

Járművek trajektóriájának előrejelzése machine learning modellekkel

PÉTER BENCE MÉRNÖKINFORMATIKA BSC 6. FÉLÉV*, Széchenyi István Egyetem, Hungary
DR. HORVÁTH ANDRÁS, Széchenyi István Egyetem, Hungary
AGG ÁRON PHD HALLGATÓ, Széchenyi István Egyetem, Hungary



Az ITS (intelligent transportation system) egyre nagyobb teret hódít napjainkban és rengeteg különböző területen alkalmazzák ezeket a rendszereket. A közlekedési csomópontok elemzése egy frekventált terület az ITS alkalmazásában. Célunk, gépi látás és gépi tanulás felhasználásával, közlekedési csomópontok elemzésének automatizálása és felgyorsítása. A kutatásban lefektetett alapgondolatokat, kifejlesztett keretrendszerét és a felmerülő problémák megoldásait, a gyakorlatban balesetek megelőzésére, renitens viselke-dések kiszűrésére és forgalomirányító rendszerek támogatására lehet használni. A kutatásban egy trajektória

osztályozó módszert ismertetünk, amely objektumdetektálás és objektumkövetés segítségével elemezi a közlekedési csomópontokban elhaladó járművek mozgását. A mozgásuk alapján klaszterezzi a trajektóriákat, majd gépi tanulás segítségével predikciót ad az újonnan belépő járművek kilépési pontjára. A módszerhez 6 különböző közlekedési csomópont-ban készített saját videó adatbázisunkat használtuk fel. A tesztelt klaszterezési mód-szerek közül (OPTICS, BIRCH, KMeans, DBSCAN) az OPTICS algoritmus bizonyult legjobbnak trajektórák klaszterezésére. Összehasonlítottunk több különböző klasszifikációs módszert a legfontosabb predikció eléréséhez, amelyek: KNN, SVM, GP, DT, GNB, MLP, SGD. A tanul-mányban bemutatott eljárások közül az KNN adta átlagban a legfontosabb 90%-os eredményt.

Authors' addresses: Péter Bence Mérnökinformatika BSc 6. félév, Széchenyi István Egyetem, Győr, Hungary; Dr. Horváth András, Széchenyi István Egyetem, Győr, Hungary; Agg Áron PhD hallgató, Széchenyi István Egyetem, Győr, Hungary.

2023. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in , <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>.

ACM Reference Format:

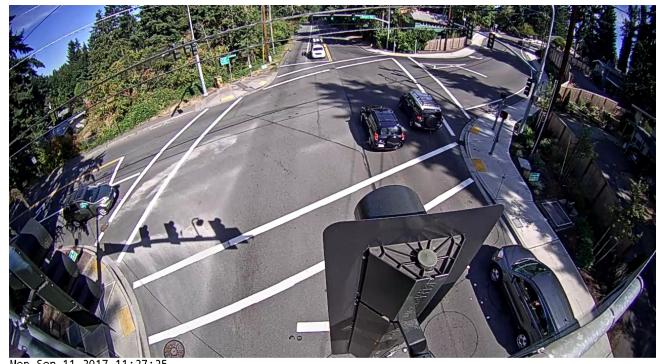
Péter Bence Mérnökinformatika BSc 6. félév, Dr. Horváth András, and Agg Áron PhD hallgató. 2023. Járművek trajektóriájának előrejelzése machine learning modellekkel. 1, 1 (April 2023), 9 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

CONTENTS

Abstract	1
Contents	2
1 Bevezetés	2
2 Kapcsolódó kutatások	3
2.1 YOLO	3
2.2 DeepSORT	4
3 Adathalmazok kialakítása	4
3.1 Adatstruktúra	4
3.2 Objektumdetektálás	4
3.3 Objektumkövetés	4
4 Klaszterezés	5
4.1 Adattisztítás	5
4.2 Feature vektorok	5
4.3 Klaszterezési algoritmusok	5
4.4 Paraméterek kiválasztása	6
5 Klasszifikáció	7
5.1 Multiclass	7
5.2 Binary	7
5.3 OneVsRest	7
5.4 Machine Learning modellek	7
5.5 Feature vektorok	7
5.6 Pontosság mérése	8
References	8

1 BEVEZETÉS

A városok növekedése egyre nagyobb forgalomhoz vezet, ami a balesetek, forgalmi dugók számát növeli és a levegő minősége is romlik. Az ITS (intelligent transportation system) fejlesztése a városokban erre megoldást jelenthet. Ez magába foglalja az információs és kommunikációs technológiák, mint például szenzorok, kamerák, kommunikációs hálózatok és adat elemzés fejlesztését. 5G hálózatokon keresztül, ezek a technológiák összeköthetők a közlekedési eszközökkel. Ehhez okos forgalomirányítási rendszerek kifejlesztésére van szükség, amik információval tudnak szolgálni a járművekbe szerelt informatikai rendszereknek. A legértékesebb információt a közlekedésben részvevő járművek jelen és jövőbeli pozíciója jelenti. Pontos és gyors trajektória előrejelző rendszerek kifejlesztése egy nagy kihívás és egyre növekszik irántuk a kereslet. E kutatási terület kiforrottanságából eredően, kevés létező keretrendszer és adathalmaz található, így a tanító adathalmaz gyűjtése, adatok kinyerésének formátuma, tárolása és mérőszámok kifejlesztése (amivel a tesztelni kívánt modellek pontosságát tudjuk mérni) is a kutatáshoz tartoznak. Ebben a kutatásban erre a problémára törekszünk egy módszertani és keretrendszer kifejleszteni, emellett klaszterezési és klasszifikációs algoritmusokat tesztelni. A tanító adatok előállításához, objektumok detektálására a YOLOv7 [Wang et al. 2022] konvolúciós neurális hálót használtuk, ez a konvolúciós neurális háló architektúra nem csak nagy pontosságot hanem sebességet is nyújt nekünk. Emellett képkockáról képkockára követni is kell tudni a detektált objektumokat. Erre is sok megoldás található



