Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Horváth Ádám

AUTOMATIKUS OSZTÁLYOZÁSI MÓDSZEREK AZ OFFLINE ALÁÍRÁSHITELESÍTÉSBEN

KONZULENS

Kővári Bence

BUDAPEST, 2011

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Horváth Ádám**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2011. 12. 06.

.....

Horváth Ádám

Összefoglaló

Az aláírás egyike a legrégebbi biometrikus azonosító módszereknek, amelyet praktikussága és hatékonysága miatt ma is széles körben használnak. A mesterséges intelligencia által elvégzett aláírás hitelesítés esetén különbséget teszünk az online és az offline forgatókönyv között aszerint, hogy az aláírás és a hitelesítés folyamata időben egybeesik vagy elkülönül egymástól. Az offline aláírás hitelesítés kihasználja az aláírás igazi gyakorlati előnyét, mivel a gyakorlatban legtöbbször csak egy aláírt papír áll a rendelkezésünkre. Ez a módszer ezen kívül sokkal inkább felhasználóbarát és széles körben alkalmazható.

Annak ellenére, hogy a szakterület mára már jelentős háttérrel rendelkezik, és évtizedek óta próbálnak mesterséges intelligencia felhasználásával különféle módon offline aláírás hitelesítő rendszert létrehozni, még napjainkban sem találhatunk olyat, amely képes a 10 százalékos hibahatár átlépésére.

Az AHR (Aláírás Hitelesítő Rendszer) projektben egy olyan offline aláírás hitelesítő rendszert próbáltunk meg felépíteni, amely az aláírás hitelesítés folyamatát a lehető legkisebb, egymástól független lépésekre bontja le, majd megpróbáltuk ezeket a lépéseket külön-külön optimalizálni.

Jelen diplomamunka túlnyomó részben az aláírás hitelesítés utolsó fázisával, az osztályozással foglalkozik. Az osztályozó módszerek és a jelenleg megtalálható aláírás hitelesítő rendszerek áttekintése után az osztályozási folyamat két lehetséges megoldását tárgyaljuk. Megvizsgálunk egy neurális hálón és statisztikai modellen alapuló megközelítést. Szemügyre vesszük a tervezési és implementációs folyamat főbb kihívásait és az alkalmazott megoldásokat, majd megvizsgáljuk és elemezzük az osztályozás során elért eredményeket.

Abstract

Siganture verification is one of the oldest methods of biometric identification, which is – due to practicality and effectiveness – still widely used. In the case of artificial intelligence based signature verification, a distinction is made between online and offline scenarios depending on whether the signing and the verification processes coincide, or are separated in time. Off-line signature verification takes has a real practical benefit, because in practice most of the time only a signed paper is available. This method is also much more user-friendly and widely applicable. Even though the field now has a significant background, and people tried to create off-line signature verification system for decades using artificial intelligence in various ways, even today you cannot find one that is capable of crossing the 10 percent margin of error.

In AHR (Signature Verification System) project we tried to build an off-line signature verification system, which decomposes the signature verification process into the smallest possible independent steps and we tried to optimize these steps individually.

This thesis mainly focuses on the classification, the last phase of signature verification. After the review of classification methods and the currently available signature verification systems we discuss two possible solutions for the classification problem. We examine a neural network based and a statistical model based approach. We look at the main challenges of the design, the implementation process, and finally we examine and analyze the results achieved by the classification.

1 Bevezetés

1.1 Motváció, a feladat indokolhatósága

Biometrikus hitelesítésre egyre nagyobb szükség van az mind a hétköznap élet, mind a jog különböző területein. Az aláírás egyike a legjobban felhasználható biometrikus azonosítási módszereknek. Előnye, hogy két ember aláírása legalább annyira különböző, mint két ember ujjlenyomata, viszont az ujjlenyomathoz képest sokkal egyszerűbben, gyorsabban és olcsóbban rögzíthető, mivel mindössze egy papírra és egy íróeszközre van hozzá szükség. Éppen emiatt a praktikussága miatt talán ennek az azonosítási formának a legelterjedtebb a hamisítása. Tekintve, hogy egyre több helyen használjuk személyazonosság ellenőrzésére az aláírásokat, gyakorlatilag nem kivitelezhető, a képzett írásszakértők alkalmazása az esetek döntő többségében. Mindemellett napjainkban egyre gyorsabb hardver eszközök válnak elérhetővé, és e két tényező kívánatossá teszi valamilyen automatizált aláírás hitelesítő rendszer meglétét.

Szem előtt kell tartanunk, hogy a rendszer célja a gyors, egyszerű és olcsó aláírás ellenőrzés megvalósítása, ezért nem kielégítő a megoldás, ha valamilyen drága és kevés számban elérhető hardverre van hozzá szükség.

1.2 Automatikus osztályozás

1.2.1 Osztályozó módszerek

Az osztályozó módszereket általános esetben arra használják, hogy az emberhez hasonló „intuitív” módon próbáljon következtetni bizonyos összefüggésekre. Matematikai megfogalmazásban a rendszernek egy függvény összes pontját kell „kitalálnia”, annak néhány megadott pontja alapján. Az offline aláírás hitelesítés során először megpróbálunk az aláírás képéből következtetni a fent említett írásjellemzőkre, illetve ezeket összehasonlítjuk más aláírásokkal és az eredményt számszerűen fejezzük ki. Ezután valamilyen osztályozó módszert használunk, amelynek az a feladata, hogy felismerje az összefüggést a kialakul számok és az aláírás hitelessége vagy hamissága között. Az osztályozónak feladata még az is, hogy egyfajta visszacsatolást adjon a korábbi fázisok felé azáltal, hogy utal arra, hogy melyik írásjellemző az, amely által az

eredeti aláírásokat hatékonyabban lehet elkülöníteni a hamisaktól, és melyik az, amely kevéssé használható.

1.2.2 Az offline aláírás hitelesítés nehézségei

Mivel egy ember általában nem minden esetben ugyanúgy ír alá, a rendszer betanításához először az illető aláírás mintáit kell felhasználni. Ezen a ponton elkülönülnek az online és offline hitelesítő rendszerek. Az utóbbiak alkalmazása a legtöbb esetben sokkal kézenfekvőbb és nem utolsó sorban olcsóbb, mivel nincs szükség drága hardver eszközökre, csak egy egyszerű beolvasóra. Sajnos egyúttal az offline hitelesítők jóval kevésbé hatékonyak, mivel az aláírásból származó információk jelentős része (pl.: tollvonások, nyomás, az írás sebessége) elvész, és csak közvetett módszerekkel lehet az aláírási folyamatot rekonstruálni.

Az online és offline aláírás hitelesítő rendszerek között az alapvető különbség az, hogy online rendszer esetén az aláírás és a hitelesítési folyamat egy időben történik, míg offline rendszer esetén ez a két folyamat egymástól elkülönül. Így az offline rendszerek amiatt is praktikusak lehetnek, mert lehetőség van akár olyan aláírások utólagos megvizsgálására is, amelyek még a rendszer elkészülte előtt lettek papírra vetve. Ez a praktikusság egyben az ilyen rendszerek egyetlen igazi hátrányának az okai is, mivel az offline rendszerek kizárólag az aláírás beolvasott, legtöbbször szürke árnyalatos képét kapják csak meg, és ebből kell ezután az eredeti írásjellemzőket rekonstruálniuk. Emiatt a jelenlegi online és offline rendszerek pontosságában is nagyságrendi különbségeket találhatunk. Ezek az eltérések annyira jelentősek, hogy míg olyan online rendszert könnyen találhatunk, amely készen áll a valódi, éles helyzetben történő alkalmazásra, az offline rendszerekről ugyanezt már nem mondhatjuk el.

1.2.3 A feladat értelmezése

A megvalósított feladat több különálló részre bontható le: Először szükségünk volt egy kis kutatómunkára, az automatikus osztályozás és az aláírás hitelesítés területén. Meg kellett ismernünk az automatikus osztályozási lehetőségeket, és hogy ezek közül melyeket és hogyan használták fel eddig az aláírások azonosítására. Ezután létrehoztuk és teszteltük a saját fejlesztésű osztályozóinkat. Itt alapvetően két irányba indultunk el. Először neurális hálókat használtunk az osztályozás megvalósításához, abban bízva, hogy egy ilyen intuitív és széles körben elterjedt technika itt is közelebb vihet minket a sikerhez. Habár sikerült több ígéretesnek mondható részeredményt

elérnünk az ilyen osztályozókkal, úgy döntöttünk inkább olyan megoldást választunk, amelynek jobban nyomon követhető matematikai alapjai vannak. Kidolgoztuk az aláírás hitelesítés egy olyan statisztikai modelljét, amely a lehető legkevesebb előfeltételt támasztja az aláírásokkal szemben, és szilárd matematikai alapokon nyugszik. Az ígéretes teszteredmények után felismertük, hogy a rendszer további fejlesztéséhez új aláírás adatbázisokra is szükségünk lesz, ezért aláírásgyűjtésbe fogtunk, majd az aláírások feldolgozásával létrehoztunk egy új, minden eddiginél több aláírást tartalmazó adatbázist. Méréseket végeztünk a korábbi és az új adatbázisokon, és következtetéseket vontunk le a mérésekből a későbbi lehetséges irányokra vonatkozóan.

1.3 Neurális hálók

Sokféle osztályozó módszer létezik, amelyek mindegyikét az élet más-más területén lehet hatékonyan alkalmazni. Az írásfelismerés és aláírás felismerés estén leggyakrabban valamilyen neurális háló alapú osztályozót használnak. A neurális háló egy komplex struktúra, amely egyszerű elemekből épül fel, emiatt az implementációja meglehetősen egyszerűen elkészíthető egy objektum orientált környezetben, ugyanakkor hálórendszerek kiépítése valamint a háló struktúrájának variálhatósága számos lehetőséget biztosít.

A sok variáció egyúttal azt a veszélyt is magában hordozza, hogy nagyon nehéz megtalálni a megfelelő hálóstruktúrát. Az egyes hálókból felépíthető hálórendszerek lehetősége szintén növeli a variációk számát, így az ideális megoldás megkereséséhez nagy mennyiségű teszt futtatása szükséges.

Az aláírások osztályozásának egyik – talán – legnagyobb kihívása az a peremfeltétel, hogy az osztályozó rendszer betanítása során kizárólag eredeti aláírások állnak rendelkezésre. A gyakorlati alkalmazást szemügyre véve ez logikus is, mivel általában nincs lehetőség profi hamisítókat megkérni, hogy próbáljanak meg hamisítani egy-egy aláírást, amivel azután az osztályozó rendszer tanítható lenne. Az egyik lehetőség, amivel kompenzálni tudjuk ezt a problémát, hogy megpróbálunk mesterségesen olyan adatokat generálni, amelyek valószínűleg hasonlóak azokhoz az adatokhoz, amelyeket hamisított aláírásokból tudnánk kinyerni. A neurális háló alapú osztályozókról tervezéséről a 3.2-es alfejezetben olvashatunk bővebben.

1.4 Statisztikai módszerek

A neurális hálók legnagyobb hátránya, hogy a struktúra rendkívül bonyolult lehet, és így idővel nagyon nehéz a belső működését átlátni. Emiatt egy olyan statisztikai modell kidolgozása mellett döntöttünk, amely egyszerű, ugyanakkor pontos és átlátható képet ad az aláírásokról. A modell egyik legnagyobb előnye, hogy valószínűségekkel számol, így pontosan meg lehet mondani előre, hogy ideális esetben mekkora lesz helyes osztályozási eredmény valószínűsége. Ha a gyakorlatban ettől eltérő eredményt tapasztalunk, akkor felderíthetjük az eltérések okait és testre szabhatjuk a rendszert. A létrehozott statisztikai modellről a 3.3-as alfejezetben olvashatunk bővebben.

1.5 A diplomamunka felépítése

A következő fejezetekben először áttekintjük és összehasonlítjuk a szakterület irodalmában fellelhető aláírás hitelesítő rendszereket, majd megvizsgáljuk az osztályozó módszerek elméleti alapjait. Ezután megvizsgáljuk a korábban említett két osztályozó típus tervezési, és implementációs kérdéseit. A diplomaterv utolsó részében a fenti rendszerek tesztelési eredményeit mutatjuk be és szemügyre vesszük azokat a következtetéseket és tanulságokat, amelyeket ezekből az eredményekből levonhatunk.

2 Az automatikus osztályozás és az aláírás hitelesítés irodalmi áttekintése

Ebben a fejezetben először szemügyre vesszük az aláírás hitelesítő rendszerek összehasonlításának alapelveit és korlátait valamint a jelenlegi offline rendszerek eredményeit, majd áttekintjük a leggyakrabban használt automatikus osztályozási módszereket. Ezután megvizsgálunk néhány konkrét példát a korábban tárgyalt osztályozási módszerek aláírás hitelesítésben való alkalmazására, végül megnézzük miképpen lehet az írásjellemzőket biometrikus jellemzőként kezelni és hogyan lehet ezt felhasználni az osztályozási feladathoz.

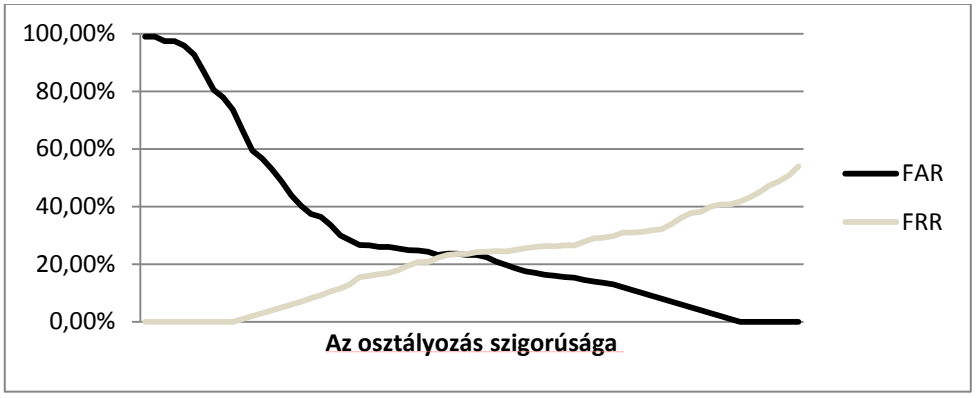
2.1 Aláírás hitelesítő rendszerek összehasonlítása

2.1.1 A teljesítmény mérése

Amikor aláírás hitelesítési rendszerek eredményességét hasonlítjuk össze tekintettel kell lennünk a rendszer tesztelése során használt aláírás adatbázisra és az adatbázis felhasználásának módjára. Az aláírás adatbázisokban általában több aláíró fejenként 10-20 aláírása található meg és legtöbb esetben tartozik minden aláíróhoz ugyanennyi hamisítvány is. Széles skálán mozog ezeknek a hamisítványoknak a minősége: Az úgy nevezett **véletlenszerű hamisítványok (random forgeries)** olyan aláírások, amelyeket véletlenszerűen választottak ki a többi aláíróhoz tartozó eredeti aláírások közül, az **egyszerű hamisítványokat (simple forgeries)** az eredeti aláírások rövid megfigyelése után, míg a **képzett hamisítványokat (skilled forgeries)** az eredeti aláírások hosszas és alapos tanulmányozása, valamint az aláírási folyamat gyakorlása után hozzák létre. A valós forgatókönyvnek az felel meg a leginkább, ha az osztályozó rendszer csak eredeti aláírásokat használhat fel a tanuláshoz, és ha tesztelés során eredeti aláírásokat és képzett hamisítványokat használunk.

Egy aláírás hitelesítő rendszer teljesítményét általában hibaarányokkal szokták kifejezni. Az osztályozási hiba két fajtáját különböztetjük meg: Előfordulhat olyan eset, hogy a rendszer tévesen utasít vissza eredeti aláírásokat (első fajú hiba, False Rejection Rate, röviden: **FRR**). A másik hibalehetőség, amikor a rendszer tévesen fogad el hamis aláírásokat (másodfajú hiba, False Acceptance Rate, röviden: **FAR**). A legtöbb

hitelesítő rendszer rendelkezik valamilyen paraméterrel, amelyet a rendszer „szigorúságának” is szoktak nevezni. Ennek a paraméternek a módosításával az **FAR** jellegű hibákat csökkenteni lehet az **FRR** jellegű hibák növelése árán és fordítva. (Lásd a 2.1-es ábrát)



2.1 ábra: FAR és FRR értékek változása az osztályozás szigorúság szerint

Azt a pontot ahol az FAR értéke azonos az FRR értékével „equal error rate”-nek (röviden: **EER)** nevezik. Ez a mérőszám általánosan használható a különböző osztályozó rendszerek összehasonlítására.

2.1.2 Nemzetközi aláírás hitelesítési versenyek

A „First International Signature Verification Competition”-ön (első nemzetközi aláírás hitelesítési verseny, [1]) 16 különböző online aláírás hitelesítési rendszert teszteltek le hasonló körülmények között. Ezen rendszerek között számos olyan is található, melyet a szakterület vezető kutatócsoportjai fejlesztettek ki. Az itt szereplő legjobb rendszer **2.8%**-os, míg a második helyezett **4.4%**-os átlagos EER arányt produkált. Néhány tanulmány arra hívja fel a figyelmünket, hogy még ennél is jobb arányokat lehet elérni: **0.35%**-os EER arányról olvashatunk [2]-ben és **0.4%**-ról [3]-ban, fontos azonban megjegyeznünk, hogy ezeket az arányokat más adatbázisokon mérték.

Tudomásunk szerint nem létezik olyan tanulmány, amivel offline aláírás hitelesítő rendszereket hasonlítottak volna össze azonos feltételek mellett, ám egy nemrég készült felmérés [4] kiváló áttekintést ad a területről. A [5] szerzői 3%-os FRR arányt értek el neurális háló alapú osztályozóval, azonban csupán 10 aláíró, összesen 160 aláírását használták fel, ráadásul a feladatot csak az aláírás felismerésére korlátozták le: A rendszernek 60 aláírást kellett hozzárendelnie a korábban már megismert aláírókhoz. A [6]-ban arról olvashatunk, hogy 1%-os EER-t sikerült elérniük

amikor véletlenszerű hamisítványokkal szemben tesztelték a rendszerüket, viszont a hibaarány 19.8%-ra növekszik ha ugyanezt a rendszert képzett hamisítványokkal szemben vizsgálják. Annak érdekében, hogy kiszűrjük az előzőekhez hasonló irreális eredményeket az alábbiakban formalizáljuk azokat az előfeltételeket, amelyek illeszkednek a gyakorlatban is előforduló esetekhez:

- A hitelesítő rendszer nem használhatja ki azt a tényt, hogy ismeri a teszt során előforduló összes lehetséges aláírást, mivel valós körülmények között sincs ilyen ismeretünk.
- A rendszer a tanulás során nem használhat fel egyetlen hamisított aláírást sem. Így tehát a rendszer nem ismert egyetlen ember alkotta hamisítvány sem a tanulási fázisban, habár a mesterségesen generált hamisítványok megengedettek.
- Feltételezzük, hogy az a rendszer, amelyik képes felismerni a képzett hamisítványokat még jobb teljesítményt nyújt a véletlenszerű és az egyszerű hamisítványokkal szemben, ezért a vizsgálat során csak képzett hamisítványokat használunk.
- A hitelesítési tesztet olyan adatbázison kell elvégezni, amely elegendően nagyméretű, és statisztika szempontból reprezentatív. Tudomásunk szerint nyilvánosan nem elérhető olyan adatbázis, amely egyáltalán megközelíti ezt a feltételt, így ennek teljesülésétől eltekintünk, ám megjegyezzük, hogy a kis mintákon kimutatott eredményeket óvatosan kell kezelni.

Azok közül a rendszerek közül, amelyek teljesítik a fenti feltételeket, a legjobb eredményeket [7] (9%-os FRR és 13.3%-os FAR), [8] (13.3%-os EER) és [9] (9.8%-os EER) éri el. Azok alapján a felmérések alapján, amelyek együttesen több mint 500 cikket ölelnek fel az elmúlt 30 évből ([4], [10], [11], [12] és [13]) bátran kijelenthetjük, hogy az offline aláírás hitelesítés hiba aránya **10%** körül mozog és az elmúlt tíz évben nem volt szignifikáns fejlődés ezen a területen.

2.1.3 Az írásszakértők hibaaránya

Ahhoz hogy még jobban megértsük ezeket az eredményeket, érdemes még összehasonlítanunk az aláírás hitelesítő rendszerek és az írásszakértők teljesítményét. A [14]-ben olyan tanulmányt olvashatunk, amely 28 írásszakértő 3 éves munkáját összegezte. Ez idő alatt összesen 11643 aláírást (3387 eredetit és 8256 képzett

hamisítványt) vizsgáltak meg a szakértők, akik **7%** alatti **FRR** hibaaránnyal tudták azonosítani az eredeti aláírásokat. Sajnos ez a felmérés nem nyújt betekintést a FAR hibaarányba, ezért ebből még könnyen vonhatnánk le téves következtetéseket. A [15]-ban olvasható tanulmányban 432 eredeti és 333 hamis aláírást vizsgáltak meg 51 különböző aláírótól és szakértőnek nem tekinthető embereket kértek meg aláírások azonosítására, akik **9%**-os **FRR** és **10.5%**-os **FAR** hibaarányokkal teljesítették feladatot. További tanulmányok ([16] és [17]) azt mutatják ki, hogy az írásszakértők akár **0.5%**-os **FAR** és **7%**-os **FRR** elérésére is képesek, míg a laikusok csak 6.5%-os FAR és 26%-os FRR arányokra.

Amíg az emberi teljesítmény felülmúlja az automatikus offline aláírás hitelesítés teljesítményét, addig szükség van az osztályozási módszerek továbbfejlesztésére. Az elmúlt évtizedben több megoldást is bemutatattak az offline aláírás hitelesítés korlátainak legyőzésére és az online aláírás hitelesítéshez képest jelentkező információ veszteség kompenzálására [13]. A 10%-os határ áttöréséhez alapvetően az algoritmus során keletkező különféle hibaforrások azonosítására, megértésére és kompenzálására lenne szükség. Habár több ilyen kutatást is végeztek az online hitelesítés esetén ([18] és [19]) az offline hitelesítés esetén egyelőre hiányzik az analitikus megközelítés.

2.2 Osztályozó módszerek

Amikor automatikus osztályozásról beszülünk, akkor általában egy olyan rendszert értünk alatta, amely képes megtanulni egy függvényt annak néhány megadott pontja alapján. Ezeket a pontokat nevezzük tanító mintáknak. A tanító minták természetesen nem határozzák meg a teljes függvényt, így annak „kitalálása” a rendszer feladata. A tanulási folyamat során a rendszer vagy sorra veszi a tanító mintákat, vagy egyszerre veszi őket figyelembe. A következő szakaszokban néhány osztályozó módszert ismertetünk.

2.2.1 Döntési fák

A döntési fákat akkor alkalmazhatjuk, ha a megtanulandó függvény értéke diszkrét. A fa csomópontjai – a levelek kivételével – a bemeneti attribútumok, vagyis a tanulandó függvény értelmezési tartományának egy-egy dimenziója. Az egyes csomópontokban a fa a csomópont értéke szerint ágazik el. A levél elemekben a függvény értéke található. Ha a fa által megtanult függvényt ki akarjuk értékelni egy

adott pontban, akkor a gyökérelemtől kezdve végig kell lépkednünk az egyes attribútumok értéke szerint mindaddig, amíg levél elemhez nem érkezünk. Ekkor az ott talált érték a válasz.

A tanulási folyamat ebben az esetben a fa felépítését jelenti. Ehhez az összes tanító mintát egyszerre vesszük figyelembe, majd valamilyen szabály alapján kiválasztjuk a legmegfelelőbb attribútumot, aminek segítségével egy csomópontot hozunk létre. Az attribútum egyes értékei alapján a tanító mintákat több csoportra osztjuk, majd az egyes csoportokra rekurzívan megismételjük a fenti lépést. A fa akkor készült el, ha minden levelében csak egyféle értékű tanítóminták vannak, ekkor a levelekbe beírhatjuk ezeket az értékeket. A részletes algoritmus példákkal szemléltetve megtalálható [20]-ban.

