Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Bíró Dávid

MESTERSÉGES INTELLIGENCIA ALAPÚ OSZTÁLYOZÁSI TECHNIKÁK AZ ALÁÍRÁS-HITELESÍTÉSBEN

KONZULENS

Dr. Kővári Bence

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló

Napjainkban a technológia fejlődésével, egyre fontosabb szerepet töltenek be az automatizált biometrikus hitelesítő rendszerek, mert ezektől gyorsabb és pontosabb eredmények várhatóak el. Az automatizált aláírás-hitelesítés is egy e rendszerek közül. Az aláírás-hitelesítés a legrégebbi a meglévő rendszerek közül. Több évtized kutatása van mögötte, de a probléma nehézségét jól mutatja, hogy még mindig nincs minden igényt kielégítő automatizált megoldás.

Két különböző automatizált aláírás-hitelesítési megközelítés ismert. Az egyik az úgynevezett online aláírás-hitelesítés, amelynél speciális és drága eszközöket használunk az aláírások rögzítésére. A másik az offline aláírás-hitelesítés, ahol csak az aláírás képével dolgozunk. A szakdolgozatomban ez utóbbi területre kell koncentrálnom.

Kutatásom során két eszköz használhatóságát vizsgáltam meg az offline aláírás-hitelesítésben ezek a Matlab és a Tensorflow. Mindkettő mesterséges intelligencia alapú osztályozási módszereket tett elérhetővé. Célom az volt, hogy ezen eszközök megoldásait kipróbáljam és a lehető legkisebb osztályozási hibaarányt elérjem.

Dolgozatomban kipróbálom az eszközök által nyújtott lehetőségeket az osztályozás elvégzésében, összehasonlítom ezek hatékonyságát a különböző bemeneti adathalmazokra. Összességében 20 aláíró egyenként 40 aláírásának, darabonként közel 150 jellemzője áll rendelkezésemre ebben.

Abstract

The automatic check systems are more and more important these days, since they are more efficient and reliable. The automatic signature check is one of these systems and it is one of the oldest systems of all. Scientist have been researching this system for decades even though there is no perfect solution for this challenge yet.

There are 2 different automatic signiture check systems at the moment. One of them is the online automatic signature check system where special and expensive tools are applied to record signatures. The other one is the off-line signature check system where the image of the signature is recorded. I have focused on the second one in my thesis.

While doing research I have analysed the use or 2 different applications. Both Matlab and Tensorflow are able to check signatures. Both Matlab and Tensorflow enable us to apply record systems based on artificial intelligence. I was going to try these solutions while achieving the highest possible accuracy.

This thesis describes classification based on Matlab and Tensorflow and compares efficiency in case of different input databases. There are 4O signatures from 2O signatories each and 15O features have been checked altogether.

1 Bevezetés

Az aláírás-hitelesítés az egyik legnagyobb múlttal rendelkező biometrikus azonosító, amely a mai napig vezető szerepet tölt be a különböző alternatív megoldások mellett, például ujjlenyomat-olvasók vagy a retinavizsgálók. Előnyei közé tartozik többek közt, hogy bármikor elvégezhető és nem igényel külső eszközt.

Az embernek az aláírása az egy nagyon fontos tulajdonsága, hiszen, ha belegondolunk, bármilyen ügyet is intézünk, végül a dokumentumokat az aláírásunkkal látjuk el. Ezért fontos, hogy el tudjuk dönteni egy dokumentumról, hogy eredeti aláírás van-e rajta az érintett személytől vagy egy hamisítvány. A törvény az aláírás hamisítást törvény által bünteti. Egy aláhamisított dokumentum akár komoly pénzbeli károkat is okozhat, például cégeknek.

Kétes dokumentumok vizsgálatát jelenleg törvényszéki írásszakértő végzi. Átlagemberként egy profi hamisítványt vizsgálva a tévedési arányunk 10% és 26% között mozog [1], míg egy szintén profi hamisítványokat vizsgáló szakértő tévedési aránya 0.5% és 7% közötti. [1] Ezekből is jól látszik, hogy a törvényszéki szakértők is hajlamosak tévedni, ezért nem haszontalan azt vizsgálni, hogy a hitelesítést eltudjuk-e végezni egy számítógépes rendszerrel, ami kisebb hibájú eredményt adna.

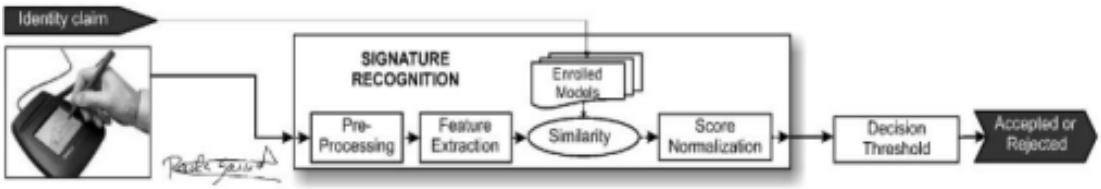
Két különböző automatizált aláírás-hitelesítési megközelítés ismert. Az egyik az úgynevezett online aláírás-hitelesítés, amelynél speciális és drága eszközöket használunk az aláírások rögzítésére. A másik az offline aláírás-hitelesítés, ahol csak az aláírás képével dolgozunk.

A feladatnak több nehézsége is van. Talán az első és legfontosabb, hogy az aláírások az egyes személyeknél nagyon eltérhetnek. A második a kevés mintaszám, természetesen egy írásszakértő is pontosabban eldönti egy aláírásról, hogy hamis-e, ha sok eredeti minta áll a rendelkezésére, de ez általában nincs így.

A szakdolgozatomban az offline aláírás hitelesítésre koncentrálok.

1.1 Az aláírás-hitelesítés folyamata

Ebben a fejezetben áttekintem az aláírás-hitelesítés folyamatát.



1-1.ábra Egy tipikus off-line aláírás-hitelesítő threshold osztályozóval [1]

1.1.1 Aláírások digitalizálása

A papírlapon lévő aláírásokat lapolvasóval vagy fényképező géppel lehet digitalizálni. A legtöbb megfigyelt rendszerben ez a digitalizáló lépés hiányzik, mert ezek a rendszerek már feltételezik, hogy az aláírások már digitalizálva vannak. [1]

1.1.2 Elő feldolgozás

Ennek során a képeket transzformációknak vetik alá. Erre azért van szükség, hogy a képeket egységesen tudja kezelni a feldolgozó algoritmus. A transzformációkat úgy kell végrehajtani, hogy minimális információ vesszen el. Ilyen transzformációk lehetnek például vágás, forgatás, zajszűrés, kicsinyítés-nagyítás stb. [1]

1.1.3 Jellemzők kinyerése

Itt történik az aláírások jellemzőinek a kinyerése ahol a képből a rendszer számszerűsített adatokat nyer ki. Ilyen adatok lehetnek pl. az alapvonalak kezdő koordinátái, az ékezetek száma, hurkok száma, nagysága stb.

1.1.4 Feldolgozás

A feldolgozási fázis eltér az előző fázisoktól, mivel itt nem egy képen dolgozunk, hanem már több aláírás képén. A kinyert jellemzőket itt párosítjuk össze, tehát itt rendeljük össze az azonos részeket akár több aláíráson keresztül. A bepárosított jellemzők után további számításokon át egyéb jellemzőket készítünk, esetleg normalizáljuk ezeket, hogy a következő fázis kényelmesebben tudjon ezekkel dolgozni. [1]

1.1.5 Osztályozás

Az osztályozásnál már rendelkezésünkre áll egy aláírást jellemző sorozat. Ennek segítségével kell betanítanunk az osztályozónkat, aminek ezek után el kell tudnia dönteni, minél kisebb hibával azt, hogy egy kapott aláírás hamisított-e vagy sem. Több típusú osztályozási technikát alkalmazhatunk itt pl. SVM, RBF, NN stb. [1]

1.2 Célkitűzés

Dolgozatomban az aláírás-hitelesítés egy részterületére fogok koncentrálni, mégpedig az utolsó osztályozási fázisra. Célom megvizsgálni, milyen eredményeket tudok elérni különböző osztályozási technikákkal. Ezzel párhuzamosan a vizsgált módszerek előnyeit és hátrányait felismerve, olyan saját konfigurációt próbálok létrehozni, amivel javíthatok az eredményeken.

Kísérleteimhez a Matlab és a TensorFlow eszközeit használom. Végül összehasonlítom a kapott eredményeket és levonom a következtetéseket.

1.3 Dolgozat felépítése

A következő fejezetben a munkám során felhasznált elméleti alapokat ismertetem. Bemutatok több osztályozási technikát úgymint MLP (2.2 Többrétegű perceptron (Multi-Layer Perceptron)), RBF (2.3 RBF (Radial Basis Function)), SVM (2.4 Szupport vektor gép (Support Vector Machine)). Röviden áttekintem ezek után az általam használt két eszköz a TensorFlow (2.6 TensorFlow bemutatása) és Matlab (2.5 Matlab eszköztárak bemutatása (toolboxes)) jellemzőit. A szakdolgozat közepén ismertetem a munkám során létrehozott konfigurációkat és az ezekhez vezető gondolatmeneteket. (3 Osztályozási módszerek) Végül bemutatom az elért eredményeket. (Eredmények)

2 Irodalom áttekintése

Ebben a fejezetben áttekintem a különböző általam használt osztályozási technikákat.

2.1 Osztályozási feladat

Egy mesterséges intelligencia alapú osztályozónak, azt kell tudni megmondania, hogy a bemenetére kapott adatok, mely osztályba tartoznak. Az ilyen típusú rendszereknek általánosan szükségük van az összes osztályból mintákra a tanuláshoz.

Az aláírás-hitelesítésben két osztályos osztályozásról beszélhetünk. Az egyik osztályba tartoznak az eredeti, a másikba a hamisított aláírások. Az aláírás-hitelesítésben az általánostól eltérő módon kell alkalmazni az osztályozási technikákat, mert itt egy olyan különleges eset áll fent, hogy nem rendelkezünk negatív mintákkal, vagyis hamisítványokkal. Tovább nehezíti a hatékony osztályozást a rendkívül alacsony mintaszám is.

Két megközelítést vizsgáltam meg. Első, hogy a feladatra úgy tekintek, mint egy anomáliadetektálásra, vagyis azt próbálom elérni, hogy az osztályozóm felismerje az eredeti minták tulajdonságait és így el tudja dönteni, hová tartozik az aktuálisan vizsgált aláírás. A másik, hogy megpróbálok az eredeti aláírásokból, mesterséges hamisítványokat létrehozni. Így már kétosztályos osztályozással dolgozhatom.

A szakdolgozatomhoz a Budapesti Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai Tanszéken fejlesztett off-line aláírás-hitelesítő rendszer kimeneti adatsorát használtam fel. Az automatizálás miatt valószínűsíthetően sok hiba került az adatokba, ezért 0%-os hibarátára nem számítottam. Ezen hibák az osztályozást nagyban befolyásolhatják. Nézzünk, egy példát a rendszerben van egy párosítási folyamat, ha ez két jellemzőt rosszul párosít össze, akkor ez eredményezheti azt, hogy két eredeti aláírás közül az osztályozó valamelyiket hamisítványnak tekinti.

A kapott adatsorban soronként vannak az aláírásokról a tulajdonságok. Aláírónként közel 150 jellemzőt tartalmazhat, természetesen nem minden aláíróra van,

ennyi, előfordul, ahol kevesebb, néhol több van, ez az aláírástól függ. Ezeket a jellemzőket 5 nagyobb csoportba sorolhatjuk:[2]

- ***Alapvonal jellemzők***: Az aláírásoknak általában több része van vezetéknév és keresztnév. Alapvonalnak azt az egyenest tekintjük, ami egy adott részre alulról ráilleszthető. Minden részhez egy alapvonalat rendelünk, amelyek a következő tulajdonságai lehetnek: kezdőpont, végpont, dőlésszög, hosszúság.

2-1. ábra Adott aláírás alapvonalai

- ***Hurokjellemzők***: A hurkoknak a zárt vonalakat tekintjük egy aláírásban. Jellemzői kerület, terület és egyéb származtatott jellemzők.
- ***Alakzatjellemzők***: Az aláírás elhelyezkedése a képen, milyen területtel rendelkezik stb.
- ***Ékezetjellemzők***: Gyakran szerepelnek ékezetek az aláírásokban, ezek elkülönített vonalszerű részei az aláírásoknak. Jellemzői hossza, íve, szöge, kezdete vége.
- ***Globális jellemzők***: Az aláírás globális jellemzői tartoznak ide, úgymint méret, elhelyezkedés, arányok stb.

Ezekből már nagyon jól látszik, hogy mennyire is különböznek az emberek aláírásai, nem véletlenül használják a mai napig is előszeretettel hitelesítésre. Ezen különbözőségek miatt nehéz olyan automatikus aláírás-hitelesítő rendszert létrehozni, ami minden típusú aláírásra kis hibaszázalékú eredményt adna.

2.2 Hibamutatók

Miután az előző részben megismerkedtünk az aláírás-hitelesítésben használatos osztályozás jellemzőivel, a következő lépés, hogy össze tudjuk hasonlítani az egyes osztályozó konfigurációk teljesítményeit.