Mon Sep 11 2017 11:27:25

Fig. 1. Bellevue Newport kereszteződés



Mon Sep 11 2017 14:27:05

Fig. 2. Bellevue Eastgate kereszteződés

manapság, erre a feladatra a DeepSORT [Wojke and Bewley 2018] nevezetű algoritmust használtuk, ez kálmán filtert és konvolúciós neurális hálót használ az objektumok követésére. A tanító adatok 5 különböző helyszín forgalmát tartalmazzák. minden helyszín más tulajdonságokkal bír, ezért nem lehet generalizálni a tanítási folyamatot, nem lehet egy univerzális modellt betanítani ami minden közlekedési helyszínre alkalmazható egyaránt. 5 videót Bellevue város github oldaláról gyűjtöttük, amiknek az elérhetőségét függelékként csatoljuk, a kereszteződések így néznek ki 1 2 3 4. A klaszterezés során megpróbáljuk minél pontosabban meghatározni a be és kimeneti pontok által leírt klasztereket, amelyek majd alapul szolgálnak a klasszifikáció tanítása során. Több fajta klaszterezési algoritmust megvizsgáltunk a kutatás során, KMeans, OPTICS [Ankerst et al. 1999], BIRCH [Zhang et al. 1996] és DBSCAN [Ester et al. 1996][Schubert et al. 2017]. A klasszifikációhoz bináris klasszifikációs modelleket kombinálunk, így több osztályos klasszifikációs modelt kapunk. minden bináris modelnél, egy osztály az összes többivel szemben van betanítva. A modellek pontosságának kiértékelésére 3 mérőszámot alkalmaztunk, amik az *Accuracy Score*, *Balanced Accuracy Score* [Brodersen et al. 2010] és *Top-k Accuracy*



Fig. 3. Bellevue NE kereszteződés



Fig. 4. Bellevue SE kereszteződés

Score. Mindegyik mérőszám kiszámolásához *K-Fold Cross-Validation* [Anguita et al. 2012] metódust alkalmaztunk, ahol $K = 5$.

2 KAPCSOLÓDÓ KUTATÁSOK

Sok ITS-el kapcsolatos kutatásban tárgyalják a forgalom folyás (traffic flow) előrejelzését. [Paul et al. 2017] összehasonlítja az eddig kutatott és használt modellek, mint például Kalman Filtering, k-nearest neighbor (k-NN), mesterséges neurális hálók, stb., pontosságát és sebességét, ezen modellek tovább-kutatását, mivel egyre növekednek a különböző szenzorok által begyűjtött traffic flow adatok, így ez a terület belépett a *Big Data* korszakába. [Rossi et al. 2021] is a traffic flow előrejelzését és generálását tárgyalja, Floating Car Data (FCD) adathalmazokon betanított, Hosszú-Rövid-Táví memóriájú és Generatív versengő hálókkal.

2.1 YOLO

”You look only once” (YOLO) egy state-of-the-art, valós idejű objektum detektáló rendszer. Legfrissebb változata a Yolov7 felülműlja sebességeben és pontosságban a modern konvolúciós hálókat (lásd 5 6 7). Beágyazott rendszerekben és videókártyákon is egyaránt jó a teljesítménye, ezért az ITS területén alkalmazható.