Az aláírás hitelesítés során a döntési fák módszere nem használható hatékonyan, mivel esetünkben a megtanulandó függvény értéke nem diszkrét, valamint viszonylag kisméretű tanító halmaz áll rendelkezésre és a tanító minták csak egyféle függvényértéket tartalmaznak.

2.2.2 Legközelebbi szomszéd modell

A modell alapötlete, hogy a megtanulandó függvény valószínűleg olyan, hogy az értelmezési tartományban egymáshoz közel eső pontok függvényértéke is közeli. Ez alapján, ha meg akarjuk határozni a függvény egy eddig ismeretlen pontjának értékét, akkor arra úgy következtethetünk, hogy megvizsgáljuk, milyen sűrűn találhatók pontok a tulajdonságtérben a kérdéses ponthoz közel. Mivel kulcsfontosságú annak definiálása, hogy mit értünk „közel” alatt, a feladatot átfogalmazhatjuk úgy is, hogy mekkora annak az „n” dimenziós gömbnek a sugara, amelynek középpontja a kérdéses pont, és legalább „k” pontot tartalmaz.

A döntési fákhoz képest nagy előnye a módszernek, hogy egyféle értéket tartalmazó tanító minták esetén is működik, ám a korábbinál is nagyobb problémát jelent, hogy a rendelkezésre álló tanítóhalmaz igen kisméretű. [20]

2.2.3 Kernelmódszer

A kernelmodellben úgy tekintünk minden tanító mintára, mintha egy kis saját függvényt generálna. Az eredő függvénybecslés, nemes egyszerűséggel a kis kernelfüggvények súlyozott összege. A legnépszerűbb kernelfüggvény egy olyan gömbszimmetrikus Gauss függvény, amelynek minden tengely mentén w a szórása.

Az i. kernelfüggvény tehát:

$$K(\bar{x}, \bar{x}_i) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-\frac{D(\bar{x},\bar{x}_i)^2}{2w^2}}$$

Ahol D(**x**, **y**) az x és y pontok távolsága, és d az értelmezési tartomány dimenzióinak száma. A kernelfüggvény a tér minden pontjához egy valószínűséget rendel, amelyből sűrűségbecsléssel kaphatjuk a megtanulandó függvényt, a következőképpen:

$$P(x) = \frac{1}{N}\sum_{i=i}^{N} K(\bar{x}, \bar{x}_i)$$

A módszer előnye, hogy ha később kiderül, hogy valamelyik tulajdonság fontosabb másoknál, akkor az átlag súlyozásakor ezt könnyen figyelembe vehetjük, azonban nehézkes az egyes tulajdonságok fontosságának megállapítása. További probléma lehet w paraméter megfelelő behangolása. [20]

2.2.4 Neurális hálók

A neurális hálók irányított kapcsolatokkal összekötött egységekből állnak. Az *i.* egységtől a *j.* egységig vezető összeköttetés súlyát $\boldsymbol{W_{ij}}$-vel, az egységek aktivációs függvényét $\boldsymbol{g}$-vel, míg a kimenetét $\boldsymbol{a_i}$-vel szoktuk jelölni. A csomópont működése:

$$a_i = g\left(\sum_j W_{ji} a_j\right)$$

Ha a bemeneti neuronokat az értelmezési tartomány, a kimeneti neuronokat az értékkészlet dimenzióinak tekintjük, akkor a neurális háló tulajdonképpen egy függvény. A háló tanulási képessége függ háló struktúrájától, a csomópontok számától és az aktivációs függvénytől is.

A háló struktúrájának tekintetében beszélhetünk előrecsatolt és visszacsatolt hálókról. Az előrecsatolt hálónak nincs belső állapota (a súlyoktól eltekintve), így a pillanatnyi bemenet függvényét reprezentálja, míg a visszacsatolt hálók memóriával is rendelkeznek, és lehetnek stabil állapotban, de mutathatnak oszcillációt, sőt kaotikus viselkedést is.

Az előrecsatolt hálókat tovább bonthatjuk egyrétegű hálókra (perceptronok) és több rétegű hálókra. Egyrétegű esetben az összes bemenet közvetlenül egy kimeneti egységre csatlakozik. Főként akkor használhatóak, ha a megtanulandó függvény az

értelmezési tartományban lineárisan szeparálható (azaz az értelmezési tartományt egy hipersíkkal fel tudjuk osztani úgy, hogy az igaz és hamis értékek a sík egy-egy oldalára kerüljenek). A több rétegű hálók esetén, van legalább egy rejtett réteg, azaz olyan csomóponthalmaz, amelynek semelyik eleme sem kimenet.

Hálóarchitektúra tekintetében választhatunk teljesen vagy nem teljesen összekötött hálókat, valamint az egyes rejtett rétegek csomópont száma sem triviális.

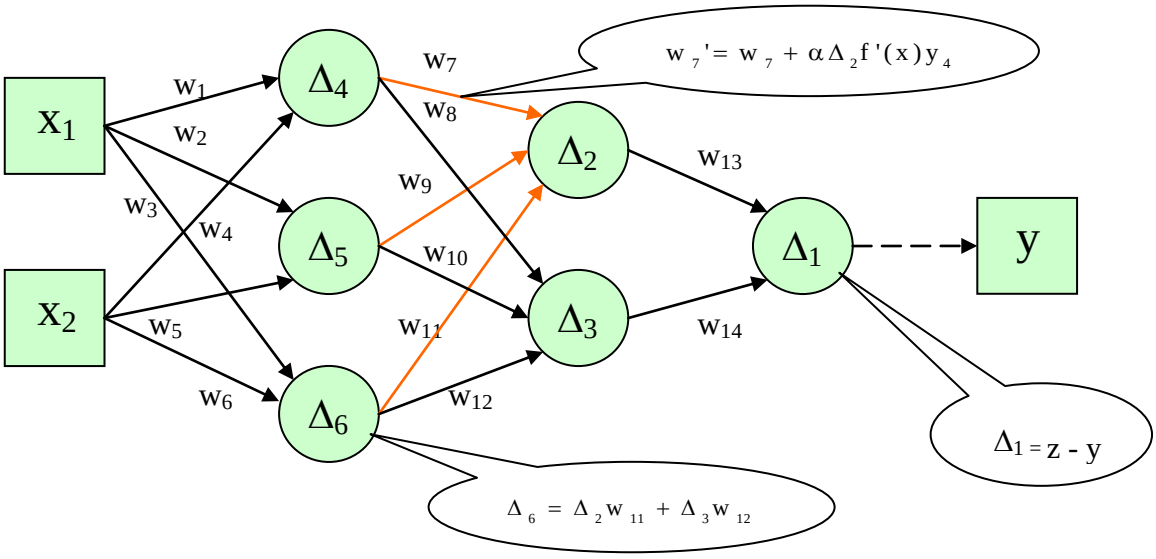
Az aktivációs függvényt szemügyre véve két kedvelt megoldás létezik, a küszöbfüggvény, amely 0-nál kisebb értékekre 0, nagyobb értékekre 1 eredményt ad, és a szigmoid függvény, amely: $1/(1 + e^{-x})$

Neurális hálók tanítását, egyetlen tanító mintával úgy végezhetjük, hogy az adott válasz hibájának megfelelően korrigáljuk az összeköttetések súlyait. Több elemű tanító mintahalmaz esetén sokféleképpen eljárhatunk, az egyik lehetséges megoldás, hogy sorra vesszük a mintákat, és ezt addig ismételjük, míg a háló válaszának hibája, egy adott szint alá nem csökken.

Népszerű súlyfrissítési szabály: $W_{ji} \leftarrow W_{ji} + \alpha * a_j * \Delta_i * g'(in_i)$, ahol Δ_i a kimenet eltérése a kívánt értéktől, és α, az úgy nevezett bátorsági faktor. (Bizonyos szakirodalmakban tanulási sebességként is hivatkoznak rá.)

Felmerülhet még a kérdés, hogy milyen módon határozzuk meg a „kívánt értéket” egy középső réteg kimenete esetén. Erre a hiba visszaterjesztés módszerét szokták használni, ami a 2.2-es ábrán látható:

- Számítsuk ki a kimeneti neuronokra Δ értékét a megfigyelt hiba alapján.
- A kimeneti réteggel kezdve ismételjük a következőket minden rétegre:
  - Terjesszük vissza a Δ értékeket a megelőző rétegre: $\Delta_j = \sum_i W_{ji} \Delta_i$
  - Frissítsük a két réteg közötti súlyokat.



2.2 ábra: Hiba visszaterjesztés és élek frissítése

Bizonyítható, hogy egyetlen rejtett réteggel tetszőleges folytonos függvény megtanulható. Az aláírás hitelesítés során a megtanulandó függvény valószínűleg folytonos, vagy folytonos függvénnyel jól közelíthető. További előnye a neurális hálóknak, hogy az egyes bemeneti tulajdonságokhoz kapcsolódó összeköttetések tanulás közben kialakuló súlya alapján következtethetünk a tulajdonság fontosságára. [20]

2.2.5 Szupport vektor gépek (SVM)

Az SVM alapötlete, hogy ha származtatott tulajdonságokat hozunk létre az eredetiekből, úgy hogy a származtatott tulajdonságokból több legyen, és függjenek minden eredetitől, akkor az új tulajdonságtérben van rá esély, hogy a megtanulandó függvény lineárisan szeparálható lesz. A lineáris szeparátorhoz közel eső pontokat nevezzük szupport vektoroknak. A túlilleszkedés elkerülése érdekében, úgy választjuk meg a lineáris szeparátort, hogy a szupport vektoroktól minél nagyobb távolságra legyen (ezt az üres sávot tartaléknak hívjuk).

A származtatott tulajdonságok megválasztása nehéz feladat és sokdimenziós tulajdonságtér esetén a származtatott tulajdonságok tere nagyon sokdimenziós is lehet, ami megnehezíti a lineáris szeparátor kiszámítását. Másrészről, mivel új tulajdonsághalmazra térünk át, nem triviális annak megállapítása, hogy az eredetiben mely tulajdonságok voltak szignifikánsak. [20]

2.3 Osztályozási módszerek alkalmazása

Az alábbi szakaszokban néhány példát láthatunk a korábban tárgyalt osztályozási módszerek felhasználására az aláírás hitelesítésben.

2.3.1 Neurális háló alapú osztályozó

Az osztályozási feladat egy megközelítést olvashatjuk [21]-ben: Két lépcsős neurális háló alapú osztályozót. A globális jellemzők különböző csoportjait definiálják, és egymástól elkülönített MLP (több rétegű percepton) alapú osztályozókat rendelnek hozzájuk. Ezek az osztályozók önmagukban véve egyszerűeknek mondhatók, mivel csak egyetlen rejtett réteget tartalmaznak. A tanulás viszont ebben az esetben nem a hiba visszaterjesztésen alapul, hanem az ALOPEX algoritmuson, amely megakadályozza, hogy a háló „beragadjon" a megtanulandó függvény lokális minimumaiba vagy maximumaiba. A hálók viszonylag nagyméretű bemeneti adathalmaz kapnak, rendre 16, 96 és 48 adatot. Az első háló bemeneti adatai a korábban kinyert globális írásjellemzők. A második háló az aláírás egy egyszerűsített változatát kapja oly módon, hogy leképezik a képet egy 12×8 méretű négyzetrácsra, és minden cellában megmérik az intenzitást. A kimeneti réteg egyetlen neuront tartalmaz, amely egy 0 és 1 közötti értéket ad vissza, amely a hasonlóságot jelenti az adott aláírás és a tanító halmaz között. A végső választ végül egy RBF (Radial Basis Function) segítségével számítják ki.

2.3.2 SVM használata az aláírás hitelesítésben

Egy hasonló megközelítést láthatunk a [22] forrásban. Itt szintén globális írásjellemzőket használnak (szélesség és magasság aránya, középpont, sarok pontok, stb.) bemenetként valamint az előző szakaszban ismertetett négyzetrács értékeket. A kinyert jellemzők felhasználásával elvégezték az osztályozást egyszerű, MLP osztályozókkal, és SVM alapú osztályozókkal is. Az SVM alapú osztályozókat kipróbálták kernel, lineáris, polinomiális, és radiális bázis függvényekkel. Az eredmények szerint az SVM osztályozók legjobb eredménye 7-8%-os hibaarányt mutat átlagosan, míg ugyanez az érék az MLP osztályozók esetén 16-22% között mozog.

2.3.3 CGS vektorok és Bayes osztályozók

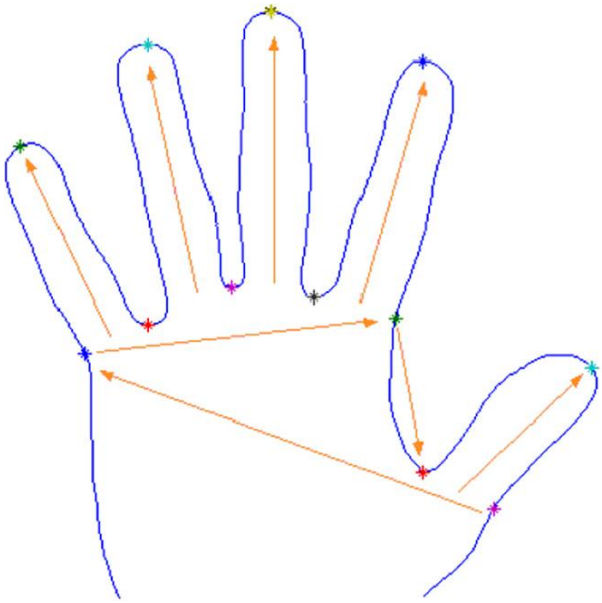
Egy másik érdekes megközelítést láthatunk a [23] forrásban. Ebben az esetben CGS vektorokat használtak fel (ezt eredetileg karakterfelismeréshez fejlesztették ki).

Minden képhez egy 1024 hosszúságú bináris vektort rendeltek hozzá, és később ezeket a vektorokat hasonlították össze. Az aláírás képét 4×8 szegmensre osztották fel, és az egyes szegmensekből származó információkat (domborúság, grádiens, szerkezeti jellemzők) a vektor egyes részein kódolták.

A vektorok összehasonlítására többféle módszert alkalmaztak. Ebben az esetben az SVM alapú osztályozók meglehetősen gyengén, 46% körüli átlagos hibaaránnyal teljesítettek, ezzel szemben a Naív Bayes osztályozók 20-25% közötti eredménnyel.

2.4 Az aláírás, mint biometrikus jellemző

A [24] forrásban egy formális megközelítését olvashatjuk a biometrikus azonosításnak, amelyet „boimetrikus disztperziós párosításnak” neveznek. Ha egy személy valamely biometrikus jellemzőjét többször lemérjük, általában nem ugyanazt a számot kapjuk. Több egymás utáni mérés eredményét tekintve azt láthatjuk, hogy a mérési eredmények valamilyen normál eloszlást követnek. Ha egy több személyből álló populáció esetén végezzük el ugyanezt a mérést, akkor általában szintén normál eloszlást kapunk. A 2.3-as ábrán a kézfej néhány normál eloszlást követő jellemzőjét láthatjuk, míg a 2.4-es ábrán olyan arcok szürkeárnyalatos képét, amelyből szintén kinyerhetőek normál eloszlású jellemzők, néhány alapvető képfeldolgozási lépés után.



2.3 ábra: A kézfej biometrikus jellemzői

1st images set of AR Database

2nd images set of AR Database

2.4 ábra: Arcfelismeréshez használt szürkeárnyalatos képek

Ha az egyes mért értékek különbségét vesszük, akkor olyan normál eloszlást kapunk, amelynek a várható értéke nulla. Ebben az esetben a szórás jelentősen nagyobb akkor, ha két véletlenszerű személy biometrikus jellemzőit hasonlítjuk össze, mintha ugyanannak a személynek a két különböző időpontban mért jellemzőivel tennénk ezt.

Ha az aláírások írásjellemzőit, mint biometrikus jellemzőket tekintjük, a fenti gondolatmenet helytálló, ám fontos megjegyeznünk, hogy ebben az esetben az egyetlen személyre jellemző értékek szignifikánsan nagyobb szórást követnek, mint más biometrikus jellemzők esetén, továbbá a fenti populációkra jellemző eloszlás a véletlenszerű hamisítványok eloszlását jelentené. A **3.3**-as szakaszban látni fogjuk, hogy az aláírások statisztikai alapú osztályozásakor a fenti tanulmánnyal ellentétben nem vettük figyelembe az egyes írásjellemzőknek a teljes populációra jellemző eloszlását, továbbá az egyes mért értékek különbségének eloszlása helyett maguknak a mért értékeknek az eloszlására koncentráltunk.

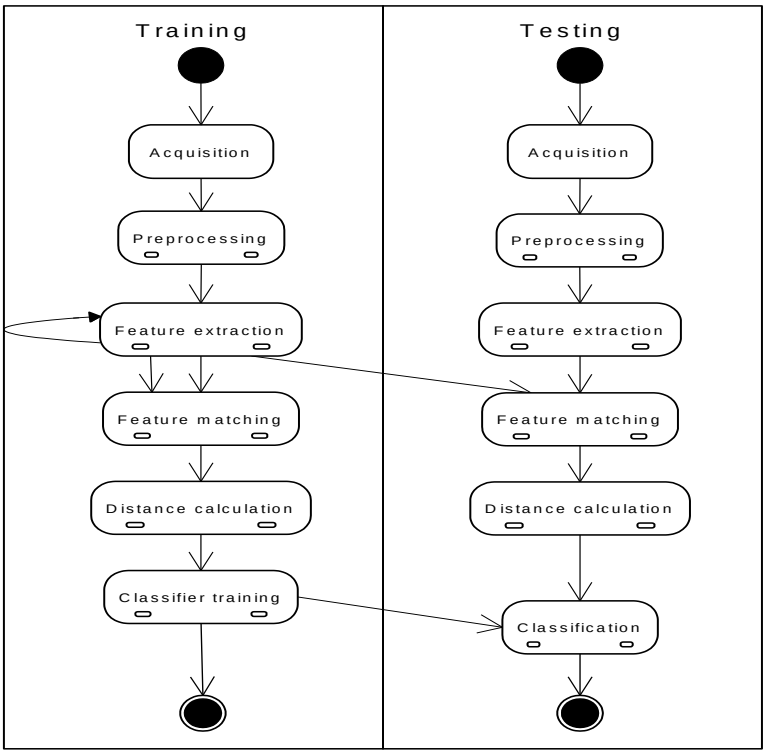
A [25] forrásban láthatunk példát a kézírás jellemzőinek statisztikai modellezésére. Ebben a tanulmányban az írásjellemzőket normál- és gammaeloszlással próbálták modellezni. A mi méréseink is azt mutatják, hogy nevezetes eloszlások közül ezzel a kettővel lehet a leginkább közelíteni a mért eloszlásokat.

3 Osztályozó rendszerek tervezése

Ebben a fejezetben először bemutatjuk az Aláírás Hitelesítő Rendszer felépítését, majd szemügyre veszünk két konkrét osztályozási módszert. A második szakaszban bemutatjuk az osztályozásra használandó neurális hálók illetve hálórendszerek megtervezése során felmerülő kérdéseket és a tervezés menetét, a harmadik szakaszban pedig a statisztikai alapú osztályozó tervezését részletezzük. A két megközelítés alapjaiban különbözik, mivel míg a neurális háló esetén függvénytanulási feladatként közelítjük meg az osztályozási problémát, addig a statisztikai osztályozó esetén valószínűségi modellt próbálunk konstruálni az osztályozáshoz.

3.1 Az AHR projekt felépítése

Az Aláírás Hitelesítő Rendszer (AHR) alapvetően két fő lépéssorozatból áll: Tanulás (Training) és Tesztelés (Testing). A tanulás során aláírásokat gyűjtünk az aláírótól, amelyek alapján az osztályozó rendszert be tudjuk tanítani, míg teszteléskor egy hasonló lépéssorozat segítségével egyetlen tesztelendő aláírás eredetiségét állapítjuk meg a korábban betanított osztályozóval. Az egyes lépések a 3.1-es ábrán láthatóak:



3.1 ábra: Az AHR projekt két fő lépéssorozata

A tanulási és tesztelési folyamat lépései:

- **Beolvasás (Acquisition):** A kép beolvasása
- **Elő feldolgozás (Preprocessing):** A kép javítása a hatékonyabb jellemzőkinyerés érdekében (keret levágása, vékonyítás/vastagítás, zajszűrés stb.)
- **Jellemző kinyerés (Feature extraction):** Különböző, számszerűen mérhető jellemzők kinyerése az aláírásokból (hurkok területe, ékezetek száma, aláírás szélessége stb.)
- **Jellemzők összerendelése (Feature matching):** Az egyes jellemzőkből bizonyos aláírások esetében több is lehet, ezért egyértelmű hozzárendelést kell adni két aláírás jellemzői között. (Pl: Az ékezeteket befoglaló téglalap mérete esetén, ha több ékezet is van, előbb meg kell állapítanunk, hogy az egyik aláíráson található egyes ékezetek a másikon melyeknek feleltethetőek meg.)
- **Távolság számítás (Distance calculation):** Távolságok kiszámítása a párosított jellemzőkre.
- **Osztályozó tanítása (Classifier training):** A kinyert távolságok segítségével betanítjuk az osztályozót az illető aláíró aláírásainak felismerésére.
- **Osztályozás (Classification):** A betanított osztályozó a tesztelés során kapott mintáról eldönti, hogy az eredeti vagy hamis.

További részletek találhatóak a [26] forrásban.

3.2 Neurális háló alapú osztályozók tervezése

3.2.1 Az osztályozás előtt rendelkezésre álló adatok

Fontos kiemelnünk, hogy a neurális hálókkal való osztályozás során nem az írásjellemzőkből származtatható abszolút számadatokkal dolgoztunk, hanem az egyes aláírások írásjellemzőinek egymástól vett távolságával. A következőkben megmutatjuk, hogy milyen transzformációkat szükséges a fenti távolságadatokon végeznünk, ahhoz, hogy a neurális hálók általi osztályozás elvégezhető legyen.

3.2.1.1 A feladat megfogalmazása függvényként

Első lépésként célszerű az aláírás hitelesítést függvény megtanulási folyamatra átfogalmaznunk, mivel a neurális hálókat alapvetően erre tudjuk felhasználni. A keresendő függvény értelmezési tartománya legyen annyi dimenziós, ahányféle különböző adat áll rendelkezésre az aláírásokra, az értékkészlete pedig legyen a (0; 1) intervallum. A függvény minden pontban vegye fel azt a valószínűséget, hogy az adott értékekkel rendelkező aláírás mennyire eredeti. Ideális esetben ez a függvény kizárólag 0 és 1 értékeket vehetne fel, azért van szükség mégis folytonos értékkészletre, mert így az osztályozó rendszernek lehetősége van arra, hogy bizonytalan eredményt szolgáltasson. (Például: 80% valószínűséggel eredeti.)