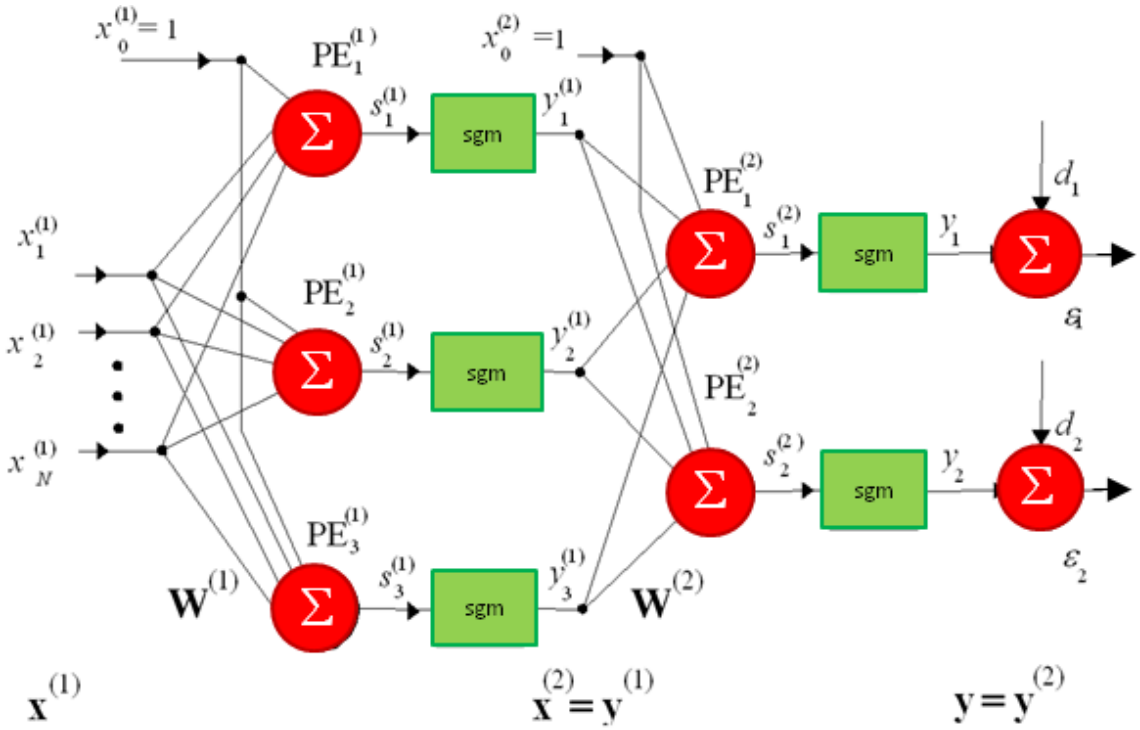
Az osztályozási feladatokban leggyakrabban használt hibamértékekkel dolgoztam:

- **FAR (False Acceptance Rate):** azon hamisítványok aránya, amelyeket az osztályozó rendszer eredetinek osztályozott.
  - FAR = $\frac{eredetinek\ tekintett\ hamisítványok\ száma}{összes\ hamisítvány\ száma} * 100\%$
- **FRR (False Rejection Rate):** azon eredeti aláírások aránya, amelyet az osztályozó rendszer hamisítottnak tekintett.
  - FFR = $\frac{hamisítványnak\ tekintett\ eredeti\ aláírások\ száma}{összes\ eredeti\ aláírás\ száma} * 100\%$
- **AER (Average Error Rate):** átlagos hibamérték, a FAR és az FRR átlagából származtatható.
  - AER = $\frac{FAR+FRR}{2} * 100\%$

A dolgozatomban az AER hibamértéket használtam az osztályozók összehasonlítására.[1]

2.3 Többrétegű perceptron (Multi-Layer Perceptron)

A többrétegű perceptron (multi-layer perceptron, MLP) a gyakorlati feladatok megoldásánál talán a leggyakrabban alkalmazott hálózat-architektúra. Az MLP rétegekbe szervezett neuronokból áll, ahol annak biztosítására, hogy a hálózat kimenete a súlyok folytonos, differenciálható függvénye legyen, a neuronok differenciálható kimeneti nem linearitással rendelkeznek.



2-2.ábra A többrétegű perceptron felépítése a hibaszámítással[3]

Az ábra egy két aktív réteget tartalmazó hálózatot mutat, amelyben az első aktív rétegben − a rejtett rétegben − három, a második aktív rétegben − jelen esetben, a kimeneti rétegben − két processzáló elem található. A hálózat tehát egy többrétegű előrecsatolt hálózat.[3]

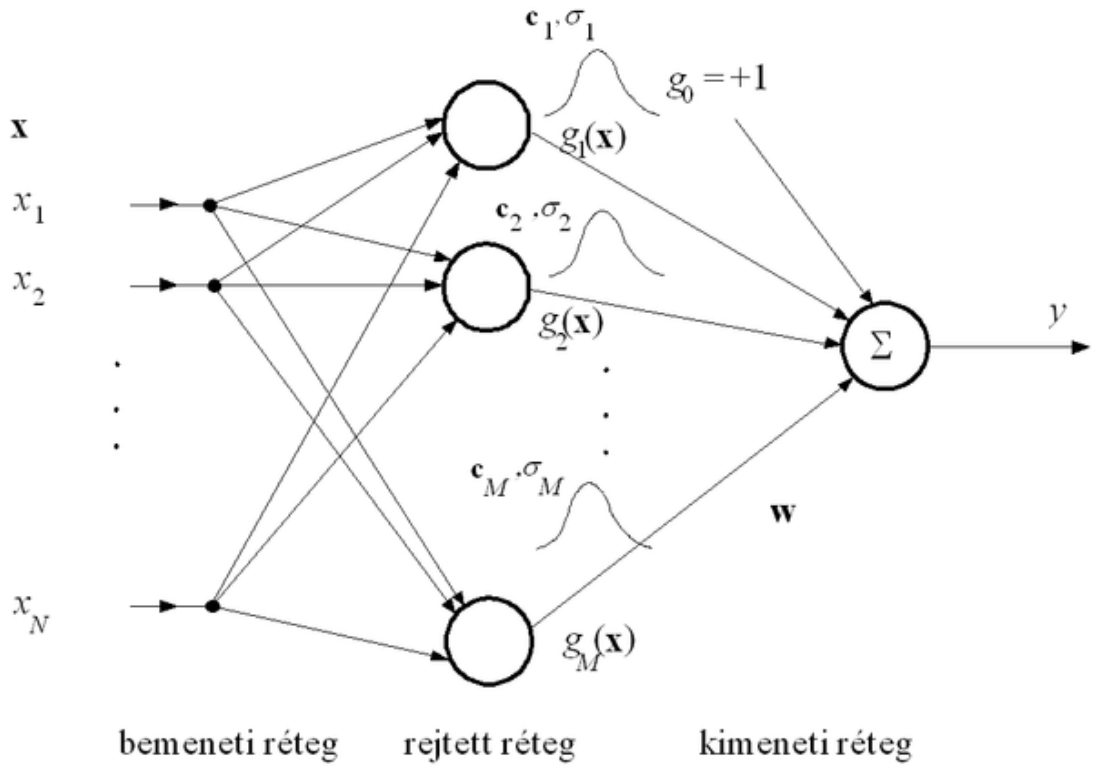
Az MLP tanítása ellenőrzött tanítás, ahol a hálózat kimenetén értelmezett hiba felhasználásával határozhatjuk meg a kritériumfüggvény vagy kockázat paraméterfüggését. Bár az MLP-nél is szinte kizárólag négyzetes hibafüggvényt szoktak alkalmazni, a nemlineáris paraméterfüggés miatt a hibafelület nem lesz kvadratikus. Mindössze abban lehetünk biztosak, hogy a hibafelület a tanítandó paraméterek folytonos, differenciálható függvénye, ami a gradiens alapú tanuló algoritmusok alkalmazását lehetővé teszi.[3]

Az egyik legelterjedtebb tanítási módszer a hibavisszaterjesztéses algoritmus. Az algoritmus egy pillanatnyi gradiensen alapuló iteratív tanuló eljárás. Az MLP gyakorlati alkalmazásnál számos kérdés merül fel. Ezek a kérdések a konkrét hálózatok konstruálásával, ill. tanításával, stb. kapcsolatosak. Ilyenek lehetnek például hány rétegű legyen a hálózat, mekkora legyen a tanulási tényező, meddig tanítsuk a hálózatot, hogyan válasszuk meg a tanító és teszt mintakészletet. Ezekre és hasonló kérdésekre, melyeket konkrét alkalmazásoknál meg kell válaszolni, minden szempontból kielégítő

általános válaszok jelenleg még nincsenek. A saját MLP konstrukciómban is előjöttek ezek a kérdések, tapasztalataim alapján próbáltam beállítani a legjobb konfigurációt.

2.4 RBF (Radial Basis Function)

Az RBF hálózat olyan két aktív rétegű előrecsatolt hálózat, melyben a rejtett réteg radiális bázisfüggvényekkel dolgozó nemlineáris leképezést valósít meg. [3]



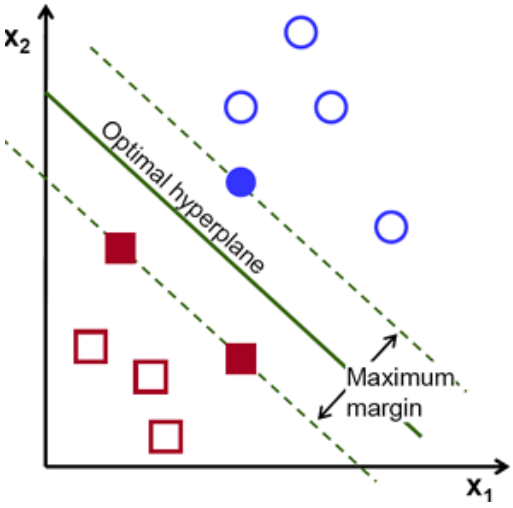
2-3. ábra (egy kimenetű) RBF hálózat felépítése [3]

Az ábrán látható hálózatban a rejtett rétegbeli processzáló elemek valósítják meg a bázisfüggvényeket. Az RBF hálóknál a bázisfüggvények gömbszimmetrikusak és általában két paraméterük van a középpont és szélességparaméter. A leggyakrabban használt bázisfüggvény a Gauss függvény.

Az RBF hálók konstrukciója a háló méretének (a rejtett rétegbeli elemek számának), a bázisfüggvények paramétereinek (amennyiben vannak a bázisfüggvényeknek szabad paraméterei) és a kimeneti réteg súlyainak meghatározását jelenti. A hálózat komplexitása a rejtett rétegbeli processzáló elemek számától, vagyis az eltérő középpontú bázisfüggvények számától függ.

2.5 Szupport vektor gép (Support Vector Machine)

Vladimir Vapnik fejlesztette ki ezt a módszert 1963-ban. A szupport gépek az előrecsatolt osztályozó/regressziós hálózatok strukturális kockázat minimalizáláson alapuló változatai. Ellenőrzött tanulást valósít meg, tehát előre meghatározott bemenet-kimenet párokat használ, melyekről feltételezzük, hogy korrekt információt tartalmaznak. Gyakorlatilag egy olyan speciális hipersíkot keres, amely nemcsak, hogy elválasztja a két osztály elemeit egymástól, de ezektől a lehető legtávolabb helyezkedik el. [3]

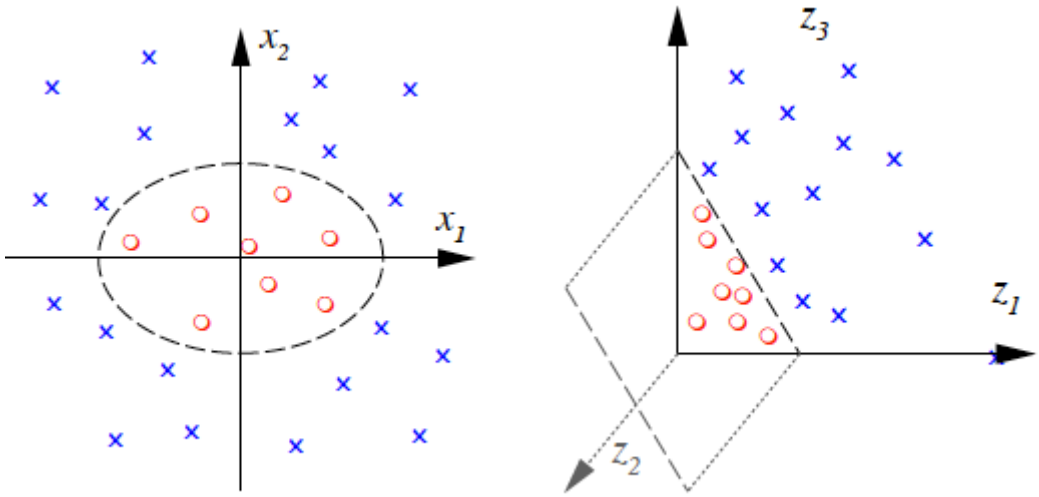


2-4. ábra SVM szupport vektorok, hipersík és maximális margó

Megkülönböztetünk lineárisan szeparálható és lineárisan nem szeparálható feladatokat. [3]

Lineárisan szeparálható egy feladat, ha van olyan lineáris felület, amellyel az osztályok elemei egymástól elválaszthatóak hibamentesen vagy kis hibával.

A valós osztályozási feladatok túlnyomó része nem szeparálható lineárisan. A lineárisan nem szeparálható feladatoknál egy megfelelő (és általában dimenziónövelő) nemlineáris transzformációval lineárisan szeparálhatóvá alakítjuk a feladatot. Ezt a szakirodalom Kernel trükknek nevezi. Az SVM előnyeinek kihasználására az így kapott jellemzőtérben az előzőeknek megfelelő optimális (maximális margójú) megoldást keressük. [3]



2-5. ábra Kernel trükk [4]

A bal oldali képen nem tudunk olyan megoldást találni, ami lineárisan helyesen szétválasztaná az osztályokat. A Kernel trükk segítségével, 2 dimenzióból 3 dimenzióba vittük át az adatokat és itt már tudunk találni helyes lineáris megoldást is.

2.6 Matlab eszköztárak bemutatása (toolboxes)

A munkám során a Matlab volt az egyik eszköz, amivel dolgoztam. A MATLAB speciális programrendszer, amelyet numerikus számítások elvégzésére fejlesztettek ki és emellett egy programozási nyelv is.

