszeri többséggel kiválasztja, milyen osztályba sorolja be a kérdéses elemet.

A feldolgozandó adatok

A házi feladatban karaktereket kell felismerni származtatott adatok alapján. A kapott forrásfájlban 20.000 sor található, soronként egy betű az angol abc-ből, és 16 integer típusú származtatott adat. A program összes futásakor keresztkiértékelést használunk, azaz először az első 10.000 adatot töltjük be a programba ismert adatként, és kiértékeljük a másik felét, majd fordítva is elvégezzük. Az összes eredmény ilyen kiértékelés végeredménye.

A távolságfüggvények

Négy féle távolságfüggvényt valósítottunk meg:

- **Euklideszi** távolság: Az adatok négyzetösszege.
- **Csebisev** távolság: Az adatokat páronként összehasonlítja, és a legnagyobb különbségű a távolság
- **Taxicab** (Manhattan) távolság: Az adatok különbségösszege
- **Hamming** távolság: A nem egyező adatpárok számával tér vissza

Számításmódok

Kétféleképpen értelmeztük a k paramétert. Az első módszerben fixen vesz k db szomszédot, és ezek közül egyszerű többséggel választjuk ki a megfelelőt. Ha egyenlőség áll fent, akkor véletlenszerűen választunk.

A másik értelmezés szerint elkezdjük sorra venni a szomszédokat, és amint az egyik betűből többség lép fel, visszaadjuk találatként. Itt a k paraméter egyfajta felső határként szerepel, maximum k szomszédot vizsgálunk meg. Ha ez alatt nincs többség, akkor véletlenszerűen térünk vissza.

Az adatok súlyozása

A távolságfüggvények felhasználásakor lehetőség van az adott adatokat súlyozni, azaz meghatározni, milyen arányban vegyük figyelembe. Ezáltal a kevésbé meghatározó származtatott adatokat ki lehet szűrni, és egyszerűsíteni, pontosítani az osztályozást.

A program működése

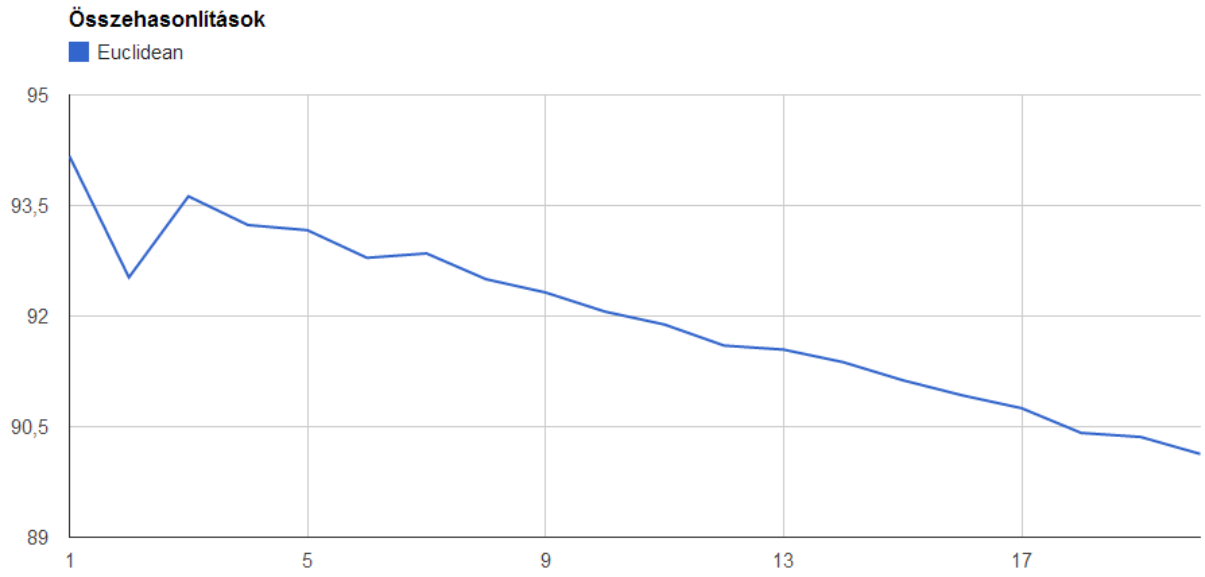
A programot elindítva lehetőség van egy adott számításmóddal és egy választott k paraméterrel lefuttatni. Ebben az esetben egy új tabon megtekinthetjük az eredményt százalékos arányban, confusion mátrixszal vagy a leggyakoribb tévesztésekkel.

Ezen kívül el lehet indítani egy súlyozás-optimalizálót is, ami adott k paraméterhez kiszámolja, hogy milyen súlyozással érdemes az adatokat kezelni. (A súlyokat 0-10-es skálán osztja, ahol 10 a legfontosabb). Ennek futásideje jóval hosszabb, mint egy egyszerű kiértékelésé.

A harmadik lehetőség, hogy egy adott súlyozással futtatjuk az osztályozást.