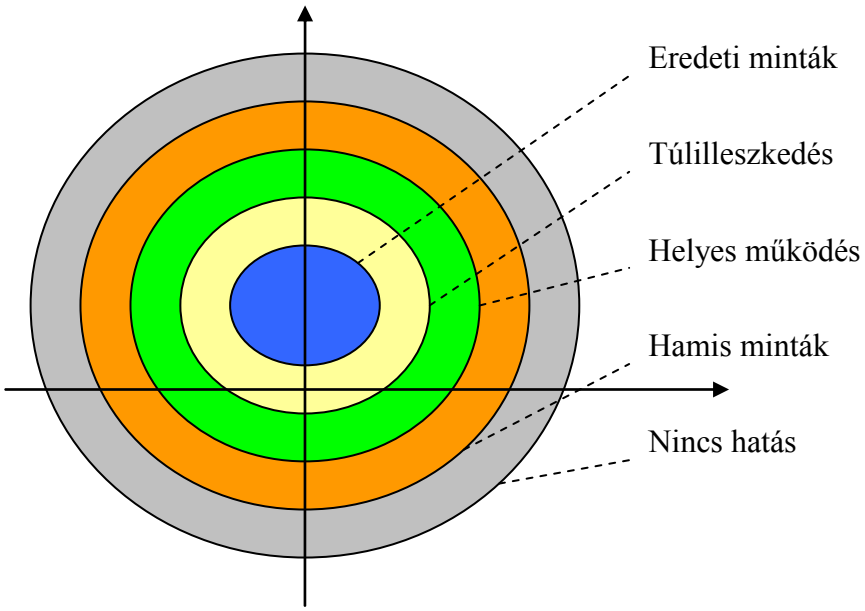
3.2.1.2 Normalizálás és a hiányzó adatok kitöltése

Minden egyes aláíró esetén 20 eredeti és 20 hamis aláírás mintánk van, és minden minta össze van hasonlítva minden eredeti aláírással. Ez tehát összesen 590 összehasonlítás egyetlen aláíró esetén. Fontos, hogy az osztályozó rendszer alapvetően arra van felkészülve, hogy egy bizonyos mintáról fog adatokat kapni, és nem minta párokról, ezért az adatokat át kell alakítani (Bővebben a 3.2.2-es szakaszban). További probléma, hogy vannak „hiányzó” elemek az adatok között, például azért, mert az egyik aláírásban 3 hurok szerepelt, a másikban csak 2, és ekkor a hiányzó hurokról nincsenek adatok. Az osztályozás megkezdéséhez a hiányzó elemeket, valahogyan fel kell tölteni, ám ez nem triviális feladat. Az egyes jellemzők igen sokfélék, ezért sokféle skálán értelmezendők, tehát konstans értékkel kitölteni nem lehet, mivel ez mindenhol mást jelent. Egy lehetséges megoldás, hogy először normalizáljuk az egyes jellemzőket úgy, hogy minden érték a [-1; 1] intervallumba essen, ezután a hiányzó elemek helyére vagy nullát írunk, vagy az adott jellemző maximális értékét. A normalizálás során némi torzítást okozhat az, hogy tanításkor csak az adatok egy része áll rendelkezésre, ezért ha teszteléskor is ugyanígy transzformáljuk az adatokat, akkor nem fog minden adat szükségképpen a [-1; 1] intervallumba esni.

3.2.1.3 Generált hamis minták

A projekt egyik legfontosabb peremfeltétele, hogy az osztályozó rendszer tanításához kizárólag eredeti aláírás minták állnak rendelkezésre, mivel éles helyzetben sincs lehetőség profi aláírás hamisítókat megkérni, hogy gyártsanak hamisítványokat. Ha azonban csak olyan tanító mintákat használunk, ahol a függvény 1 értéket vesz fel,

akkor a neurális háló könnyedén megtanulja a konstans 1 értéket, hiszen ez a függvény illeszkedik a teljes tanító halmazra. Nincs más lehetőség, mint mesterségesen létrehozni olyan tanító halmazokat, amelyekben a függvény 1-től különböző értéket vesz fel. A hamis mintákat legegyszerűbben úgy generálhatjuk, ha az eredeti mintákat vesszük alapul, és azokat eltoljuk valamilyen „kis" értékkel az értelmezési tartományban. Döntő fontosságú, hogy ezt a „kis" értéket megfelelően válasszuk meg. Ha túl nagyra választjuk, akkor a generált hamis minták az értelmezési tartományban a ténylegesen hamis aláírásoknak megfelelő érték „mögé" kerülnek, és a háló egy olyan függvényt tanul meg, amely a konstans 1-hez nagyon hasonlóan túl sok helyen 1 értékű. Ha viszont túl kicsire választjuk, akkor a generált hamis minták ugyan a valódi hamis, és az eredeti minták „közé" kerülnek, de az eredeti mintákhoz túlságosan közel. Ennek eredménye, hogy a háló azt tanulja meg, hogy egy aláírás pontosan akkor eredeti, ha azt tanuláskor is megkapta (túlilleszkedés). A cél tehát az, hogy a generált hamis minták az eredetiek és a hamisak közé kerüljenek, és minél közelebb legyenek a valódi hamisakhoz. A 3.2-es ábra a fent leírt eseteket szemlélteti, természetesen az értelmezési tartomány a valóságban sokkal több dimenziós.



3.2 ábra: Hamis minták generálásának esetei

3.2.2 Hálóstruktúra megválasztása

3.2.2.1 Párosításokból tanító és teszt halmaz

Az előző szakaszban említett párosításokból olyan adatokat kell létrehoznunk, ahol az egyes aláírás mintákról van információnk. Erre két megoldást alkalmaztunk. Az

első, hogy az egyes aláírások esetében az egyes jellemzők párosításaiból származó értékeket összeátlagoltam, így minden aláírás esetén a valódiaktól való átlagos távolsága áll rendelkezésre. Ez egy egyszerű megoldás, ám az információ jelentős része elveszik.

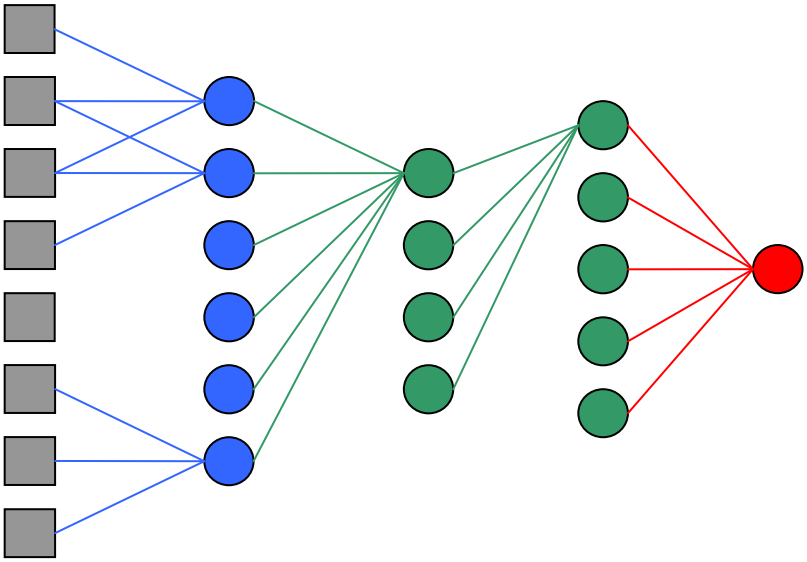
Egy másik lehetséges megközelítés, hogy nem egy neurális hálót használunk az osztályozásra, hanem egy hálórendszert, amelyben minden háló egy adott eredeti aláírástól való távolságra koncentrál. Annyi hálónk van tehát, ahány eredeti minta áll rendelkezésre tanításkor, és az adatokat szétosztjuk a hálók között úgy, hogy mindenki csak egy adott aláírástól való távolságot kap a tanítóhalmaz minden eleme esetén. Teszteléskor ugyanígy osztjuk szét a tesztadatokat a hálók között. Ebben az esetben nincs információ veszteség, de nehezebbé válik a rendszer monitorozása.

3.2.2.2 A hálók belső szerkezete

A neurális háló neuronokból és azok közötti kapcsolatokból áll. Ha az információ a hálóban csak egy irányba haladhat, vagyis ha a neurális hálót gráfnak tekintve körmentes gráfról beszélünk, akkor előrecsatolt hálóról van szó, egyébként visszacsatoltról. Mivel a visszacsatolt hálók adott inputra két időpillanatban különböző választ is adhatnak, a projekt során csak előrecsatolt hálókkal foglalkoztunk. Ebben az esetben a neuronokat rétegekre oszthatjuk úgy, hogy azok, amelyek közvetlenül rá vannak kapcsolódva a bemenetre, az első réteget alkotják, amelyek ezekkel állnak közvetlen kapcsolatban a második réteget, és így tovább.

A háló belső szerkezetét tehát úgy választhatjuk meg, ha specifikáljuk a rétegek számát, méreteit, valamint a köztük lévő kapcsolatokat. Alapvetően három rétegtípust használtunk a projekt során: Előfeldolgozó réteg, középső réteg és kimeneti réteg.

Az előfeldolgozó réteg kapcsolódik közvetlenül a bemenetekre. (Ennek hiányában az első középső réteg.) Ezen réteg neuronjai valamilyen szabály szerint csoportokba szervezik a bemeneti adatokat. A következő szabályt használtuk: Az előfeldolgozó réteg *n.* neuronja össze van kapcsolva a bemenetek közül az *n*-ediktől az *(n+k)*-adikkal, így tehát összesen *n-k+1* neuronból áll az így kialakított előfeldolgozó réteg. További példákat és alkalmazásokat olvashatunk [27]-ben. A tesztjeink alapján úgy tűnt, hogy ennek a rétegnek a használata javítja az osztályozás pontosságát.



3.3 ábra: Az egyes rétegek összekötése

A középső rétegek olyan neuron halmazokból állnak, amelyeknek minden eleme össze van kapcsolva a megelőző réteg (előfeldolgozó vagy egy másik középső) minden elemével. A tesztek alapján láttuk, hogy középső rétegek használatával stabilabb működés érhető el, a háló kimenete kevesebbszer fog ugrásszerűen változni és kevesebb tanítási iterációra lesz szükség. Természetesen minél több ilyen réteget használunk, annál hosszabb ideig fog tartani egy-egy tanítás. Az egyes középső rétegek méretének megválasztása nem triviális feladat, általában a többi paraméter függvényében van egy optimális érték.

Végezetül a kimeneti „réteg” egyetlen neuronból fog állni, amely összegzi és transzformálja az utolsó középső réteg értékeit. A 3.3-as ábrán láthatjuk az egyes rétegek összekötését: Szürkével a bemeneti puffereket, kékkel az előfeldolgozó réteget, zölddel a középső rétegeket és vörössel a kimeneti réteget jelöltük.

3.2.3 A hálórendszer tanítása

A korábbiakban láthattuk, hogy miként lehet az adatainkból tanuló és teszthalmazokat létrehozni (3.2.1), valamint milyen hálórendszer építhető (3.2.2). Ebben a szakaszban megnézzük, miképpen lehet e tanító és teszthalmaz segítségével a hálórendszert használni.

A tanulási folyamat nem más, mint az egyes hálók élsúlyainak megfelelő behangolása, így a megtanulandó függvényt az élsúlyok terében keressük. A tanulás hatékonyságához megfelelően kell megválasztani a hálók bátorsági faktorát (tanulási sebesség) és a tanulás iterációinak számát. Ha az iterációk számát túl nagyra választjuk,

akkor vagy az iterációk jelentős részében nem változnak a háló élsúlyai, vagy túltanulás léphet fel, amely esetében a hálók csak a tanítás során megkapott halmazra fognak megfelelően válaszolni. A tanításra a 2.2.4-ben leírt hiba visszaterjesztés módszerét használtuk, tehát valamilyen módon beállítjuk kezdetben a súlyokat, majd megnézzük, hogy mit válaszolnak a hálók a tanítóhalmaz egyes elemeire, végül frissítjük az élsúlyokat.

Ahhoz, hogy a 3.2.1.3-ban ismertetett túlilleszkedés is tesztelhető legyen, az osztályozó nem kaphatja meg az összes eredeti aláírás mintát tanuláskor. A mi esetünkben az eredeti minták felét a tanulás és a tesztelés során is felhasználtuk a másik felét csak a teszteléskor.

Több hálóból álló rendszer esetén a válasz kiolvasása nem triviális feladat. A hálórendszer eredő válasza szavazással alakul ki. Minden háló szavazhat egy 0 és 1 közötti számmal. Ha a szavazat egy bizonytalan érték (pl.: 0.4 és 0.6 közötti), akkor tartózkodásnak tekintjük. A végső válasz úgy alakul ki, hogy átlagoljuk a nem tartózkodó hálók szavazatait (Ha mindenki tartózkodott, akkor pedig: 0.5).

3.3 Statisztikai osztályozók tervezése

Míg a neurális háló alapú osztályozó esetén az osztályozási feladatot egy függvény megtanulásként közelítettük meg, a statisztikai osztályozó tervezésekor az osztályozást egy bizonyos esemény valószínűségének kiszámításaként értelmezzük. A korábbiaknál is fontosabbnak tartjuk, hogy jobban belelássunk az osztályozó belső szerkezetébe, pontosan követhető legyen, hogy az osztályozási eredményt mi okozza. Azt reméljük, hogy ezáltal az osztályzási folyamatot könnyebben tudjuk később továbbfejleszteni, és az osztályozás előtti lépésekről is több visszacsatolást kapunk. A későbbiekben látni fogjuk, hogy a statisztikai osztályozó alapja, hogy az aláírásokból származó számadataink eloszlását minél pontosabban ismerjük, ezért ebben az esetben nem távolság adatokkal dolgoztunk, hanem az írásjellemzők abszolút numerikus adataival.

3.3.1 Modellezés és alapfeltevések

Az osztályozó tervezéséhez először az osztályozási problémát valószínűségi változókkal modellezzük, miközben néhány a priori feltételezéssel is élünk:

- Egy adott aláíró esetén legyen n darab mérhető írásjellemzőnk, amelyek eloszlását a $x_1, x_2, \ldots x_n$ páronként független valószínűségi változók adják

meg eredeti aláírások esetén, és $y_1, y_2, \ldots y_n$ páronként független valószínűségi változók adják meg hamis aláírások esetén.

- Az írásjellemzők normáleloszlást követnek, mégpedig úgy, hogy az eredeti és a hamis aláírások esetén is ugyanaz a várható értékük, ám a hamis aláírások írásjellemzőinek a szórása nagyobb. Formálisan:
  - $x_i \sim N(\mu_i, \sigma_i)$, minden *i* esetén
  - $y_i \sim N(\mu_i, \sigma_i w_i)$, minden *i* esetén, ahol $w_i > 1$
- Jelentse *O* azt az eseményt, hogy a tesztelendő aláírás eredeti, míg *F* azt az eseményt, hogy a tesztelendő aláírás hamis. Kezdetben semmit sem tudunk a tesztelendő aláírásról, ezért $P(O) = P(F) = 0.5$.
- Jelöljék a tesztelendő aláírás írásjellemzőinek eloszlását $v_1, v_2 \ldots v_n$ valószínűségi változók. Ezek nyilván eloszlásban azonosak a megfelelő $x_1, x_2, \ldots x_n$ vagy $y_1, y_2, \ldots y_n$ valószínűségi változókkal attól függően, hogy eredeti vagy hamis aláírást tesztelünk.

3.3.2 Osztályozás egyetlen írásjellemzővel

Ebben a szakaszban megmutatjuk, hogyan lehet meghatározni egy aláírás eredetiségének valószínűségét egyetlen írásjellemző mért értéke alapján. A következő szakaszban látni fogjuk, hogy miként lehet ezt a valószínűséget felhasználni több írásjellemző mért értéke alapján adódó együttes valószínűség meghatározására.

Legyen tehát adott a *k*-adik írásjellemző m_k mért értéke és keressük annak valószínűségét, hogy ilyen m_k mért érték mellett az aláírás eredeti, a korábbi jelöléseinket használva tehát a $P(O \mid v_k = m_k)$ feltételes valószínűséget keressük.

$$P(O \mid v_k = m_k) = \frac{P(O \wedge v_k = m_k)}{P(v_k = m_k)} = \frac{P(O \wedge v_k = m_k)}{P(O \wedge v_k = m_k) + P(F \wedge v_k = m_k)}$$

A korábbi jelöléseink miatt, *O* esemény bekövetkezésekor v_k eloszlása, azonos x_k eloszlásával, továbbá *F* esemény bekövetkezésekor v_k eloszlása azonos y_k eloszlásával így:

$$\frac{P(O \wedge v_k = m_k)}{P(O \wedge v_k = m_k) + P(F \wedge v_k = m_k)} = \frac{P(O \wedge x_k = m_k)}{P(O \wedge x_k = m_k) + P(F \wedge y_k = m_k)}$$

Vegyük észre, hogy a $P(O \wedge x_k = m_k)$ és a $P(F \wedge y_k = m_k)$ valószínűségek a sűrűségfüggvény tulajdonságai és a korábbi feltevéseink szerint a következőképpen is felírhatók:

$$P(O \wedge x_k = m_k) = P(x_k = m_k \mid O)P(O) \approx \frac{1}{2} f_{x_k}(m_k)$$

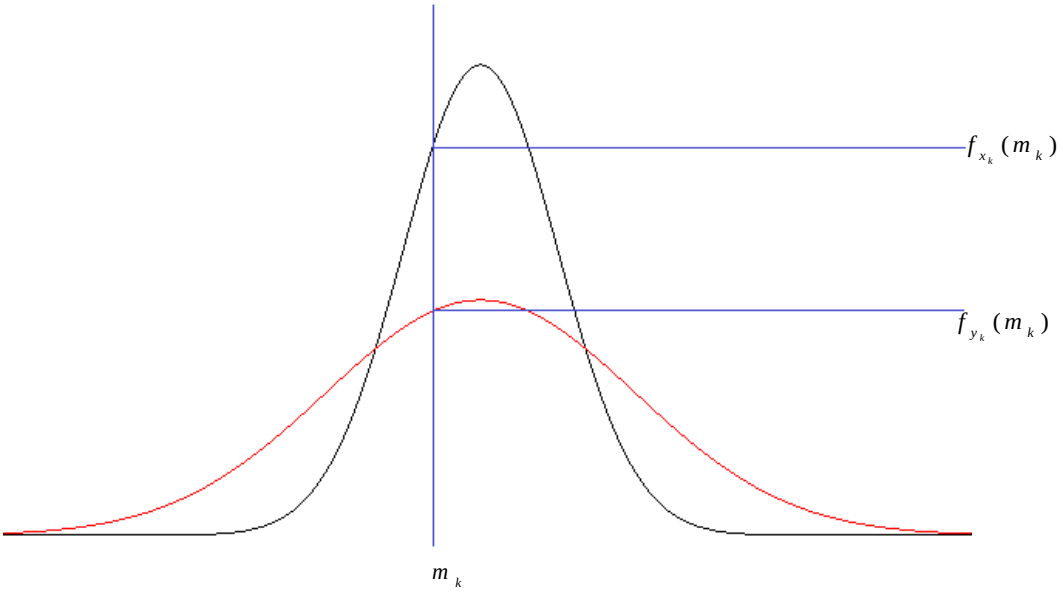
Ahol f_{x_k} az $N(\mu_k, \sigma_k)$ eloszlású x_k valószínűségi változó sűrűségfüggvénye. Hasonlóképpen:

$$P(F \wedge y_k = m_k) = P(y_k = m_k \mid F)P(F) \approx \frac{1}{2} f_{y_k}(m_k)$$

Így végül:

$$P(O \mid v_k = m_k) = \frac{\frac{1}{2} f_{x_k}(m_k)}{\frac{1}{2} f_{x_k}(m_k) + \frac{1}{2} f_{y_k}(m_k)} = \frac{f_{x_k}(m_k)}{f_{x_k}(m_k) + f_{y_k}(m_k)}$$

A következő ábra szemlélteti az aláírás eredetiségének a fenti képlet alapján való meghatározását. A fekete Gauss-görbe jelképezi az írásjellemző eredeti, a vörös pedig a hamis aláírások esetén érvényes eloszlását.



3.4 ábra: Osztályozás egyetlen írásjellemző segítségével

3.3.3 Osztályozás több írásjellemző felhasználásával

Az előző fejezetben láttuk, hogyan határozható meg egyetlen írásjellemző mért értéke alapján a tesztelendő aláírás eredetisége. Tegyük fel, hogy a fenti mérést és számítást már elvégeztük *n* darab független eloszlású írásjellemzőre és az így a p_1, p_2 ... p_n valószínűségeket kaptuk. Definiáljuk a következő eseményeket:

- *A* esemény: Minden méréskor egy eredeti aláírást teszteltünk
- *B* esemény: Minden méréskor egy hamis aláírást teszteltünk

- U esemény: Minden méréskor eredeti vagy minden méréskor hamis aláírást teszteltünk

A definíciókból az alábbi állítások következnek.

- $P(A) = p_1 p_2 \dots p_n$
- $P(B) = (1 - p_1)(1 - p_2)\dots(1 - p_n)$
- $P(U) = P(A) + P(B)$
- $P(A \wedge U) = P(A)$
- $P(B \wedge U) = P(B)$

Ezek után keressük annak valószínűségét, hogy minden méréskor eredeti aláírást teszteltünk, azzal a feltevéssel, hogy minden méréskor ugyanazt az aláírást teszteltük, ami tehát az alábbi feltételes valószínűségként írható fel:

$$P(A \mid U) = \frac{P(A \wedge U)}{P(U)} = \frac{P(A)}{P(A) + P(B)} = \frac{p_1 p_2 \dots p_n}{p_1 p_2 \dots p_n + (1 - p_1)(1 - p_2)\dots(1 - p_n)}$$

Ez tehát azt jelenti, hogy az eredő valószínűség egészen másképp alakul, mintha az elemi valószínűségek intuitívan adódó összeátlagolásából számolnák, például ha $p_1=0.6$ és $p_2=0.7$, akkor az átlagolás 0.65-ös valószínűséget adna a fenti összegzés viszont 0.78-at. További érdekesség, hogy ugyanezt a számítási módszer használják annak kiszámítására is, hogy egy beérkező n szóból álló e-mail üzenet mekkora valószínűséggel kéretlen reklámlevél (spam). Ebben az esetben az elemi valószínűségek azt jelentik, hogy az egyes szavak előfordulásának okán mekkora valószínűséggel tekinthető az illető levél spamnek.

3.3.4 Eloszlások azonosítása, függetlenségvizsgálat

A korábban tárgyalt modellnek két fontos feltétele a gyakorlatban általában nem teljesül:

- A mérhető írásjellemzők, általában nem függetlenek
- Néhány írásjellemző nem követ normáleloszlást

Az első probléma áthidalására a faktoranalízis módszere alkalmazható, így a modellben nem a korábbi írásjellemzőket, hanem a belőlük kinyerhető független faktorokat használjuk. A faktoranalízis előnye, hogy az írásjellemzők faktorokra bontásával további információkat kaphatunk az aláíróról, ami a későbbiekben egy pontosabb modell létrehozását is lehetővé teheti. A módszer hátránya, hogy az így kinyert faktorok esetén lehet, hogy még annyira sem garantálható a normáleloszlás,

mint az eredeti írásjellemzők estén, amely szintén előtérbe hozza az itt tárgyalt második problémát.

A korábbi modellünkben valójában a végső számításokat kivéve sehol sem használtuk ki az írásjellemzők normáleloszlását csupán azt, hogy az eloszlásuk ismert. Ezért lehetőségünk van először meghatározni az írásjellemzők eloszlását, és majd a korábbi modell szerint az új eloszlással számolni. A feladat egy lehetséges megközelítése, hogy Kolmogorov-Szmirnov teszt segítségével az írásjellemzőkhöz tartozó adatsorokat minél több megfelelően paraméterezett nevezetes eloszláshoz hasonlítjuk, majd ezek közül azt választjuk, amelyiktől a teszt alapján a legkevésbé tér el. Fontos azonban, hogy a fenti teszt csak nagyszámú mintahalmaz esetén garantál pontos eredményt, ezért az ilyen jellegű eloszlás felismerés akár az eredmények torzulásához is vezethet. Ezen kívül megjegyezzük, hogy a méréseink szerint, még a fenti két probléma figyelmen kívül hagyásával is ígéretes osztályozási eredmények érhetőek el.