Az alap Matlab funkcionalitás mellé könnyedén lehet új eszköztárakat (toolboxes) felvenni. A szakdolgozatom során több Matlab eszköztárat felhasználtam. Az összes különböző osztályozó algoritmusokat tett elérhetővé. A következőkben az általam használtakat ismertetem röviden.

2.6.1 Neural Network Toolbox

Ezen toolbox segítségével, neurális hálózatok hozhatóak létre, taníthatók és szimulálhatók. Függvényeket, algoritmusokat nyújt regressziós, osztályozós, klaszterezős stb. feladatok megoldására. [6]

Ebből az eszköztárból két osztályos osztályozókat használtam fel. Röviden bemutatom, mely függvények segítségével hozhatóak létre az előző három fejezetben ismertetett hálók.

2.6.1.1 MLP

Első lépésként létre kell hoznunk a hálót, ezt a *newff* függvénnyel tehetjük meg. Ennek a függvénynek meg kell adnunk mindenképp a bemenő adathalmazt, az elemek osztályainak az azonosítóit és a háló méretét. Ezeken kívül természetesen sokkal több paraméter megadható. Ezután jöhet a háló tanítása a *train* és tesztelése a *sim* függvénnyel. Például:

```
net=newff(data,class,[10 3]);
net=train(net,data,class);
t=sim(net, testdata);
```

2.6.1.2 SVM

Itt is létre kell hoznunk legelőször a struktúrát. Erre és egyben a tanításra az *svmtrain* függvény szolgál. Két kötelező paramétere egy tanító adathalmaz és az adathalmaz elemeihez rendelt osztályazonosító. Lehetőségünk van megadni a kernel függvény típusát pl. lineáris, kvadratikus, rbf, polinomiális stb. Az SVM-et osztályozásra az *svmclassify* függvény meghívásával lehet használni. Első paramétere az *svmtrain* visszatérési értéke, a másik pedig a tesztelni kívánt adathalmaz. Például:

```
svm = svmtrain(data,class,'kernel_function','rbf');
t = svmclassify(svm, testdata);
```

2.6.1.3 RBF

Ez a toolbox két rbf megvalósítást tartalmaz. Az első a *newrbe* függvénnyel hozható létre, amely minden tanítópontra illeszt egy Gauss görbét. A másik megvalósítás a *newrb* függvénnyel hozható létre, amely abban különbözik az elsőtől, hogy nem illeszt minden tanítópontra görbét, hanem iteratívan növeli a neuronok számát egy kis hibahatárig. A háló tesztelését az MLP-hez hasonlóan a *sim* függvénnyel lehet megtenni. Például:

```
mse = 0.02; //cél hibaérték
std = 1; //szórás konstans
net = newrb(data,class,mse,std);
t = sim(net, testdata);
```

2.6.2 Statistics and Machine Learning Toolbox

Amint az a nevéből is következik ebben az eszköztárban statisztikai és gépi tanulási módszerek találhatóak. Osztályozó, klaszterező, ellenőrzött és nem ellenőrzött tanulást stb. elvégző algoritmusok találhatóak benne. Az eszköztárnak az egyosztályos osztályozásra, vagyis anomáliadetektálásra használható részeivel dolgoztam.[7]

Az osztályozáshoz három függvényt használtam fel.

- fitcsvm: modell létrehozásáért felelős
- crossval: kereszt kiértékelést végez
- kfoldpredict: osztályoz

2.6.3 A Library for Support Vector Machines (LibSVM) Toolbox

Ezt az eszköztárat Chih-Chung Chang and Chih-Jen Lin írták, tehát nem hivatalosan a Matlab csapata fejlesztette. Ez több programozási nyelven is megvalósításra került én a Matlab-os változatot használtam. Több típusú SVC (Support Vector Classifier) alapú osztályozásra képes pl. C-SVC, NU-SVC vagy one-class classification. Ezen lehetőségei miatt használtam fel a szakdolgozatomhoz.[5]

Két függvény volt számomra fontos az *svmtrain* és az *svmpredict*. Az *svmtrain* függvénynek rengeteg paraméter megadható, ezek közül bemutatom azt a paraméterezést, amellyel egy osztályos osztályozásra képes.

model = svmtrain(data, class, '-s 2 –t 2'), az -s kapcsoló jelentése SVM típusa, a kettesnek az egy osztályos osztályozás felel meg, a -t jelentése kernel típusa, ez is kettes, ami azt jelenti, hogy RBF lesz a kernel függvény.

2.7 TensorFlow bemutatása

A TensorFlow egy nyílt forráskódú gépi és mély tanulás elvégzésére alkalmas szoftver. Ezt a rendszert a Google kutatói és mérnökei dolgozták ki. 2015 novemberében adták ki az első verziót, tehát egy nagyon friss újonnan elérhető rendszerről van szó. A rugalmas architektúrája lehetővé teszi több különböző platformra való telepítését. A Google-ön belül jelenleg is fejlesztik a jelenlegi verziónak az egyik célja az, hogy az ezt használó kutatókat, diákokat, mérnököket stb. közösségbe foglalja, hogy a visszajelzéseik és esetleges fejlesztéseik alapján még hatékonyabbá tegyék a rendszert. [8]

A számítások elvégzéséhez adatfolyam gráfot használ. Az adatfolyam gráf egy irányított gráf, amelynek vannak csomópontjai és élei. A csomópontok a matematikai műveleteknek felelnek meg és e mellett az éleknek továbbítják az adatokat. Az élek az egyes csomópontok között bementként/kimenetként viselkednek. Az éleken az adatok többdimenziós adattömbökként szállítódnak, ezeket nevezzük tenzoroknak. [8]

2.7.1 Deep Learning

A TensorFlow mély tanulást (Deep Learning) használ a tanuláshoz. A mély tanulás, olyan feladatokban tud hatékonyan működni, ahol rengeteg adattal rendelkezünk. A Deep Learning olyan algoritmusokat takar, amelyek a hatalmas adattömegeket (big data) hatékonyan és gyorsan tudják feldolgozni. Egyik legfontosabb eltérése a hagyományos gépi tanulási módszerektől, hogy több rétegben dolgozza fel az információkat. A mély tanulásnak eddig a képfelismerésben voltak kimagasló eredményei.

A munkám során a TensorFlow-ban logisztikus osztályozást használtam, röviden bemutatom, hogy ez itt mit is jelent.

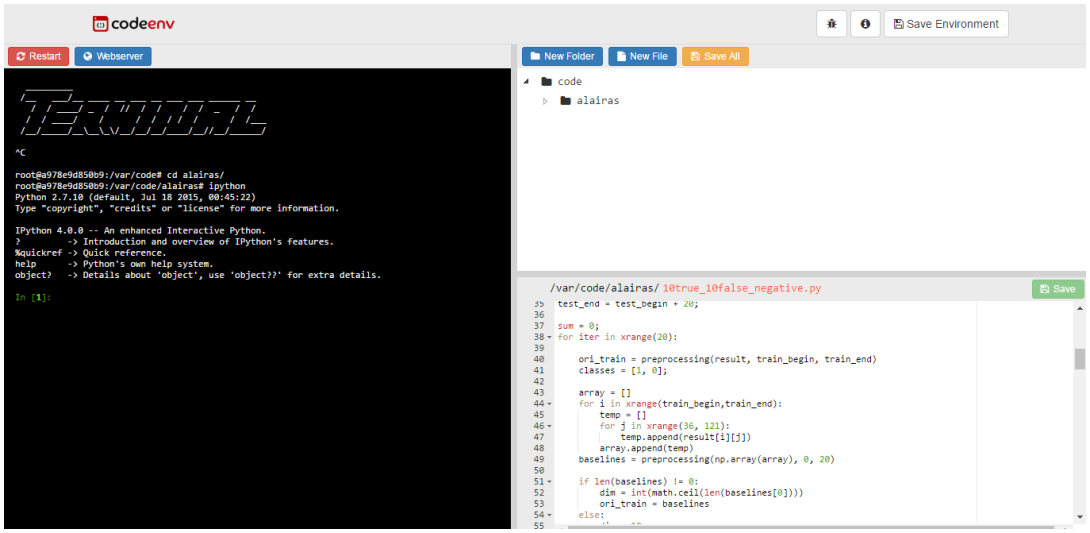
A fő képlet: $Y = WX + b$. A W a súlyokat tartalmazó mátrix, X a bemeneti adathalmaz, a b pedig egy úgynevezett eltolás vektor. Az Y értéke egy számokat tartalmazó vektor/tömb, amely annyi elemű ahány osztályunk van. Például: ha egy két osztályos osztályozót akarunk létrehozni (eredeti vagy hamisítvány), akkor az Y értéke egy adott elemre lehet [3 6], ami azt jelenti, hogy az osztályozó szerint az aktuális elem a kettes osztályba tarozik, ha a 6 a kettes osztálynak felel meg. Érezhetően ezek a rangsorolások nem igazán követhetőek, ezért ezt a rangsort át kell alakítani valószínűségekké. Az átalakítás után lehet a [3 6] vektorból, [0.3 0.7] vektor, ahol az

elemek már valószínűségeknek felelnek meg. Tehát ez utóbbi vektor azt jósolja, hogy a kapott aláírás 70%-ban a kettes osztályba tartozik.

Így sem lehetünk teljesen elégedettek, jó lenne egy olyan megfeleltetés, amely a valószínűségekből, azonnal visszaadja, hogy melyik osztályba tartozik az elem. Ezt „One Hot Encoding” kódolásnak nevezzük. Ez azt jelenti, hogy egy kétosztályos osztályozónál az osztályoknak [1 0] és [0 1] vektorok felelnek meg [9].

2.7.2 TensorFlow használata

A munkám során a TensorFlow-nak nem közvetlenül az aláírásokat, hanem a kinyert jellemzőket adtam bemenetként. A gyorsabb működés miatt, nem saját gépre telepített TensorFlow-val dolgoztam, hanem a codeenv nevű online oldalon, már készen összerakottal. Ide mindössze regisztrálnom kellett és azonnal használhattam is a szolgáltatásait. Jelen esetben itt a fejlesztési nyelv a python volt [10]. A TensorFlow-t python modulként lehet beimportálni, ezután már elérhetőek voltak a szolgáltatásai.



2-6. ábra TensorFlow környezet a codeenv-n

A környezet 3 részre tagolódik. A felület egyik fele a python terminál, ahol a scripteket lehet futtatni. A jobb oldal további két részre tagolódik. A felső részben hozhatjuk létre és tekinthetjük meg a mappáinkat és a fájljainkat. Az alsó rész egy szerkesztőfelület, amellyel a scriptek megírhatóak. A változtatásokat egyszerűen a Save Environment gombbal elmenthetjük.

3 Osztályozási módszerek

A következő fejezetben bemutatom, milyen lépések sorozatával értem el, az egyes osztályozó konfigurációimat.

3.1 A mérési környezet

A méréseim során egy SVC (Signature Verification Competition)[11] nevű adatbázist használtam fel. Ebben az adatbázisban főleg képzett hamisítók által készített hamisítványok voltak, így jóval nehezebb kiszűrni a hamis aláírásokat. Azért választottam ezt az adatbázist, mert a profi hamisítások miatt jóval nagyobb kihívást jelent jó osztályozót létrehozni. Aláírónként 40 aláírást tartalmaz az adatbázis, ezek eloszlása 20 eredeti és 20 hamisítvány.

Az SVC adatbázisban csak képek voltak az aláírásokról. A képek feldolgozását a már említett a Budapesti Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai Tanszéken fejlesztett off-line aláírás-hitelesítő rendszer végezte. A rendszer egy adathalmazt biztosított számomra, amelyben soronként vannak az aláírásokból kinyert jellemzők, amelyek száma akár közel 150 darab is lehet. Az automatizálás miatt egyes aláírások jellemzői nem lettek pontosan kinyerve, ezek az osztályozó konfigurációimban rendre a legrosszabb eredményeket érték el.

A mérések során több csoportosítást alkalmaztam. Meg kellett először is határoznom a tanító- és teszt mintákat. Ahhoz, hogy valamiféle képet kapjak az adatok minőségéről, meghatároztam egy optimális hibaarányt. Ezt úgy tudtam elérni, hogy a tanítás során a valós problémától eltérően hamisítványokkal is tanítottam a rendszert. Az itt kapott hibaarányt tűztem ki célul.

Másodszor a tanításhoz már csak eredeti mintákat használtam fel. Két megközelítést vizsgáltam meg, az első, hogy anomáliadetektálásként tekintettem a feladatra, a másik pedig, hogy megpróbáltam minél pontosabb hamisítványokat készíteni az eredeti aláírások felhasználásával és így már kétosztályos osztályozás volt a feladat.

Egy aláíróra mindössze 40 aláírás elérhető. Ebből a tanításhoz, csak a 20 eredeti mintát használhatom fel. Ezekből a számokból is jól látszik, hogy rendkívül kicsi a

mintaszám, ami az osztályozók eredményét nagyban befolyásolja. Ezért készítettem egy mintageneráló algoritmust, amellyel a 20 meglévő mintából két nagyságrenddel többet generálok. A részletes megvalósításokat az egyes általam használt eszközökön a következő fejezetekben ismertetem.

3.2 Matlab megoldások

Ebben a fejezetben bemutatom a Matlab-ban készített osztályozó megoldásaimat. A tárgyalást az adatok elő feldolgozásának részleteivel kezdem, ezután az előző fejezetben említett optimális hibaarány meghatározása következett. Az anomáliadetektáló, majd a mesterséges hamisítványokat használó osztályozóim bemutatása ezek után következik.

3.2.1 Adatok elő feldolgozása

Mielőtt elkezdtem az osztályozókat összerakni, az adatokat elő feldolgoznom kellett. Már említettem, hogy egy aláírásból akár 150 jellemző is elérhető lehet, viszont a legtöbb esetben több jellemző is üresen marad az adathalmazban. Legelőször ezen üres elemeket kellett kiszűrnöm. A további felhasználás céljából az elő feldolgozó függvényem az adatokat öt részre bontotta, az öt nagy jellemzőcsoport mentén, úgymint alapvonalak, hurkok, alak, ékezetek és globális jellemzők.