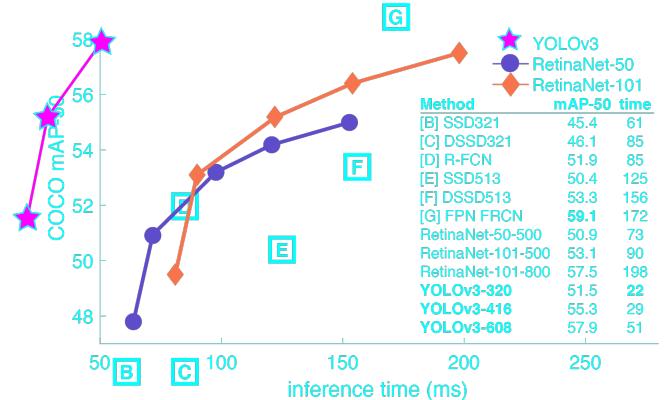


Fig. 5. YOLOv3 Performance

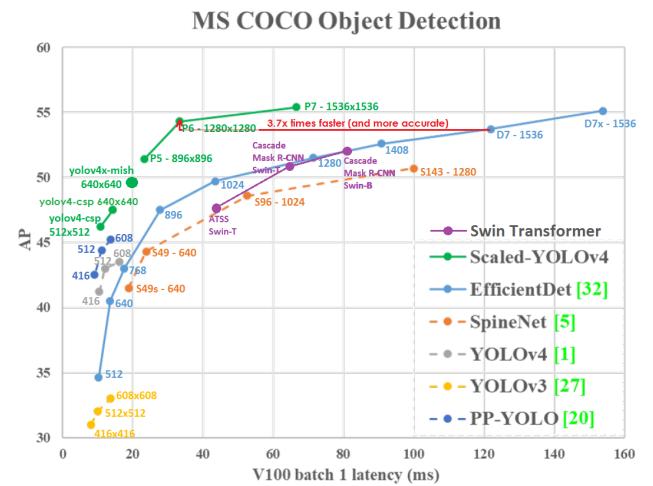


Fig. 6. YOLOv4 Performance

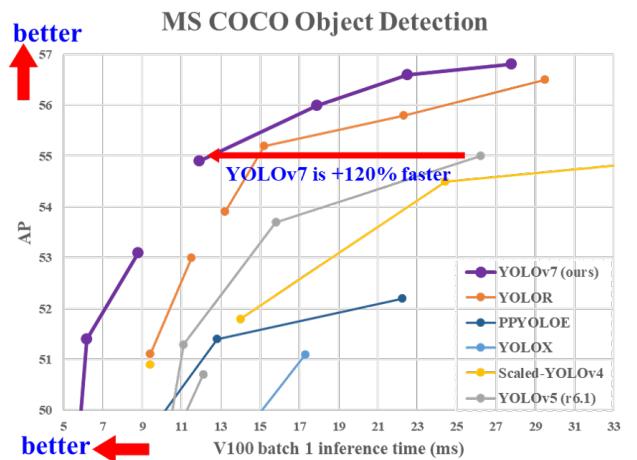


Fig. 7. YOLOv7 Performance

2.2 DeepSORT

Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT) a SORT algoritmus egy továbbfejlesztett változata. Az eredeti algoritmus sebességét törekednek növelni egy offline tanítási lépéssel. A betanított mély asszociációs metrika segítségével nagyban lecsökkentették az identitás vál-tásokat, és hosszabb elfedés után is követni tudja az objektumokat.

3 ADATHALMAZOK KIALAKÍTÁSA

A kutatás során saját adathalmazok kialakítására volt szükség. Az adatok begyűjtésére és eltárolására saját alkalmazást és keretrendszer fejlesztettük ki. A szoftver keretrendszer python nyelven írtuk meg, a forráskód ezen a linken megtalálható http://github.com/Pecneb/computer_vision_research. A fejlesztés során a következő programkönyvtárat használtuk OpenCV [Bradski 2000], Numpy [Harris et al. 2020], Pandas [pandas development team 2020], Scikit-Learn [Pedregosa et al. 2011], Matplotlib [Hunter 2007], SQLite [Hipp 2020], Joblib [Joblib Development Team 2020]. Az adathalmazokat SQLite adatbázisban és joblib fájlokban tároltuk el. Azért döntöttünk így, hogy két féle módon is eltároljuk az adathalmazokat, mert az SQL adatbázist univerzálisan bármilyen adatbáziskezelővel, vagy más programnyelvvel be lehet olvasni, viszont a joblib fájlokat sokkal gyorsabban be lehet tölteni pythonnal.

3.1 Adatstruktúra

Az adatstruktúrát SQL schema-ként, és python class-ként is definiáltuk.

```
CREATE TABLE IF NOT EXISTS objects (
    objID INTEGER PRIMARY KEY NOT NULL,
    label TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS detections (
    objID INTEGER NOT NULL,
    frameNum INTEGER NOT NULL,
    confidence REAL NOT NULL,
    x REAL NOT NULL,
    y REAL NOT NULL,
    width REAL NOT NULL,
    height REAL NOT NULL,
    vx REAL NOT NULL,
    vy REAL NOT NULL,
    ax REAL NOT NULL,
    ay REAL NOT NULL,
    vx_c REAL NOT NULL,
    vy_c REAL NOT NULL,
    ax_c REAL NOT NULL,
    ay_c REAL NOT NULL,
    FOREIGN KEY(objID) REFERENCES objects(objID)
);
CREATE TABLE IF NOT EXISTS metadata (
    historyDepth INTEGER NOT NULL,
    yoloVersion TEXT NOT NULL,
```

```
device TEXT NOT NULL,
imgsize INTEGER NOT NULL,
stride INTEGER NOT NULL,
confidence_threshold REAL NOT NULL,
iou_threshold REAL NOT NULL,
k_velocity REAL NOT NULL,
k_acceleration REAL NOT NULL
);
```

Minden követett objektum egyedi azonosítóval lett ellátva. Az objektumhoz tartozó detektálások külön táblába lett kiszervezve, ahol az *objID* idegen kulcsal kapcsoljuk az *objektumok* táblához. Egy objektumhoz az egyedi azonosítón kívül tartozik egy *label* amit a YOLO objektum detektálótól kap, ez lehet pl. autó, személy, teherautó, stb. Az objektumokhoz tartozó detektálások tartalmazzák a képkocka számát, amikor a detektálás történt, a konfidenciát, hogy mennyire biztos az objektumfelismerő a hozzárendelt *label*-ben, az objektum *X*, *Y* kordinátáját, az objektum szélességét *width* és magasságát *height*, sebességét *vx*, *vy* és gyorsulását *ax*, *ay*, amik a deepSORT által kalkulált értékek, így még külön a koordinátákból kiszámolt *v_{x_c}*, *v_{y_c}* sebességet és *a_{x_c}*, *a_{y_c}* gyorsulást is eltároltuk. Ezek mellett még a konfigurációs adatokat is külön táblában tároljuk, hogy később meg lehessen ismételni a detektálást. A koordinátákat a videó méretének megfelelően leskálázzuk 0 - 1 értékek köré. Ha a videó kép szélesség *w*, magasság *h*, akkor a képarány *r* = $\frac{w}{h}$, és az eltárolt koordináták *X* = $\frac{x_0}{w} * r$, *Y* = $\frac{y_0}{w} * r$, *v_x* = $\frac{v_{x0}}{w} * r$, *v_y* = $\frac{v_{y0}}{w} * r$, *a_x* = $\frac{a_{x0}}{w} * r$, *a_y* = $\frac{a_{y0}}{w} * r$, *v_{x_c}* = $\frac{v_{xc0}}{w} * r$, *v_{y_c}* = $\frac{v_{yc0}}{w} * r$, *a_{x_c}* = $\frac{a_{xc0}}{w} * r$, *a_{y_c}* = $\frac{a_{yc0}}{w} * r$.

