Járművek trajektóriájinak 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 keretrendszert és a felmerülő probélmák megoldásait, a gyakorlatban balesetek megelőzésére, renitens viselkedések kiszűrésére és forgalomirányító renszerek támogatásá-ra lehet használni. A kutatásban egy trajektória osztályozó módszert ismertetünk, amely objektumdetektálás és objektumkövetés segítségével elemezi a közlekedési csomópontokban elhaladó járművek mozgását. A mozgásuk alapján klaszterezi a trajektóriákat, majd gépi tanulás segítségével predikciót ad az újonnan belépő járművek kilépési pontjára. A módszerhez 6 különböző közlekedési csomópontban készített saját videó adatbázisunkat használtuk fel. A tesztelt klaszterezési mód-szerek közül (OPTICS, BIRCH, KMeans, DBSCAN) az OPTICS algoritmus bizonyult legjobbnak trajektórák klaszterezésére. Összehasonlí-tottunk több különböző klasszifikációs módszert a legpontosabb predikció eléréséhez, amelyek: KNN, SVM, GP, DT, GNB, MLP, SGD. A tanul-mányban bemutatott eljárások közül az SVM adta a legpontosabb 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.

ACM Reference Format:
Péter Bence Mérnökinformatika BSc 6. félév, Dr. Horváth András, and Agg Áron PhD hallgató. 2023. Járművek trajektóriájinak előrejelzése machine learning modellekkel. 1, 1 (March 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Contents

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 szolgáni 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 iránytuk a kereslet. E kutatási terület kiforratlanságából 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 módszertant és keretrendszert 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 konvolúciósl 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 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 6 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. 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 klasszos klasszifikációs modelt kapunk. Minden bináris modelnél, egy klassz 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 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-Távú 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 felülmúlja sebességben és pontosságban a modern konvolúciós hálókat (lásd 1 2 3). Beágyazott rendszerekben és videókártyá-kon is egyaránt jó a teljesítménye, ezért az ITS területén alkalmazható.