A függvény bemeneti paramétere a 20 hamisított/eredeti aláírása az egyes aláíróknak. Azon jellemzőket kell kiszűrni, amelyek mind a 20 aláírásban 0 értéket vesznek fel, tehát nincs ilyen jellemzőjük. A Matlab *mean()* függvényével, az egyes oszlopokat az adathalmazban átlagoltam. Ennek az eredménye egy 148 elemű egydimenziós vektor lett. Azon jellemzőket, amelyeknek az átlaga 0 nem vettem figyelembe, így például a 148 dimenziós adatból, 48 dimenziós lett.

A jellemzőcsoportok különválasztásához felhasználtam azon előzetes információkat, hogy az egyes csoportok maximálisan mennyi jellemzővel rendelkezhetnek.

A függvény visszatérési értéke a szűrt adathalmaz és az ebből származtatott jellemzőcsoportok. A következő ábrán a megvalósított kódrészlet látható.

```
function [T, BaseLines, Loops, Shapes, Accents, Globals] = preprocessing(Signature)
%a bemenő adat oszlopinak átlagát tartalmazza
M = mean(Signature);
%a szűrt jellemzők
filteredData = [];
%a bemenő adatból az egyes jellemzőcsoportokhoz, mely indexek tartoznak
bases = [];loops = [];accents = [];shapes = [];globals = [];

%üres sorok törlése és az egyes jellemzőcsoportok létrehozása
for iter=1:length(M)
    if M(iter) ~= 0
        filteredData(end+1) = iter;
        if iter < 37
            bases(end+1) = iter;
        elseif iter < 122;
            loops(end+1) = iter;
        elseif iter < 125;
            shapes(end+1) = iter;
        elseif iter < 143;
            accents(end+1) = iter;
        else iter < 149;
            globals(end+1) = iter;
        end
    end
end
T = Signature(:,filteredData);
BaseLines = Signature(:,bases);
Loops = Signature(:,loops);
Shapes = Signature(:,shapes);
Accents = Signature(:, accents);
Globals = Signature(:,globals);
end
```

3-1. ábra Elő feldolgozó függvény Matlab megvalósítása

3.2.2 Hamisított aláírások felhasználása

A célom azzal, hogy a hamisított aláírásokkal is tanítom a rendszert, hogy lássam az egyes osztályozási módszerek milyen hibaarányú eredményt tudnak elérni. Mindhárom osztályozó konfigurációban először az adatokat elő feldolgozom, az előző alfejezetben bemutatott függvénnyel. Ezután megvalósítom az egyes osztályozási módszereket, osztályozom a teszt adathalmazt, majd az eredményeket excel fájlba mentem. A tanulásokhoz és a tesztelésekhez is 10 hamis és 10 eredeti aláírást használtam fel.

3.2.2.1 Kétosztályos MLP

Az MLP hálónak a tanuláshoz, legelőször meg kellett adni a rejtett rétegek és az egyes rétegeken belüli neuronok számát. Erre nincs olyan módszer, amely minden problémánál beválna. Ezért én is több variációt kipróbáltam és ezek közül a legkisebb átlagos hibaarányú konfigurációt választottam ki.

Minél több réteggel próbálkoztam, annál rosszabb eredmények születtek. A legjobb eredményt egy, egy rejtett rétegű 10 neuront tartalmazó hálóval értem el. A háló a tanításkor három részre bontja a tanító adathalmazt: tesztelő, validáló és tanító halmazokra. Alapértelmezetten ezek a felosztások véletlenszerűen történnek meg. Én explicite megadtam, 10-10%-a legyen az adatoknak validálásra és tesztelésre és 80%-a tanításra.

A következő ábrán az látszik, milyen eredmények érhetőek el az egyes aláírásoknál ezen osztályozó konfigurációval:

Azonosító	*AER*	*FAR*	*FFR*
2	35%	20%	50%
4	45%	30%	60%
6	15%	20%	10%
8	60%	20%	100%
10	15%	0%	30%
13	30%	50%	10%
15	45%	80%	10%
18	25%	30%	20%
20	20%	40%	0%
22	50%	90%	10%
24	50%	20%	80%
25	30%	60%	0%
28	20%	30%	10%
32	5%	10%	0%
33	30%	20%	40%
34	95%	100%	90%
35	30%	60%	0%
37	35%	10%	60%
38	25%	50%	0%
40	25%	50%	0%
Átlag	34,3%	39,5%	29,0%

3-2.ábra MLP hamisított mintákkal

Láthatóan nem sikerült jó eredményt elérni. A FAR értéke kimagaslóan magas, gyakorlatilag az osztályozó nem tud különbséget tenni a hamisítványok és az eredeti aláírások között.

3.2.2.2 Kétosztályos RBF

A Matlab programcsomag NN eszköztárában két lehetőség van RBF háló építésére. Jelen fejezetben mindkettő használhatóságát megvizsgáltam. A különbség a

két megvalósítás között, hogy amíg az egyik minden tanítópontra illeszt, egy Gauss görbét és így minden tanítópontban 0 hibát képes elérni, addig a másiknak meglehet adni egy minimális hibaértéket, amelyet ha elér, nem folytatja tovább a háló tanítását.

A tanítást mindkét megvalósításnál a szórás paraméter befolyásolta a legnagyobb mértékben. Kis értékű szórást beállítva az RBF háló rossz eredményeket adott. A szórás értékét növelve, az osztályozó teljesítménye is javult. Sem a túl kicsi, sem a túl nagy szórás érték nem adott jó eredményt. A legkisebb hibaarányú osztályozóknak $2e^6$-os szórás értéket állítottam be.

Az egyik megvalósításnál, az átlagos jobb hibaarány eléréséhez, nem vártam el 0%-os hibát. Lehetőségem volt beállítani egy olyan minimális hibaarányt, amely elérésekor nem folytatódik tovább a tanulás. Én ezt 7.5%-os MSE (Mean Square Error) elérésénél húztam meg.

A következő ábrákon a két megvalósítás eredményei láthatóak:

Azonosító	*AER*	*FAR*	*FFR*	*Neuronszám*
2	30%	40%	20%	20
4	0%	0%	0%	20
6	40%	0%	80%	20
8	65%	30%	100%	20
10	20%	10%	30%	20
13	5%	10%	0%	20
15	40%	70%	10%	20
18	50%	30%	70%	20
20	0%	0%	0%	20
22	80%	70%	90%	20
24	40%	20%	60%	20
25	55%	70%	40%	20
28	25%	50%	0%	20
32	20%	40%	0%	20
33	35%	30%	40%	20
34	60%	100%	20%	20
35	25%	50%	0%	20
37	20%	20%	20%	20
38	45%	90%	0%	20
40	20%	40%	0%	20
Átlag	33,8%	38,5%	29,0%	20

3-3.ábra RBF Gauss görbe minden tanítópontra

Azonosító	AER	FAR	FFR	Neuronszám
2	15%	20%	10%	14
4	20%	40%	0%	13
6	25%	40%	10%	10
8	75%	60%	90%	11
10	25%	30%	20%	10
13	20%	40%	0%	14
15	30%	60%	0%	8
18	35%	40%	30%	9
20	0%	0%	0%	5
22	75%	60%	90%	12
24	50%	40%	60%	11
25	30%	50%	10%	12
28	5%	10%	0%	12
32	20%	40%	0%	8
33	20%	20%	20%	16
34	30%	20%	40%	12
35	10%	20%	0%	9
37	10%	20%	0%	7
38	25%	50%	0%	3
40	20%	40%	0%	9
Átlag	27%	35%	19%	10,25

3-4. ábra RBF megadott minimális MSE-vel

A fenti két ábrából látható, hogy a második konfiguráció közel 7%-al jobb eredményt adott. Ezt méghozzá úgy érte el, hogy az átlagos neuron szám 20-ról 10-re csökkent. Ha megnézzük a FAR értékek közel egyenlők maradtak, az FFR értékek pedig javultak. Mindkét megvalósítási konfiguráció jobb lett, mint az MLP-s.

3.2.2.3 Kétosztályos SVM

Az SVM konfigurációm megalkotásakor a kernel függvény megválasztása volt az egyik legmeghatározóbb feladat. Több típusú kernel függvény volt adott, mint például lineáris, polinomiális, rbf, mlp stb. A kiválasztott kernel függvénynek további paramétereket lehetet megadni.

A lineáris kernel alapú megoldás adta a legrosszabb eredményt, de ez várható is volt, hiszen a feladat nem lineáris. Ebben a konfigurációban a polinomiális kernel függvényt választottam, mert ez adta a legkisebb hibaszázalékú eredményt. A függvény polinom fokát egyre, kettőre és háromra állítottam, ez után minden aláíráshoz kiválasztottam, hogy mely értékkel kapom a legjobb eredményt. A Matlab kód megtalálható az A. Függelékben.

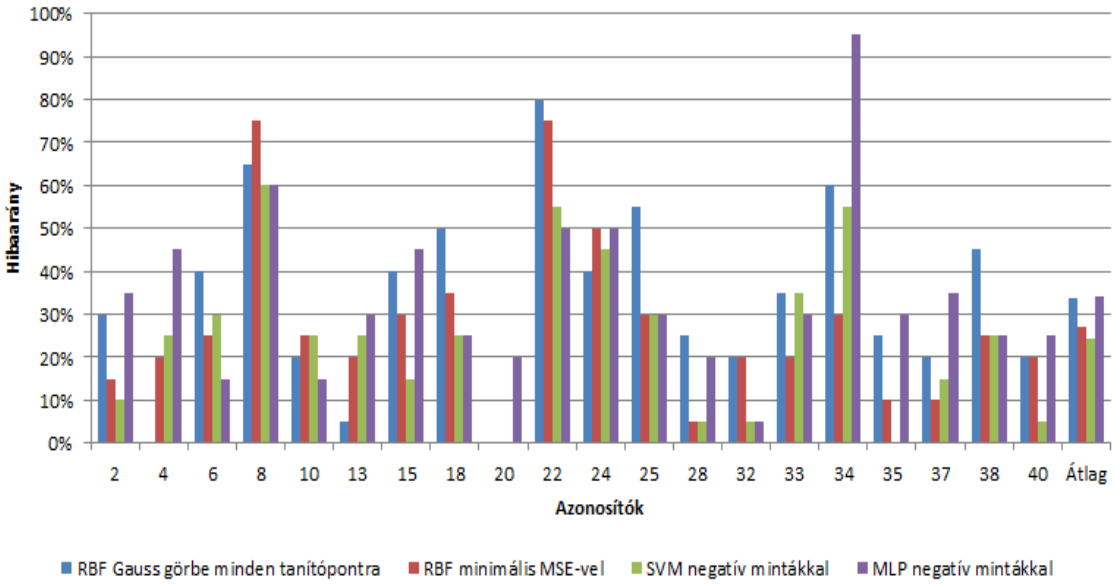
A következő ábrán az elért osztályozási hibaarányok láthatóak:

Azonosító	*AER*	*FAR*	*FFR*
2	10%	10%	10%
4	25%	50%	0%
6	30%	30%	30%
8	60%	20%	100%
10	25%	0%	50%
13	25%	30%	20%
15	15%	20%	10%
18	25%	20%	30%
20	0%	0%	0%
22	55%	30%	80%
24	45%	30%	60%
25	30%	60%	0%
28	5%	10%	0%
32	5%	0%	10%
33	35%	40%	30%
34	55%	70%	40%
35	0%	0%	0%
37	15%	30%	0%
38	25%	50%	0%
40	5%	10%	0%
Átlag	24,5%	25,5%	23,5%

3-5. ábra SVM hamisított mintákkal

Az eddigi konfigurációk közül ez adta a legkisebb átlagos eredményt. Megfigyelhetjük azt, az előzőektől eltérően, hogy a FAR és FFR értéke átlagosan közel egyenlő, emellett pedig azt is, hogy az osztályozó kisebb hibával osztályozza a hamisítványokat.

3.2.2.4 Hamisított aláírások felhasználásának értékelése



3-6. ábra Eredmények összehasonlítása

A legkisebb hibaarányú eredményt az SVM konfigurációm adta 24.5%-al. Ettől az értéktől jobb eredményre számítottam előzőleg. További tesztelések során rájöttem, hogy fontos szerepe van annak, mely aláírásokkal tanítok és melyekkel tesztelek.

Egy tolóablakos módszert alkalmaztam, az aláírás halmazok kiválasztására. Az eredeti aláírások 10-es szomszédos részhalmazait, a hamisított aláírások 10-es szomszédos részhalmazaival kapcsoltam össze. Ennek a megvalósítása két egymásba ágyazott ciklust jelent. Minden ciklusban tanítottam a hálót, azon halmazokat választottam ki, amelyekkel a legkisebb hibaszázalékot kaptam.

Ezzel a módszerrel jelentősen javítottam az előző alfejezetekben bemutatott legtöbb osztályozó konfiguráción. Egyedül az MLP-vel nem tudtam lényegesen jobb eredményt elérni. A Matlab kód megtalálható a B. Függelékben.