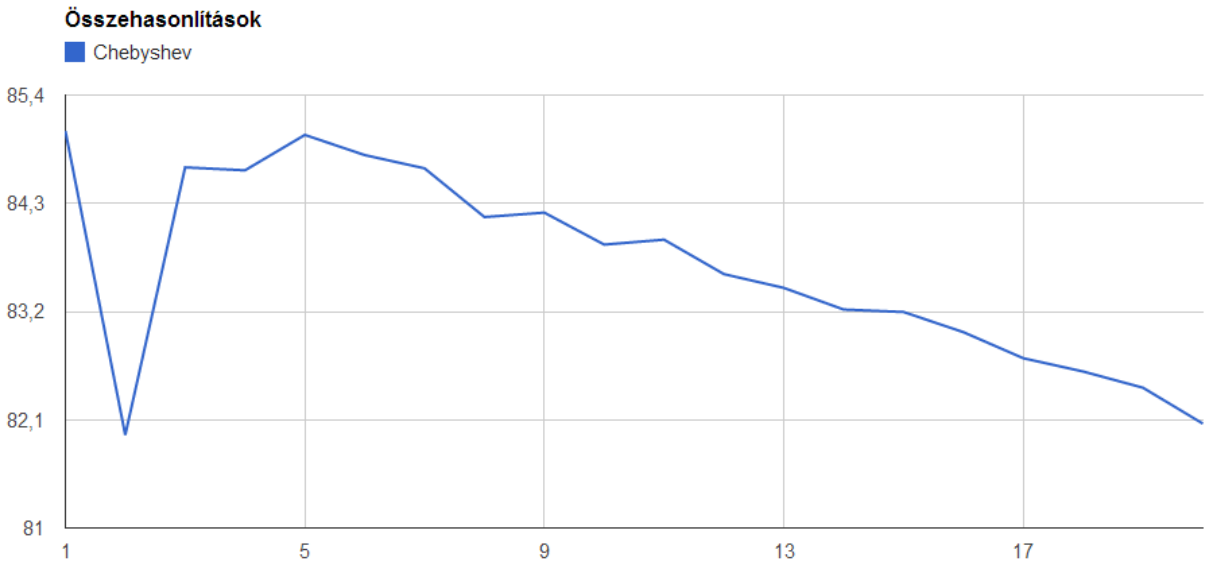
Tapasztalatok

Fix szomszédok esetében k paraméter változtatása



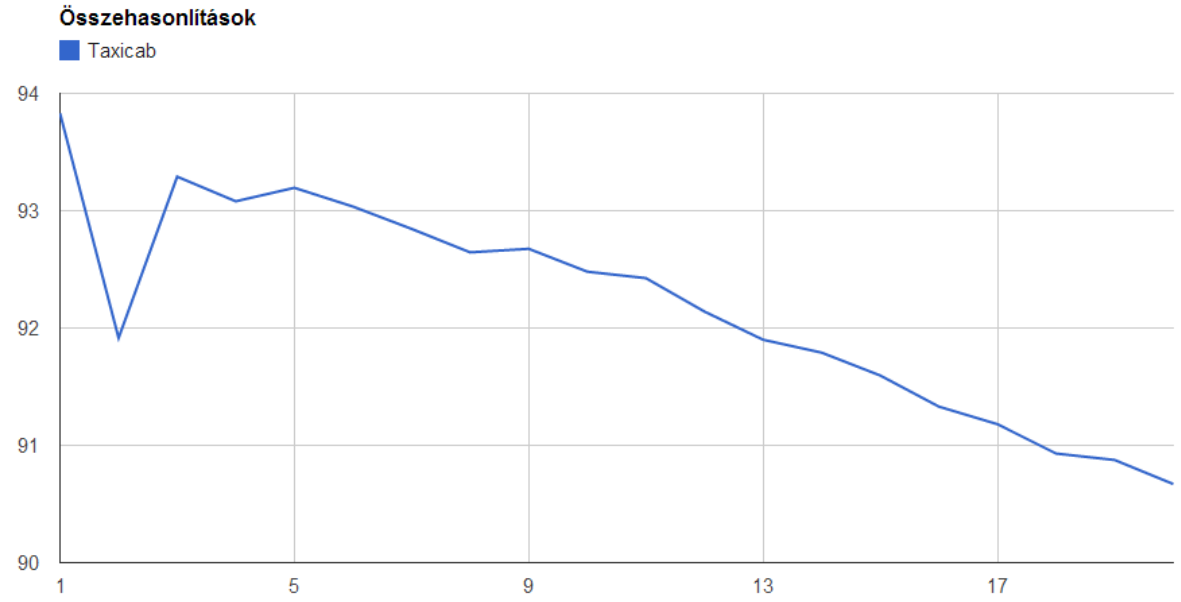
k növelésével egyre jobban csökken a hatékonyság. Megfigyelhető, hogy a páratlan k értékeknél jobban teljesít, mint a párosoknál (azaz kevesebb egál eset áll fent).