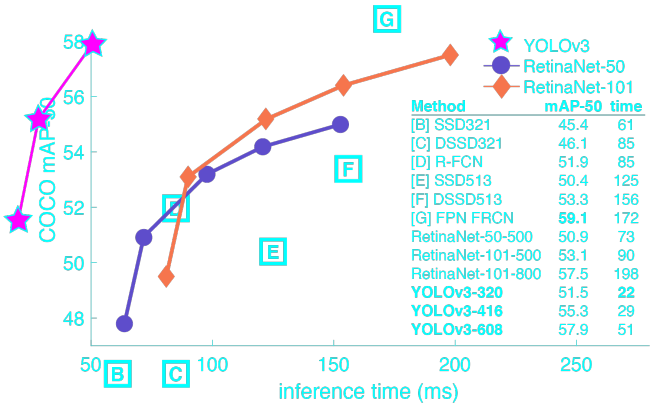

Fig. 1. YOLOv3 Performance

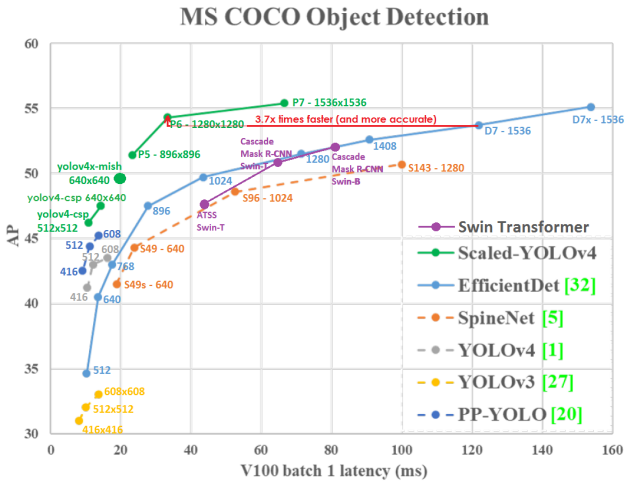

Fig. 2. YOLOv4 Performance

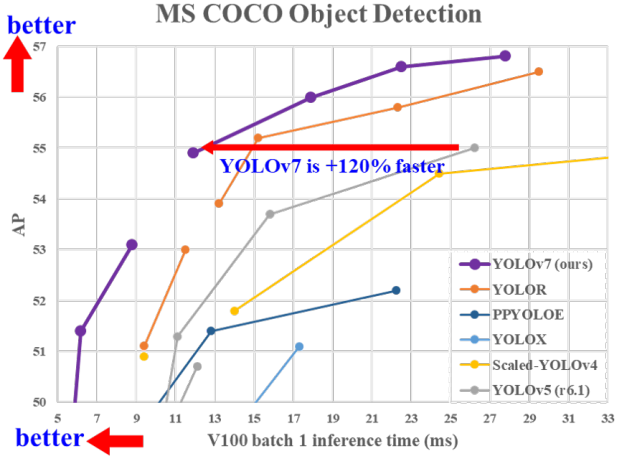

Fig. 3. YOLOv7 Performance

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 keretrendszert fejlesztettünk ki. A szoftver keretrendszert python nyelven írtuk meg, a forráskód ezen a linken megtatlálható http://github.com/Pecneb/computer_vision_research. A fejlesztés során a következő programkönyvtárakat 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 töletni 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,
    FOREIGN KEY(objID) REFERENCES objects(objID)
);
CREATE TABLE IF NOT EXISTS metadata (
    historyDepth INTEGER NOT NULL,
    futureDepth 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
);
```

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 tartózó 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 és magasságát, sebességét és gyorsulását. Ezek mellett még a konfigurációs adatokat is külön táblában tároljuk, hogy később meg lehessen ismételni a detektálást.

3.2 Objektumdetektálás

Az objektumdetektáláshoz a fennt említett YOLO modellt haszáltuk. Kutatásunk kezdetekor, a YOLO 4-es verziójával kezdtünk dolgozni, de később átváltottunk a jobb pontosságot és sebességet ígérő 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 felülmúlta a 4-es verziót (lásd 3).

3.3 Objektumkövetés

Ahhoz, hogy trajektóriák alapján tudjunk 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 alőállítani a klasszifikáció alapjául szolgáló klasszokat. Ahhoz, hogy az a rengeteg trajektóriából és detektálásból számunkra felhasználható információ keletkezzen, meg kell határoznunk feature vectorokat, amik a trajektóriákra jellemző értékeket tartalmaznak. Ebben a feature térben fogja a klaszterező algoritmus megtalálni az egymáshoz közeli, hasonló trajektóriákat.

4.1 Adattisztítás

A klaszterezés előtt a nyers adatokat fel kell dolgoznunk, hogy az esetleges hibás, zajos detektálások, trajektóriák miatt kapjunk fals klasztereket. Az objektum detektálás és követés nem tökéletes, rossz fényviszonyok, hosszabb eltakarások miatt a trajektóriák megszakadhatnak, ezért ki kell választani az egyben maradt trajektóriákat. Három szűrő algoritmust futtattunk az adathalmazon. Elsőnek a trajektóriák belépő és kilépő pontjainak az euklideszi távolsága alapján szűrtünk. Majd a kép széleit meghatározzuk min max kiválasztással, és azokat a trajektórákat választjuk 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 lámpákat 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 trajektóriákat hoz létre. A DeepSORT-nak lehet finomhangolni A paramétereit, ami nem bizonyult akkora javulásnak, ezért útólagos szűréssel kellett korrigálnunk ezt a hibát. Az algoritmus végig iterál a trajektóriák pontjain és kiszámítja az egymást követő detektálások euklideszi távolságát, és ha egy küszöbérték felett vannak akkor eldobjuk a trajektóriát. A következő képeken láthatók a klaszterek szűrés előtt és után. A koordinátákl plottolása mellett, hisztogramot is készítettünk, hogy lássuk a trajektóriák hosszának változását.

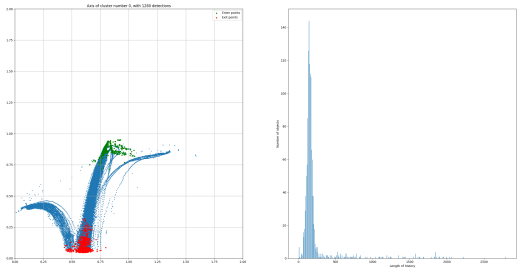

Fig. 4. KlaszterV1 1

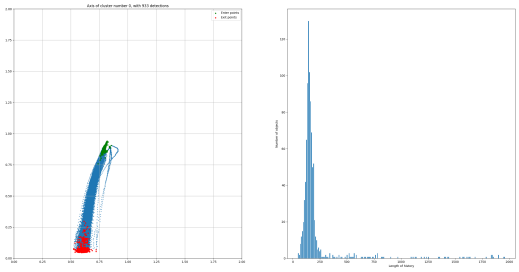

Fig. 5. KlaszterV2 1

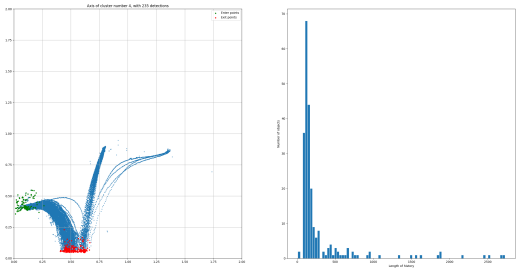

Fig. 6. KlaszterV1 2

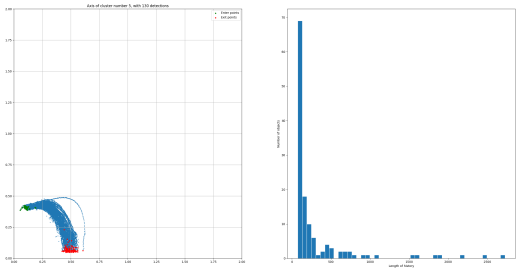

Fig. 7. KlaszterV2 2

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

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. 4 5 6 7). Ez az algoritmus megkeresi a nagy sűrűségű halmazokat az adathalmazban, majd fokozatossan bővíti őket.