3.2 Objektumdetektálás

Az objektumdetektáláshoz a fennt említett YOLO modellt használtuk. Kutatásunk kezdetekor, a YOLO 4-es verziójával kezdtük dolgozni, de később átváltottunk a jobb pontosságot és sebességet igérő 7-es verzióra.

3.2.1 YOLOv4. YOLO 4-es verzióját, C-ben implementálták. Hogy fel tudjuk használni, írnunk kellett egy python API-t, ami meg tudtunk hívni a deketáló programunkban.

3.2.2 YOLOv7. A YOLOv7 viszont már python-ban implementálták amihez már sokkal könnyebb volt API-t programozni és használni. Emellett, gyorsaságban és pontosságban is felülmúlt a 4-es verziót (lásd 7).

3.3 Objektumkövetés

Ahhoz, hogy trajektóriák alapján tudunk szabájosságokat felismerni a forgalomban, pontos objektumkövetésre volt szükségünk. Eleinte saját objektumkövető algoritmust használtunk, ami deketálások euklideszi távolsága alapján próbálta meg követni az objektumokat. Ezzel az volt a gond, hogy hosszabb kitakarás után nem találta meg az objektumot, így egy új objektumnak számított, ami a kép közepéből bukkant fel. Ennek a problémának a kiküszöbölésére próbáltuk ki a DeepSORT algoritmust.

3.3.1 DeepSORT. A DeepSORT algoritmus pythonban implementált változatát integráltuk a mi programunkba.

4 KLASZTEREZÉS

A klaszterezés segítségével lehet az adathalmazból alóállítani a klasszifikáció alapjául szolgáló klasszokat. Ahhoz, hogy az a rengeteg trajektoriából és detektálásból számunkra felhasználható információ keletkezzen, meg kell határoznunk feature vectorokat, amik a trajektoriákra jellemző értékeket tartalmaznak. Ebben a feature térben fogja a klaszterező algoritmus megtalálni az egymáshoz közeli, hasonló trajektoriákat.

4.1 Adattisztítás

A klaszterezés előtt a nyers adatokat fel kell dolgoznunk, hogy az esetleges hibás, zajos detektálások, trajektoriák miatt kapjunk fals klasztereket. Az objektum detektálás és követés nem tökéletes, rossz fényviszonyok, hosszabb eltakarások miatt a trajektoriák megszakadhatnak, ezért ki kell választani az egyben maradt trajektoriákat. Három szűrő algoritmust futtattunk az adathalmazon. Elsőnek a trajektoriák belépő és kilépő pontjainak az euklideszi távolsága alapján szűrtünk. Majd a kép széleit meghatározzuk min max kiválasztással, és azokat a trajektoriákat válasszuk ki amiknek a szélektől meghatározott távolságra vannak a belépő és kilépő pontjaik.

4.1.1 DeepSORT pontatlanság. Kutatásunk során azt tapasztaltuk, hogy a DeepSORT és a YOLO pontatlanságai felerősítik egymást. A YOLO hajlamos néha táblákat vagy rendőr lám-pákat autóknak nézni, és ekkor a DeepSORT is elkezdi követni. Egy olyan hibáját is felfedeztük a DeepSORT-nak, hogy egy objektumról áttapad a követés egy másik objektumra, ami fals trajektoriákat hoz létre. A DeepSORT-nak lehet finomhangolni a paramétereit, ami nem bizonyult akkora javulásnak, ezért útóból szűréssel kellett korrigálnunk ezt a hibát. Az algoritmus véig iterál a trajektoriák pontjain és kiszámítja az egymást követő detektálások euklideszi távoltságát, és ha egy küszöbérték felett vannak akkor eldobjuk a trajektoriát. A következő képeken láthatók a klaszterek szűrés előtt és után (lásd 8 9).

4.2 Feature vektorok

Klaszterezéshez 4 és 6 dimenziós feature vektorokat használtunk. A 6 dimenziós vektorokat a DeepSORT hibájának a kiszűrésére hoztuk létre, felépítésük a következő [belépő x,y középső x,y kilépő x,y], de a kifejlesztett szűrő hatékonyabbnak bizonyult, és a kevesebb dimenzió is előnyt jelent, ezért maradtunk a 4 dimenziós feature vektor mellett, aminek a felépítése: [belépő x,y kilépő x,y].

4.3 Klaszterezési algoritmusok

Klaszterezéshez több fajta algoritmust teszteltünk. A legjobb eredményeket az OPTICS (Ordering Points To Identify the Clustering Structure) [Ankerst et al. 1999] algoritmus adta. Aminek az eredményei fenti képeken látható (lásd. 8). OPTICS-on kívül leteszteltük a KMeans, DBSCAN és