k=100-nál 77,09%-os találati aránya van



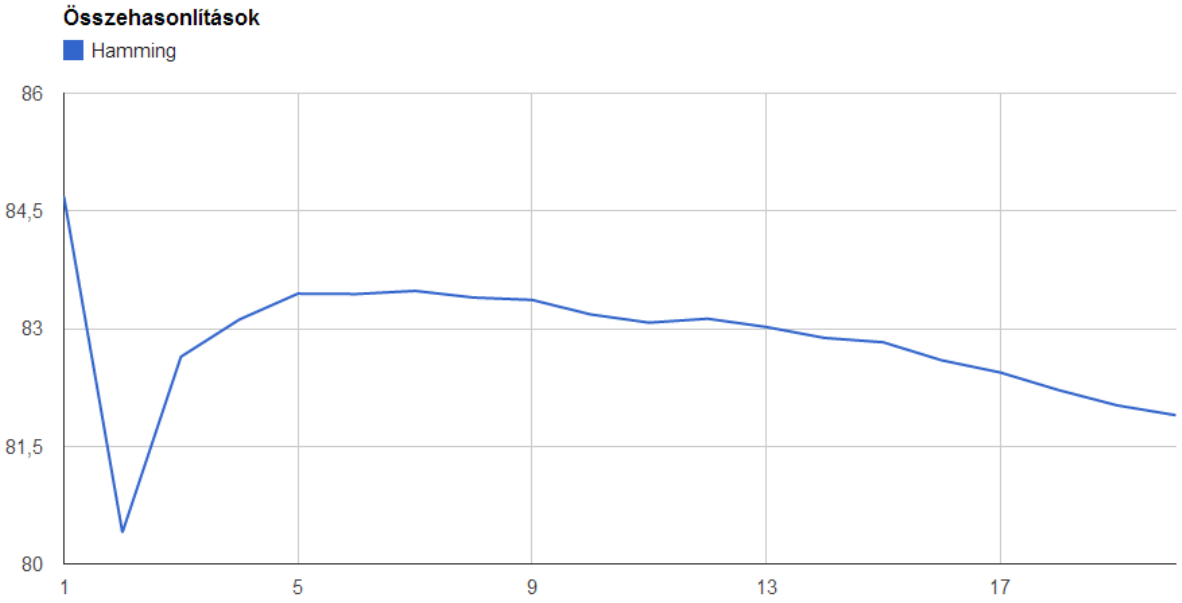
k növelésével egy ideig (k=5-ig) növekszik a hatékonyság, majd utána szépen lassan csökken. Itt is megfigyelhető, hogy a páratlanok jobban teljesítenek.