4 Osztályozók implementálása

Ebben a fejezetben először áttekintjük az AHR (Aláírás Hitelesítési Rendszer) interfészeit, majd szemügyre vesszünk néhány előfeldolgozási feladathoz kapcsolódó implementációt. Végezetül előbb egy neurális háló, majd egy statisztikai alapú osztályozó rendszer és a kapcsolódó tesztelő környezet implementációját vizsgáljuk meg.

4.1 AHR interfészek

Jelen alfejezetben az AHR keretrendszer által specifikált interfészeket mutatjuk be. A 3.1-es fejezetben leírt lépéseket moduloknak feleltetjük meg, amelyekre jellemző a lazán csatoltság, vagyis az egyes modulok csakis saját bemeneti és kimeneti adataikat ismerik. A modulok összekapcsolása egy komplett rendszerré egy külön futtató rendszer feladata. Az interfészek részletes osztálydiagramja az A függelékben látható. Tekintsük most át ezeket az interfészeket:

- `IModule` – Minden modulnak ebből kell származnia, a modulok egyedi név alapján történő megkeresését teszi lehetővé.
- `ITransformation` – Az elő feldolgozási lépésben történő, általában nem visszavonható átalakításokat reprezentálja.
- `IFeatureExtracion` – A jellemzők kinyerésére szolgáló modulokat reprezentálja. A modul kinyert jellemzőknek megfelelően kitölti a képhez tartozó meta adatokat.
- `IModelFeatureExtraction` – Bizonyos jellemzők nem az egyes aláírásokra, hanem az aláíróra vonatkoznak. Ezek a modulok az ilyen metaadatok kitöltésére hivatottak.
- `SignatureMatch` és `ModelMatch` – A kinyert adatokat reprezentáló osztályok.
- `IFeatureMatching` – A jellemzők összerendelését végző modulokat reprezentálja.
- `IModelFeatureMatching` – Az aláírás és az aláíró jellemzőit összerendelő modulokat reprezentáló interfész.

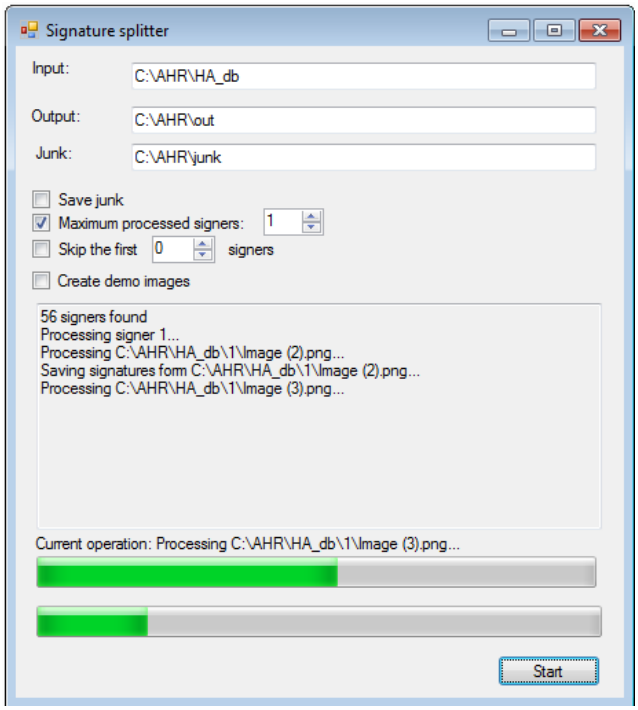
- `IFeatureaComparison` – Az összehasonlítások alapján távolságértékeket számoló modulok interfésze.
- `IModelFeatureComparison` – Az aláíró és az egyes aláírások távolságait számító modulokat reprezentálja.
- `IClassification` – Az osztályozó rendszerek interfésze.
- `ILogger, IProgress, ICache` – További interfészek, amelyek a kényelmesebb kommunikációt teszik lehetővé a futtatórendszerrel.

További részletek találhatóak a [26] forrásban.

4.2 Képfeldolgozási feladatok

Ebben az alfejezetben két olyan szoftvert fogunk ismertetni, amelyeknek az aláírás hitelesítés előfeldolgozási szakaszában van szerepe. E programok létrehozására akkor merült fel az igény, amikor úgy döntöttünk, hogy saját aláírás adatbázissal kívánjuk bővíteni a teszt adatbázisaink sorát, és ehhez egy aláírásgyűjtést kezdeményeztünk. Olyan adatbázist szándékoztunk létrehozni, amelyben minden egyes aláírótól nagy mennyiségű aláírás minta áll rendelkezésre. Ehhez minden aláírótól fejenként öt a B függelékben található aláírás ív kitöltését kértük. A beolvasás után először el kellett különítenünk az egyes íveken található tíz aláírást egymástól, tehát szükség volt egy „**kivágó**” alkalmazásra. Ezután a kivágott aláírásokról még el kellett távolítanunk a vonalat, amihez szükségünk volt egy „**vonal eltávolító**” alkalmazásra.

4.2.1 A képfeldolgozó alkalmazások felhasználói interfésze

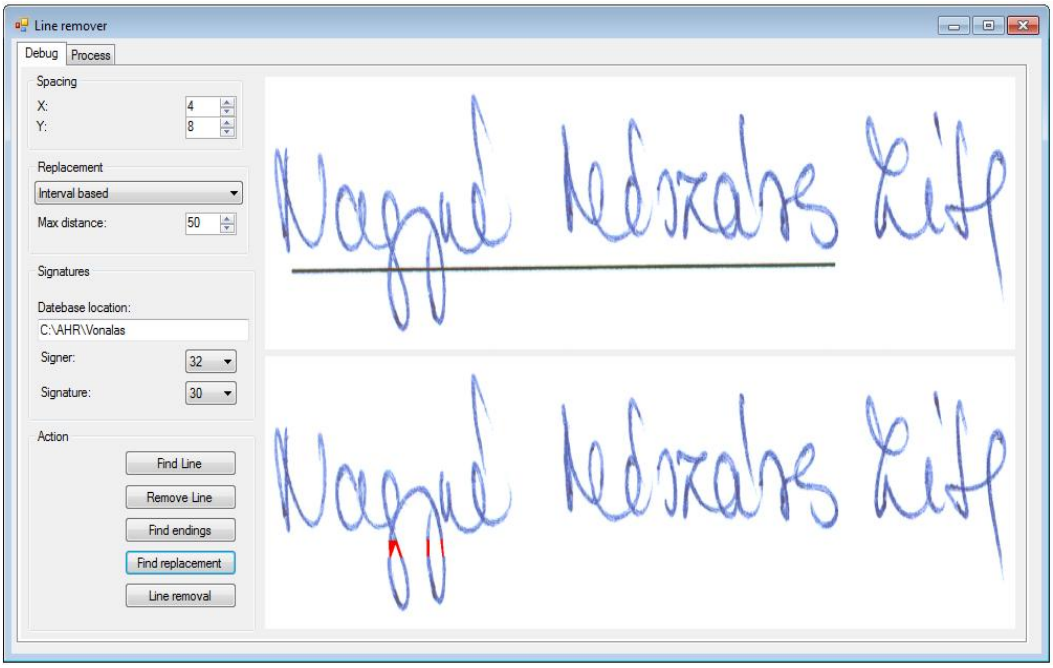


4.1 ábra: Az aláírás kivágó alkalmazás felhasználói felülete

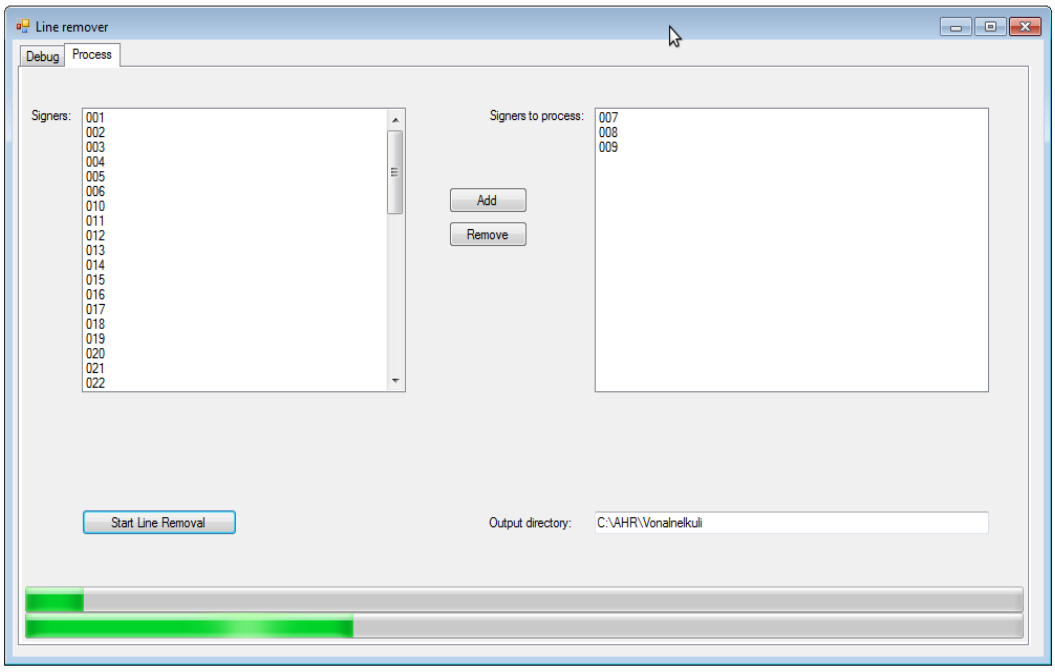
A 4.1-es ábrán láthatjuk az aláírás kivágó alkalmazás grafikus felületét. Az első három szövegdobozban megadhatjuk a beolvasott aláírás íveket tartalmazó könyvtárat (**input**), a létrehozandó szétvágott aláírások kimeneti könyvtárát (**output**), és a vágás során keletkező „hulladék” kimeneti könyvtárát (**junk**). Ezek a hulladékok olyan, az ábrán talált kisebb alakzatok, amelyek a program szerint nagy valószínűséggel nem képezik semelyik aláírás részét sem, azonban minden aláíró első feldolgozásakor érdemes átvizsgálni ezeket a képeket, mivel előfordulhat, hogy belekerül egy-egy ékezet.

A következő három jelölőnégyzet segítségével a program működését szabhatjuk testre: Beállíthatjuk, hogy legfeljebb hány aláíró aláírásait kívánjuk feldolgozni és melyek azok az aláírások, amelyeket mellőzni szeretnénk a feldolgozás során. Ugyancsak itt adhatjuk meg azt is, hogy akarjuk-e menteni a „hulladékokat” és hogy akarunk-e olyan „demo” képet létrehozni a folyamat közben, amelyen megfigyelhető a vágási folyamat egy köztes állapota.

A „Start” nyomógombra kattintva megkezdődik a megadott aláírások feldolgozása, és az eredmény a fent specifikált könyvtárakba kerül. A folyamat közben részletes, szöveges tájékoztatást kapunk arról, hogy mi történik, valamint két állapotsáv is jelzi, hogy éppen hol tart a feldolgozás.



4.2 ábra: A vonal eltávolító alkalmazás tesztelési felülete



4.3 ábra: A vonal eltávolító alkalmazás felhasználói felülete

A 4.2-es és a 4.3-as ábrán a vonal eltávolító alkalmazás felhasználói felületét láthatjuk. A „**Debug**” fülön megfigyelhetjük, hogy az egyes paraméterek beállítása milyen hatással van a vonal eltávolítási folyamatra. Ezen az oldalon beállíthatjuk a vonal pixelei körül eltávolítandó hely nagyságát (**spacing**), kiválaszthatjuk az aláírások vonal eltávolítás utáni kipótolására használandó algoritmust (**replacement**) és a pótolandó intervallumok keresésének maximális távolságát (**max distance**).

Ezután betölthetünk egy aláírást az adatbázisból és megtekintjük rajta a vonal eltávolítási művelet egyes lépéseinek eredményét.

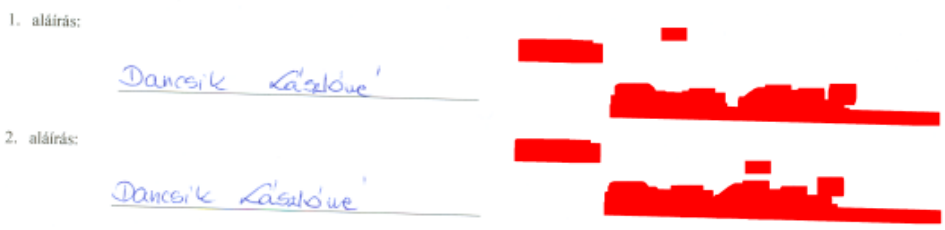
A „**Find line**” nyomógomb hatására vörössel kijelöli az alkalmazás az eltávolítandó vonalat, míg a „**Remove line**” megnyomására az alkalmazás kiradírozza a képről a megtalált vonalat. A „**Find endings**” funkció a vonal eltávolítása után megjelöli a „szabaddá vált”, valószínűleg összetartozó végződéseket, amelyeket a „**Find replacement**” művelet megpróbál összekötni. Végül a „**Line removal**” nyomógomb hatására az aláírásba helyezett „pótlékok” felveszik az aláírás színét, így elkészül a vonal nélküli aláírás.

A „**Process**” fülön a korábbi beállításoknak megfelelően végrehajthatjuk a vonal eltávolítási műveletet az egyes aláírók aláírásain. A bal oldali listában láthatjuk az adatbázisban tárolt aláírókat, míg a jobb oldaliban azokat, amelyeken a vonal eltávolítási műveletet el akarjuk végezni. Az „**Add**” és „**Remove**” nyomógombok segítésével mozgathatjuk a kijelölt elemeket a listák között, míg végül a **„Start Line Removal**” nyomógomb segítésével elkezdhetjük a vonal eltávolításokat. A folyamat állapotát az állapotsávokon tudjuk az alkalmazás futása közben nyomon követni.

4.2.2 Képkivágási algoritmusok

Ebben a szakaszban áttekintjük a képkivágó alkalmazás főbb algoritmusait. A kép kivágás négy főbb lépésből áll: Először (1) létrehozunk egy bitmaszkot, amely tartalmazza azokat a pixeleket, amelyek feltehetően részei a képen található valamely alakzatnak, majd a bitmaszk felhasználásával (2) elkülönítjük egymástól az alakzatokat, ezután (3) egyesítjük azokat az alakzatokat, amelyek nagyon közel vannak egymáshoz, (4) végül kiválasztjuk az alakzatok közül azokat, amelyek feltehetően valamely aláírást tartalmazzák. Nézzük most ezeket a lépéseket részletesebben:

A bitmaszk létrehozásához egyszerűen vesszük az eredeti képet, majd minden olyan pontja esetén, amely számunkra „érdekes” elhelyezünk egy bizonyos sugarú kört a maszkon, az adott pontnak megfelelő helyen. Egy pont általában akkor érdekes számunkra, ha a színében megfelelő mértékben találhatóak meg az aláírásokra jellemző színkomponensek. Szemlétesen a bitmaszk olyan, mintha az aláírás íven egy picit „szétkennénk” az aláírásokat, annak érdekében, hogy az egyes aláírások egy zárt alakzatot alkossanak (lásd a 4.4-es ábrát).



4.4 ábra: Bitmaszk létrehozása a beolvasott képből

Ezután következik az alakzatok egymástól való elkülönítése. Ehhez vízszintes irányú pásztákat indítunk a bitmaszkon lentről felfelé, egy olyan sort keresve, ahol van megjelölt pont. Ezután ettől a sortól indulva keresünk egy üres sort, így megtaláltuk az alakzat alsó és felső határoló vonalát. Most függőleges irányú pásztákat indítunk a bitmaszkon egy olyan oszlopot keresve, ahol a korábban talált két sor között van megjelölt pixel, majd innen továbbindulva egy üres oszlopot. E módszerrel megtalálhatjuk a képen található egyik alakzat határoló téglalapját (lásd a 4.5-ös ábrát). A téglalapon belüli pixeleket távolítsuk el a bitmaszkról és ismételjük az eljárást addig, amíg találunk újabb alakzatokat.

4.5 ábra: Alakzatok szeparálása a bitmaszk alapján

Miután megtaláltuk a képen fellelhető alakzatokat célszerű egyesíteni azokat, amelyek egymáshoz közel vannak. Ez főleg azért praktikus, mert gyakran előfordul, hogy egy-egy ékezet az aláírástól elkülönülten, önálló alakzatba kerül ha az aláíró túlságosan messze írja az ékezetet az aláírásától.

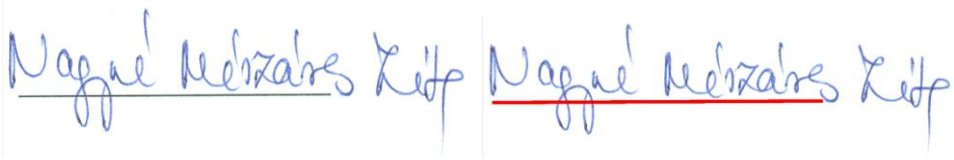
Az alakzatok egyesítése után már csak ki kell válogatni közülük azokat, amelyek az íven található aláírásokat tartalmazzák. Erre a legegyszerűbb módszer a 10 legnagyobb területű alakzat kiválasztása. Érdemes ezután az összes alakzatot elmenteni

és később átnézni a „hulladékot” is, nehogy véletlenül kidobjuk az egyik aláírás egy darabját.

4.2.3 A vonal eltávolítás algoritmusai

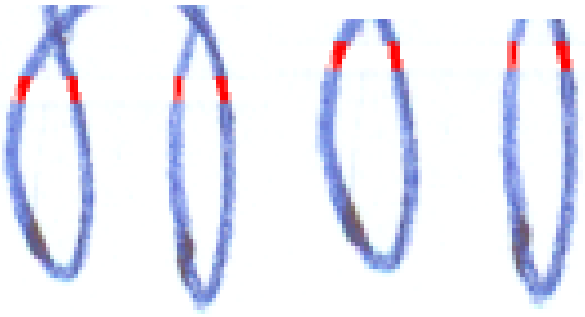
Ebben a szakaszban áttekintjük a vonal eltávolítási folyamat főbb algoritmusait. A vonal eltávolítás 3 főbb szakaszból áll: Először (1) megkeressük a vonalhoz tartozó pixeleket, a felső és alsó határait, valamint a végpontjait és kifehérítjük a vonalat. Ezután a felső és alsó határoló pontok alapján (2) megkeressük és párosítjuk a szabadon maradt végződéseket. Végül (3) összekötjük a korábban talált végződéseket az aláírás színének felhasználásával. Nézzük most ezeket a szakaszokat részletesebben:

A vonal végpontjainak megtalálásához egyszerűen végigpásztázzuk a képet balról-jobbra, fentről-lefelé haladva, és a pásztázás során először megtalált „vonalszínű” pixel lesz vonal kezdőpontja, az utolsó ilyen pixel pedig a végpontja. Ezután a két pont közötti pixeleket bejárjuk, és a köztük lévő a összes pixelt, valamint minden ilyen pixeltől fel- és lefelé a vastagsági paraméternek megfelelő számú további pixeleket felvesszük a vonal pontjai közé. A vonal pontjai között minden oszlopban a legfelső pontok összessége alkotja a felső határoló pontokat, míg a legalsó pontok összessége az alsó határoló pontokat. Egy ilyen vonalat láthatunk a 4.6-os ábrán.



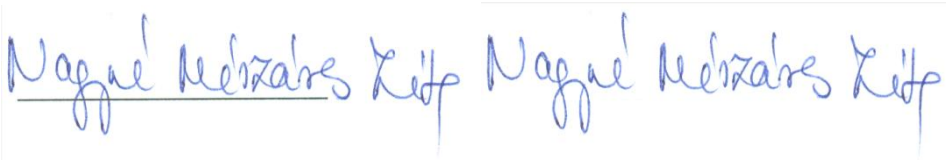
4.6 ábra: Vonal azonosítása

A szabadon maradt végződések megtalálása és párosítása kétféleképpen történhet: A végződéseket vagy pontokként, vagy intervallumokként definiálhatjuk, és ennek megfelelően az összekötések vonalak vagy négyszögek lehetnek. A végződéseket minden esetben a felső és alsó határoló pontokon talált „aláírás színű” pixelekből hozzuk létre, és a párosítás során a legközelebbi elemet keressük. A 4.7-es ábrán láthatunk egy-egy példát a kép összekötő algoritmus eredményére.



4.7 ábra: Pixel (balra) és intervallum (jobbra) alapú pótlékok

A megtalált és párosított végződések közötti rész már csak ki kell töltenünk az aláírás színével, amihez praktikusan az aláírás pixeleinek átlagszínét használhatjuk. A 4.8-as ábra az elkészült vonaltalanított aláírást mutatja be.

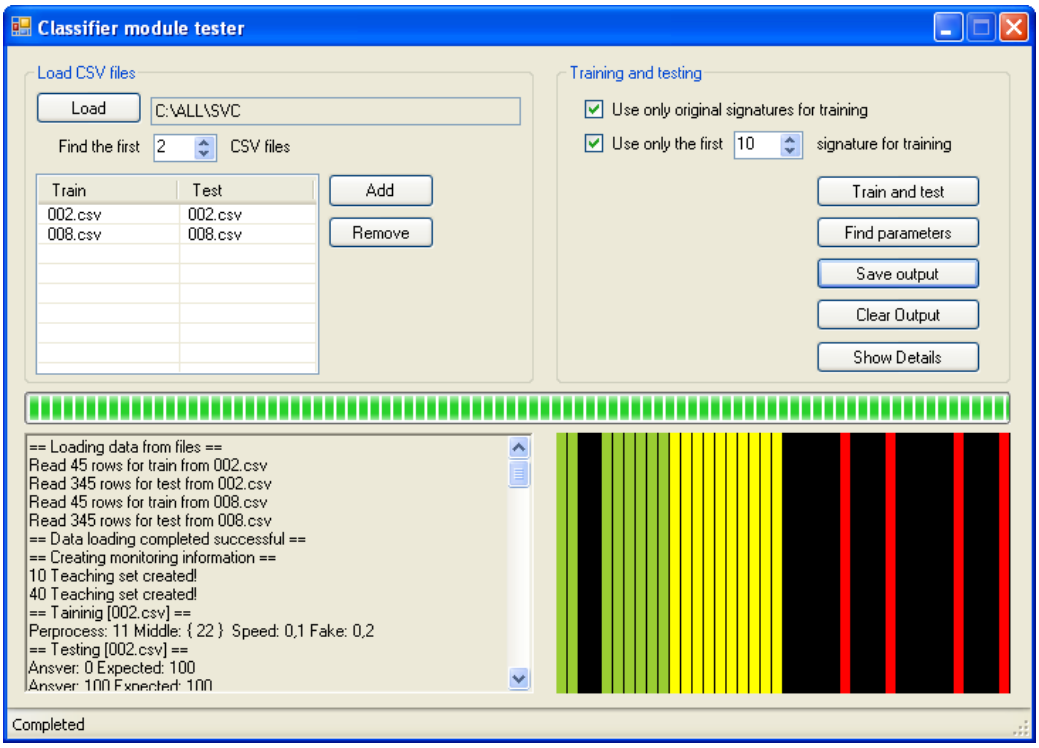


4.8 ábra: A vonal eltávolítás előtti és utáni állapot

4.3 Neurális háló tesztelő és monitorozó

A neurális háló tesztelő és monitorozó programnak az a célja, hogy neurális háló alapú osztályozó rendszereket tudjunk önmagukban tesztelni, és a működést menet közben megfigyelni. A program alapvetően úgy készült, hogy bármilyen osztályozó modult képes legyen tesztelni, de ha a monitorozó és paraméterkereső képességét ki akarjuk használni, akkor a megfelelő üzenetek formájában kommunikáló neurális hálót használó osztályozót érdemes vele tesztelni.