A következő két ábrán a tolóablakos tanítóhalmaz kiválasztó algoritmust használó, RBF és SVM osztályozó konfigurációk eredményei láthatóak:

Azonosító	*AER*	*FAR*	*FFR*
2	5%	0%	10%
4	0%	0%	0%
6	10%	20%	0%
8	5%	0%	10%
10	0%	0%	0%
13	0%	0%	0%
15	0%	0%	0%
18	5%	0%	10%
20	0%	0%	0%
22	15%	20%	10%
24	15%	30%	0%
25	10%	20%	0%
28	5%	10%	0%
32	5%	0%	10%
33	5%	10%	0%
34	15%	30%	0%
35	0%	0%	0%
37	0%	0%	0%
38	0%	0%	0%
40	0%	0%	0%
Átlag	4,8%	7,0%	2,5%

3-7. ábra RBF tolóablakos tanító halmaz kiválasztással

Azonosító	*AER*	*FAR*	*FFR*
2	0%	0%	0%
4	20%	30%	10%
6	0%	0%	0%
8	5%	10%	0%
10	0%	0%	0%
13	10%	20%	0%
15	0%	0%	0%
18	0%	0%	0%
20	0%	0%	0%
22	15%	20%	10%
24	5%	10%	0%
25	10%	10%	10%
28	0%	0%	0%
32	0%	0%	0%
33	10%	10%	10%
34	5%	10%	0%
35	0%	0%	0%
37	0%	0%	0%
38	5%	10%	0%
40	0%	0%	0%
Átlag	4,3%	6,5%	2,0%

3-8. ábra SVM tolóablakos tanító halmaz kiválasztással

Láthatóan jelentősen javultak az eredmények, a tolóablakos tanító halmaz kiválasztással. A két megvalósítás eredményei között számottevő különbség nincs. Ezek után a célom, hogy megközelítsem ezen eredményeket csak eredeti aláírások felhasználásával a tanításhoz.

3.2.3 Egy osztályos osztályozók

Amikor egy osztályos osztályozókról beszélünk, valójában anomáliadetektálásról van szó. Az ilyen típusú osztályozók egy normál működést tudnak megtanulni és ebből következtetve ismerik fel az ettől eltérő működést. Maga az anomáliadetektáló alkalmas aláírások felismerésére, hiszen a normál működésnek feleltethetjük meg az eredeti, míg az ettől eltérő működésűnek pedig a hamisított aláírásokat.

A munkám során két Matlab eszköztár egy osztályos osztályozóit használtam fel. Mindkettő esetben, kezdetben a tanításhoz, a tanító halmaz minden jellemzőjét felhasználtam, de mivel az SVC adatbázisban szakértéktől származó hamisított aláírások találhatóak meg, a jellemzők nagy részét sikeresen tudták reprodukálni, ezért a tanítás során ezek elnyomták azokat, amelyek különböztek az eredeti és hamisított aláírások között. Az osztályozók nem tudtak különbséget tenni a két osztály között, ezért a jellemzőket redukálnom kellett.

3.2.3.1 Jellemzők szelektálása

Azokat a jellemzőket választottam ki, amelyek az egyes aláírásoknál a legkevésbé térnek el egymástól. Ezt arra alapoztam, hogy minden aláírásban vannak olyan jellemzők, amelyek szinte tökéletesen megegyeznek. Ezeket a jellemzőket jóval nehezebb hamisítani, mert az aláíró is önkéntelenül használja őket.

A legkisebb eltérésű jellemzők kiválasztására egy jó módszer, ha szórás alapján rangsoroljuk az egyes jellemzőket. Ha egy jellemző szórása az aláírások között nagy, ez azt jelenti, hogy az egyes aláírásokban többé/kevésbé eltérnek, míg ha a szórása kicsi, akkor pedig nagyban hasonlóak. A minimális szórások meghatározását nem a teljes jellemzőhalmazra, hanem az egyes jellemzőcsoportokra alkalmaztam. Ennek az volt az oka, hogy nem akartam, olyat, hogy valamely jellemzőcsoportból ne kerüljön jellemző a redukált halmazba.

Nehéz feladatnak bizonyult meghatározni azt, hogy az egyes aláíróknál, a jellemzőcsoportjaikból a szórások alapján hány elemet válasszak ki. Ezt a feladatot nem is sikerült maradéktalanul megoldanom.

A redukált jellemző halmaz meghatározásának lépései:

1. Az adatok normalizálása jellemzők szerint
2. Jellemzők csoportokra bontása
3. Csoportonként a jellemzők szórásának kiszámolása
4. Csoporton belüli rendezések végrehajtása
5. Csoportokból a legkisebb szórású részhalmazok kiválasztása

Az adatokat első lépésként normalizálnom kellett, erre Matlab-ban készítettem egy *normalize_data* nevű függvényt.

```
function norm_data = normalize_data(data)
    %Normalizálás
    maximum = ones(20,1)*max(data);
    minimum = ones(20,1)*min(data);
    oszto = maximum - minimum;
    norm_data = (data - minimum) ./ oszto;
end
```

3-9. ábra Adatok normalizálása

Az Adatok elő feldolgozása alfejezetben bemutatott *preprocessing* függvény a paraméterében kapott adathalmazt felbontja az 5 nagy jellemzőcsoportra. Ezzel a második lépést is megvalósítottam.

A jellemzőcsoportokon belüli jellemzők szórásának a kiszámításához, kihasználtam a Matlab által nyújtott *std* nevű függvényt. A függvény oszloponként (azaz jellemzőkként) meghatározta a szórásokat. A csoportonkénti szórások kiszámítására létrehoztam egy *calculate_std* nevű Matlab függvényt, amely visszatérési értékként visszaadja jellemzőcsoportonként a szórások értékeit. Erre az újrafelhasználhatóság szempontjából volt szükség. A kód megtalálható a C. Függelékben.

A következő lépés a jellemzőcsoportonkénti szórások rendezése volt. A Matlab *sort* függvényét használtam erre. A *sort* függvénynek a szórásokat tartalmazó listákat adtam át paraméterként. Visszatérési értékként a rendezett listát és az átrendezett

értékekhez tartozó lista indexeket kaptam vissza. A lista indexekre a jellemzők kiválasztásánál volt szükségem.

Utolsó lépésként meg kellett határoznom, hogy az egyes csoportokból, mennyi jellemzőt használjak fel a tanításhoz. Ennek megoldására nem tudtam olyan függvényszerűséget meghatározni, amellyel ez explicite megmondható lenne. Ezért keresztkiértékelést alkalmaztam. Ez azt jelenti, hogy 5 egymásba ágyazott ciklussal meghatároztam a legkisebb hibaarányt adó csoport részhalmazokat.

3.2.3.2 LibSVM toolbox nyújtotta egy osztályos osztályozó

A toolbox nevéből is következően az ebben lévő egy osztályos osztályozó SVM alapú. A tanítás során csak az eredeti aláírásokat használtam fel. Ahhoz, hogy tudjam tesztelni, milyen eredményt ad az osztályozó eredeti aláírásokra a 20 eredeti aláírásból 15-öt használtam tanításra és 5-öt tesztelésre. A hamisítványokat értelemszerűen csak tesztelésre használtam. A tanítás előtt az előző fejezetben bemutatott jellemző szelektálás lépésein végigmentem.

Az egy osztályos osztályozónak első lépésként meg kellett adni, hogy milyen legyen a kernel függvényének a típusa. Én az RBF kernel függvény mellett döntöttem. Teszteltem a másik három kernel típust is, de azokkal nem sikerült megközelítenem az RBF-es eredményemet.

A LibSVM eszköztárban az RBF kernel függvény Gauss függvény. A következő képlettel írható fel:

$$K(u, v) = \exp(-gamma * |u - v|^2)$$

A képletben egyedül a gamma paraméter változtatható. A gamma alapértelmezett értéke 1/jellemzők száma. A gamma gyakorlatilag az általánosan felírt Gauss függvény szélességparaméterének a reciproka. Az alapértelmezett érték nem adott megfelelő osztályozási eredményt, ezért elkezdtem változtatni rajta. Több iterációt lefuttatva kiválasztottam azt a gamma értéket, amellyel az osztályozó a legkisebb hibát érte el. A teljes kód megtalálható a D. Függelékben.

A LibSVM osztályozóval elért eredmények:

Azonosító	AER	FAR	FFR
2	10%	20%	0%
4	20%	0%	40%
6	15%	10%	20%
8	5%	10%	0%
10	23%	25%	20%
13	13%	25%	0%
15	8%	15%	0%
18	10%	20%	0%
20	0%	0%	0%
22	35%	10%	60%
24	20%	20%	20%
25	3%	5%	0%
28	30%	20%	40%
32	25%	30%	20%
33	18%	15%	20%
34	28%	35%	20%
35	20%	0%	40%
37	20%	0%	40%
38	10%	0%	20%
40	0%	0%	0%
Átlag	15,5%	13,0%	18,0%

3-10. ábra LibSVM egy osztályos osztályozó eredménye

Az eredmények nagyon jónak mondhatóak, de így is jelentősen eltérnek az optimális konfigurációval elért eredményeimtől. Az egyik érdekes észrevétel, hogy itt a FAR átlagos értéke kisebb, mint az FFR értéke. Tehát ez az osztályozó kicsivel pontosabban ismeri fel a hamisítványokat, mint az eredeti aláírásokat.

3.2.3.3 Statistics and Machine Learning Toolbox nyújtotta egy osztályos osztályozó

Ebben az eszköztárban megvalósított egy osztályos osztályozó is SVM-re épül, viszont a működése eltér az előző fejezetben bemutatottól. Az osztályozónak meg kell adni mind az eredeti, mind a hamisított aláírásokat. Kimenetként pedig visszaadja a kiugró értékeket. Az eszköztárban ezt a *fitcsvm()* függvény valósítja meg.

Két paramétert kell beállítani ehhez, az egyik azt mondja meg, hogy valószínűsíthetően hány százaléka lesz, a bemenetnek kiugró érték. Ideális esetben nyílván ennek 50%-nak kellene lenni, hisz a bemenő aláírások fele eredeti, fele pedig

hamisítvány. A kiértékelések során ezzel az értékkel nem kaptam jó eredményeket. Mivel professzionális hamisítványokról van szó, ezért a várható kiugró értékek százalékos arányát 35%-ra állítottam be. [12]

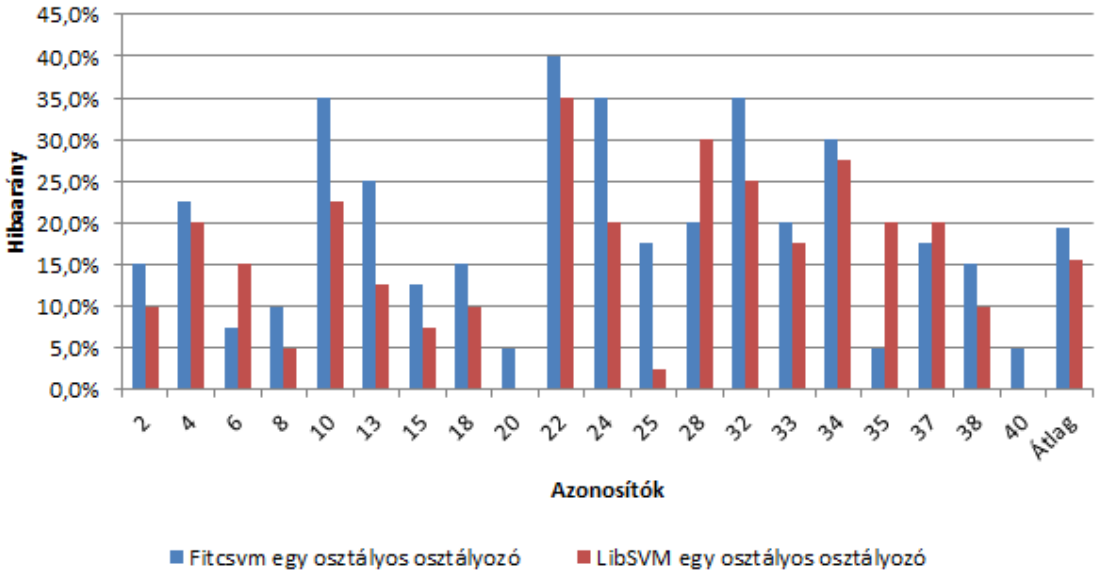
A másik paraméter az elválasztó felületet befolyásolta. Kis értéket megadva kevés tartó vektora lesz a hálónak, tehát a szeparáló felület sima, nyers döntési határt ad. Nagy értékeknél a hálónak több tartóvektora van, tehát a szeparáló felület kanyargós döntési határt ad. Úgy kell beállítani az optimális értékét a paraméternek, hogy elég nagy legyen ahhoz, hogy a komplex feladatoknál használható legyen, viszont elég kicsi, hogy ne okozzon túltanulást. [12]

A tanítás előtt jelen konfigurációban is elő feldogoztam és szelektáltam a jellemzőket. A kód az előző megvalósítástól egyedül az osztályozási módszerben különbözik. Az elért eredmények a következő ábrán láthatóak:

Azonosító	AER	FAR	FFR
2	15,0%	25,0%	5,0%
4	22,5%	15,0%	30,0%
6	7,5%	15,0%	0,0%
8	10,0%	10,0%	10,0%
10	35,0%	45,0%	25,0%
13	25,0%	35,0%	15,0%
15	12,5%	15,0%	10,0%
18	15,0%	15,0%	15,0%
20	5,0%	10,0%	0,0%
22	40,0%	50,0%	30,0%
24	35,0%	40,0%	30,0%
25	17,5%	15,0%	20,0%
28	20,0%	15,0%	25,0%
32	35,0%	35,0%	35,0%
33	20,0%	30,0%	10,0%
34	30,0%	15,0%	45,0%
35	5,0%	5,0%	5,0%
37	17,5%	15,0%	20,0%
38	15,0%	5,0%	25,0%
40	5,0%	5,0%	5,0%
Átlag	19,4%	20,8%	18,0%

3-11. ábra Fitcsvm egy osztályos osztályozó eredménye

Az osztályozó hibaaránya elmarad az előző konfigurációmtól, de nem mondható rossznak ez sem. Láthatóan itt az előzőtől eltérően az eredeti aláírásokat kisebb hibával osztályozta az osztályozó.