k=100-nál 68,445%-os találati aránya van



k=3 és k=5 értékeknél a legpontosabb, ezután itt is lassú csökkenésbe kezd

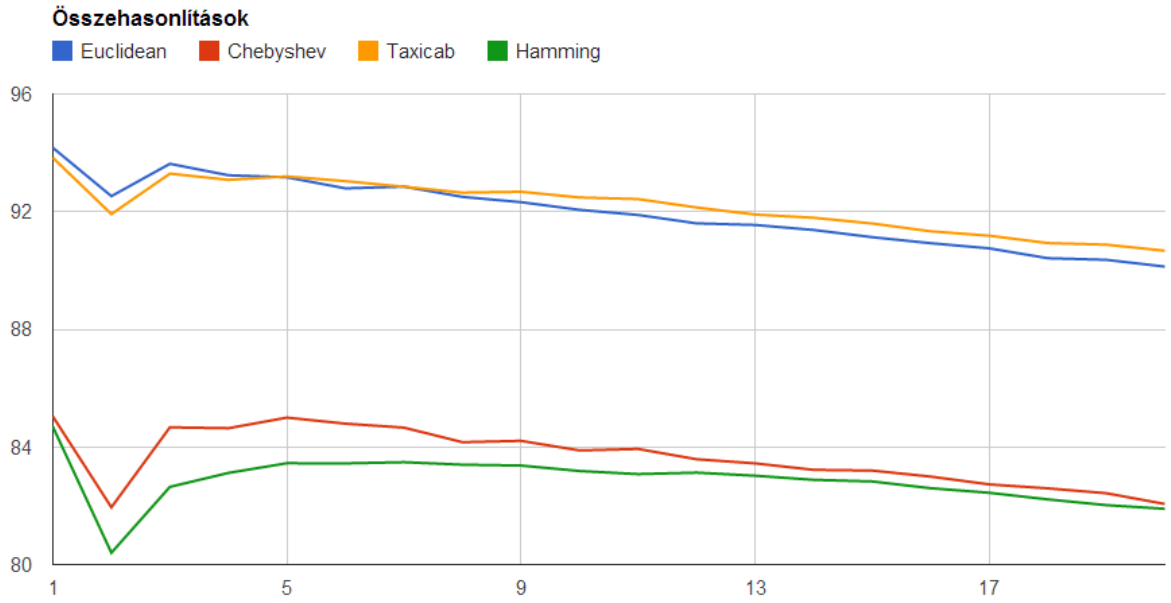
k=100 esetén 80,19%



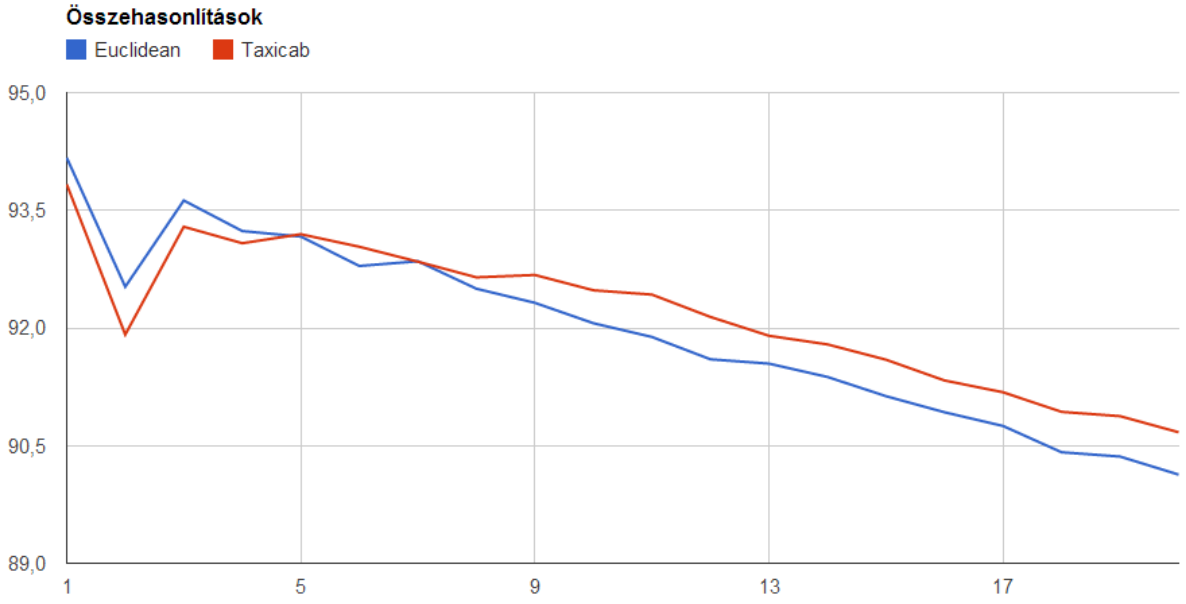
A hamming távolságnál k=7 adja a legjobb találati arányt, majd utána szépen lassan ez is csökken

k=100 esetén 72,995%

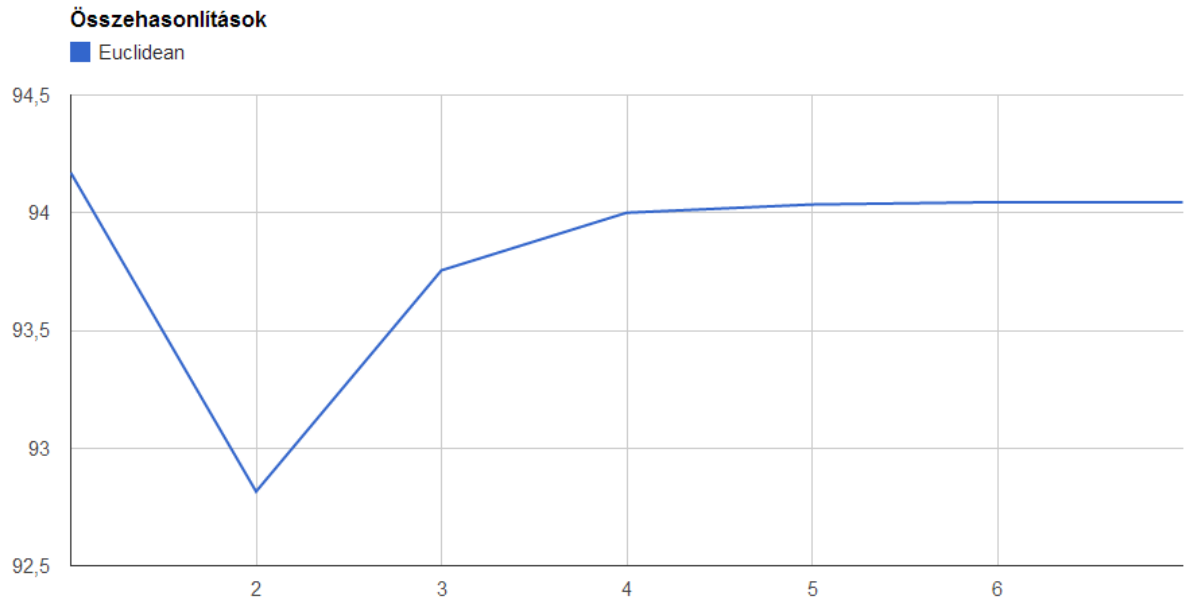
Konklúzió: Mind a négy távolságfüggvénynek van egy ideális k!=1 értéke, ahol a legjobb teljesítményt nyújtja, majd ez után lassan elkezd csökkenni a hatékonysága. Mindenhol megfigyelhető, hogy a páratlan k értékek jobb eredményt adnak.