4.3.1 A program grafikus felülete



4.9 ábra: A tesztelő modul fő ablaka

A 4.9-es ábrán láthatjuk a program fő ablakát. Az ablak bal felső részében tudjuk elvégezni az adatok betöltését. A tesztelési fázisban az adatok CSV fájlokban vannak melyek szerkezete a következő: A signer1, index1 és isOriginal1 oszlopok azonosítják az összehasonlításban résztvevő egyik aláírást, a signer2, index2 és isOriginal2 oszlopok pedig értelemszerűen a másikat, a többi oszlop a megfelelő attribútumok szerinti távolságot tartalmazza. A **Load** gombra kattintva kiválaszthatjuk a CSV fájlok helyét, ezután a megadott könyvtárban talált megadott számú fájlt betölti. Kiválaszthatjuk, hogy melyik fájl tartalmazza a tanulási és melyik a tesztelési adatokat. Alapértelmezésben minden CSV fájl tartalmazza mindkettőt.

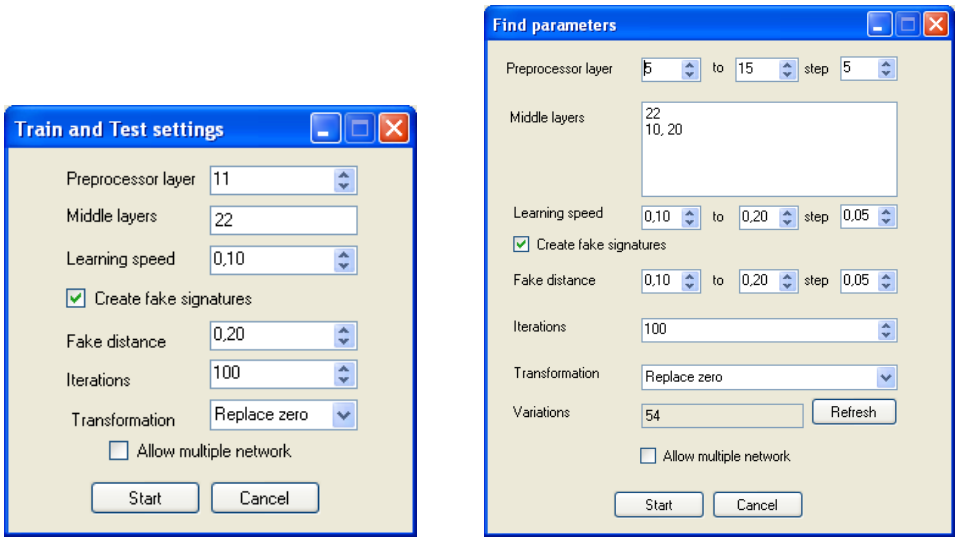
Az ablak jobb felső részében található beállításokkal tovább szűrhetjük a tanulásra szánt aláírások halmazát, valamint az itt található nyomógombokkal a program fő funkcióit érhetjük el:

- **Train and Test** – A gomb megnyomásával a tanulási paraméterek beállítása után tesztelhetjük a háló teljesítményét az összes betöltött adaton.
- **Find parameters –** Ennek a funkciónak a célja az ideális paraméterezés megtalálása. A megjelenő párbeszédpanelen paraméter intervallumokat

állíthatunk be, és minden variációt megvizsgálhatunk az összes betöltött adaton.

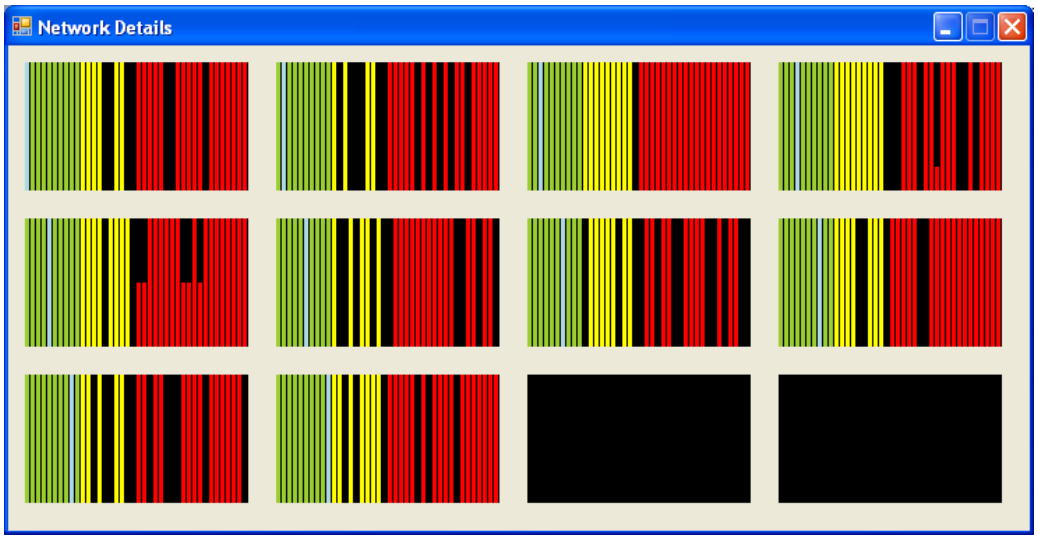
- **Save output –** A tesztelő program napló üzeneteit menthetjük szöveges fájlba.
- **Clear output –** Naplóbejegyzések törlése.
- **Show Details –** Többhálós modell esetén az egyes hálók állapotainak megjelenítése.

Az ablak bal alsó részén a futás közben keletkező naplóbejegyzéseket tekinthetjük meg, míg a jobb alsó részben a hálórendszer teljesítményét mutató grafikonokat láthatjuk. A zöld oszlopok jelzik a háló válaszát azokra a tesztesetekre, amelyeket eredetiek és tanításkor is rendelkezésre álltak, a sárga oszlopok jelentik azokat, amelyek ugyan eredetiek, de a tanítás során nem voltak jelen, míg a vörös oszlopok a hamisítványokra adott válaszokat mutatják. A cél természetesen az, hogy a zöld és sárga oszlopok magassága maximális a vörösöké minimális legyen.



4.10 ábra: Paraméterek hangolása

A 4.10-es ábrán láthatjuk, a paraméterek beállításra szolgáló ablakokat. Rendre beállíthatjuk az előfeldolgozó réteg méretét, a középső rétegeket, a bátorsági faktort, a generálandó hamis minták távolságát az eredetiektől, az iterációk számát, a hiányzó értékek pótlásának mikéntjét, valamint hogy egy vagy többhálós modellt kívánunk használni. Paraméterkeresés esetén a legtöbb számérték esetén a legkisebb és legnagyobb értéket és a lépésközt állíthatjuk be, továbbá a **Variations** mezőben az így létrejövő paramétervariációk számát láthatjuk.

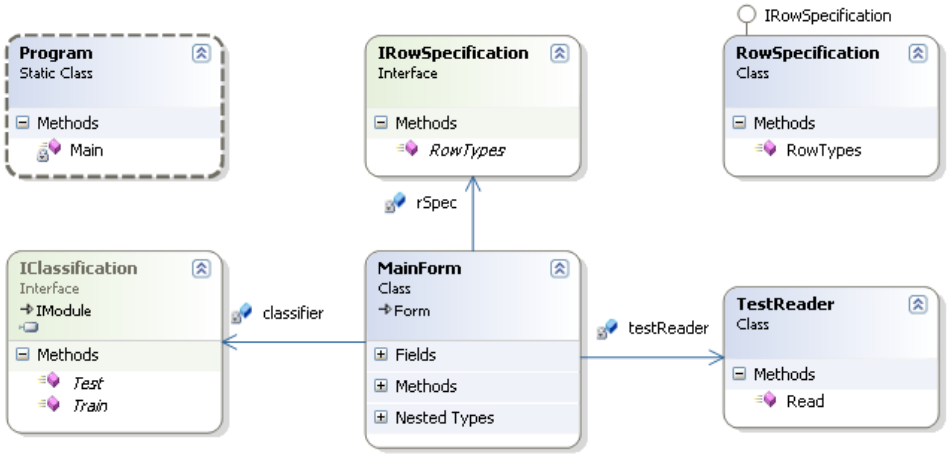


4.11 ábra: Részhálók monitorozása

A **Show Details** nyomógomb megnyomására jön elő a 4.11-as árán látható ablak, melyben nyomon követhetjük az egyes hálók teljesítményét. A zöld, sárga és vörös oszlopok szemantikája azonos az eredő teljesítmény monitorozásánál leírtaknál, míg a kék oszlop azt jelöli, hogy az adott aláírásról a hálónak nincs információja. (Mert önmagával kellene összehasonlítani.)

4.3.2 Az implementáció részletei

A 4.12.-es ábrán a főbb osztályokat láthatjuk. A program teljes osztálydiagramja az C függelékben található.



4.12 ábra: A tesztelő modul egyszerűsített osztálydiagramja

A `TestReader` osztály feladata, az adott CSV fájlok beolvasása, és `DataRow[]` formátumba konvertálása. Az egyes oszlopok típusát úgy adhatjuk meg, hogy az `IRowSpecification` interfészből leszármaztatunk.

A következő fontosabb művelet az adatok szétválogatása, mivel nem minden adatot akarunk tanításkor felhasználni. Ehhez az `Alairas.Common` névtérben létrehozott `SignUtil` osztály `GetIndexSet` és `HasSignIDs` függvényeit használtam fel. Az előbbivel lekérdezhető az indexek halmaza, az utóbbival pedig egy sorról megmondhatjuk, hogy adott indexek szerepelnek-e benne. Mivel egy aláírást a CSV fájlban szereplő indexe és az eredetisége együtt azonosít, létrehoztam egy `SignID` osztályt, amely összefogja ezeket, és így minden aláíráshoz egyedi azonosító rendelhető. A programban, és jelen dolgozatban ha egy aláírás „indexére” hivatkozom, akkor ezt az azonosítót értem alatta.

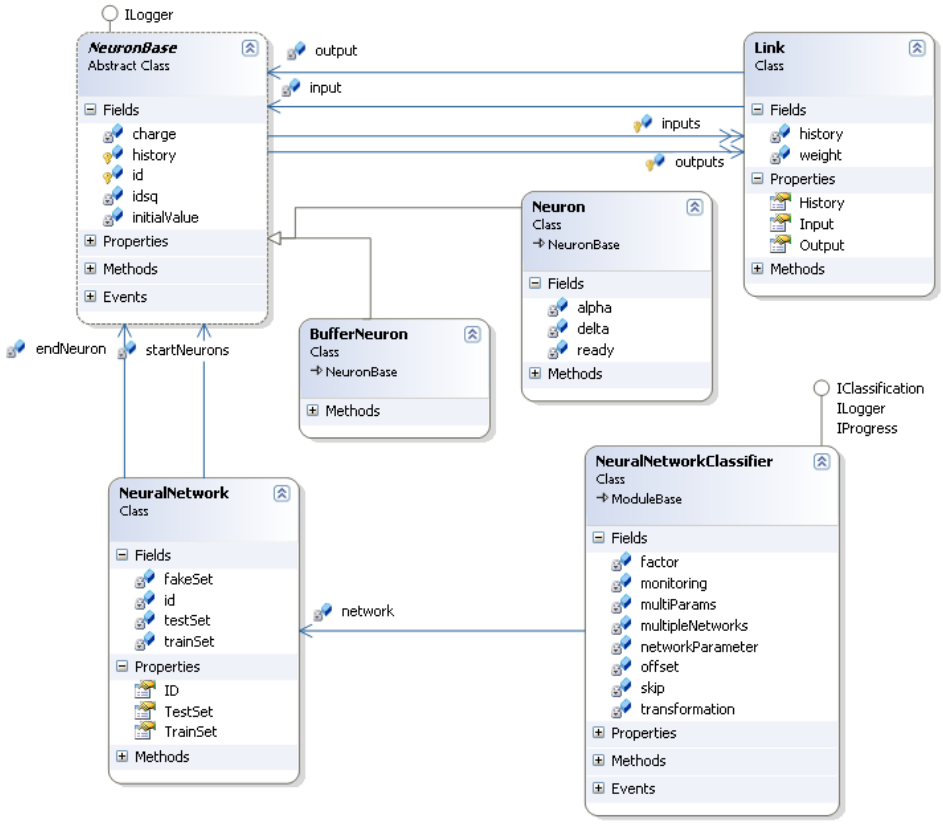
Ha az adatokat szétválogattuk kezdődhet a tényleges osztályozás. Általános osztályozó esetén csupán a `Train` és `Test` függvények meghívását jelentené, ám ha a hálókat monitorozni is akarjuk, akkor még a tanulás megkezdéses előtt minden adatot szükséges átadnunk az osztályozónak, ezért ebben az esetben a program a `CreateMonitoringSet` függvényt hívja először.

A fentieken kívül szükség van még az osztályozó modul üzeneteinek feldolgozására. Általános esetben csak ki kell írnunk a naplózó felületre a beérkező üzenetet, ám előfordulhatnak speciális üzenetek is:

- **Státusz üzenet –** Ezek az üzenetek „--” sorozattal kezdődnek, és azt jelzik, hogy az üzenetet a státusz sorba kell írni
- **Diagram üzenet –** Ezek „@@” karakterekkel kezdődnek és utána a diagram rajzolásához szükséges adatok érkeznek
- **Részletes diagram üzenek –** Ezek „$$” sorozattal kezdődnek és a részleteket megjelenítő diagram kirajzolásához szükségesek

4.4 A neurális háló alapú osztályozó implementációja

A 4.13-as ábrán látható az osztályozó rendszer egyszerűsített osztálydiagramja, a teljes diagram a D függelékben található.



4.13 ábra: Az osztályozó egyszerűsített osztálydiagramja

4.4.1 A neuronokkal kapcsolatos osztályok

4.4.1.1 NeuronBase

A működésük szempontjából kétféle neuront különböztethetünk meg. Az egyszerű pufferként viselkedő bemeneti neuronokat, és a transzformációt is végző „normál” neuronokat. A közös tulajdonságok a `NeuronBase` absztrakt osztályban vannak implementálva.

Minden neuron rendelkezik valamilyen kezdeti töltéssel (`initialValue`), amelyre kisülés után be kell állítani a töltöttségét, valamint aktuális töltéssel (`charge`) bemeneti és kimeneti élekkel (`inputs`, `outputs`), valamint a hibavisszaterjesztés érdekében a legutóbbi kisüléskori töltését (`history`) is el kell tárolni.

A konstruktorban az értékek beállítása mellett megadható egy neuron halmaz, melynek minden elemével összekötődik a neuron (előző rétegbeli neuronok). Említést érdemel még a `Send` metódus (4.1-es kódrészlet), amely egy üzentet fogad. A töltés beállításán túl, az is a funkciója, hogy ha már minden bemeneti élről érkezett üzenet, akkor a neuron tüzelni kezd, így valósul meg a neuronok kommunikációja.

```
public void Send(double d)
{
	charge += d;
	if (isReady())
	{
		double outValue = ComputeOutput(charge);
		if (outputs != null) foreach (Link li in outputs)
			li.Send(outValue);
		history = charge;
		charge = initialValue;
	}
}
```

4.1 Kódrészlet: A neuronok üzenetküldése

A hiba visszaterjesztő algoritmus két fő fázisa, a hibák kiszámítása és az él súlyok frissítése megfigyelhető a `Teach` függvényen (4.2-es kódrészlet).

```
public void Teach(double expected)
{
	Backprop(expected - Out);
	LearnFromError();
}
```

4.2 Kódrészlet: A neurális háló tanítása

Az osztály által specifikált absztrakt függvények:

- `double ComputeOutput(double input)` – a kimenet számítása a bemenet alapján
- `void Backprop(double d)` – hiba visszaterjesztés futtatása, ezáltal az egyes neuronok hibáinak meghatározása
- `void LearnFromError()` – él súlyok frissítése a korábban kiszámolt hibák alapján
- `bool isReady()` – a neuron feladata eldönteni, hogy minden információ rendelkezésre áll-e a tüzeléshez.

4.4.1.2 BufferNeuron

A `BufferNeuron` és a `Neuron` származik a fenti osztályból, amelyek a fenti két fajta neuront valósítják meg. A `BufferNeuron` nem tartalmaz bejövő éleket, nem végez transzformációt, és a hiba visszaterjesztési algoritmusa sem csinál semmit, hiszen mögötte már nincsenek élek.

4.4.1.3 Neuron

A `Neuron` osztály reprezentálja a normál neuronokat. Itt található a szigmoid függvény, a derivált szigmoid, az általunk aktivációs függvényként használt transzformált szigmoid függvény, és annak deriváltja (4.3 – 4.6 kódrészletek). A szigmoid függvény deriváltja az alábbi módon az eredeti függvényből is előállítható:

$$\frac{d}{dx}g(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = -1\frac{1}{\left(1+e^{-x}\right)^2}e^{-x} * (-1) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = g(x)(1-g(x))$$

```
double Sigmoid(double x)
{
	double num = 1;
	double den = 1 + Math.Pow(Math.E, -x);
	return num / den;
}
```

4.3 Kódrészlet: A szigmoid függvény

```
double Sigmoid_diff(double x)
{
	return Sigmoid(x) * (1 - Sigmoid(x));
}
```

4.4 Kódrészlet: Derivált szigmoid függvény

Az eredeti szigmoid függvény értékkészlete a (0; 1) intevallum, azonban a bemeneti adataink a [-1; 1] intervallumba esnek. Mivel többrétegű hálót használunk szükséges, hogy bármelyik rétegbeli neuron ugyanabba a tartományba eső értékeket kapja, ezért a függvény transzformáltuk, hogy a kimenete a (-1; 1) tartományba essen. A transzformáció hatására a derivált is változik.

```
private double G(double x)
{
	return 2 * Sigmoid(x) - 1;
}
```

4.5 Kódrészlet: A transzformált szigmoid függvény

```
private double G_diff(double x)
{
	return Sigmoid_diff(x) * 2;
}
```

4.6 Kódrészlet: A transzformált szigmoid függvény deriváltja

A hiba visszaterjesztés a következőképpen valósul meg: Ha egy neuronra `Backprop`-ot hívnak egy adott delta értékkel, akkor az élek mentén szétterjeszti az hátrafelé, így végül minden neuronban összegződnek a megfelelő delta értékek. Hasonló módon minden neuronra `LearFromError`-t hívhatunk, amit szintén elterjeszt visszafelé. A különbség annyi, hogy ez utóbbit minden neuronnak pontosan egyszer kell végrehajtania. A 4.7-os és a 4.8-es kódrészleten láthatjuk ezeket a függvényeket.

```
public override void Backprop(double d)
{
      delta += d;
      if (inputs != null)
      {
            foreach (Link li in inputs)
                  li.Backprop(d);
      }
}
```

4.7 Kódrészlet: Hibavisszaterjesztés

```
public override void LearnFromError()
{
      if ((delta != 0) && inputs != null)
            foreach (Link li in inputs)
            {
                  double dw =
                  alpha * delta * G_diff(history) * li.History;
                  li.LearnFromError(dw);
            }
      delta = 0;
}
```

4.8 Kódrészlet: Élek frissítése

4.4.1.4 Link

Ide tartozik még a `Link` osztály is, amely egy egyszerű irányított él szerepét tölti be, a rajta átfolyó üzeneteket megszorozza a súlyával valamint a `LearnFromError` üzenetet továbbterjeszti hátrafelé. Az üzenetküldés a 4.9-es kódrészletben olvasható.

```
public void Send(double d)
{
      history = d;
      if (output != null)
      {
            output.Send(weight * d);
      }
}
```

4.9 Kódrészlet: Üzenetküldés az éleken keresztül

4.4.1.5 NeuralNetwork

Ennek az osztálynak a feladata a hálózat felépítése, valamint tartalmazza a hozzá tartozó tanító- és teszthalmazokat, így képes tanulásra, és lekérhető az aktuális válasz is egy adott mintára. Fontos megemlíteni, hogy ezen osztály hivatott az egy- és többhálós modell közötti különbségek elfedésére. Az `Id` osztálytag többhálós modell esetén a háló által felügyelt aláírás indexe, egyhálós modell esetén pedig `null` értékű. A háló felépítését végző függvény az E függelékben olvasható.

Érdemes megjegyezni, hogy a program korai szakaszaiban először csak az egy hálós modellel próbálkoztunk, ezért akkor még nem volt ilyen osztály, és a `NeuralNetworkClassifier` közvetlenül a be- és kimeneti neuronok referenciáját tartalmazta.

4.4.1.6 NeuralNetworkClassifier

Ez az osztály reprezentálja a neurális háló alapú osztályozót, ennek megfelelően az IClassification interfészt valósítja meg. Ahhoz, hogy a hálórendszer működését menet közben is monitorozni lehessen, a tesztelési adatokat is át kell adni már tanításkor, ezért nem elegendő az IClassification interfész által specifikált Train és Test függvények megvalósítására szorítkozni. Hasonlóképpen, ha paraméterhangolást is végezni kívánunk, akkor az adatok megadásának sorrendje eltérő lehet. Másrészről szintén ügyelni kell arra is, hogy a neurális háló alapú osztályozót tesztelő modul kompatibilis maradjon a többi osztályozóval, így kétféle kompatibilitást kell szem előtt tartanunk a speciális funkciók beépítésénél.

Az osztályozást, és a kapcsolódó speciális funkciókat megvalósító függvények:

- `Train` – Paraméterként a tanító adatokat várja. Ha korábban másképp nem adtuk át az adatokat, akkor létrehozza a transzformációhoz szükséges információkat, transzformálja a tanító adatokat, létrehozza a hálórendszert és hozzáadja a tanító halmazokat. Végül elindítja a tanulási folyamatot a korábban megadott háló paraméterekkel, vagy alapértelmezett értékekkel, ha nem adtunk meg mást.
- `Test` – Paraméterként a tesztelő adatokat várja. Ha korábban másképp nem adtuk át az adatokat, akkor transzformálja a tesztelési adatokat és hozzáadja a hálórendszerhez a tesztelő halmazokat. Végül elindítja a kiértékelő folyamatot, és az utolsó kiértékelés eredményét visszaadja. Erre azért van szükség, mert a tesztelés során ez a függvény sok aláírást

tesztel, és az eredményeket üzenetek formájában továbbítja, míg a keretrendszer által specifikált működés szerint egyetlen aláírást tesztel, és a teszt eredményét vissza kell adnia.

- `FindParameters` – A tanító és a tesztelő adatokat várja paraméterként. Az előzőekhez hasonlóan először elvégzi a transzformációkat, és létrehozza a hálórendszereket a megfelelő tanító- és teszthalmazokkal, ha korábban ezt még nem tettük meg, majd a korábban megadott paraméterkombinációkkal elvégzi a tanítást és a tesztelést.
- `CreateMonitoringSet` – Paraméterként a tanító és a teszt adatokat várja. Elvégzi a transzformációkat, felépíti a hálót, és hozzáadja a tanító- és teszthalmazokat.
- `SetParams` – Beállítja a hálóparamétereket.
- `SetMultiParams` – Beállítja a paraméterkeresés alapjául szolgáló paraméterhalmazokat.

A fenti függvények meghívása tehát az alábbi módon történhet:

1. `[SetParams]`
2. `[CreateMonitoringSet]`
3. `Train`
4. `Test`

Vagy:

1. `SetMultiParams`
2. `[CreateMonitoringSet]`
3. `FindParameters`

(A szögletes zárójelbe írt részek opcionálisak)

A hálórendszer tanításában és tesztelésében az alábbi segédfüggvények vesznek részt:

- `VoteAnswer` – a szavazás útján kialakuló válasz kiszámítása.
- `ParameterizedTrain` – a hálórendszer tanítása a megadott paraméterek szerint. Fontos, hogy amennyiben bekapcsoltuk a monitorozást, úgy ennek a függvénynek a futása közben fognak keletkezni az ezzel kapcsolatos üzenetek
- `EvaluateTrain` – a hálórendszer tesztelése.