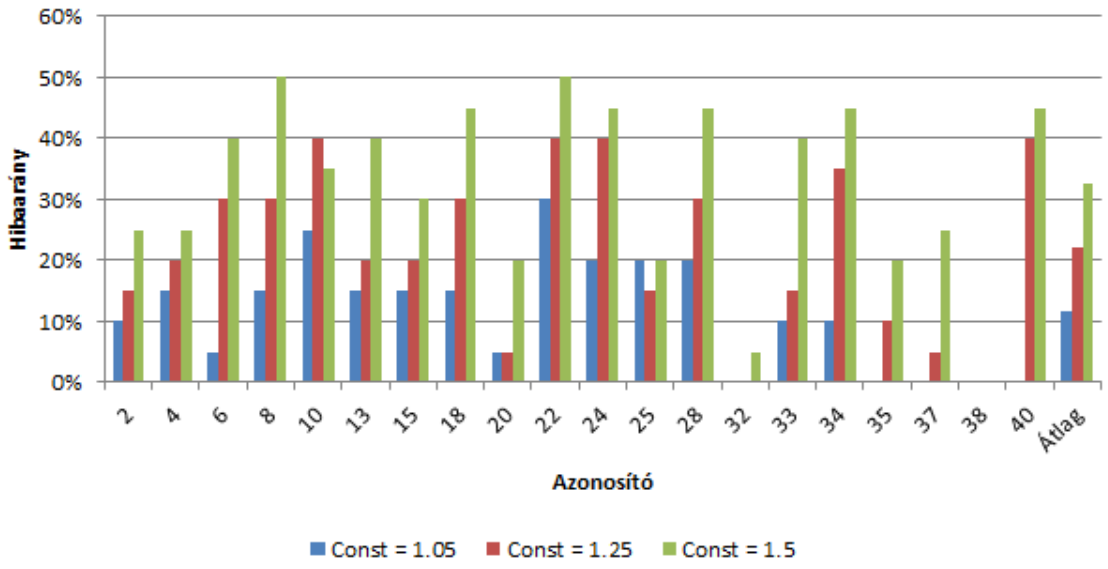
3-12. ábra Egy osztályos osztályozók eredményeinek összehasonlítása

3.2.4 Kétosztályos osztályozó mesterségesen generált aláírásokkal

A munkám során csak az eredeti aláírásokból generáltam mesterséges hamisított mintákat. A hamisítványokat az eredeti aláírások néhány jellemzőjének eltolásával hoztam létre. Itt nagyon fontos az eltolás mértéke a túl nagy és a túl kicsi eltolás is félretanítja a hálózatot.

A tanításhoz 10 eredeti és 10 generált hamisítványt használtam fel. Az aláírások jellemzőit ebben a konfigurációban is kezdetben redukálnom kellett. Az osztályozásra használt osztályozó a Kétosztályos SVM alfejezetben volt részletesen bemutatva.

A kód szintjén a generált hamisítványokat nagyon egyszerűen létre tudtam hozni: $gen\_false\_signatures = tru\_signatures * const$ A teljes kód megtalálható az E. Függelékben. A következő ábrán 3 különböző eltolás értékekkel elért eredmények láthatóak:



3-13. ábra Különböző eltolás értékekkel elért eredmények

Az ábráról láthatóan az 1.05-ös eltolás érték eredményezi a legkisebb hibaarányt, tehát az ebből generált minták közelítik meg legjobban az eredeti hamisítványokat. A következő ábrán az ezen eltolást használó eredmények láthatóak:

Azonosító	*AER*	*FAR*	*FFR*
2	10%	0%	20%
4	15%	30%	0%
6	5%	10%	0%
8	15%	0%	30%
10	25%	40%	10%
13	15%	30%	0%
15	15%	0%	30%
18	15%	20%	10%
20	5%	0%	10%
22	30%	30%	30%
24	20%	30%	10%
25	20%	30%	10%
28	20%	30%	10%
32	0%	0%	0%
33	10%	10%	10%
34	10%	10%	10%
35	0%	0%	0%
37	0%	0%	0%
38	0%	0%	0%
40	0%	0%	0%
Átlag	11,5%	13,5%	9,5%

3-14. ábra Generált hamisítványokkal elért eredmények

Itt sikerült igazán jó eredményeket elérnem, igaz ez is eléggé elmarad az optimális eredményektől. Láthatóan sok, számszerűsítve 10 aláíróra sikerült 0%-10% közötti eredményeket elérnem. Egyedül egy aláíróra kaptam 30%-os hibaarányt.

3.3 TensorFlow megoldások

Ebben a fejezetben bemutatom a TensorFlow segítségével létrehozott osztályozó konfigurációimat. Kezdetben a tanításhoz felhasználtam hamisított mintákat is, hogy lássam, milyen eredményeket tudok elérni ily módon. A következőkben pedig egyedül az eredeti aláírások felhasználásával tanítottam a hálót.

A tanítás előtt minden konfigurációban elvégeztem a következő lépéseket:

1. adatok elő feldolgozása
2. minták generálása
3. jellemzők szűrése szórás alapján

Az elő feldolgozást, a Matlab megoldásokhoz hasonlóan végeztem. (3.2.1Adatok elő feldolgozása) Az irodalmi áttekintésben már említettem, hogy a TensorFlow nagy tanító adathalmazzal tud megfelelően működni. Nekem 20 eredeti és 20 hamisítvány mintám volt, ez nagyon kevés, ahhoz, hogy egyáltalán eredményeket érjek el az eszközzel. Ezért a tanításra kiválasztott halmaz elemeiből új elemeket kellett generálnom. Végül, a jellemzőket szelektáltam a már bemutatott módon. (3.2.3.1Jellemzők szelektálása) Természetesen minden a Matlab megoldásokban lévő algoritmust python nyelven kellett megvalósítani a TensorFlow-nak.

3.3.1 Minták generálása

A mesterséges eredeti minták generálását a már meglévő minták ötvözésével értem el. A célom az volt, hogy létrehozzak olyan mintákat, amelyeket az aláíró írhatott volna. A generálás során a minták közötti különbségeket használtam arra, hogy növeljem a mintaszámot. A konkrét kinyert különbségeket még egy konstans értékkel leosztottam, mert e-nélkül a generált minták minősége nem volt megfelelő. Számokban kifejezve az osztályozások között 20% különbség volt.

A következő kódrészlet a generálás megvalósítását tartalmazza:

```
train = [];
test = [];
```

```
for i in xrange(15):
    for j in xrange(i+1,15):
        diff1 = np.absolute(np.subtract(filtered_true_data[i],filtered_true_data[j])/60)
        diff2 = np.absolute(np.subtract(filtered_false_data[i],filtered_false_data[j])/60)
        for row in xrange(15):
            if row != i:
                train.append(filtered_true_data[row] + diff1)
                test.append(filtered_false_data[row] + diff2)
```

A kódban 15 mintából generálok további mintákat. A *train* változóban az eredeti mintákból generált értékek, a *test* változóban pedig a hamisítványokból generált értékek találhatóak meg. Ezzel az algoritmussal a 15 mintából 1470 mintát hoztam létre.

3.3.2 Tanítás hamisított aláírások felhasználásával

A célom megvizsgálni azt, hogy a hamisított aláírások felhasználásával, milyen eredményeket tudok elérni. A tanításhoz jelen konfigurációban 15 eredeti és 15 hamisítvány aláírást használtam fel, ennek a magyarázata az, hogy 10-10 mintából túl kevés mintát tudtam csak generálni, ahhoz, hogy megfelelően taníthassam a hálót.