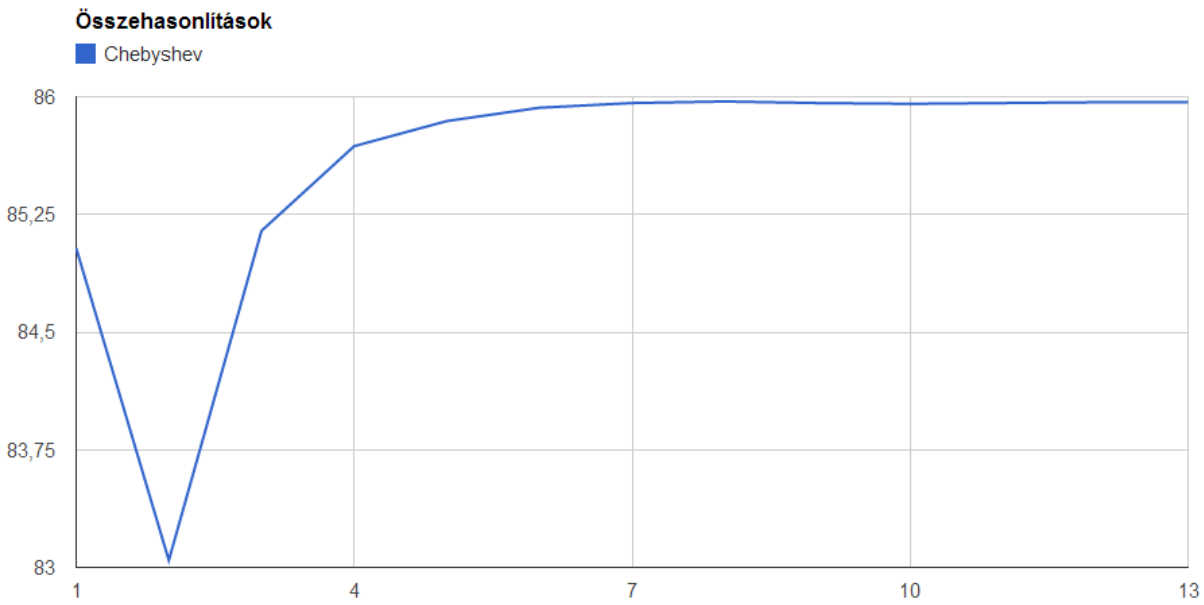
Egymással összehasonlítva látszik, hogy kisebb k értékekre az euklideszi teljesít a legjobban, majd utána átveszi a vezetést a taxicab a vezetést.



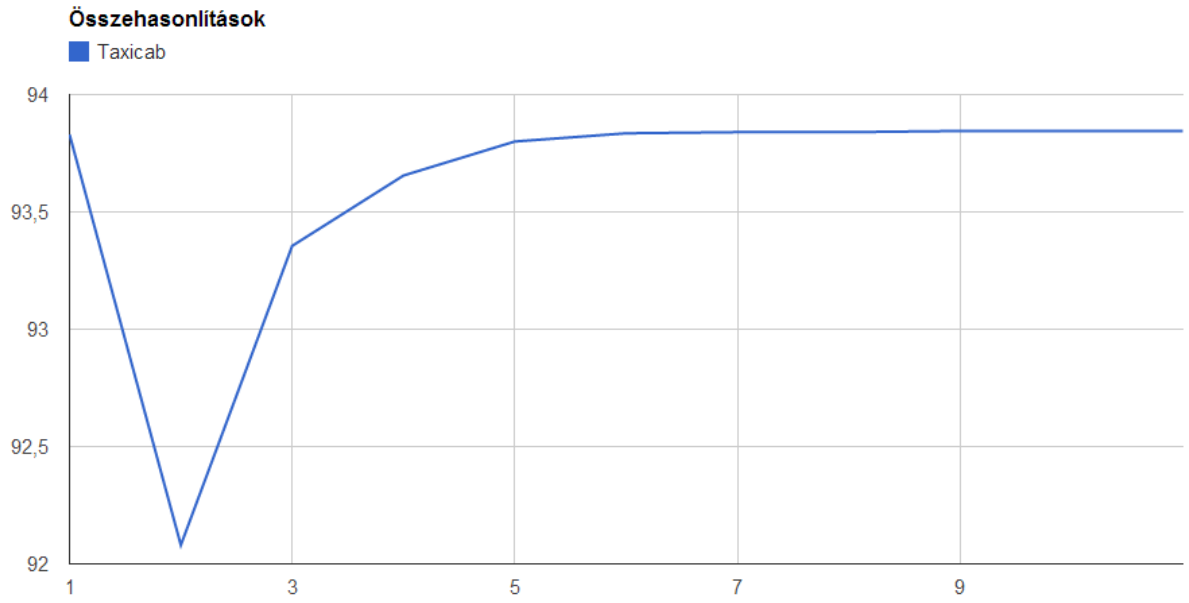
A másik számítási móddal is hasonlóan végignézhetőek a távolságfüggvények