4.4.1.7 További segédosztályok

Praktikussági okokból bevezettünk néhány további segédosztály, ezek közül három (`SignID`, `TablePart`, `TeachSet`) egy-egy kisebb adatszerkezetet definiál, míg a negyedig (`SignUtil`) az aláírásokkal és az adatfeldolgozással kapcsolatos, gyakran használt műveleteket fogja egybe.

Egyszerű adatszerkezetek:

- `SignID` – Az aláírást eredetileg az eredetisége és az indexe együtt azonosítja. Ez az osztály összefogja ezt a két adatot.
- `TablePart` – Egy adott azonosítóval rendelkező aláíráshoz tartozó táblarészeket tárolhatunk az ilyen típusú struktúrákban
- `TeachSet` – A tanítóhalmazt tárolhatunk az ilyen típusú struktúrákban. Egységbe foglalja a bementeket az elvárt kimentet és a forrás aláírás azonosítóját. Egy hálót tartalmazó modell esetén az egyes tanítóhalmazokhoz egyetlen forrás aláírás rendelhető, még több hálós esetben ez az azonosító az mutatja meg, hogy az illető háló által „figyelt” aláírás melyikkel van összehasonlítva. (A `TeachSet`-ekhez egyértelműen hozzárendelhető egy háló)

A `SignUtil` osztály függvényei

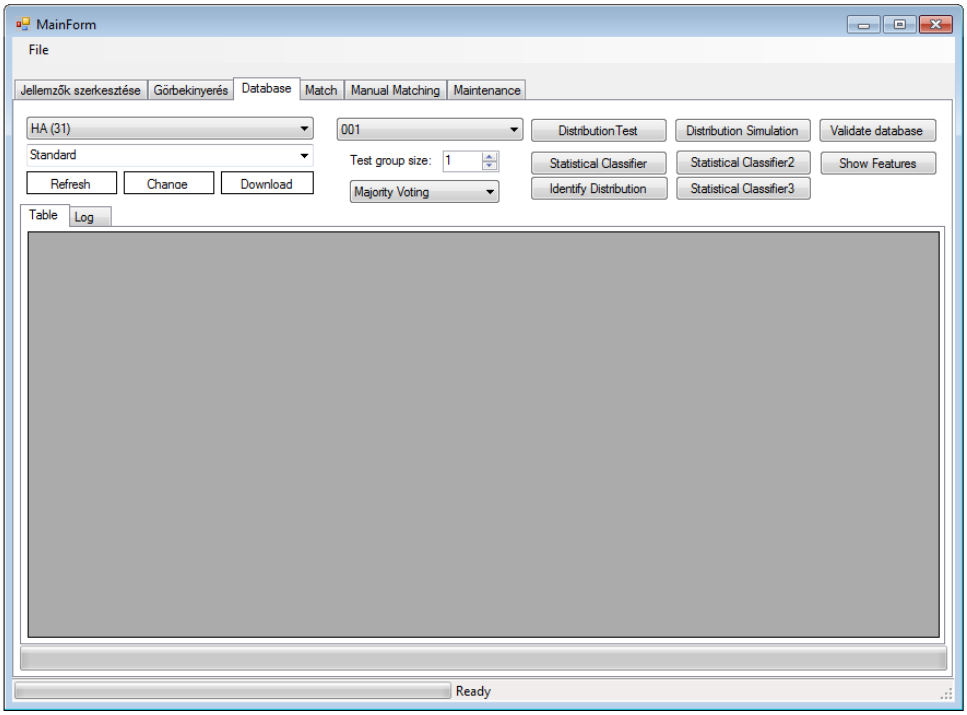
- `SignID[] GetIndexSet(DataRow[] rows, bool allowFake)` – Az adatsorokból kiolvassa az aláírás azonosítókat. Beállítható az is, hogy a hamis aláírások indexére is kíváncsiak vagyunk-e.
- `SignID GetSignID(DataRow row, int index)` – Az adott indexű aláírás azonosítóját adja vissza. (Egy adatsorban két aláírás azonosítója szerepel.)
- `bool HasSignIDs(DataRow row, SignID[] IDSet1, SignID[] IDSet2)` – Igaz értékkel tér vissza, ha a megadott adatsorban az aláírások azonosítója megfelel a megadott azonosító halmazok egy-egy elemének. Ha az első halmaz null, akkor az a minden azonosítót tartalmazó halmazt jelenti.

4.5 A Statisztikai osztályozás implementációja

Az AHR projekt keretei között több alkalmazást is létrehoztunk, amelyek a statisztikai osztályozási módszer valamely aspektusát vizsgálják. Jelen alfejezetben a 3.3-as szakaszban tárgyalt elméleti modell alapján létrehozott statisztikai osztályozók, illetve olyan tesztelő programok implementációs kérdéseire fókuszálunk, amelyek ezeket az osztályozókat szimuláció segítségével vagy valós aláírások feldolgozásából nyert adatok felhasználásával tesztelik. Először áttekintjük azokat az osztályokat és egyéb elemeket, amelyeket több kapcsolódó alkalmazásban is használtunk, majd megvizsgáljuk a statisztikai osztályozás szimulálására használt alkalmazást. Az alfejezet további részeiben az eloszlás azonosító rendszer és a statisztikai osztályozók implementációjával foglalkozunk.

4.5.1 Közös elemek

Az itt tárgyalt implementációk közül a szimulátor kivételével minden alkalmazást az AHR-ben már meglévő **DevelperGui** elnevezésű felhasználói felületet használja. Ennek egyik előnye, hogy nincs szükség külön felhasználói felület létrehozására minden egyes alkalmazáshoz így a feladat lényegi részének megoldására tudunk koncentrálni. Egy további előnye a közös felhasználói felületnek, hogy minden az aláírás hitelesítéssel kapcsolatos funkció egy helyről elérhető, mint például az adatbázisok letöltése, a jellemző kinyerés, a kézi vagy automatizált jellemző párosítás vagy éppen az osztályozók tesztelése. A 4.14-es ábra a közös felhasználói felületet mutatja be.



4.14-es ábra: A DeveloperGui elnevezésű közös felhasználói felület

A közös felhasználói felület ismertetése után most bemutatunk két olyan osztályt, amelyet minden az alfejezetben tárgyalt implementációban felhasználtunk. Az egyik `IStatisticalClassifier` interfész, amely a különböző statisztikai osztályozók közös funkcióit emeli ki.

```
public interface IStatisticalClassifier
{
        void Train( double[,] features );
        double Test( double[,] features, bool majorityVoting);
}
```

4.10 Kódrészlet: Az IStatisticalClassifier interfész

A `Train` függvény megadja az osztályozónak azt az adathalmazt, amelyet felhasználhat a tanulási fázisban. Mivel a statisztikai osztályozók tervezése során végig azt a feltevést követtük, hogy a tanulási fázisban biztosan soha nem fogunk hamisított aláírásokat használni, itt sincs lehetőség annak megadására, hogy a kapott tanítóhalmazban mely aláírások eredetiek, vagyis az osztályozónak azt kell feltételeznie, hogy mindegyik az. A `Test` függvény `majorityVoting` paramétere azt mondja meg, hogy miként kell összegezni a valószínűségeket, ha az osztályozó egy aláírás csoportot kap teszteléskor.

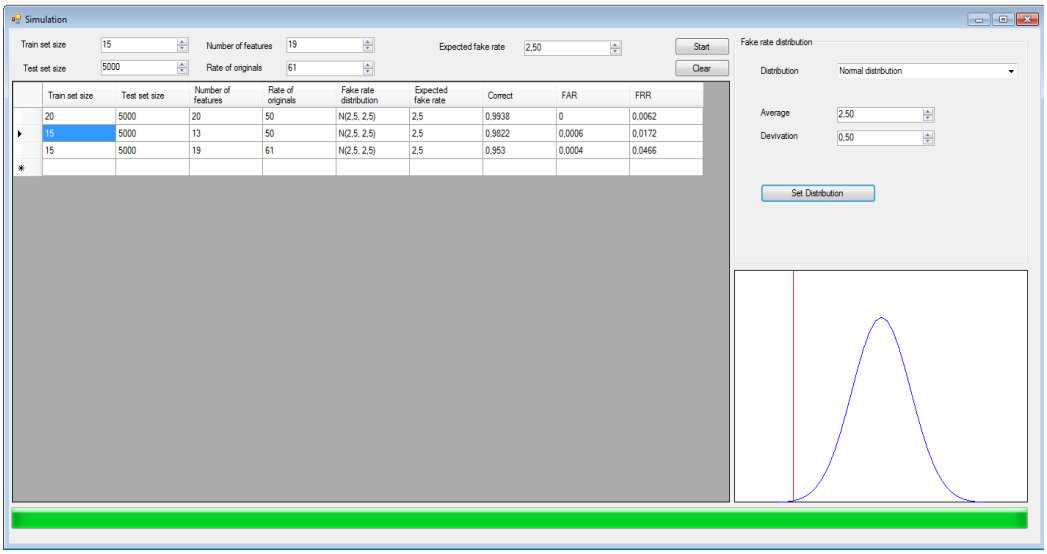
A másik közösen használt osztályt azért vezettük be, hogy tesztelés során könnyebben lehessen az eredményeket kezelni és egyszerűen ki lehessen mutatni a különböző hibatípusokat. Ennek az osztálynak a segítségével összeadhatjuk az osztályozási folyamatok eredményét, és hozzáadhatjuk egy új teszt eredményét a jelenlegi osztályozáshoz. A 4.11-es kódrészlet az osztály forrásának vázlatát mutatja be.

```
public class ClassifiacionTestResult
{

    public void Add(ClassifiacionTestResult r) {…}

    public void DoTest(bool result, bool expected) {…}

    public double FAR
    {
        get {…}
    }

    public double FRR
    {
        get {…}
    }

    public double CorrectRate
    {
        get {…}
    }
}
```

4.11 Kódrészlet: Az ClassificationTestResult osztály

4.5.2 Statisztikai osztályozás szimulált adatokon

Az osztályozás szimulációjának célja egyrészt hogy bizonyítsuk az elméleti modell működésének helyességét, másrészt hogy megfigyelhessük és megvizsgálhassuk azokat a motívumokat, amelyekben az éles adatok eltérnek a feltételezéseinktől. A 4.15-ös ábrán a szimulátor program felhasználói felületét tekinthetjük meg.



4.15-ös ábra: Az osztályozás szimuláló felhasználói felülete

A bal felső terülten beállíthatjuk a szimuláció paramétereit. Meghatározhatjuk a tanító és teszt adathalmazok méretét (**Train set size** és **Test set size**), beállíthatjuk a független írásjellemzők számát és a teszthalmazban található eredeti aláírások százalékos arányát (**Number of features** és **Rate of originals**), továbbá itt adhatjuk meg az osztályozás szigorúságának mértékét is (**Expected fake rate**).

Mivel az eredeti és hamis aláírások írásjellemzői szórásának aránya kulcsfontosságú az elméleti modell szempontjából, ez az arány a szimulátorban is kiemelt szerepet kapott. A jobb oldali területen beállíthatjuk ennek az aránynak az eloszlását és az eloszlás paramétereit. A szimulátorban uniform, normál és exponenciális eloszlások közül lehet választani.

A paraméterek beállítása után a **Start** gomb megnyomásával indíthatjuk el a szimulációt. A szimuláció lefutása közben az alsó állapotsávon követhetjük nyomon a folyamatot, míg a szimuláció futása után az eredménye a felület nagy részét elfoglaló táblázatban jelenik meg. A táblázatban nyomon követhetjük a beállított paraméter értékeket és megfigyelhetjük az egyes hibatípusok százalékos arányát.

A szimuláció algoritmusa nagyon egyszerű: Először létrehozzuk az írásjellemzőknek megfelelő eloszlásokat. Ezek normál eloszlások lesznek véletlenszerű paraméterezéssel az eredeti aláírások esetén, amelyek szórását a megfelelő számmal – melyet szintén egy valószínűségi változó ír le – beszorozva megkapjuk a hamisítványok eloszlását. Ezután legeneráljuk tanító halmazt a fentiek közül az eredeti aláírásokra vonatkozó eloszlások használatával, majd betanítunk egy példányt a tesztelendő

osztályozóból a kapott tanítóhalmazzal. Ezután a korábbi eloszlásokat felhasználva legeneráljuk a tesztelő halmazt is és végül elvégezzük a betanított osztályozó tesztelését.

4.5.3 Eloszlás azonosító

Az eloszlások azonosításának célja, hogy egy megadott mintahalmaz alapján megkeressük azt a nevezetes eloszlást, amelyhez a mintahalmaz eloszlása a legközelebb áll. Ehhez először megállapítjuk a kapott mintahalmaz várható értékét és szórását, majd felparaméterezzük az alábbi nevezetes eloszlásokat úgy, hogy a szórás és a várható érték a mintahalmazéval azonos legyen:

- Normál eloszlás
- Logaritmikus normál eloszlás
- Uniform eloszlás
- Háromszöges eloszlás
- Laplace eloszlás
- Gamma eloszlás
- Erlang eloszlás

A felparaméterezett eloszlásokat ezután a Kolmogorov-Szmirnov próba segítségével teszteljük le a mintahalmazra. Ez tehát azt jelenti, hogy a mintahalmaz tapasztalati eloszlásfüggvényének és a tesztelendő eloszlásfüggvény különbségének maximumát vesszük és ezt az értéket korrigáljuk a mintahalmaz elemszámával. A próba pontos algoritmusát 4.12-es kódrészlet mutatja be:

```
public double TestDirstribution(
        double[] sample,
        ContinuousDistribution distribution
){
        double Dx = 0.0;

        foreach( double x in sample )
        {
                double Fn =
                (double)( sample.Where( y => y <= x ).ToArray().Length ) /
                (double)( sample.Length );

                double F = distribution.CumulativeDistribution( x );

                double D = Math.Abs( Fn - F );

                if( D >= Dx )
                {
                        Dx = D;
                }
        }

        return Dx * Math.Sqrt( sample.Length );
}
```

4.12 Kódrészlet: A Kolmogorov-Szmirnov próba megvalósítása

4.5.4 Statsztikai osztályozó

A statisztikai osztályozót két változatban készítettük el. Az első változat minden esetben normál eloszlást feltételez, míg a második változat a korábban tárgyalt eloszlás azonosító algoritmust használja. Mivel a második változatból könnyen visszakaphatjuk az elsőt, ha az eloszlás azonosító részt a normál eloszlásokra korlátozzuk jelen szakaszban csak az eloszlás azonosítást is használó verziót tárgyaljuk.

Az osztályozó tanulási fázisában a kapott mintaadatokat írásjellemzők szerint csoportosítja, majd az eloszlás felismerő algoritmus segítségével minden egyes írásjellemző eloszlását felismeri és tárolja. A tesztelési fázisban az osztályozó egyetlen aláírást, vagy egy csupa eredeti vagy csupa hamis aláírásokból álló (homogén) aláírás csoportot kaphat. A csoport tesztelése esetén kétféleképpen is összegezhetjük az eredményt. Kiszámíthatjuk az egyes aláírásokra eredetiségének valószínűségét és a valószínűségek számtani közepét vehetjük, vagy számolhatunk úgy, mintha *k* elemű, külön-külön *f* írásjellemzőt tartalmazó aláírásunk helyett egyetlen, *f*k* írásjellemzőt tartalmazó aláírásunk lenne. Az írásjellemzők alapján az aláírások eredetiségének valószínűségét a 3.3.2-ban tárgyalt modell és képletek alapján számoljuk és a 3.3.3-ban megadott módon összegezzük. A valószínűségi összegzésre használt módszert a 4.13-as kódrészlet mutatja be.

```
private double summarize( double[] p )
{
	double PiPs = 1.0;
	double PiNPs = 1.0;

	for( int j = 0; j < p.Length; j++ )
	{
		if( !Double.IsNaN( p[j] ) )
		{
			PiPs *= p[j];
			PiNPs *= ( 1.0 - p[j] );
		}
	}

	double pA = PiPs / ( PiPs + PiNPs );
	return pA;
}
```

4.13 Kódrészlet: Valószínűségek összegzése

5 Tesztelés és az eredmények értelmezése

A most következő fejezetben a korábban bemutatott osztályozók tesztelésének eredményeit mutatjuk be és azt, hogy milyen következtetéseket vonhatunk le ezekből az eredményekből. Természetesen a tesztelések végső célja az, hogy a későbbiek során javítsunk az osztályozás és végül az egész aláírás hitelesítő rendszer pontosságán. Először áttekintjük a tesztek során felhasznált adatbázisokat, majd részletesen megvizsgáljuk a neurális hálók osztályozási eredményeit. Ezután látni fogjuk, hogy eloszlás felismerés és faktor analízis használatával miképpen tudunk modellt alkotni az aláírókról, végül a statisztikai osztályozás különféle eredményeit vehetjük szemügyre.

5.1 Aláírás adatbázisok

5.1.1 Az SVC20 adatbázis

Az irodalmi áttekintésben már említett „Signature Verifiacation Competiton” alkalmával is használt adatbázist alakítottuk át úgy, hogy offline tesztelésre használhassuk. Az eredeti adatbázisban tollvonás (stroke) adatok voltak tárolva, amelyek alapján rekonstruáltuk aláírások képeit. Ezek a képek egyszerű PNG fájlok, amelyeket akár egy offline aláírás hitelesítés során a beolvasási fázis után is kaphatunk volna. Ezeken képeken ezután alapvonal- és hurokdetektáló algoritmusokat futtattunk, melyeket később összepárosítottunk, és az így kinyert írásjellemzők kerültek végül az adatbázisunkba. Húsz eredeti és húsz hamis aláírás áll a rendelkezésünkre összesen húsz aláírótól.

5.1.2 A BME9 és a BME9-CA adatbázis

Ebben az esetben az Budapesti Műszaki és Gazdaságtudományi Egyetemen végeztünk aláírásgyűjtés és szintén az egyetemen belül kerestünk együttműködő személyeket a hamisítványok előállításához. Itt összesen kilenc aláíróhoz kaptunk tizenhat eredeti és tizenhat hamis aláírás mintát. Először itt is csak alapvonal és hurok adatok kinyerésével foglalkozzunk, így alakult ki a **BME9** adatbázis, később ezt kiegészítettük irányváltoztatásokkal és remegésekkel így jött végül létre a **BME9-CA** nevű új adatbázisunk.

5.1.3 A HA31 adatbázis

Ezt az adatbázist elsősorban az írásjellemzők eloszlásának tanulmányozása céljából hoztuk létre. Az aláírások gyűjtését és az előfeldolgozást jelen diplomaterv írója hajtotta végre a B függelékben található aláírás ívek és a 4.2 alfejezetben tárgyalt programok segítségével. A különlegessége, hogy nagyszámú – ötven – aláírást tartalmaz összesen harmincegy aláírótól, jelenleg nincsenek azonban hamisítványok ebben az adatbázisban így az osztályozó módszerek közvetlen tesztelésére még nem alkalmas.

5.2 A neurális háló alapú osztályozók eredményei

Mivel az aláírás hitelesítési folyamat több, egymásra épülő lépésből áll, először arra a kérdésre kerestük a választ, hogy mennyire lehetséges egyáltalán osztályozni azokat az adatokat, amiket az osztályozó kap. Másképpen: Létezik-e olyan háló vagy háló rendszer, amely minden aláírást a megfelelő módon osztályoz? Ehhez a hálórendszer már tanuláskor megkapta az összes aláírást (a hamisítványokat is). Ezzel azt az esetet próbáltuk kiszűrni, amikor már a jellemzők kinyerése sem elegendően pontos az osztályozáshoz.

A második lépésben azt vizsgáltuk meg, hogy a korábbiakban „osztályozhatónak” minősített aláírásokat mennyire tudjuk valójában osztályozni. Itt az osztályozó már nem kapta meg a hamisítványokat tanuláskor, és az eredeti aláírásoknak is csak a felét kapta meg. Ez utóbbira a 3.2.1.3 szakaszban említett túlilleszkedés elkerülése végett van szükség.

A neurális háló alapú osztályozók elkészülésekor még csak az SVC20 és a BME9 adatbázisok álltak a rendelkezésünkre, ezért a teszteket csak ezen a két adatbázison végeztük el.

5.2.1 Osztályozhatóság megállapítása

Az alábbi táblázat az SVC20 adatbázisban található adatok osztályozhatóságára kapott eredményeket foglalja össze.

Azonosító	Maximális találat (%)	Találat a legjobb hálórendszer esetén (%)
002.csv	**100**	**100**
006.csv	80	80
008.csv	**100**	**100**
010.csv	77,5	75
013.csv	**100**	**100**
015.csv	**100**	**100**
018.csv	**100**	**100**
020.csv	77,5	77,5
025.csv	**100**	**100**
028.csv	**100**	**100**
032.csv	92,5	92,5
033.csv	70	65
035.csv	**100**	**100**
037.csv	**100**	**100**
038.csv	82,5	82,5
040.csv	**100**	**100**

5.1 ábra: Az SVC20 adatbázis osztályozhatósága

Mivel ekkor főként hurok információkkal dolgoztunk, és néhány aláírásból ilyeneket egyáltalán nem is lehet kinyerni, a táblázatban nem jelenik meg minden azonosító. A maximális találat oszlop jelentése az, hogy a különböző hálórendszere milyen legjobb eredményt értek el az aláírás osztályozásában, míg a harmadik oszlopban a találati arány alapján legjobbra minősített osztályozó eredményei láthatóak. Az ábrából kitűnik, hogy a megvizsgált 16 aláírás esetén 10 bizonyult osztályozhatónak.

A legjobban teljesítő osztályozás paraméterei:

- Bátorsági faktor: 0.25
- Elő feldolgozó réteg: neurononként 5 kapcsolat
- Középső réteg: 1 réteg 20 neuronnal
- Iterációk száma: 400
- Hiányzó értékek kicserélése: Közép érték helyettesítés

A következő táblázat a BME9 adatbázison futtatott teszt eredményét mutatja, ebben az esetben a 7 aláírásból 4-et találtunk osztályozhatónak.

Azonosító	Maximális találat (%)	Találat a legjobb hálórendszer esetén (%)
037.csv	96,875	96,875
038.csv	**100**	**100**
039.csv	**100**	**100**
041.csv	**100**	**100**
043.csv	93,75	93,75
044.csv	96,875	96,875
045.csv	**100**	**100**

5.2 ábra: A BME9 adatbázis osztályozhatósága

A legjobban teljesítő osztályozás paraméterei:

- Bátorsági faktor: 0.15
- Elő feldolgozó réteg: neurononként 5 kapcsolat
- Középső réteg: 1 réteg 20 neuronnal
- Iterációk száma: 400
- Hiányzó értékek kicserélése: Maximális érték helyettesítés

5.2.2 Az osztályozás hatékonysága

Az alábbi táblázat az SVC20 adatbázis korábban osztályozhatónak minősített adatain lefutatott osztályozás eredményét mutatja. Az ábrából kitűnik, hogy 10-ből 7 esetben a tanító adatokra adott válasz legalább 80%-os volt, és a teszt adatokra adott válasz legalább 70%-os volt a legjobb osztályozás során. Azt is láthatjuk, hogy a legjobbnak bizonyult osztályozó kb. 70%-s találati arányt mutatott,

Azonosító	Maximális találat (%)		Találat a legjobb háló rendszer esetén (%)	
	Tanító adatok	Teszt adatok	Tanító adatok	Teszt adatok
002.csv	100	56,67	100	40
008.csv	90	90	70	83,33
013.csv	100	76,67	90	63,33
015.csv	100	73,33	100	66,67
018.csv	80	83,33	60	83,33
025.csv	90	70	80	56,67
028.csv	40	53,33	40	53,33
035.csv	100	96,67	100	93,33
037.csv	70	73,33	50	66,67
040.csv	100	93,33	100	86,67
Átlag	**87**	**76.67**	**79**	**69.33**

5.3 ábra: Az osztályozható SVC20 aláírások osztályozása

A legjobban teljesítő osztályozás paraméterei:

- Bátorsági faktor: 0.13
- Elő feldolgozó réteg: neurononként 9 kapcsolat
- Középső réteg: 1 réteg 10 neuronnal
- Iterációk száma: 400
- Hiányzó értékek kicserélése: Közép érték helyettesítés
- Generált hamis minták távolsága: 0.225

A következő táblázat hasonlóképpen a BME9 adatbázison lefuttatott tesztek eredményeit mutatja. Látható, hogy itt is vannak különösen jól osztályozható aláírások, és az átlagos találati arány szintén 70% körüli.