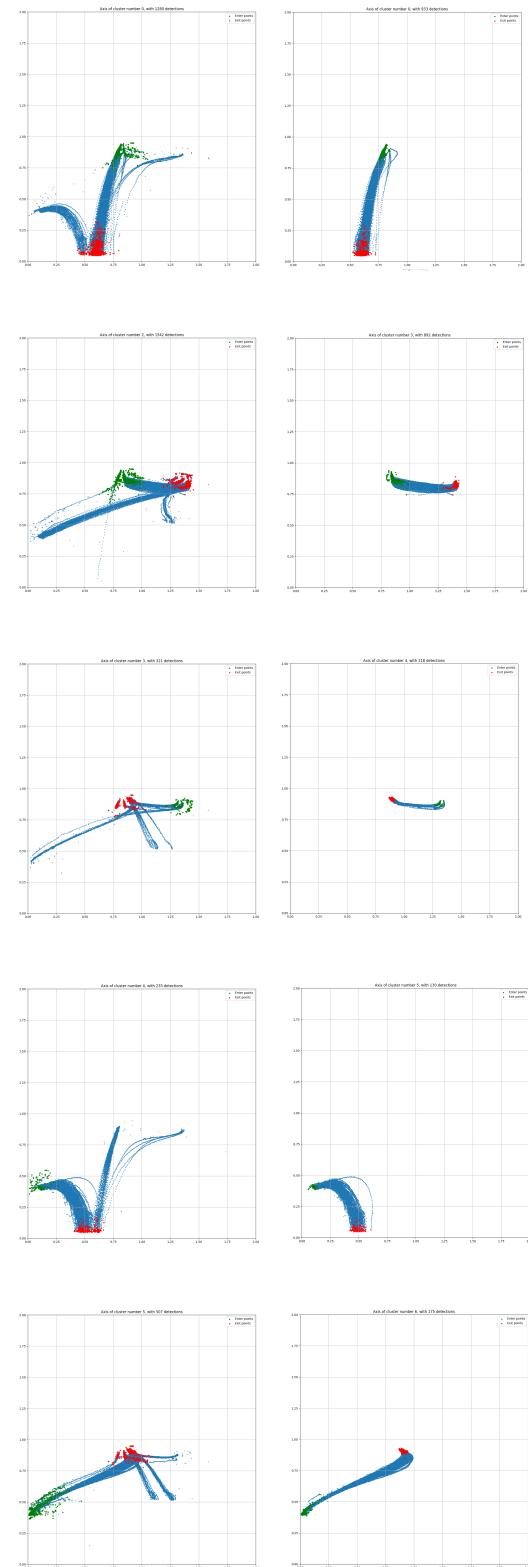


Fig. 8. Klaszterezés szűrő
, Vol. 1, No. 1, Article . Publication date: April 2023.

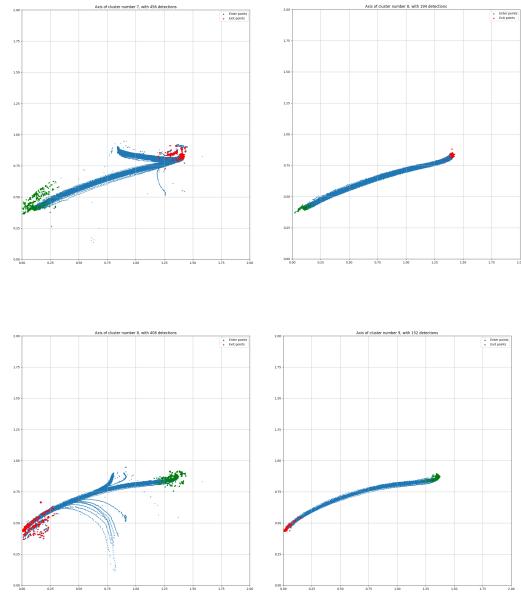


Fig. 9. Klaszterezés szűrő

BIRCH algoritmusokat. KMeans algoritmus egyenlő variációjú csoportokba osztja a mintákat úgy, hogy az *inercia* kritériumot csökkentse. A KMeans egy fontos paramétere a $n_{clusters}$, amivel a várható klaszterek számát kell megadni. Ennek a paraméternek a meghatározására próbálkoztunk különböző metrikákat felhasználni, hogy automatizálható legyen a klaszterezési lépés. Ezek a metrikák a Silhouette Coefficient [Rousseeuw 1987], Calinski-Harabasz Index [Caliński and JA 1974] és Davies-Bouldin Index [Davies and Bouldin 1979]. Elbow diagramok segítségével próbáltuk eldönteni, hogy milyen értéket érdemes adni az $n_{clusters}$ paraméternek lásd 10. A mérőszámok konzisztensen alacsony értéket adtak, egy négy ágyú kereszteződésnél, ahol jóval több klaszterbe sorolhatók a trajektóriák. A KMeans használatát ezért elvetettük.

A BIRCH algoritmus sokszor egybevon klaszter be- meneteket vagy kimeneteket, ami miatt több más irányból jövő, vagy több más irányba kilépő objektumokat sorol azonos klaszterekbe. *threshold* paraméterrel lehet a klaszterek méretét szabályozni, amivel javítható az egybevolt klaszterek száma, a kutatás során futtatott tesztek alapján, még így is az OPTICS adta a legtisztább klasztereket. DBSCAN algoritmus sokban hasonlít az OPTICS-hoz. A *max_eps* paraméter helyett, ami egy távolság tartományt ad meg, az *eps* paraméter pontos távolságot adja meg, hogy két minta egymás szomszédságában van-e vagy sem, ez egy éles határ, ezért az OPTICS könnyebben finomhangolható.