4.4 Paraméterek kiválasztása

A megfelelő paraméterek kiválasztása a klaszterezéshez igen fonzosnak bizonyult. Ezt a gyűjtött adathalmazokon kézzel kellett finomhangolnunk. A halmazok minimum számosságát a $min\_samples$ paraméterrel lehet szabályozni, a pontok egymástól való távolságának felső határát max_eps-el lehet megadni. Az távolság kiszámítására használt methódust $metric$-el lehet megadni. A xi paraméterrel az elérési plot minimum meredekségét lehet megadni. ami a klaszterek határát szabja meg. Az adathalmazra alkalmazható megfelelő paramétereket nem tudtuk generalizálni, kézzel kellett finomhangolnunk. A plotokon látható klaszterek megtalálásához $min\_samples = 50$, $max\_eps = 0.1$, $metric =' minkowski'$ és $xi = 0.15$ paramétereket használtunk.

5 KLASSZIFIKÁCIÓ

5.1 Multiclass

A több klasszos klasszifikálás egy olyan feladat, amikor több mint 2 klassz van, és minden feature vektor csa egy klasszba tartozhat.

5.2 Binary

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

5.3 OneVsRest

Binráris klasszifikációs modellek kombinációjából, egy több klasszos modellt állítottunk össze, ami ... %-al pontosabb eredményt adott.

5.4 Machine Learning modellek

Kutatásunk során több klasszifikációs modellt is teszteltünk, ezek közül a két legjobban teljesítő algoritmust tárgyaljuk, KNN és SVM modellt. ...

5.5 Feature vektorok

Ahogyan klaszterezésnél is, fontos meghatározni egy olyan feature vektort, ami a legjobban jellemzi a trajektóriát. 7 fajta feature vektorral teszteltük a kiválasztott klasszifikációs modelleket, a legjobban az 1. és 7. verzió teljesített. ...

5.5.1 Adatdúsítás. Hogy minél pontosabban reprezentáljuk a valód idejű futást, a gyűjtött trajektóriákból nem trajektóriánként egy feture vektort generáltunk. Ezeknek a számosságát 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 hozzátartozó valódi érték, akkor az eltalált predikciók és összes predikció hányadosa, amit így lehet leírni:

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

Balanced Accuracy. amit ezért használtunk, hogy az adathalmaz kiegyensúlyozatlansága miatt ne kapjunk fals pontosságot. Ha minden klasszra egyenlően jól teljesít a klasszifikációs modellünk, akkor a sima *Accuracy*-t kapjuk vissza. Ha a teszt adathalmaz kiegyensúlyozatlansága miatt az egyik klassznak jobb a pontossága mint egy másiknak, akkor ezt az értéket elosztja a klasszok számával. Ha az y_i a valódi értéke az i. mintának, és w_i a hozzátartozó súly, akkor ezt a súlyt a következőképpen korrigáljuk:

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

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

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

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

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

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

5.6.1 Adathalmaz szétválasztás. A pontosság méréséhez el kell választanunk egy teszt adatahalmazt 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 modellünket, egy olyan teszt adathalmazon is le kell tesztelnünk, amit nem használtunk fel cross-validáció alatt.

5.6.4 Modellek tárolása. A betanított modelleket joblib fileként 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*.

Mihael 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).

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

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

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