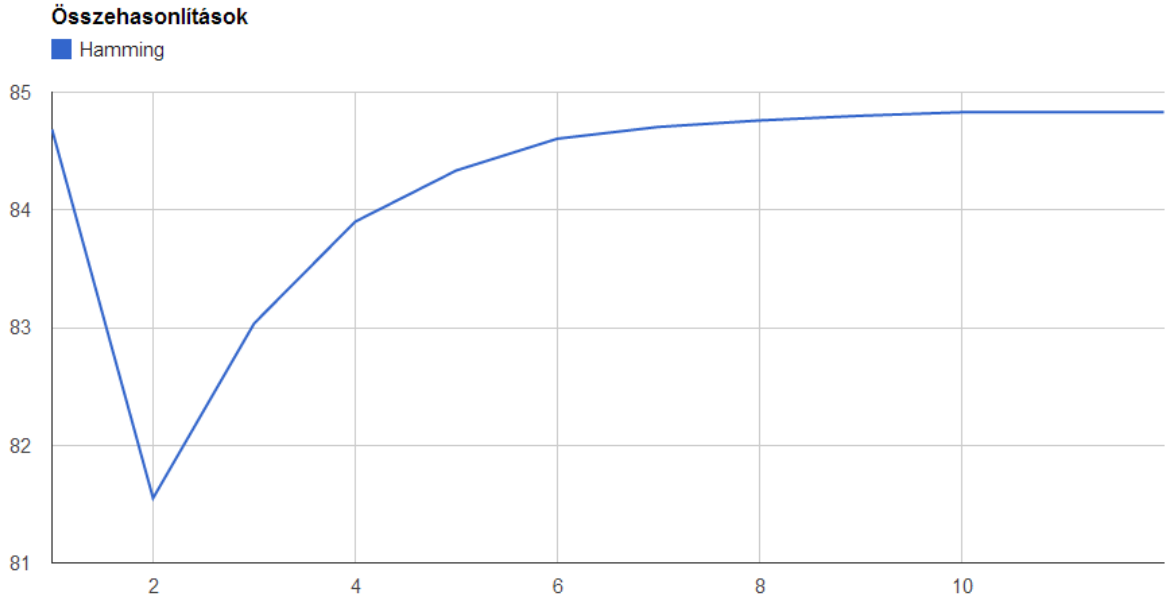


A számítás módjából adódik, hogy egy fix értékhez fog beállni a grafikon, mivel ha valamiről eldönti (akár rosszul is), hogy hova tartozik, akkor nem fog újabb szomszédot hozzávenni. Jelen esetben k=5-től kezdve nem változik a grafikon.

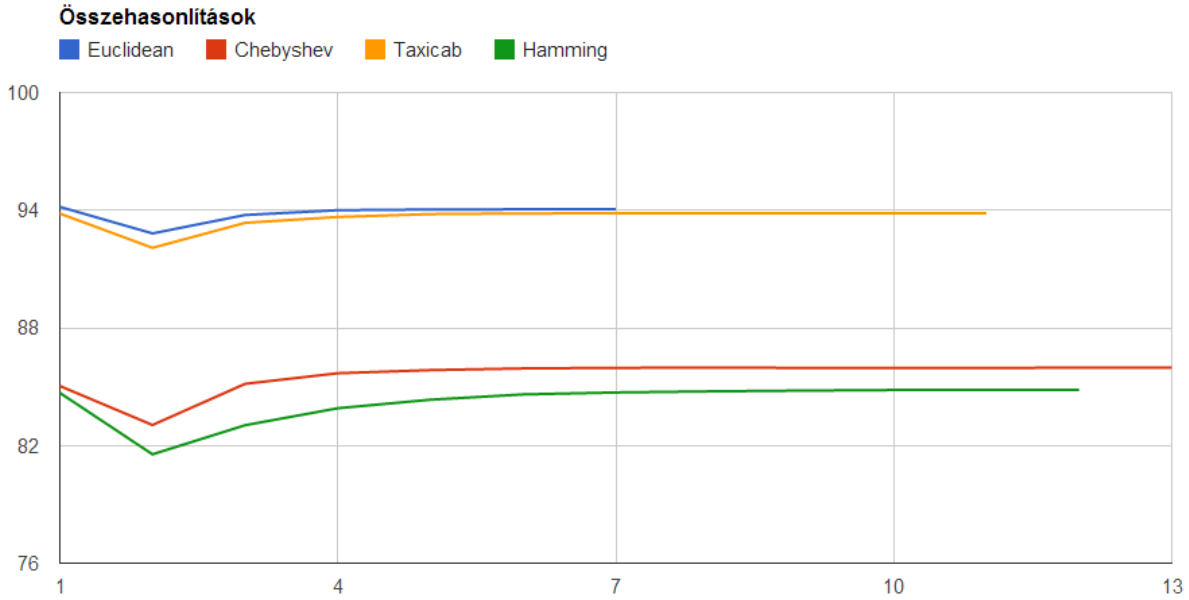


Az előzőhöz hasonlóan itt is megfigyelhető, hogy k=7-től kezdve nem növekszik tovább.