A TensorFlow-ban a tanítást, úgynevezett csomónként (batch) kell elvégezni. A tanulás egy iteratív folyamat, ezért van erre szükség. A következő kódrészletben ez látható:

```
begin = 0
end = 60
for i in range(49):
    train_step.run(feed_dict={x: batch_data[begin:end], y_: batch_label[begin:end]})
    begin = end
    end = begin + 60
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

A *train_step* egy már előzőleg létrehozott TensorFlow tenzor, amelyben a tanuló algoritmus van megadva. A *run ()* metódusával futtathatjuk a tanítást. Bemenetként 60 aláírás mintát kap - 30 eredeti és 30 hamisítvány - a hozzájuk tartozó osztály címkékkel.

A *correct_prediction* változóban az eredeti és az osztályozó által adott osztálycímkék összehasonlításának eredménye található. Az *accurary* változóban pedig az osztályozó teljesítménye. Miután a hálót megtanítottuk, a következő lépés, hogy egy teszt adathalmazra megvizsgáljuk, milyen hibával osztályoz. A már eddig is használt hibamutatókat használom itt is.

far = accuracy.eval(feed_dict={x: far_test_array, y_: far_label_test})

ffr = accuracy.eval(feed_dict={x: ffr_test_array, y_: ffr_label_test})

A következő ábrán az elért eredmények láthatóak:

Azonosító	AER	FAR	FFR
2	20%	20%	20%
4	20%	40%	0%
6	10%	20%	0%
8	30%	60%	0%
10	20%	0%	40%
13	20%	0%	40%
15	0%	0%	0%
18	50%	20%	80%
20	0%	0%	0%
22	10%	0%	20%
24	0%	0%	0%
25	0%	0%	0%
28	10%	0%	20%
32	0%	0%	0%
33	50%	100%	0%
34	20%	40%	0%
35	50%	0%	100%
37	60%	100%	20%
38	0%	0%	0%
40	0%	0%	0%
Átlag	18,5%	20,0%	17,0%

3-15. ábra TensorFlow hamisított mintákkal

Láthatóan elég kis hibával osztályoz a konfiguráció. Nem teljesen jogos összehasonlítani az eddigi megoldásokkal, hiszen amíg azokban 10-10 mintával teszteltem, addig itt 5-5 mintával. Az eredmények között láthatóan van néhány

kimagaslóan magasnak mondható pl. a 37-es és a 33-as azonosítóju aláírások. Összességében jó eredménynek tekintem, hiszen a tanítás során mesterségesen generált mintákat használtam, amelyeknek a minősége nyílván rosszabb, mint a 20-20 kapott mintáké. A teljes kód megtalálható az F. Függelékben.

3.3.3 Tanítás csak eredeti aláírások felhasználásával

Ennek a konfigurációnak a tanításához csak az eredeti mintákat használtam fel. Mivel a TensorFlow nem képes egy osztályos osztályozásra, ezért hamisítványokat kellett generálnom az eredeti aláírásokból.

Hamisítványokat a 3.2.4 Kétosztályos osztályozó mesterségesen generált aláírásokkal fejezetben bemutatott módon generáltam.

Sajnos nem sikerült jó eredményt elérnem, ennél az eszköznél nem vált be ez a típusú hamisított mintagenerálás. Az eredmények a következő ábrán láthatóak:

Azonosító	*AER*	*FAR*	*FFR*
2	35%	20%	50%
4	45%	30%	60%
6	15%	20%	10%
8	60%	20%	100%
10	15%	0%	30%
13	30%	50%	10%
15	45%	80%	10%
18	30%	40%	20%
20	20%	40%	0%
22	50%	90%	10%
24	65%	50%	80%
25	30%	60%	0%
28	20%	30%	10%
32	5%	10%	0%
33	30%	20%	40%
34	95%	100%	90%
35	40%	60%	20%
37	35%	10%	60%
38	50%	50%	50%
40	25%	50%	0%
Átlag	37,0%	41,5%	32,5%

3-16. ábra TensorFlow generált mintákkal

Messze elmarad ez az eredmény a hasonló konfigurációjú Matlab-os eredményektől. Az a következtetés vonható le, hogy a TensorFlow-ban alkalmazott

tanítási módszer az ilyen típusú származtatott/generált mintákat valamilyen módon kiszűri.

3.4 Eredmények

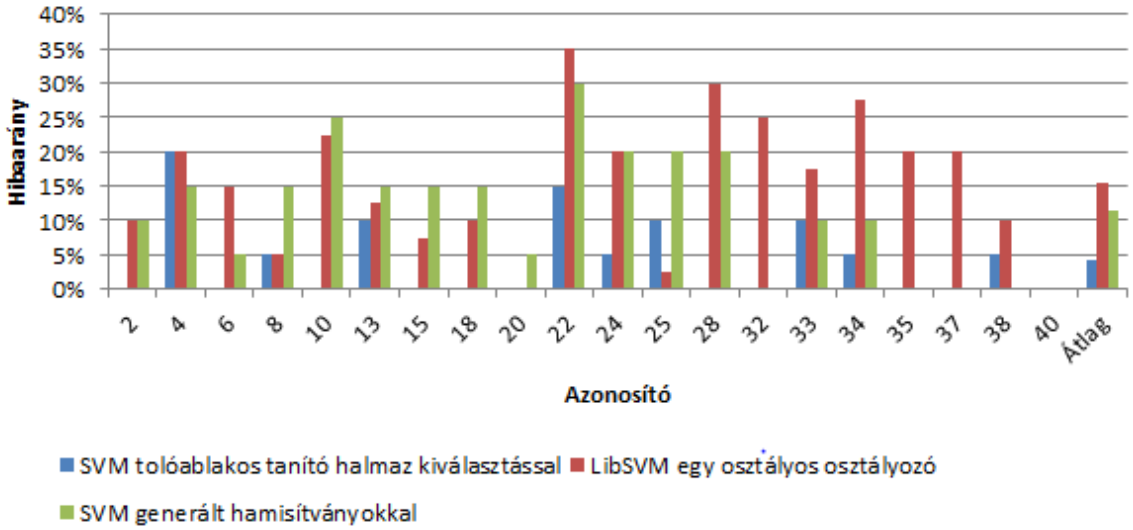
Ebben a fejezetben összesítem az elért eredményeket, összehasonlítom és értékelem őket.

A munkám során két eszköz használhatóságát vizsgáltam meg az offline aláírás hitelesítésben. A Matlab konfigurációk 3 nagyobb részre bonthatóak:

1. Kétosztályos osztályozók hamisított aláírások felhasználásával
2. Egy osztályos osztályozók (anomáliadetektálók)
3. Kétosztályos osztályozók mesterségesen generált hamisítványokkal

Az egyes konfigurációk bemutatásánál az elért eredményeket értékeltem és össze is hasonlítottam őket egymással. Ezek közül az elsőt nem lehet alkalmazni a valós probléma megoldására, hiszen itt a tanításhoz használtam a hamisítványokat is. Viszont e konfigurációk közül a legjobbat tekintettem az elérendő célnak.

A következő ábrán a fentiek 3 legjobb konfigurációinak eredményei láthatóak:

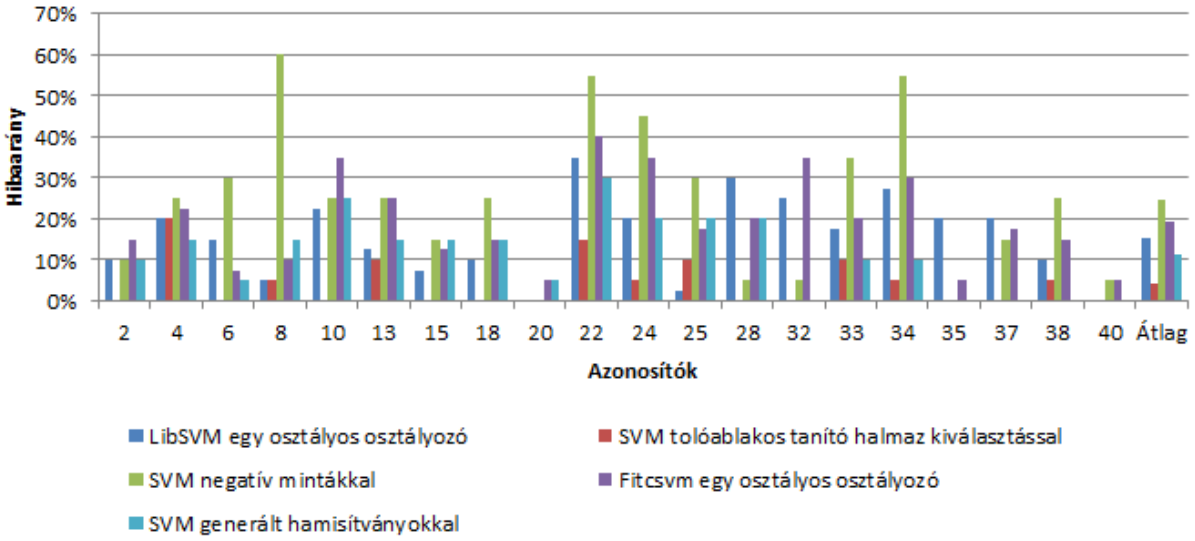


3-17. ábra Legjobb eredmények összehasonlítása

Az eredményeket látva nem sikerült megközelíteni az optimális konfigurációval elért átlagos hibaarányt. Viszont az elért eredmények a valós probléma megoldására korántsem mondhatóak rosszaknak. A hamisítványgenerálós konfigurációmmal sikerült

11.5%-os átlagos hibaarányt elérnem úgy, hogy mindössze két aláíróra a 10-es és a 22-es azonosítójúra haladta meg a hibaarány a 25%-ot. Érdekes még megfigyelni azt, hogy néhány aláíróra mindhárom hasonló eredményeket ad, de vannak olyanok, amelyekre teljesen különbözőt. Itt jegyezném meg, hogy a 22-es aláíróra kapom gyakorlatilag minden konfigurációban az egyik legrosszabb eredményeket.

Tekintettel arra, hogy elég sok SVM alapú konfigurációt hoztam létre a következő ábrán ezeket együtt ábrázolva láthatjuk:



3-18. ábra SVM konfigurációk összehasonlítása

A TensorFlow-val elért eredményeim gyengébbek lettek az előzőektől. Két konfigurációt állítottam össze, a hamisítványokat használó megoldásom viszonylag még jó eredményt ért el 18.5%-os átlagos hibaaránnyal, viszont a generált hamisítványokat használó megoldásom csak 37%-ot. Megjegyezném, hogy a munkám során nagyobb hangsúlyt fektettem a Matlab-os megoldások kidolgozására, úgyhogy valószínűleg jobb eredmények is elérhetőek lennének a TensorFlow-val is.

4 Összegzés

A szakdolgozatomban megismerkedtem az offline aláírás-hitelesítéssel. Megismertem a BME-n fejlesztett automatizált aláírás jellemzőket kinyerő rendszert. Létrehoztam és összehasonlítottam több osztályozó konfigurációt. A dolgozatomban sikerült olyan konfigurációt létrehoznom, amellyel viszonylag kis hibaarány elérhető még akkor is, ha a hamisítványok szakértőktől származnak.

Bemutattam több típusú osztályozási technikát annak jellemzőivel és esetleges problémáival. Kéttípusú megközelítést vizsgáltam meg, az elsőben két osztályos osztályozási feladatként tekintettem a problémára, a másikban pedig anomáliadetektálásként.

A munkám során két eszköz használhatóságát vizsgáltam meg a Matlab-ét és a TensorFlow-ét. A dolgozatomban a Matlab konfigurációk létrehozására fektettem a nagyobb hangsúlyt. Mindkét eszköznél megismertem és felhasználtam a mesterséges intelligencia alapú osztályozási módszereit. Az osztályozók hatékonyságát növelve több elő feldolgozó lépést is beiktattam, ezek a felesleges jellemzők törlését és a jellemzők szűrését oldották meg. Az osztályozóknak ezeken kívül az optimális paraméter értékeit is beállítottam. A munkámat nehezítették az automatikus jellemzőkinyerő rendszerben lévő hibák.

Összehasonlítottam az elért eredményeket, mind osztályozási technikán belül, mind különböző osztályozási technikák között. Jobb eredményeket is elérhetnének az osztályozóim, ha például a jellemzőkinyerés automatizálása javulna. Érdekes lenne kipróbálni, mert elviekben az eredményeknek javulniuk kellene.

Hiányosságnak érzem azt, hogy nem sikerült több időt fektetnem a TensorFlow konfigurációk létrehozására, a jellemzők szűrését a szóráson kívül más módszerrel megoldani, bár kipróbáltam néhányat, de azokkal szörnyű eredményeket kaptam.

Személy szerint én nem bántam meg, hogy ezt a témát választottam úgy érzem sokat tanultam belőle. Az tetszett a legjobban benne, hogy egy valós problémán, gyakorlatban próbálhattam ki azokat az ismereteket, amelyeket az Intelligens rendszerek szakirányomon elméletben megtanultam.

5 Irodalomjegyzék

[1] Kővári Bence, Models and algorithms in off-line, feature-based, handwritten signature verification, PhD tézis, 2013.

[2] B. Kovari, B. Toth és H. Charaf, „Classification Approaches in Off-Line Handwritten Signature Verification," WSEAS Transactions on Mathematics

[3] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József Neurális hálózatok

[4] [online] http://courses.cs.ut.ee/2011/graphmining/Main/KernelMethodsForGraphs

[5] A Library for Support Vector Machines, Chih-Chung Chang and Chih-Jen Lin 2013.

[6] NN toolbox, [online] http://www.mathworks.com/products/neural-network/

[7] SML toolbox, [online] http://www.mathworks.com/help/stats/index.html

[8] Tensorflow, [online] https://www.tensorflow.org/

[9] Deep Learning, [online] https://www.udacity.com/course/deep-learning--ud730

[10] Python, [online] https://www.python.org/

[11] e. a. D.-Y. Yeung, „SVC2004: First International Signature Verification Competition," in Lecture Notes in Computer Science, Biometric Authentication, 2004, pp. 16-22.

[12] fitcsvm, [online] http://www.mathworks.com/help/stats/fitcsvm.html

6 Függelékek

6.1 A. Függelék

```
clear all;
clc;
Data = xlsread('Features.xlsx');

trainBegin = 1;
trainEnd = 20;
testBegin = 21;
testEnd = 40;
SUM = 0;
results = []; fars = []; ffrs = [];
%SVM háló tesztelése
tic
for iter=1:20
    MIN = 500;
    True = preprocessing(Data(trainBegin:trainEnd,1:end));
    trainBegin = trainEnd + 21;
    trainEnd = trainBegin + 19;

    False = preprocessing(Data(testBegin:testEnd,1:end));
    testBegin = testEnd + 21;
    testEnd = testBegin + 19;

    %hány jellemzőt vegyünk figyelembe
    dim = length(True(1,:));

    Train = True(1:10,1:dim);
    Train(11:20,1:dim) = False(1:10,1:dim);

    %a vegyes tanításhoz az osztálycimkék
    d = [1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
    t=Train';

    for param=1:3

        svmStruct = svmtrain(t,d,'kernel_function','polynomial',…
'polyorder',param);

        r = svmclassify(svmStruct,False(11:20,1:dim));
        false = r(r>0);
        FAR = (length(false)/length(r))*100; %hány rosszat osztályzott jónak

        r1 = svmclassify(svmStruct,True(11:20,1:dim));
        false1 = r1(r1<0);
        FFR = (length(false1)/length(r1))*100; %hány jót osztályzott rossznak

        %átlagos hiba
        AER = (FFR + FAR)/2;

        if AER < MIN
           MIN = AER;
           far = FAR;
           ffr = FFR;
```

```
        end

    end

    results(end+1) = MIN;
    fars(end+1) = far;
    ffrs(end+1) = ffr;

    SUM = SUM + MIN;
end
toc
results
%átlagos osztályozási hibaaránya az osztályozónak
res = SUM/iter
%eredmények excel fájlba mentése
xlswrite('svm.xlsx',[results' fars' ffrs']);
```

6.2 B. Függelék

```
clear all;
clc;

Data = xlsread('Features.xlsx');
tic
trainBegin = 1;
trainEnd = 20;
testBegin = 21;
testEnd = 40;
SUM = 0;
results = []; fars = []; ffrs = [];
for iter = 1:20
    True = preprocessing(Data(trainBegin:trainEnd,:));
    trainBegin = trainEnd + 21;
    trainEnd = trainBegin + 19;

    False = preprocessing(Data(testBegin:testEnd,:));
    testBegin = testEnd + 21;
    testEnd = testBegin + 19;

    d = [1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]; %a vegyes
tanításhoz, elvárt kimenet

    MIN = 100;
    offset = 9;
    signature_counts = 1:20;
    %kiválasztom azokat a tanítómintákat, amelyekkel a legkisebb az elért hiba
    for trueFirstIdx = 1:11
        Train = True(trueFirstIdx:trueFirstIdx+offset,:);
        for falseFirstIdx = 1:11
             Train(11:20,1:end) = False(falseFirstIdx:falseFirstIdx+offset,:);
             t=Train';

             for param = 1:3
                  svmStruct =
svmtrain(t,d,'kernel_function','polynomial','polyorder',param);
                  %net = newrbe(t,d,2e6);
                  %net = newrb(t,d,0.075,2e6);
                  test_true_idx = signature_counts(signature_counts >
trueFirstIdx + 9 | signature_counts < trueFirstIdx);
                  test_false_idx = signature_counts(signature_counts >
falseFirstIdx + 9 | signature_counts < falseFirstIdx);
```

```
                    %r= sim(net, False(test_false_idx,:)');%háló tesztelése
                    r = svmclassify(svmStruct,False(test_false_idx,:));
                    false = r(r>0);
                    FAR = (length(false)/length(r)); %hány rosszat osztályzott
jónak
                    %r1= sim(net, True(test_true_idx,:)');%háló tesztelése
                    r1 = svmclassify(svmStruct,True(test_true_idx,:));
                    false = r1(r1<0);
                    FFR = (length(false)/length(r1)); %hány jót osztályzott
rossznak

                    AER = (FAR + FFR)/2;

                    if AER < MIN
                          MIN = AER;
                          far = FAR;
                          ffr = FFR;
                    end

              end
          end
      end
      SUM = SUM + MIN;
      results(end+1) = MIN;
      fars(end+1) = far;
      ffrs(end+1) = ffr;
end
toc
SUM/iter*100
%eredmények kiírása
xlswrite('ablak_svm.xlsx',[results' fars' ffrs']);
```

6.3 C. Függelék

```
function [base_lines_std, loops_std, shapes_std, accents_std, global_std] =
calculate_std(data)
    %jellemzőcsoportok meghatározása
    [T, base_lines, loops, shapes, accents, globals] = preprocessing(data);

    %szórások meghatározása és a NaN értékű szórások helyettesítése
    base_lines_std = std(normalize_data(base_lines));
    base_lines_std(find(isnan(base_lines_std))) = 1;

    loops_std = std(normalize_data(loops));
    loops_std(find(isnan(loops_std))) = 1;

    shapes_std = std(normalize_data(shapes));
    shapes_std(find(isnan(shapes_std))) = 1;

    accents_std = std(normalize_data(accents));
    accents_std(find(isnan(accents_std))) = 1;

    global_std = std(normalize_data(globals));
    global_std(find(isnan(global_std))) = 1;

end
```

6.4 D. Függelék

```
clear all;
clc;
Data = xlsread('Features.xlsx');

trainBegin = 1;
trainEnd = 20;
testBegin = 21;
testEnd = 40;
SUM = 0;
results = []; fars = []; ffrs = [];
%LIB_SVM
tic
for iter=1:20
    MIN = 500;
    [True, base_lines, loops, shapes, accents, globals] =
preprocessing(Data(trainBegin:trainEnd,:));
    [base_lines_std, loops_std, shapes_std, accents_std, global_std] =
calculate_std(Data(trainBegin:trainEnd,:));

    trainBegin = trainEnd + 21;
    trainEnd = trainBegin + 19;

    [False, base_lines_f, loops_f, shapes_f, accents_f, globals_f] =
preprocessing(Data(testBegin:testEnd,:));
    testBegin = testEnd + 21;
    testEnd = testBegin + 19;