4.4 Paraméterek kiválasztása

A megfelelő paraméterek kiválasztása a klaszterezéshez igen fonzosnak bizonyult. Ezt a gyűjtött adathalmazokon kézzel

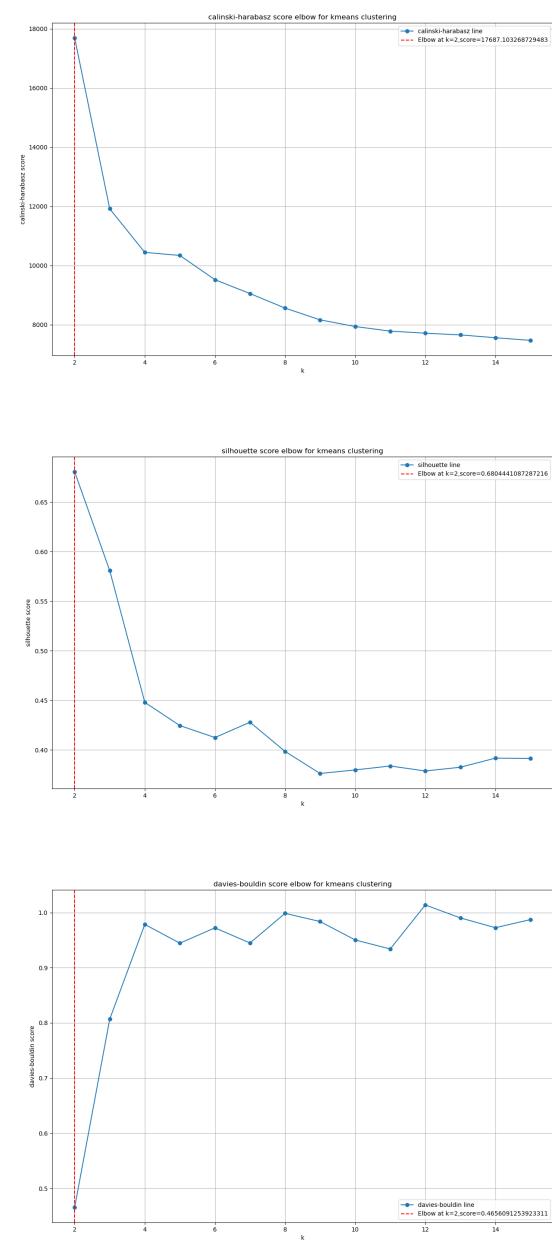


Fig. 10. Elbow diagramok

kellett finomhangolnunk. A halmazok minimum számoságát a *min_samples* paraméterrel lehet szabályozni, a pontok egymástól való távolságának felső határát *max_eps*-el lehet megadni. Az távolság kiszámítására használt méthódust *metric*-el lehet megadni. A *xi* paraméterrel az elérési plot minimum meredekségét lehet megadni. ami a klaszterek határát szabja meg. Az adathalmazra alkalmazható

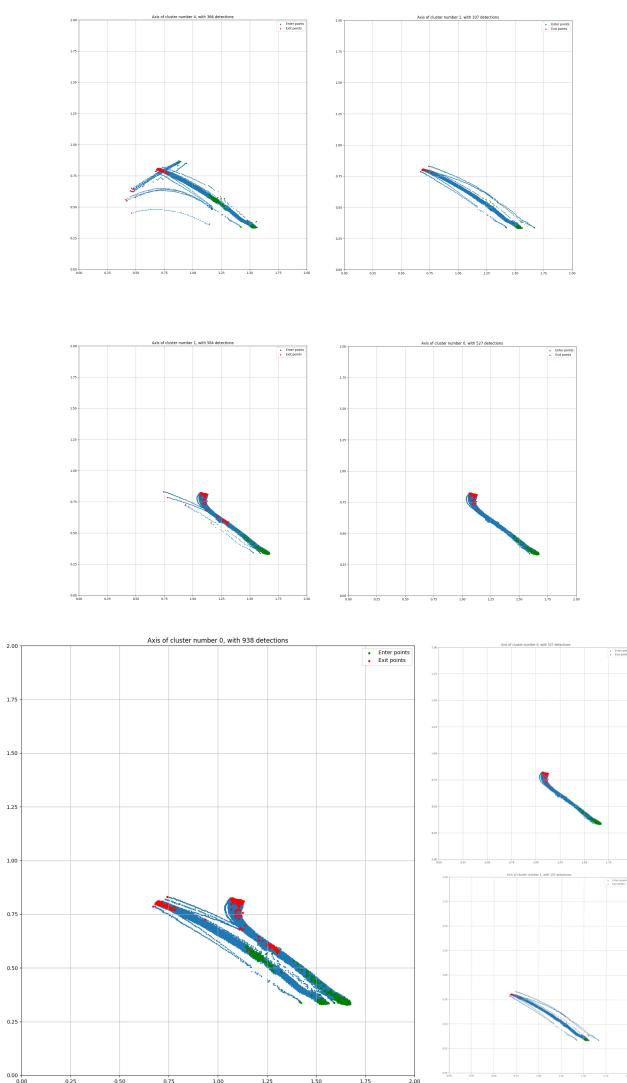


Fig. 11. KMeans, BIRCH, DBSCAN vs OPTICS

megfelelő paramétereket nem tudtuk generalizálni, kézzel kellett finomhangolnunk. A plotokon látható klaszterek megtalálásához $\text{min_samples} = 50$, $\text{max_eps} = 0.1$, $\text{metric} = \text{'minkowski'}$ és $xi = 0.15$ paramétereket használtunk.

5 KLASSZIFIKÁCIÓ

5.1 Multiclass

A több osztályos klasszifikálás egy olyan feladat, amikor több mint 2 osztály van, és minden feature vektor csak egy osztályba tartozhat. Ez kevesebb osztály számánál jó eredményt adhat, de a mi esetünkben ahol 10-15 osztály is lehet, ami azt jelenti, hogy nem lehet elérni nagy pontosságot. Ennyi osztály közül néhányszor pontosan eltalálni melyik osztályba tartozik egy trajektória.