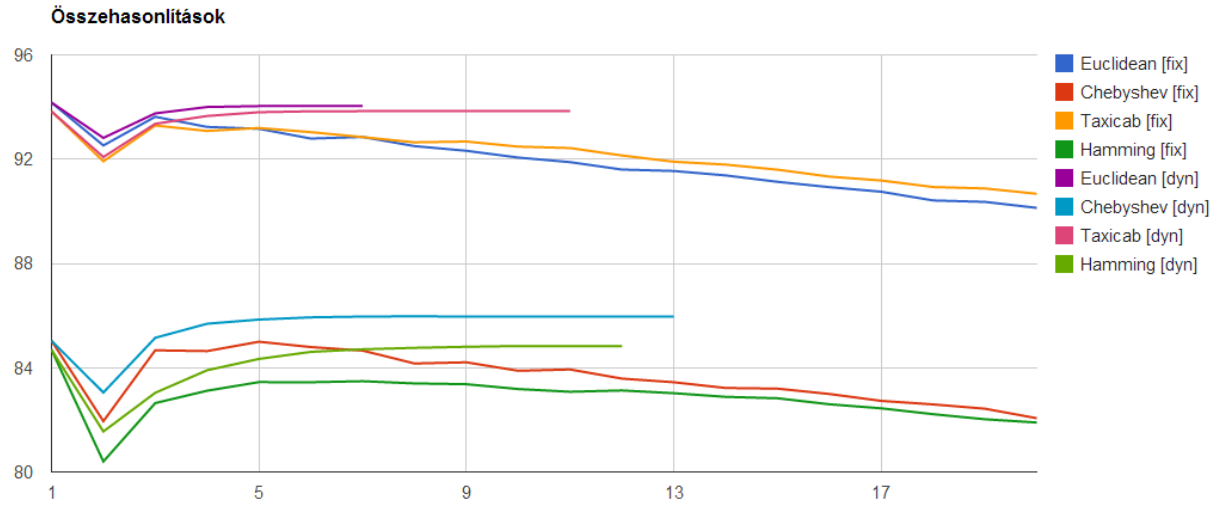


Az összes grafikont egymás mellé rakva láthatjuk, hogy ebben az esetben hogyan teljesítenek egymáshoz képes.

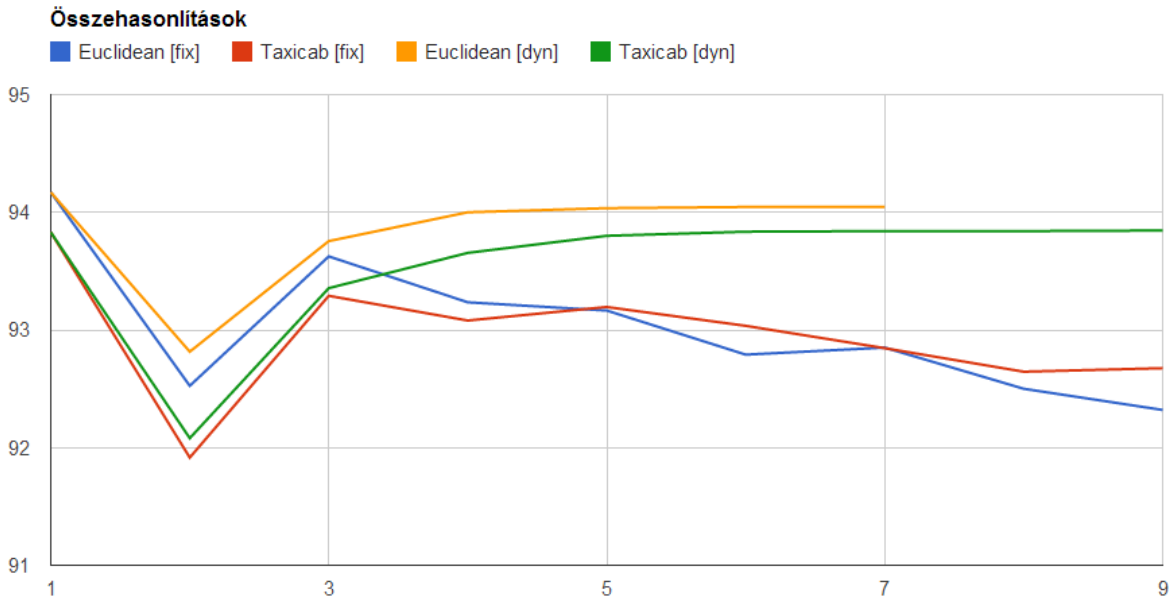


Látható, hogy a leghatékonyabb az euklideszi, majd itt is a taxicab követi.

A két módszert összevetve az alábbi grafikonokat kapjuk



A legjobb 2-2 eredmény összehasonlításából látszik, hogy a legjobb eredményt akkor érhetjük el, ha folyamatosan vesszük hozzá a szomszédokat, és a maximum értéket 5-re állítjuk (feltételezve, hogy a k=1 esetet elvetjük, azaz legalább két szomszédot megnézünk)



Jellemző parametrizálás

A továbbiakban az euklideszi távolsággal és fixen 3 szomszéddal dolgozunk tovább.

A program a paramétereket fontosság szerint 0-tól 10-es skálán értékeli, ahol a 0 azt jelenti, hogy egyáltalán nem veszi figyelembe. Alapértelmezés szerint minden paraméter 5-ös fontosságú, majd végignézi, hogy kaphatunk-e ennél jobb értékeket.

A program lefutása után kiderült, hogy az adatok közül nem mind ugyanolyan fontos az osztályozás szempontjából. A súlyozás nélküli 93,625%-os találati arányt 95,31%-ra lehetett növelni megfelelő beállításokkal (ez +1,685%, ami 20000 adatnál már nem kevés tévesztés).

A kapott súlyozást (1,1,1,1,1,4,3,8,7,6,5,5,10,5,6,5) összevetve az értékek leírásával kiderült, hogy a legkevésbé fontos tulajdonságok a betűk méretére és elhelyezkedésére vonatkoznak (első 5 adat), míg a legfontosabbnak a “edge”-k számát tartotta.