    %szórások rendezése
    [base_lines_std bases_idx] = sort(base_lines_std);
    [loops_std loops_idx] = sort(loops_std);
    [shapes_std shapes_idx] = sort(shapes_std);
    [accents_std accents_idx] = sort(accents_std);
    [global_std globals_idx] = sort(global_std);

    %Keresztkiértékeléssel meghatározom a legjobb részhalmazokat
    for b=1:7
        for l=0:6
            for s=0:2
               for a=0:4
                   for g=1:5
                        bcount = b; lcount = l; scount = s;
                        acount = a; gcount = g;

                        if length(loops_idx) == 0
                            lcount = 0;
                        end

                        if length(bases_idx) < bcount
                            bcount = length(bases_idx);
                        end

                        Train = [base_lines(:,bases_idx(1:bcount))
                        loops(:,loops_idx(1:lcount))...
      shapes(:,shapes_idx(1:scount))accents(:,accents_idx(1:acount))
globals(:,globals_idx(1:gcount))];

Test = [base_lines_f(:,bases_idx(1:bcount)) loops_f(:,loops_idx(1:lcount))...
                            shapes_f(:,shapes_idx(1:scount))
accents_f(:,accents_idx(1:acount)) globals_f(:,globals_idx(1:gcount))];

 d = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];
 dtest = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
```

```
                model = svmtrain(d(1:15)',Train(1:15,:),'-s 2 -t 2 -n 0.1 -g
0.00005
');

                    r = svmpredict(d',Test,model);
                    false = r(r>0);
                    FAR = (length(false)/length(r));
                %hány rosszat osztályzott jónak

                    r1 = svmpredict(dtest(1:5)',Train(16:20,:),model);
                    false1 = r1(r1<0);
                    FFR = (length(false1)/length(r1));
                %hány jót osztályzott rossznak

                    %átlagos hiba
                    AER = (FFR + FAR)/2;

                    if AER < MIN
                       MIN = AER;
                       far = FAR;
                       ffr = FFR;
                    end
                                end
                            end
                        end
                    end
                end
    fars(end+1) = far;
    ffrs(end+1) = ffr;
    results(end+1) = MIN;
    SUM = SUM + MIN;
end
toc
results
%átlagos osztályozási hibaaránya az osztályozónak
res = SUM/iter*100
%eredmények excel fájlba mentése
xlswrite('libsvm.xlsx',[results' fars' ffrs']);
```

6.5 E. Függelék

```
clear all
clc
Data = xlsread('Features.xlsx');
d = [1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]; %a vegyes tanításhoz,
elvárt kimenet

trainBegin = 1;
trainEnd = 20;
testBegin = 21;
testEnd = 40;
SUM = 0;
results = []; fars = []; ffrs = [];
tic
for iter=1:20
    [True, base_lines, loops, shapes, accents, globals] =
preprocessing(Data(trainBegin:trainEnd,:));
    [base_lines_std, loops_std, shapes_std, accents_std, global_std] =
calculate_std(Data(trainBegin:trainEnd,:));

    trainBegin = trainEnd + 21;
    trainEnd = trainBegin + 19;
```

```
    [False, base_lines_f, loops_f, shapes_f, accents_f, globals_f] =
preprocessing(Data(testBegin:testEnd,:));
    testBegin = testEnd + 21;
    testEnd = testBegin + 19;

    %szórások rendezése
    [base_lines_std bases_idx] = sort(base_lines_std);
    [loops_std loops_idx] = sort(loops_std);
    [shapes_std shapes_idx] = sort(shapes_std);
    [accents_std accents_idx] = sort(accents_std);
    [global_std globals_idx] = sort(global_std);

    bcount = 5;
    lcount = 2;
    scount = 2;
    acount = 0;
    gcount = 3;

    if length(loops_std) == 0
        lcount = 0;
    end

    Train = [base_lines(:,bases_idx(1:bcount)) loops(:,loops_idx(1:lcount))...
                            shapes(:,shapes_idx(1:scount))
accents(:,accents_idx(1:acount)) globals(:,globals_idx(1:gcount))];

    Test = [base_lines_f(:,bases_idx(1:bcount))
loops_f(:,loops_idx(1:lcount))...
                            shapes_f(:,shapes_idx(1:scount))
accents_f(:,accents_idx(1:acount)) globals_f(:,globals_idx(1:gcount))];

    False_gen = Train * 1.05;

    MIN = 100;
    offset = 9;
    signature_counts = 1:20;
    for trueFirstIdx = 1:11
        Mix = Train(trueFirstIdx:trueFirstIdx+offset,:);
        for falseFirstIdx = 1:11
            Mix(11:20,1:end) =
False_gen(falseFirstIdx:falseFirstIdx+offset,:);
            t=Mix';

            for param = 1:3
                svmStruct =
svmtrain(t,d,'kernel_function','polynomial','polyorder',param);

                test_true_idx = signature_counts(signature_counts >
trueFirstIdx + 9 | signature_counts < trueFirstIdx);
                test_false_idx = signature_counts(signature_counts >
falseFirstIdx + 9 | signature_counts < falseFirstIdx);

                r = svmclassify(svmStruct,Test(test_false_idx,:));
                false = r(r>0);
                FAR = (length(false)/length(r)); %hány rosszat osztályzott
jónak

                r1 = svmclassify(svmStruct,Train(test_true_idx,:));
                false = r1(r1<0);
                FFR = (length(false)/length(r1)); %hány jót osztályzott
rossznak

                AER = (FAR + FFR)/2;
```

```
                    if AER < MIN
                        MIN = AER;
                        far = FAR;
                        ffr = FFR;
                    end
                end
            end
        end
        z = [iter MIN]
        SUM = SUM + MIN;
        results(end+1) = MIN;
        fars(end+1) = far;
        ffrs(end+1) = ffr;
end
toc
SUM/iter*100
%eredmények kiírása
xlswrite('negativ_mintak_generalasa_1.05.xlsx',[results' fars' ffrs']);
```

6.6 F. Függelék

```
#execfile() --> run script
# -*- coding: utf-8 -*-
import csv
import numpy as np
import os
import tensorflow as tf
def preprocessing(data, begin, end):
    f = data[begin:end].sum(axis=0)
    dim = len(data[0]);
    array = [];
    for i in xrange(begin, end):
        temp = []
        for j in xrange(dim):
            if f[j] != 0:
                temp.append(data[i][j])
        if len(temp) != 0:
            array.append(temp)
    return array


def cut_groups(begin, end, group_begin, group_end):
    #jellemzo csoportok az eredeti mintakra
    array = []
    for i in xrange(begin,end):
        temp = []
        for j in xrange(group_begin, group_end):
            temp.append(result[i][j])
        array.append(temp)
```

```
    group = preprocessing(np.array(array), 0, 20)
    return group

def normalize(data):
    #normalizalasa az adatoknak
    norm = []
    oszto = np.subtract(np.amax(data, axis=0), np.amin(data, axis = 0))
    minValue = np.amin(data, axis = 0)
    for i in xrange(20):
        a = np.subtract(data[i],minValue)
        norm.append(np.divide(a, oszto))
    return norm

def sort(data, idx):
    #jellemzők rendezése szórásuk alapján
    data_sort = []
    for i in xrange(20):
        temp = []
        for j in xrange(len(idx)):
            temp.append(data[i][idx[j]])
        data_sort.append(temp)
    return data_sort

def calculate_std(data):
    if len(data) != 0:
        norm = normalize(data)
        #szoras kiszamolasa
        std = np.std(norm, axis = 0)
        std[np.isnan(std)] = 1
        idx = np.argsort(std)
        return sort(data,idx), idx

    return [], []

def calc_true_group_count(bcount,lcount,scount,acount,gcount):

    data = []
    for i in xrange(20):
        temp = []
        temp.append(baselines_sorted_std[i][:bcount])
        if len(loops_sorted_std) != 0:
            temp.append(loops_sorted_std[i][:lcount])
        else:
```

```
            temp.append([])
        temp.append(shapes_sorted_std[i][:scount])
        temp.append(accents_sorted_std[i][:acount])
        temp.append(globals_sorted_std[i][:gcount])

        temp2 = []
        for j in xrange(5):
            for k in xrange(len(temp[j])):
                temp2.append(temp[j][k])
        data.append(temp2)
    return data

def calc_false_group_count(begin, end, bcount,lcount,scount,acount,gcount, b_idx, l_idx, s_idx, a_idx, g_idx):
    baselines = sort(cut_groups(begin, end, 0, 36),b_idx)
    loops = sort(cut_groups(begin, end, 36, 121),l_idx)
    shapes = sort(cut_groups(begin, end, 121, 124),s_idx)
    accents = sort(cut_groups(begin, end, 124, 142),a_idx)
    globales = sort(cut_groups(begin, end, 142, 148),g_idx)

    data = []
    for i in xrange(20):
        temp = []
        temp.append(baselines[i][:bcount])
        temp.append(loops[i][:lcount])
        temp.append(shapes[i][:scount])
        temp.append(accents[i][:acount])
        temp.append(globales[i][:gcount])

        temp2 = []
        for j in xrange(5):
            for k in xrange(len(temp[j])):
                temp2.append(temp[j][k])
        data.append(temp2)

    return data

reader=csv.reader(open("szakdoga/features.csv","rb"),delimiter='       ');
x=list(reader);
result=np.array(x).astype('float');

n = 2;
```

```
true_begin = 0;
true_end = 20;
false_begin = true_end;
false_end = false_begin + 20;

fars = []
ffrs = []
aers = []

sum = 0;
for iter in xrange(20):

    true = preprocessing(result, true_begin, true_end)
    classes = [1, 0];

    false = preprocessing(result, false_begin, false_end)
    classes_test = [0, 1]

    bcount = 8
    lcount = 1
    scount = 0
    acount = 1
    gcount = 2

    baselines_sorted_std, b_idx = calculate_std(cut_groups(true_begin, true_end, 0, 36))
    loops_sorted_std, l_idx = calculate_std(cut_groups(true_begin, true_end, 36, 121))
    shapes_sorted_std, s_idx = calculate_std(cut_groups(true_begin, true_end, 121, 124))
    accents_sorted_std, a_idx = calculate_std(cut_groups(true_begin, true_end, 124, 142))
    globals_sorted_std, g_idx = calculate_std(cut_groups(true_begin, true_end, 142, 148))

    filtered_true_data = calc_true_group_count(bcount,lcount,scount,acount,gcount)

    filtered_false_data=calc_false_group_count(false_begin,false_end,bcount,lcount,scount,acount,gcount,
b_idx, l_idx, s_idx, a_idx, g_idx)

    dim = len(filtered_true_data[0])

    true_begin = true_end + 20;
    true_end = true_begin + 20;

    false_begin = false_end + 20;
    false_end = false_begin + 20;
```

```
train = [];
test = [];
for i in xrange(15):
   for j in xrange(i+1,15):
         diff1=np.absolute(np.subtract(filtered_true_data[i],filtered_true_data[j])/60)
        diff2=np.absolute(np.subtract(filtered_false_data[i],filtered_false_data[j])/60)
      for row in xrange(15):
         if row != i:
            train.append(filtered_true_data[row] + diff1)
            test.append(filtered_false_data[row] + diff2)

sess = tf.InteractiveSession()

length = len(train)

x = tf.placeholder(tf.float32, shape=[None, dim])
y_ = tf.placeholder(tf.float32, shape=[None, n])

W = tf.Variable(tf.zeros([dim,n]))
b = tf.Variable(tf.zeros([n]))

#tf.nn.softmax v. sigmoid v. tanh
y = tf.nn.sigmoid(tf.matmul(x,W) + b)

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y,y_))
train_step = tf.train.AdamOptimizer(2e-3).minimize(cross_entropy)

#AdamOptimizer; sigmoid; dim=10; 84%; 2e-3; 80 - 45
sess.run(tf.initialize_all_variables())

train_array = np.zeros((2*length, dim))
label_train = np.zeros((2*length,n));

for i in range(length):
   train_array[i] = train[i][:dim]
   label_train[i] = classes

for i in range(length):
   train_array[i+length] = test[i][:dim]
   label_train[i+length] = classes_test

batch_data = []
batch_label = []
```

```
flag = False
j = length
k = 0
for i in range(2*length):
    if i % 30 == 0 and i != 0:
        flag = not flag

    if not flag:
        batch_data.append(train_array[k])
        batch_label.append(label_train[k])
        k = k + 1
    if flag:
        batch_data.append(train_array[j])
        batch_label.append(label_train[j])
        j = j + 1

#TRAIN
begin = 0
end = 60
for i in range(49):
    train_step.run(feed_dict={x:batch_data[begin:end],y_: batch_label[begin:end]})
    begin = end
    end = begin + 60

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#VALIDATION MODEL
ffr_test_array = np.zeros((5, dim))
ffr_label_test = np.zeros((5,n))

far_test_array = np.zeros((5, dim))
far_label_test = np.zeros((5,n))

for i in range(15,20):
    far_test_array[i-15] = filtered_false_data[i][:dim]
    far_label_test[i-15] = classes_test

for i in range(15,20):
    ffr_test_array[i-15] = filtered_true_data[i][:dim]
    ffr_label_test[i-15] = classes
```

```
    far = accuracy.eval(feed_dict={x: far_test_array, y_: far_label_test})
    ffr = accuracy.eval(feed_dict={x: ffr_test_array, y_: ffr_label_test})
    aer = (far+ffr)/2;
    fars.append(1-far)
    ffrs.append(1-ffr)
    aers.append(1-aer)
    sum = sum + aer
    print(str(iter+1) + " AER: %f" % aer + " FAR: %f" % far + " FFR: %f" % ffr)
    sess.close()
print str((sum/(iter+1))*100) + "%"
results = []
results.append(aers)
results.append(fars)
results.append(ffrs)
os.remove('tf_negativ_mintakkal.txt')
create = open('tf_negativ_mintakkal.txt','wb')
np.savetxt("tf_negativ_mintakkal.txt",np.array(results).transpose(),delimiter=";",fmt='%.2f')
create.close()
```