Bellevue Newport

Metrics	Balanced	Top 1	Top 2
KNN	90.57%	95.50%	99.12%
GNB	63.92%	73.25%	90.10%
MLP	58.49%	81.93%	89.43%
SGD Modified Huber	45.43%	66.39%	83.54%
SGD Log Loss	40.41%	60.70%	79.69%
SVM	77.87%	89.29%	96.61%
DT	90.95%	93.64%	95.06%

Table 1. Feature vektor verzió 1

Average Accuracy
Feature Vector v1

Metrics	Balanced	Top 1	Top 2
KNN	94.16%	96.71%	99.46%
SVM	81.65%	90.82%	98.05%
DT	93.11%	94.88%	96.03%

Table 2. Feature Vektor verzió 1

5.2 Binary

A bináris klasszifikáció a többklasszossal szemben, csak 2 osztály között dönt. 2 osztály között sokkal pontosabban el lehet dönten, hogy a feature vektor melyikbe tartozik.

5.3 OneVsRest

Binráris klasszifikációs modellek kombinációjából, egy több osztályos modellt állítottunk össze, ami átlagban 90%-os feletti pontossággal tudta előre jelezni, hogy mely klaszterbe tartozik a jármű.

5.4 Machine Learning modellek

Kutatatásunk során több féle machine learning modellt teszteltünk: KNN (KNearestNeighbors), GNB (GaussianNaiveBayes), MLP (MultiLayerPerceptron), SGD (StochasticGradientDescent), SVM/SVC (SupportVectorMachine/SupportVectorClassifier) [Chang and Lin 2011], DT (DecisionTree) [Breiman et al. 1984]. Ezekből a GNB, MLP és SGD nem adott jó eredményeket ami látható az alábbi táblázatban 1, az eredmények megismétlődtek későbbi tesztekben, ezért ezeket a modellek nem tárgyaljuk. A legjobb eredményeket a KNN adta minden esetben 90% felett teljesített. A második legjobb a DecisionTree lett, ami átlagban balanced accuracy-ban az SVM felett teljesített, és Top 1 accuracyban is csakis tizedekkel maradt le a 7. feature vektor használatakor, az 1. verzióval 4%-al jobban teljesített. A mérések eredményei a 2. és 3. táblázatban láthatók.

5.5 Feature vektorok

Ahogyan klaszterezésnél is, fontos meghatározni egy olyan feature vektort, ami a legjobban jellemzi a trajektoriát. Ebben a papírból két fajta vektor verziót fogunk tárgyalni, amik a tesztek során a legjobban teljesítettek. Az egyik a legelső

Average Accuracy Feature Vector v7			
Metrics	Balanced	Top 1	Top 2
KNN	92.08%	95.61%	98.66%
SVM	88.72%	93.86%	98.92%
DT	89.46%	93.30%	94.55%

Table 3. Feature Vektor verzió 7

verziót amit kipróbáltunk, a másik a hetedik verzió, ami jobban teljesített mint az összes többi 2-6 verziót.

Az első verzió. Felépítése a következő $[x_0, y_0, v_{x_0}, v_{y_0}, x_m, y_m, x_l, y_l, v_{x_l}, v_{y_l}]$, ahol m index jelöli az időben középen elhelyezkedő detektálást, az l index pedig az utolsó, legfrissebb detektálást jelöli. Ez a vektor jól reprezentálja a valós idejű futás közben keletkező trajektoriákat, mert egyszerre több száz detektálást objektumonként nem lehet eltárolni a memórában, hanem egy meghatározott méretű buffert kell alkalmazni, aminek mi 15 vagy 30 detektálást adtunk. Az adathalmazban eltárolt trajektoriák ennél a buffernél több detektálást tartalmaznak, ezért az első verzióról a trajektoriát k részre osztottuk, így egy trajektória szelet mérete $s_n = n_d/k$, ahol n_d a detektálások számossága a trajektoriában. Ezekből a szeletekből képeztük az egyes feature vektorokat.

A hetedik verzió. Felépítése $[x_0 * w_1, y_0 * w_2, v_{x_0} * w_3, v_{y_0} * w_4, x_l * w_5, y_l * w_6, v_{x_l} * w_7, v_{y_l} * w_8]$. Ennél a verzióval használtunk súlyokat, ahol $w_1 = 1$, $w_2 = 1$, $w_3 = 100$, $w_4 = 100$, $w_5 = 2$, $w_6 = 2$, $w_7 = 200$, $w_8 = 200$. Mivel a sebességek két nagyságrenddel kisebbek mint a koordináták, ezért felszoroztuk őket 100-as súlyokkal. Hogy a 15-30 detektálás nagyságú bufferekben nagyobb hangsúlyt kapjanak a legfrissebb koordináták és sebességek, így azok 2-es és 200-as szorzót kaptak.

Teszt eredmények. Azt mutatják, hogy az utóbbi verzió magasabb pontosságokat produkált (lásd ??).

5.5.1 Adatdúsítás. Hogy minél pontosabban reprezentáljuk a valód idejű futást, a gyűjtött trajektoriákból nem trajektoriánként egy füre vektort generáltunk. Ezeknek a számoságát a feature vektort felépítő algoritmusban szabtuk meg.

5.6 Pontosság mérése

A pontosság mérésére háromféle metrikát használtunk.