A mostani optimalizálás a teljes találati arány javítására vonatkozott, elméletben lehetne megadni más és más célokat is, például egy adott karaktert ismerjen fel minél nagyobb biztonsággal.

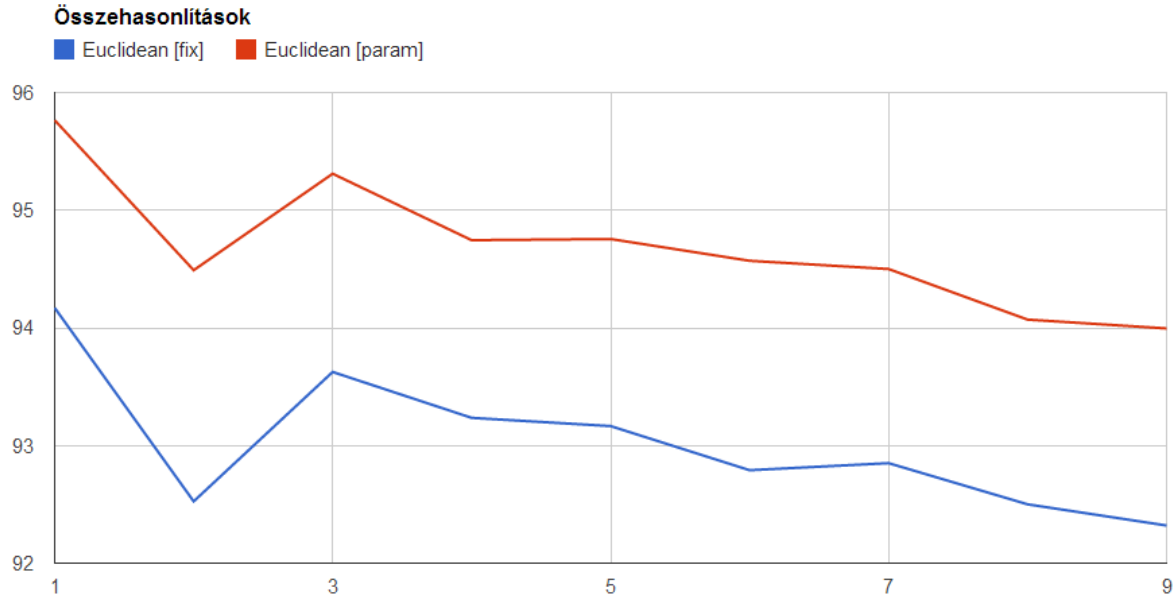
Confusion mátrix

Az euklideszi távolságméréssel, k=3 paraméterrel, súlyozás nélkül az alábbi mátrixot kapjuk (bal oldalt a helyes érték, felül pedig, hogy mit tippelt rá az algoritmus)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	780	2										1	1				1				1				3	
B		726		6	3		2	3			1		1					12	2		1	7		1		1
C			707		5		9					2			7		2			1	2		1			
D	1	18		762			3	11						4	3		1		1	1						
E		10	10		706	3	12	1			5	1					3		2		1	1		1		12
F		1		4	3	704		2	4	1				3		36		1	1	12		2	1			
G		4	8	13	14	1	722				3		1		2		2		2			1				
H		19	3	37	5	4	9	611			19		1	5	6		2	11			1				1	
I	1	1	2	2		6	1		711	29				1										1		
J	1	3		1	2	3		1	28	702				1	1		1				2					1
K		5		6	14	1	2	43			635				1			13			3			16		
L		1	1		4		5	3	1	2	3	734				1		1	2					2		1
M		11		1		1	4	1					752	12	1						1	3	5			
N		4		10				9					3	738	9			4				6				
O			6	17			2						1	5	703		16				2		1			
P		7		4	3	46	1	5				2				728	1	1							5	
Q		1	1	1	2		7					2			17	4	744	2							1	1
R		31		6	1	4	1	7			19	2	1	7		1	2	673	2			1				
S	1	8		3	11	2	2	2	2	1		1					2	1	706	1						5
T	1	5	4	2	1	5		2			3		1	1	1		2			750		1		2	14	1
U	4	3	1	5				11		1			3		2						783					
V	1	19	1			1	2	2					2	2	1	1					2	725	2		3	
W			2				1	2					13		5			2			3	5	719			
X	2	4	1	4	9			3	3	1	20						3	3	1					730		3
Y	2	2		2				2								2	1			9	1	2	1	1	761	
Z					3				1	3							9		1	3				1		713

Látható, hogy a főátlóban helyezkedik el a legtöbb érték, azaz helyesen tippelt a program. E mellett található jó pár gyakori tévesztés, például P helyett F-et mond, K helyett H-t, és így tovább.

Ha megnézzük a súlyozott futás eredményét is, akkor láthatjuk, hogy néhány betű találati aránya növekedett, de van olyan is, ami romlott. Ez nem meglepő, mert az optimalizálás az összes találati arányra vonatozott, nem egy-egy betű találati arányára.



Az összehasonlításon látszik, hogy bár a k=3-ra lett optimalizálva, a többi k értéknél is jobban teljesít a súlyozott osztályozás.