Azonosító	Maximális találat (%)		Találat a legjobb háló rendszer esetén (%)	
	Tanító adatok	Teszt adatok	Tanító adatok	Teszt adatok
038.csv	87,5	75	87,5	75
039.csv	100	91,67	87,5	79,17
041.csv	100	66,67	100	45,83
045.csv	100	87,5	100	87,5
Átlag	**96.88**	**80.21**	**93.75**	**71.88**

5.4 ábra: Az osztályozható BME9 aláírások osztályozása

A legjobban teljesítő osztályozás paraméterei:

- Bátorsági faktor: 0.2
- Elő feldolgozó réteg: neurononként 5 kapcsolat
- Középső réteg: 1 réteg 20 neuronnal
- Iterációk száma: 400
- Hiányzó értékek kicserélése: Közép érték helyettesítés
- Generált hamis minták távolsága: 0.25

5.3 Az eloszlás felismerés eredményei

Az eloszlás felismerést az SVC20 és a HA31 adatbázisokon hajtottuk végre. Az alábbi ábrákon megfigyelhetjük az eloszlások arányát.

Eloszlás	SVC20		HA31	
	Darabszám	Arány	Darabszám	Arány
Normál eloszlás	768	62,34%	3030	65,78%
Laplace eloszlás	260	21,10%	836	18,15%
Háromszöges eloszlás	48	3,90%	353	7,66%
Gamma eloszlás	148	12,01%	341	7,40%
Uniform eloszlás	8	0,65%	46	1,00%

5.5 ábra: Mért eloszlások aránya

A táblázatból kitűnik, hogy nagyságrendileg helyes volt az a feltételezésünk, hogy az írásjellemzők túlnyomó többségben normál eloszlást követnek. Az is látható, hogy csupán a normál és a Laplace eloszlás használatával az írásjellemzők 83%-a lefedhető mindkét tesztelt adatbázis esetén, ami arra enged következtetni, hogy a későbbiekben érdemes ezt a kétfajta eloszlást alaposabban megfigyelnünk.

5.4 Faktor analízis eredmények

A faktor analízis segítségével megpróbáltuk az írásjellemzők függetlenségét vizsgálni, illetve visszavezetni ezeket az írásjellemzőket független faktorokra. Nem titkolt célunk az is, hogy megpróbáljuk ezeket a független faktorokat az aláíró valamilyen biológiai jellemzőjéhez kötni, illetve természetesen a faktor analízis az osztályozó algoritmust is segíthet tovább fejleszteni, ha a későbbiekben az írásjellemzők helyett, ezekkel a független faktorokkal számolunk. Jelenleg az SVC20, a BME9-CA, és a HA31 adatbázisokból kinyerhető írásjellemzőkön végeztünk ilyen elemzést.

A BME9-CA esetén 189 írásjellemzőt sikerült 24 független komponensre visszavezetni, míg az SVC20 esetén 134-et 13-ra. A HA31 adatbázis esetén kipróbáltuk a faktoranalízist úgy, hogy csak az alapvonal és hurok adatokat hagytuk az adatbázisban, én úgy is, hogy minden adatot figyelembe vettünk. Az első esetben 170 írásjellemzőt tudtunk 15 komponensre bontani, a második esetben viszont meglepő módon 223 írásjellemzők bontottunk le mindössze 3 faktorra.

Az igazán látványos eredményeket azonban az SVC20 adatbázis komponens mátrixának vizsgálata során láttuk. A komponens mátrix terjedelmi okokból az F függelékben tekinthető meg. Az ábrából kitűnik, hogy az írásjellemzőkből kinyert faktorokat látványosan hozzá lehet rendelni a makroszkopikus írásjellemzőköz. A legtöbb makroszkopikus írásjellemzőről ugyanis világosan látható, hogy mely faktorok vesznek részt döntően a kialakításában, és mely faktorok elhanyagolható mértékben.

5.5 A statisztikai osztályozó szimulációs eredményei

A következő mérések során olyan véletlenszerűen generált adathalmazzal dolgoztunk, amelyek megfelelnek a statisztikai alapú osztályozó ideális alapfeltevéseinek.

Először arra kerestük a választ, hogy egy-egy írásjellemzőnek mekkora elválasztó ereje lehet. Az, hogy egy írásjellemző esetén a hamisítványok és az eredeti

írásjellemzők szórásának aránya milyen, miképpen befolyásolja az osztályozás sikerességét. Az eredményeket az 5.6-os ábra foglalja össze.

Tanító halmaz mérete	Teszt halmaz mérete	Írásjellemzők száma	Szórás arány	Találat (%)	FAR (%)	FRR (%)
200	5000	1	1,5	58,86	33,58	7,56
200	5000	1	2	66,2	27,36	6,44
200	5000	1	2,5	71,84	21,08	7,08
200	5000	1	3	73,58	19,04	7,38
200	5000	1	4	79,06	14,6	6,34
200	5000	1	5	82,44	11,14	6,42

5.6 ábra: Az írásjellemzők elválasztó ereje ideális statisztikai osztályozás esetén

Úgy láttuk, hogy ha a szórások aránya legalább 2.5, akkor az írásjellemző elválasztó ereje már több mint 70%-os, vagyis csupán ennek az egy írásjellemzőnek a segítségével 70%-os valószínűséggel jó osztályozási eredményt kapunk.

Később azt néztük meg, hogy a független írásjellemzők száma miképpen hat az osztályozás találati arányára. A mérés eredményei az 5.7-es ábrán láthatóak.

Tanító halmaz mérete	Teszt halmaz mérete	Írásjellemzők száma	Szórás arány	Találat (%)	FAR (%)	FRR (%)
200	5000	1	2,5	70,92	21,38	7,7
200	5000	2	2,5	78,22	18,04	3,74
200	5000	5	2,5	91,54	5,4	3,06
200	5000	10	2,5	97,46	1,54	1
200	5000	15	2,5	99,24	0,42	0,34
200	5000	20	2,5	99,68	0,14	0,18
200	5000	25	2,5	99,78	0,04	0,18
200	5000	50	2,5	100	0	0
200	5000	100	2,5	100	0	0
200	**5000**	**200**	**2,5**	**100**	**0**	**0**

5.7 ábra: Osztályozás független írásjellemzőkkel ideális statisztikai osztályozás esetén

Az ábrából kitűnik, hogy mindössze 5 olyan független írásjellemző segítségével, amelyeknek külön-külön csak 70%-os elválasztó ereje van – amit 2.5-ös szórás arány esetén mértünk – elérhető, hogy az osztályozás pontossága 90% fölötti legyen. Tizenöt ilyen jellemző pedig már 99% fölötti pontosságot biztosít.

A következő mérés során azt vizsgáltuk, hogy ideális esetben mekkora tanítóhalmaz szükséges a sikeres osztályozáshoz. Az eredményeket az 5.8-as ábrán tekinthetjük meg.

Tanító halmaz mérete	Teszt halmaz mérete	Írásjellemzők száma	Szórás arány	Találat (%)	FAR (%)	FRR (%)
5	5000	20	2,5	63,82	0	36,18
6	5000	20	2,5	80,74	0,02	19,24
7	5000	20	2,5	81,38	0	18,62
8	5000	20	2,5	93,06	0,04	6,9
9	5000	20	2,5	89,9	0,02	10,08
10	5000	20	2,5	97,9	0,04	2,06
15	5000	20	2,5	96,72	0,02	3,26
20	5000	20	2,5	99,22	0,1	0,68

5.8 ábra: Osztályozás különböző tanítóhalmazokkal ideális statisztikai osztályozás esetén

Jól látható, hogy a kezdeti bizonytalanság gyorsan eliminálódik, amint a tanító halmaz mérete tíz vagy annál több aláírásból áll.

5.6 A Statisztikai osztályozó eredményei éles adatokon

A statisztikai osztályozót az SVC20 és a BME9-CA adatbázisokon próbáltuk ki. Először az osztályozónak azzal a típusával végeztünk méréseket, amely minden írásjellemzőre normáleloszlást feltételez. Az 5.9-es és az 5.10-es ábra szemlélteti ezeket az eredményeket.

Azonosító	Találat (%)	FAR (%)	FRR (%)
2	75,5198	14,0594	10,4208
4	83,3663	8,4653	8,1683
6	79,7277	12,1535	8,1188
8	88,5644	5,495	5,9406
10	67,3267	22,4257	10,2475
13	77,6238	15,0743	7,302
15	77,0792	15,2723	7,6485
18	87,0792	3,9356	8,9851
20	90,6436	0,2723	9,0842
22	67,3515	24,6287	8,0198
24	90,396	0,0743	9,5297
25	69,6782	22,5248	7,797
28	88,8366	3,6881	7,4752
32	73,9604	17,599	8,4406
33	76,4109	16,2376	7,3515
34	73,6881	16,4356	9,8762
35	92,3267	0,2228	7,4505
37	87,005	3,5149	9,4802
38	87,995	4,5297	7,4752
40	89,5545	1,1386	9,3069
Átlag	**81,2067**	**10,3874**	**8,4059**

5.9 ábra: Osztályozás az SVC20 adatbázison statisztikai osztályozóval

Azonosító	Találat (%)	FAR (%)	FRR (%)
37	84,6535	0	15,3465
38	85,8911	0,1547	13,9542
39	87,2525	0,8354	11,9121
40	88,4282	0,8973	10,6745
41	85,5198	3,4653	11,0149
42	86,9431	0,0309	13,026
43	87,2215	0,6498	12,1287
44	83,0446	0,2166	16,7389
45	88,3973	0	11,6027
Átlag	**86,3724**	**0,6944**	**12,9332**

5.10 ábra: Osztályozás a BME9-CA adatbázison statisztikai osztályozóval

Ezután a statisztikai osztályozónak azzal a típusával végeztünk méréseket, amely az egyes írásjellemzőket többféle nevezetes eloszlással is összeveti és a legjobban illeszkedőt választja.

Azonosító	Találat (%)	FAR (%)	FRR (%)
2	72,8218	17,2525	9,9257
4	82,9703	10,3713	6,6584
6	78,7129	13,2921	7,995
8	88,4653	6,0149	5,5198
10	64,2574	25,5941	10,1485
13	77,797	15,0743	7,1287
15	74,901	18,3911	6,7079
18	85,0495	6,3861	8,5644
20	91,7327	0,7178	7,5495
22	64,9505	27,9703	7,0792
24	91,0396	0,9406	8,0198
25	69,4802	22,203	8,3168
28	89,1337	4,8515	6,0149
32	74,3069	17,698	7,995
33	75,3713	18,2673	6,3614
34	73,4901	17,3762	9,1337
35	92,3515	0,3218	7,3267
37	85,495	5,3713	9,1337
38	87,9703	5,7921	6,2376
40	90,5693	1,3366	8,0941
Átlag	**80,5433**	**11,7611**	**7,6955**

5.11 ábra: Osztályozás az SVC20 adatbázison eloszlás felismerő statisztikai osztályozóval

Azonosító	Találat (%)	FAR (%)	FRR (%)
37	86,5408	0,0619	13,3973
38	86,9121	1,2376	11,8502
39	89,8824	1,2067	8,9109
40	88,3045	1,7946	9,901
41	83,5396	7,2092	9,2512
42	88,7376	0,1547	11,1077
43	88,2735	1,5161	10,2104
44	84,9938	1,6399	13,3663
45	91,0272	0	8,9728
Átlag	**87,5791**	**1,6467**	**10,7742**

5.12 ábra: Osztályozás a BME9-CA adatbázison eloszlás felismerő statisztikai osztályozóval

Az eredményekből kiolvashatjuk, hogy az eloszlás felismerés igen kis mértékben befolyásolja az osztályozás eredményét. Ebből arra következtethetünk, hogy habár csupán az írásjellemzőknek csak közel 65%-ára illeszkedik a nevezetes eloszlások közül legjobban a normál eloszlás, a legtöbb írásjellemzőt mégis jól modellezhetjük vele. Az eredményekből az is élesen kitűnik, hogy az eredmények, habár jobbak, mint a neurális hálók eseté, így is messze elmaradnak a szimuláció által jósolt számoktól, ráadásul egy nagyságrenddel több írásjellemzőt használtunk, mint a szimulációk során. Mivel az eloszlás felismerés kimutatta, hogy normál eloszlásos modellünk javarészt helytálló, az eltérés okát két további lehetséges helyen kereshetjük: Egyrészt nincs semmi garancia a valós írásjellemzők tényleges elválasztó erejéről. Az eloszlás felismerés kimutatta, hogy némely esetben a hamisítványok írásjellemzőinek szórása akár kisebb is lehet, mint az eredeti aláírásoké, és sok helyen csak kis mértékben tér el. Másrészt az írásjellemzőink valószínűleg nem függetlenek egymástól, mint ahogy azt a faktor analízis is megmutatta.

Megvizsgáltuk még azt az esetet is, amikor az osztályozó egy homogén aláírás csoportot kap. Az 5.13-as és az 5.14-es ábra e mérések eredményeit foglalja össze.

Összegzés módja ó	Aláírások egy csoportban	Találat (%)	FAR (%)	FRR (%)
Valószínűségi összegzés	1	80,7955	9,7386	09,465909
	3	84,4274	4,9781	10,594406
	5	86,5642	3,1996	10,236185
	8	87,2193	2,0410	10,73975
Átlagolás	1	80,7955	9,7386	9,465909
	3	87,2509	7,1285	5,620629
	5	90,6684	5,8957	3,435829
	8	93,9483	4,0285	2,023173

5.13 ábra: Csoportos aláírás hitelesítés az SVC20 adatbázison

Összegzés módja ó	Aláírások egy csoportban	Találat (%)	FAR (%)	FRR (%)
Valószínűségi összegzés	1	86,8606	6,3669	6,7725
	3	89,9998	1,9284	8,0718
	5	89,1779	0,7021	10,1199
	8	85,7880	0,0129	14,1990
Átlagolás	1	86,8606	6,3669	6,7726
	3	92,9647	3,5603	3,4751
	5	95,3396	2,5647	2,0956
	8	96,456	2,0309	1,5132

5.14 ábra: Csoportos aláírás hitelesítés az BME9-CA adatbázison

Az eredményekből világosan látszik, hogy a csoportos módszer valóban sokat tud javítani az osztályozási eredményen, és az is megfigyelhetjük, hogy míg a valószínűségi összegzéses módszer a FAR jellegű hibát tudja hatékonyan csökkenteni, addig az átlagolás mind a FAR, mind az FRR jellegű hibákat egyenletesen csökkenti.

5.7 Osztályozás a HA31 adatbázison

Habár a HA31 adatbázis nem tartalmaz hamisított aláírásokat – ami miatt a teljes körű osztályozást nem tudjuk rajta kipróbálni – elvégeztünk egy osztályozást az SVC20 és a BME9-CA adatbázishoz igazított szigorúsági paraméterek mellett, hogy megfigyelhessük milyen FRR arányt produkál az osztályozó ezen az adatbázison. A mérések eredményei az 5.15 ábrán olvashatóak.

Azonosító	Találat (%)	FAR (%)	FRR (%)
1	71,1164	0	28,8836
2	65,5855	0	34,4145
3	80,915	0	19,085
6	76,2718	0	23,7282
8	73,8136	0	26,1864
9	62,9225	0	37,0775
10	69,8191	0	30,1809
12	65,3465	0	34,6535
13	76,7839	0	23,2161
14	83,3049	0	16,6951
16	76,3742	0	23,6258
17	58,8938	0	41,1062
18	60,6692	0	39,3308
26	65,6879	0	34,3121
28	72,1065	0	27,8935
30	73,6087	0	26,3913
31	77,1594	0	22,8406
35	72,4821	0	27,5179
36	72,4821	0	27,5179
39	68,7265	0	31,2735
40	67,7023	0	32,2977
41	70,4677	0	29,5323
42	80,676	0	19,324
43	88,3237	0	11,6763
44	69,5801	0	30,4199
45	72,3114	0	27,6886
50	72,4138	0	27,5862
51	73,5063	0	26,4937
53	69,7166	0	30,2834
54	74,5647	0	25,4353
55	74,0184	0	25,9816
Átlag	72,1726	0	27,8274

5.14 ábra: FRR mérés a HA31 adatbázison

Az eredmények alapján látszik, hogy az osztályozó kicsivel gyengébben teljesített, mint a többi adatbázison. Ebből azt a következtetést vonhatjuk le, hogy a jellemzőkinyerés továbbfejlesztésével is lehetne javítani az eredményeken. Ezen kívül láthatjuk, hogy sok esetben félre viheti az osztályozást, ha az írásjellemzők eloszlását a kisszámú tanító minta miatt rosszul állapítja meg az osztályozó.

6 Összefoglalás

Jelen diplomamunkában több aspektusából is megfigyelhettük az offline aláírás hitelesítési rendszereket. Láttuk, hogy milyen nehézségei vannak az offline rendszereknek az online rendszerekhez képest. Betekintést nyerhettünk a jelenleg megtalálható offline rendszerekben és megvizsgáltuk az automatikus osztályozás elméleti alapjait, miközben több osztályozási módszert is szemügyre vettünk.

Láttuk az AHR projekt felépítését, az egyes modulok szerepét. Megismertük a neurális háló alapú osztályozók felépítésének lehetőségeit, valamint az ilyen hálókból álló hálórendszerek használatát, és a hálórendszer egyes paramétereinek szerepét. Megfigyeltük az is, hogy a rendelkezésre álló adatokon milyen transzformációkat kell végre hajtanunk ahhoz, hogy a hiányzó értékeket kipótoljuk anélkül, hogy torzítanánk az adatokat. Az aláírás hitelesítés legnagyobb kihívása, hogy az osztályozó tanítása során nem állnak rendelkezésre hamis aláírás minták, ezért megismertünk egy konkrét módszert ennek a problémának az áthidalására és azt is láttuk, hogy mik ennek a módszernek a veszélyei.

Megfigyelhettük az aláírás hitelesítés statisztikai modelljének létrehozásához vezető lépéseket: Láttuk a modell előfeltevéseit és határait, majd megvizsgáltuk, hogy miként lehet a modell segítségével egyetlen írásjellemző alapján osztályozni. Ezután a modell kiterjesztésével ezt a módszert fel tudtuk használni arra az esetre, amikor több, független írásjellemző áll a rendelkezésünkre. Ezek után megismerkedtünk a modellhez kapcsolódó járulékos mérési feladatokkal úgy, mint függetlenség- és eloszlásvizsgálat.

A dolgozat legnagyobb részében a fenti osztályozók implementációs kérdéseivel foglalkozott. Nemcsak az osztályozáshoz közvetlenül kapcsolódó szoftver elemek implementációját vizsgáltuk meg, hanem a képfeldolgozási és tesztelési feladatok implementációjára is kitértünk. Láttuk a kép kivágási és vonal eltávolítási algoritmusok implementációját, és megvizsgáltuk a neurális háló alapú osztályozó és a hozzá tartozó tesztelő szoftver megvalósításának módját. Áttekintettük továbbá a statisztikai modell implementációjának lépéseit is.

Az utolsó fejezetben az osztályozók teszteléséről és a teszt eredmények értelmezéséről volt szó. Láttunk ígéretes eredményeket, ám azzal is világossá vált, hogy a rendszer még nem áll készen a valós körülmények között való használatra.

Megfigyelhettük az eloszlás- és a függetlenségvizsgálat eredményeit, amelyeket később mind a modell finomítására, mind az osztályozó pontosságának javítására fel lehet majd használni. Világossá váltak azok a pontok is, ahol az elméleti modellünk a gyakorlatban is helytállónak mutatkozott, de egyelőre az éles adatokon mért eredmények bővel elmaradnak szimulációktól. Szemügyre vettük és elemeztük az újonnan létrehozott, HA31 elnevezésű adatbázist is, amelyből szintén fontos következtetéseket tudtunk levonni.

A projekt kétségkívül jelen dolgozat után is folytatódni fog. A következő lépések egyike a statisztikai modell tökéletesítése lehet. A faktoranalízis során láttuk, hogy az írásjellemzőket lehetőség van visszavezetni független faktorokra, amelyeket érdemes lehet közelebbről is megvizsgálni és kideríteni, hogy vajon ezek valamilyen nagyobb biometrikus jellemzőhöz köthetőek. A faktoranalízis eredményeit felhasználhatjuk arra is, hogy a jelenleg használt írásjellemzők közül kiválasszuk azokat, amelyek a legjobban hozzájárulnak az osztályozás sikeréhez.

Azt is láttuk, hogy a HA31 adatbázison a várnál nagyon FRR értékek mérhetők, és ennek az eltérésnek a vizsgálata szintén közelebb vihet minket az aláírókról alkotott statisztikai modell finomításához. Érdemes lenne ehhez az adatbázishoz hamisítványokat is létrehozni, mert így az osztályozók tesztelése során is rendelkezésünkre állna egy olyan adatbázis, ami az objektív méréshez elegendően nagy mennyiségű aláírást is tartalmazna.

Habár a rendszer jelenleg nem alkalmazható éles körülmények között, a szimulációs eredmények megmutatatták, hogy a statisztikai modell tökéletesítésével és az egyes részek finomításával el lehet majd érni egy olyan pontossági arány, ami már megfelel a gyakorlatban elvárt követelményeknek.

Irodalomjegyzék

[1] D.-Y. Yeung, et al., "SVC2004: First International Signature Verification Competition," *Lecture Notes in Computer Science, Biometric Authentication, Volume 3072/2004*, pp. 16-22, 2004.

[2] J. Ortega-Garcia, J. Fierrez-Aguilar, J. Martin-Rello, and a. J. Gonzalez-Rodriguez, "Complete signal modeling and score normalization for function-based dynamic signature verification," *Lecture Notes in Computer Science, in Audio- and Video-Based Biometric Person*, vol. 2688, p. 658–667, 2003.

[3] L. Bovino, S. Impedovo, G. Pirlo, and L. Sarcinella, "Multi-expert verification of hand-written signatures," *Proc. 7th Int. Conf. Doc. Anal.*, p. 932–936, Aug. 2003.

[4] D. Impedovo and G. Pirlo, "Automatic Signature Verification: The State of the Art," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and reviews*, vol. 38, no. 5, pp. 509-635, Sep. 2008.

[5] A. N. Abu-Rezq and A. S. Tolba, "Cooperative Self-Organizing Maps for Consistency Checking and Signature Verification," *Digital Signal Processing*, vol. 9, pp. 107-119, 1999.

[6] L. P. Cordella, P. Foggia, C. Sansone, F. Tortorella, and M. Vento, "A Cascaded Multiple Expert System for Verification," *Lecture Notes in Computer Science, Multiple Classifier Systems*, vol. 1857, pp. 330-339, 2000.

[7] X.-H. Xiao and G. Leedham, "Signature verification by neural networks with selective attention," *Applied Intelligence*, vol. 11, no. 2, pp. 213-223, 1999.

[8] M. A. Ferrer, J. B. Alonso, and C. M. Travisco, "Offline geometric parameters for automatic signature verification using fixed point arithmetic," *IEEE Transactions on Pattern Analyisis and Machine Intelligence*, vol. 27, no. 6, pp. 993-997, Jun. 2005.

[9] J. Fierrez-Aguilar, S. Krawczyk, J. Ortega-Garcia, and A. K. Jain, "Fusion of Local and Regional Approaches for On-Line Signature Verification," *IWBRS 2005, LNCS 3781*, p. 188–196, 2005.