Accuracy Score. Ha \hat{y}_i az i . minta predikciója és y_i a hozzá-tartozó valódi érték, akkor az eltalált predikciók és összes predikció hányadosa, amit így lehet leírni:

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i) \quad (1)$$

Balanced Accuracy. amit ezért használtunk, hogy az adathalmaz kiegyensúlyozatlansága miatt ne kapunk fals

pontosságot. Ha minden osztályra egyenlően jól teljesít a klasszifikációs modellünk, akkor a sima Accuracy-t kapjuk vissza. Ha a teszt adathalmaz kiegyensúlyozatlansága miatt az egyik osztálynak jobb a pontossága mint egy másiknak, akkor ezt az értéket elosztja a számával. Ha az y_i a valódi értéke az i . mintának, és w_i a hozzáartozó súly, akkor ezt a súlyt a következőképpen korrigáljuk:

$$\hat{w}_i = \frac{w_i}{\sum_j 1(y_j = y_i) w_j} \quad (2)$$

ahol $1(x)$ a karakterisztikus függvény. Adott a \hat{y}_i peridkió az i . mintának, így a balanced accuracy-t így definiálhatjuk:

$$\text{balanced-accuracy}(y, \hat{y}, w) = \frac{1}{\sum \hat{w}_i} \sum_i 1(\hat{y}_i = y_i) \hat{w}_i \quad (3)$$

Top-K Accuracy. az Accuracy Score egy generalizált változata. A különbség az, hogy a predikció akkor számít igaznak, ha beletartozik a k legmagasabb valószínűségű predíciók közé. Ha $\hat{f}_{i,j}$ a i . mintának a j . legmagasabb predikciója, és y_i a hozzáartozó valódi predikció, akkor az eltalált predikciók és az összes minta hányadosát így lehet definiálni:

$$\text{top-k accuracy}(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \sum_{j=1}^k 1(\hat{f}_{i,j} = y_i) \quad (4)$$

ahol k a megengedett találhatások száma, és $1(x)$ a karakterisztikus függvény.

5.6.1 Adathalmaz szétválasztás. A pontosság méréséhez el kell választanunk egy teszt adathalmazt a tanító adathalmaztól, mivel ha azon az adathalmazon tesztelünk, amin tanítottunk akkor tökéletes pontosságot kapnánk eredményül. Ezt Overfitting-nek hívják. A szétválasztást egy általunk implementált algoritmus végzi, ami véletlen szám generátort használ a minták kiválasztásához. Az algoritmusnak meg lehet adni paraméterként a tanító adathalmaz méretét, és egy seed értéket, ami azért fontos, hogy meg lehessen ismételni a szétválasztást.

5.6.2 Cross Validation. A túltanítás elkerülése érdekében, alkalmaztunk egy elterjedt metódust a cross-validációt. ...

5.6.3 Teszthalmazos validáció. Hogy meggyőződjünk arról, hogy biztosan nem tanítottuk túl a modelltunket, egy olyan teszt adathalmazon is le kell tesztelnünk, amit nem használtunk fel cross-validáció alatt.

5.6.4 Modellek tárolása. A betanított modelleket joblib file-ként tároltuk el, amit később python-nal tudunk betölteni.

REFERENCES

- D. Anguita, Luca Ghelardoni, Alessandro Ghio, L. Oneto, and Sandro Ridella. 2012. The 'K' in K-fold Cross Validation. In *The European Symposium on Artificial Neural Networks*.
- Mihai Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Rec.* 28, 2 (jun 1999), 49–60. <https://doi.org/10.1145/304181.304187>
- G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

- L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. 1984. Classification and Regression Trees.
- Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. 2010. The Balanced Accuracy and Its Posterior Distribution. In *Proceedings of the 2010 20th International Conference on Pattern Recognition (ICPR '10)*. IEEE Computer Society, USA, 3121–3124. <https://doi.org/10.1109/ICPR.2010.764>
- Tadeusz Caliński and Harabasz JA. 1974. A Dendrite Method for Cluster Analysis. *Communications in Statistics - Theory and Methods* 3 (01 1974), 1–27. <https://doi.org/10.1080/0361092740827101>
- Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2 (2011), 27:1–27:27. Issue 3. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- David L. Davies and Donald W. Bouldin. 1979. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1, 2 (April 1979), 224–227. <https://doi.org/10.1109/TPAMI.1979.4766909>
- Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) (KDD'96). AAAI Press, 226–231.
- Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Richard D Hipp. 2020. SQLite. <https://www.sqlite.org/index.html>
- J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Joblib Development Team. 2020. *Joblib: running Python functions as pipeline jobs*. <https://joblib.readthedocs.io/>
- The pandas development team. 2020. *pandas-dev/pandas: Pandas*. <https://doi.org/10.5281/zenodo.3509134>
- Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho. 2017. Chapter 8 - Big Data collision analysis framework. In *Intelligent Vehicular Networks and Communications*, Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho (Eds.). Elsevier, 177–184. <https://doi.org/10.1016/B978-0-12-809266-8.00008-9>
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- Luca Rossi, Andrea Ajmar, Marina Paolanti, and Roberto Pierdicca. 2021. Vehicle trajectory prediction and generation using LSTM models and GANs. *PLOS ONE* 16, 7 (07 2021), 1–28. <https://doi.org/10.1371/journal.pone.0253868>
- Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
- Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (jul 2017), 21 pages. <https://doi.org/10.1145/3068335>
- Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2022. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696* (2022).
- Nicolai Wojke and Alex Bewley. 2018. Deep Cosine Metric Learning for Person Re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 748–756. <https://doi.org/10.1109/WACV.2018.00087>
- Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada) (SIGMOD '96). Association for Computing Machinery, New York, NY, USA, 103–114. <https://doi.org/10.1145/233269.233324>