[10] F. Leerle and R. Palmond, "Automatic Signature Verification - The State of the Art

1989-1993," *Int'l Pattern Recognition and Artificial Intelligence, special issue signature verification*, vol. 8, no. 3, pp. 643-660, 1994.

[11] R. Plamondon and G. Lorette, "Automatic Signature Verification and Writer Identification - The State of the Art," *Pattern Recognition*, vol. 22, no. 2, pp. 107-131, 1989.

[12] R. Plamondon and S. N. Sargur, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63-80, 2000.

[13] B. Kovari, "The development of off-line signature verification methods, comparative study," in *microCAD 2007 International Scientific Conference*, 2007.

[14] B. Found and D. Rogers, "The initial profiling trial of a program to characterise forensic handwriting examiners skill," *Journal of the American Society of Questioned Document Examiners*, vol. 6, no. 2, pp. 483-492, 2003.

[15] J. Coetzer, B. M. Herbst, and J. A. d. Preez, "Off-Line Signature Verification: A Comparison between Human and Machine Performance," *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[16] G. Fielding, K. Gummadidala, M. Kam, and R. Conn, "Signature Authentication by Forensic Document Examiners," *Journal of Forensic Sciences*, vol. 46, no. 4, Jul. 2001.

[17] K. A. Jain, "Signature Verification," *Michigan State University - Biometrics*, 2010.

[18] D. Muramatsu and T. Matsumoto, "Effectiveness of Pen Pressure, Azimuth, and Altitude Features for Online Signature Verification," *Advances in Biometrics*, no. 4642, pp. 503-512, Aug. 2007.

[19] H. Lei and V. Govindaraju, "A comparative study on the consistency of features in on-line signature verification," *Pattern Recognition Letters*, vol. 26, p. 2483–2489, 2005.

[20] S. Russell and P. Norvig, *Mesterséges intelligenca modern megközelítésben*. Panem Kiadó Kft, 1999.

[21] H. Baltzakisa and N. Papamarkos, "A new signature verification technique based on a two-stage neural network classifier," *Engineering Applications of Artificial Intelligence*, vol. 14, pp. 95-103, 2001.

[22] E. Ozgunduz, T. Senturk, and M. Karsligil, "Off-line Signature Verification and Recognition by Support Vector Machine," *Thirteenth European Signal Processing Conference*, 2005.

[23] M. K. Kalera, S. Srihari, and A. Xu, "Offline Signature Verification and Identification Using Distance Statistics," *International Journal of Pattern Recognition and Artifcial Intelligence*, vol. 18, no. 7, pp. 1339-1360, 2004.

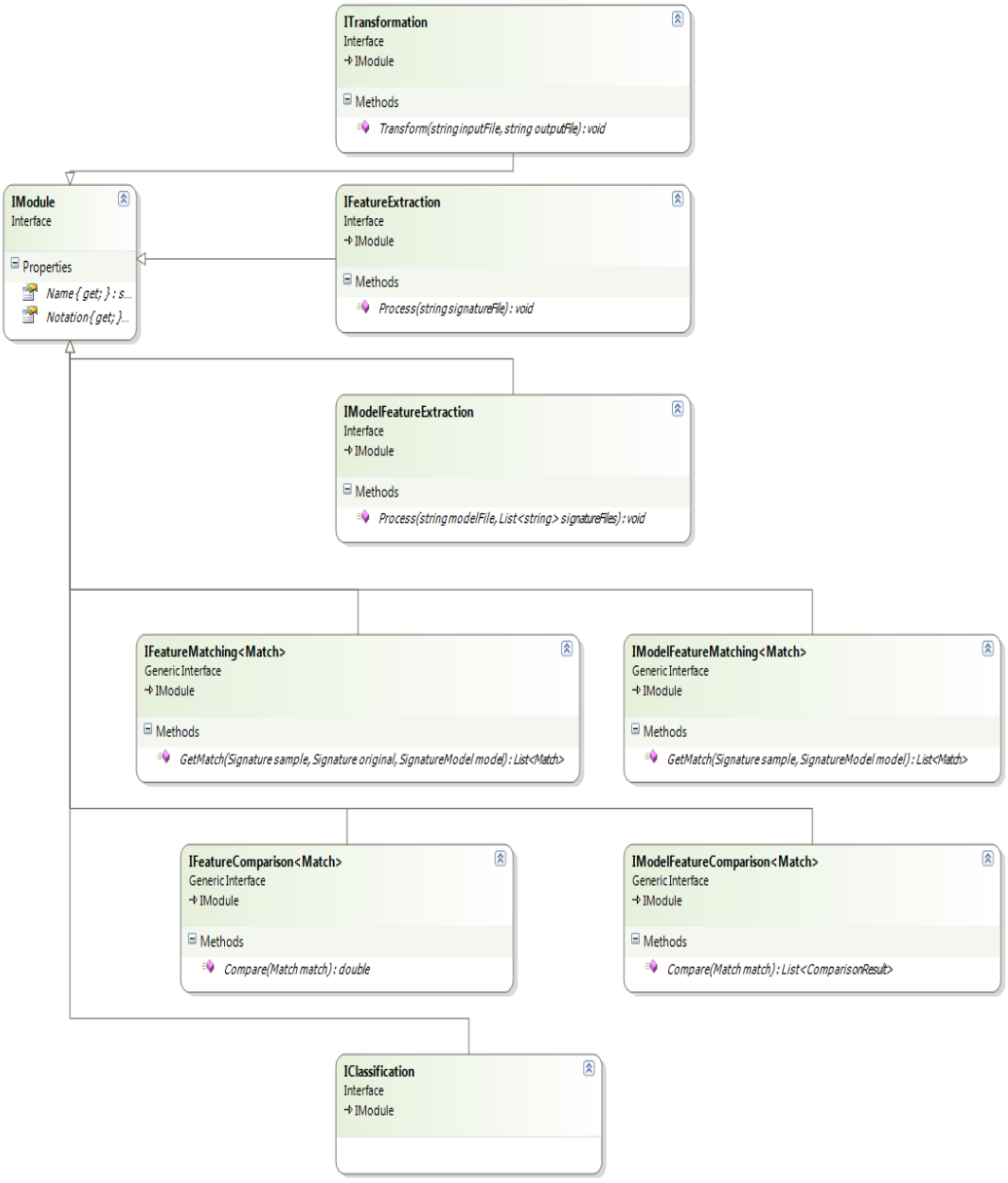
[24] J. Fàbregas and M. Faundez-Zanuy, "Biometric dispersion matcher," *Pattern Recognition*, vol. 41, pp. 3412-3426, 2008.

[25] S. N. Srihari, M. J. Beal, K. Bandi, V. Shah, and P. Krishnamurthy, "A Statistical Model For Writer Verification".

[26] B. Kővári, I. Albert, and H. Charaf, "A General Representation for Modeling and Benchmarking Off-line Signature Verifiers," 2008.

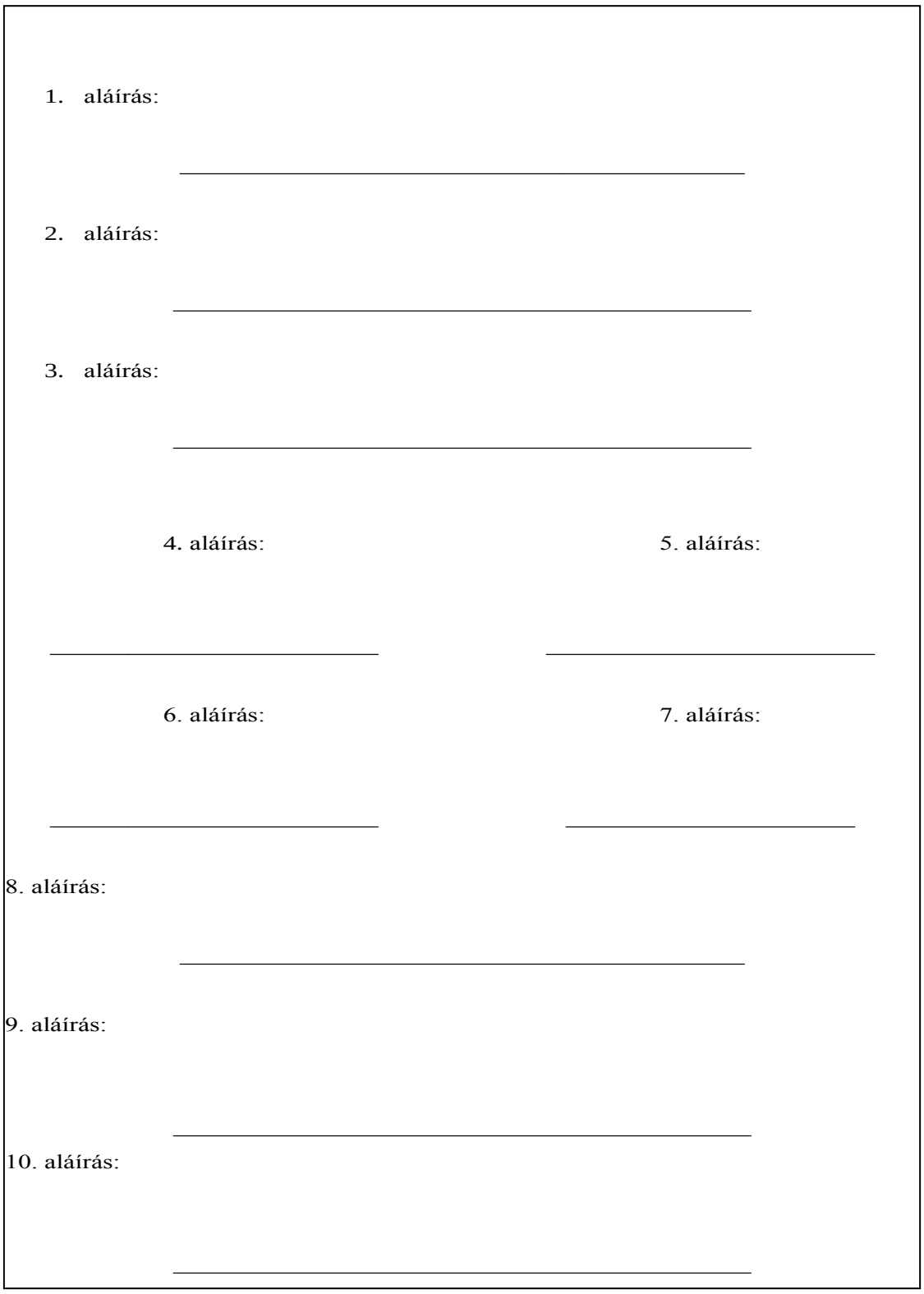
[27] G. Horváth, et al., *Neurális hálózatok és műszaki alkalmazásaik*. Budapest: Műegyetemi kiadó, 1998.

A. Függelék



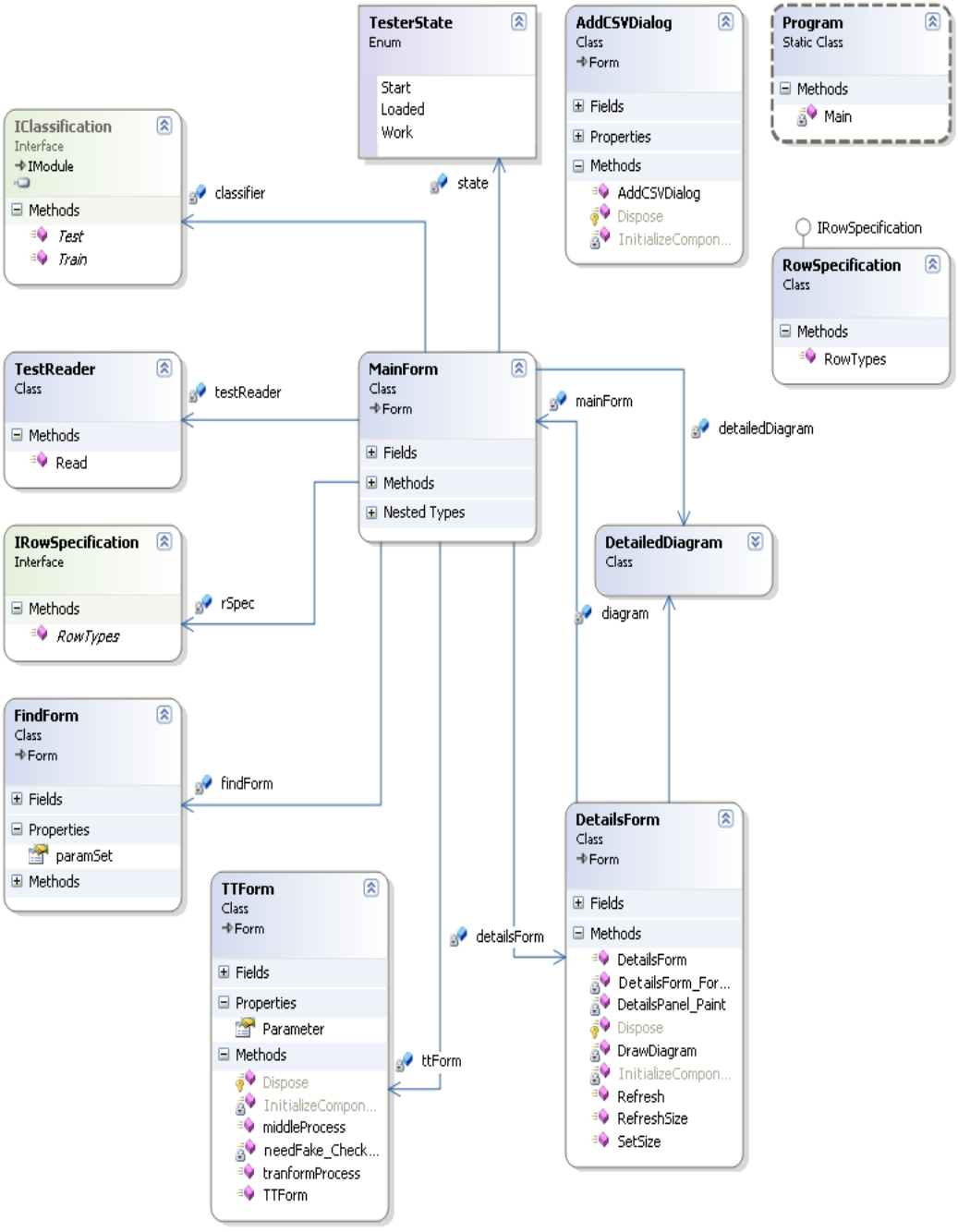
A1 ábra: Az AHR keretrendszer által specifikált interfészek osztálydiagramja

B. Függelék



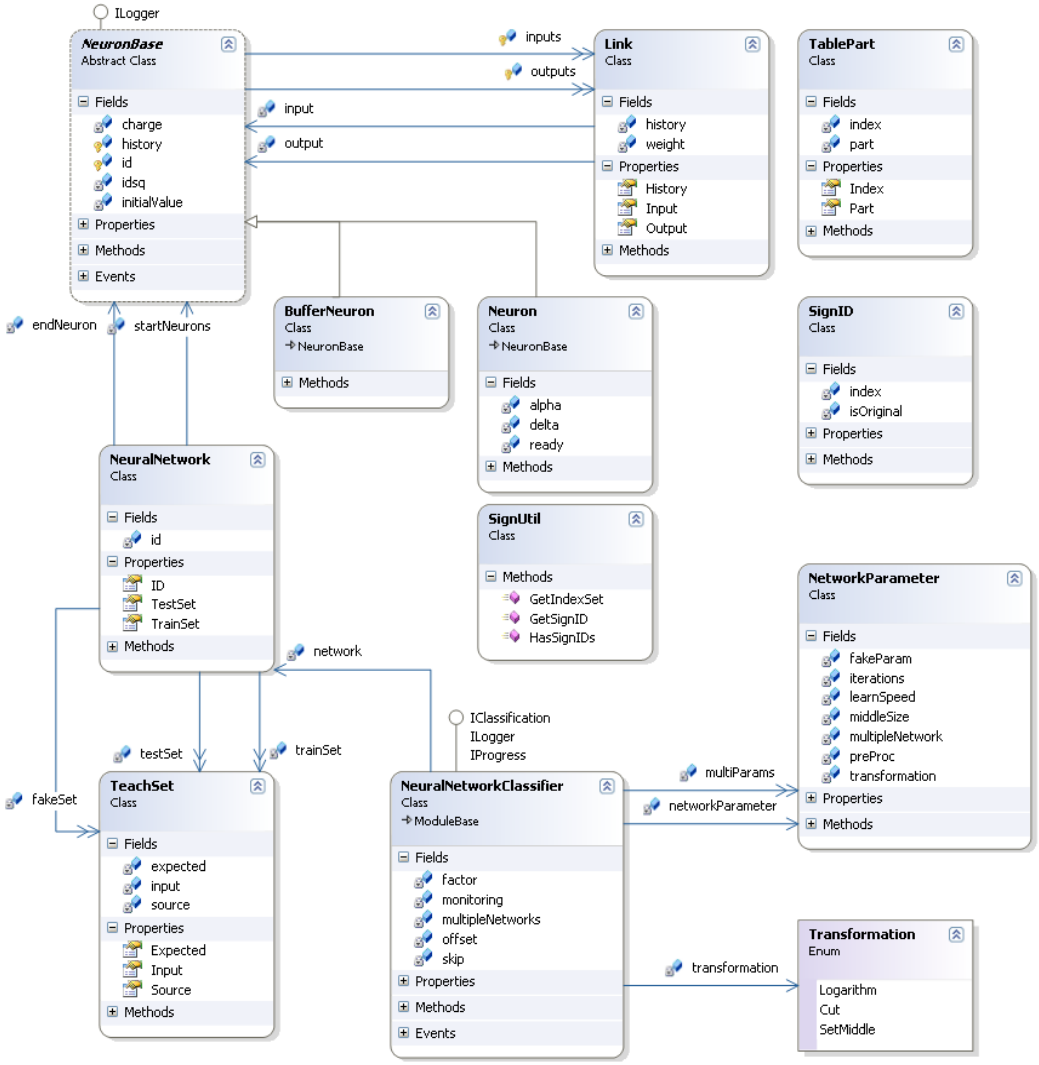
B1 ábra: Az aláírásgyűjtés során használt aláírás ív

C. Függelék



C1 ábra: A tesztelő és monitorozó program osztálydiagramja

D. Függelék



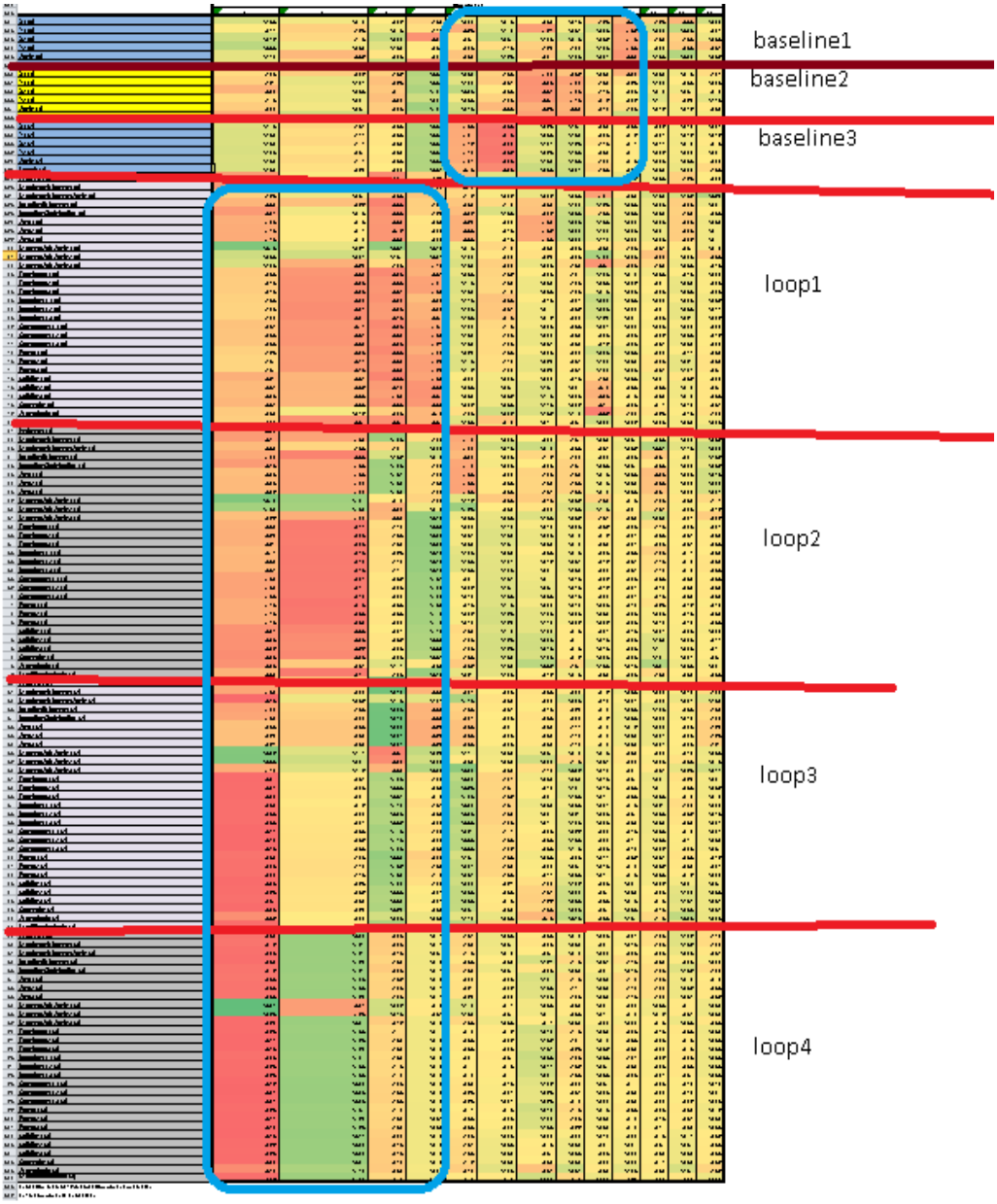
D1 ábra: A neurális háló alapú osztályozó osztálydiagramja

E. Függelék

```
public void Build
(int preProc, int[] middleSize, double learnSpeed, double weight,double initialValue)
{
        // A bemenetek szama
        int dimension = trainSet[0].Input.Length;

        // Bemeneti pufferek letrehozasa
        startNeurons = new NeuronBase[dimension];
        for (int i = 0; i < dimension; i++)
        {
                startNeurons[i] = new BufferNeuron();
        }

        NeuronBase[] lastLayer = startNeurons;

        // Elofeldolgozo reteg letrehozasa
        if (preProc != 0)
        {
                int preProcSize = (dimension - preProc) + 1;
                NeuronBase[] preProcLayer = new NeuronBase[preProcSize];
                for (int i = 0; i < preProcSize; i++)
                {
                        NeuronBase[] previousLayer =
                        startNeurons.Where((n, index) =>
                                ((index >= i) && (index < i + preProc))).ToArray();
                        preProcLayer[i] =
                        new Neuron(initialValue, previousLayer, weight, learnSpeed);
                }

                lastLayer = preProcLayer;
        }

        // Rejtett retegek letrehozasa
        if (middleSize != null)
        {
                foreach (int size in middleSize)
                {
                        NeuronBase[] middleLayer = new NeuronBase[size];

                        for (int i = 0; i < size; i++)
                        {
                                middleLayer[i] =
                                new Neuron(initialValue, lastLayer, weight,
                        learnSpeed);
                        }

                lastLayer = middleLayer;
                }
        }

        // Kimeneti neuron letrehozasa
        endNeuron = new Neuron(initialValue, lastLayer, weight, learnSpeed);
}
```

E1 Kódrészlet: Háló felépítése

F. Függelék



F1 Ábra: Faktroanalízis az SVC20 adatbázison