

TDK – Dolgozat

Forgács Attila

Intézmény: Miskolci Egyetem
Kar: Gépészmérnöki és Informatikai Kar
Tanszék: Alkalmazott Matematikai Tanszék

Szerző: Forgács Attila, programtervező informatikus hallgató, BSc
Témavezető: Dr. Fegyverneki Sándor, egyetemi docens
Dolgozat címe: Egy Markov modell alapú számjegy felismerési módszer

Tartalomjegyzék

1. Bevezetés	1
2. Lényegkiemelés	2
2.1. Vázkijelölés	3
2.2. Kontúr meghatározása	5
2.3. Élkövetés	6
2.4. Megvalósítása	6
3. Markov modellek	8
3.1. Sztochasztikus folyamatok	9
3.2. Markov-láncok	9
3.3. Rejtett Markov-láncok	11
3.3.1. Az első probléma megoldása	12
3.3.2. A második probléma megoldása	13
3.3.3. A harmadik probléma megoldása	14
3.3.4. Vezérlések típusai	16
3.3.5. Kezdőmodell választása	17
3.4. Megvalósítása	19
4. Eredmények	21
4.1. Online módszer	21
4.2. Offline módszerek és eredményeik	21
5. Összefoglalás	24
Irodalomjegyzék	25

1. fejezet

Bevezetés

A tablet PC-k és egyéb, kézírás illetve rajz befogadására képes eszközök terjedésével rugalmasabb kommunikáció valósulhat meg az ember és gép közötti. Egyszerű példa rá az alakzatok rajzolása. Rajzolunk egy kör szerű alakzatot, a gép felismeri, hogy az egy kör, és kiigazítja. Így nem szükséges megadni különösebben semmit egyebet, csak le kell rajzolni. Ennek feltétele az alakzatok kellően hatékony felismerése. Az automatikus olvasó rendszerek statisztikai modellezési technikái erőteljesen támaszkodnak a karakterek tömör jellemzőinek reprezentálására. A jellemzők kinyerése a kézírás-felismerés alapja, rendszerint olyan heurisztikus módszerek, amelyek leírják a karakterek alapvető geometriai tulajdonságait, vagy a karakter pixeleinek az eloszlását.

Markov modelleket sokféle problémára sikeresen alkalmaznak, többek között karakterek felismerésére. Karakter felismerésen belül három esetet különböztetünk meg: nyomtatott, kézzel írt (handprinted), folyóírás. Dolgozatomban a második esettel foglalkozom, számjegyekre specializálva.

Karaktereknek több egyedi jellemzője is lehet, amelyek alapján egymástól jól elkülönülnek. Csoportosítás alapja lehet a leírásuk folyamata. Azonban ezt a többlet információt a szkennelt karakterek már nem tartalmazzák, ezért olyan eljárást kell találnunk amely megfelelő mértékben közelíti. Ez alapján kapunk egy sorozatot, ami zajos. A zaj nélküli sorozatokat közelíthetjük Markov-lánccal, a zajos esetet pedig egy olyan Markov-lánccal, melyet nem tudunk közvetlenül megfigyelni.

A dolgozat első része a jellemzők kinyerésével, karakterek leírásának a folyamatának visszaadásával, második része a rejtett Markov-láncok elméletével, harmadik részben pedig az elért eredményekkel, és az elkészített programmal foglalkozik.

Egyéni eredményeimnek tekintem a kezdőmodell meghatározása szolgáló módszert, független kimenetelű modell módszereinek az átalakítását összefüggő elemű sorozatokra, és a saját fejlesztésű függvénykönyvtárat, ami elsősorban a dolgozatban szereplő rejtett Markov modell alapú képfeldolgozási módszerekhez készült.

A bemutatott kutató munka a TÁMOP-4.2.1.B-10/2/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

2. fejezet

Lényegkiemelés

Karakter felismerő módszereknek két fajtája van online és offline. Első esetben el tudjuk tárolni a karakter leírásának a folyamatát, és még más adatot is mint a toll gyorsulása de hátrányuk, hogy speciális eszközöket igényelnek. Egyik alkalmazása, hogy alakzatokat tudjunk felismerni, szebb ábrák készülhetnek egy megbeszélés közben, el lehet menteni ami a táblára került. Főbb alkalmazási területe az aláírás hitelesítés, mivel pontosabb eredmény érhető el ezekkel az online módszerekkel.

Online esetet csak a módszer működőképességének vizsgálatára használtam. Amennyiben nem érhető el jónak mondható eredmény akkor offline módszerrel se valószínű, hogy jó eredményt sikerül elérni. Ezzel szemben az offline módszerek bonyolultabbak, mivel csak a szkennelt képet használják, első lépésben elő kell állítani egy olyan sorozatot amely közelíti a leírásának a folyamatát.

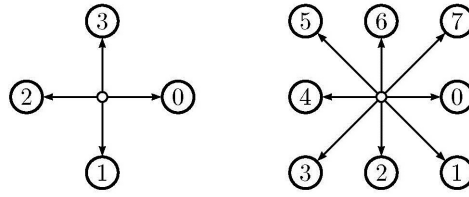
OCR-ek esetében az offline módszerek gyakoribbak, csak egy közülük az amely az írás folyamatát adja vissza, vagy a karakter kontúrját elemzi. Ezek a karakterek különféle jellegzetességeit vizsgálják, melyek lehetnek statisztikai vagy alakbeli tulajdonságok. Az utóbbi módszerek közé tartozik a kontúr összehasonlítása.

Az egyik ilyen a karakterek körvonala, kontúrja, melyet a Google által fejlesztett OCR, a Tesseract is használ [17]. Egy másik egyszerűbb ötlet az, hogy vékonyítsuk le a karaktert egy pixel vastagra, induljunk el egy kezdőpontból a megmaradt pontokon, míg be nem járjuk a karakter vázának pontjait. Az előálló lánc kód egyes esetekben hasonlít mint amikor online módszerrel eltároljuk, de például az is előfordulhat, hogy fordítva adja vissza a lánc kódot, mit ahogy leírtuk volna a karaktert.

Kontúr nem más, mint a karaktert és a háttér elválasztó pontok halmaza. Előállításának egyik alapvető módja, hogy választunk egy kezdő kontúr pontot és abból a szomszéd kontúr pontot összekötő irányt, tovább lépünk erre a szomszéd pontra, addig folytatjuk, míg marad kontúr pont. Felsoroljuk a szomszédját összekötő irányokat, amit lánc kódnak nevezzük.

Kontúrt többféle képen is leírhatjuk, függ egy pont szomszédjának az értelmezésétől is (2.1. ábra). Ha az átlós irányokat nem vesszük figyelembe akkor 4, ellenkező esetben pedig 8 szomszédságról beszélünk. A 8 szomszédság nyilván magában foglalja a 4 szomszédságot is. Két irány kód távolságát pedig egyszerűen a 2.1. táblázat alapján értelmezhetjük. Tartoznak a 4 szomszédság irányai a Σ_4 , a 8 szomszédság irányai pedig a Σ_8 halmazba.

Sorozatot előállítani, nem tudunk tetszőleges képekből, csak olyan bináris képekből, melyeken pontosan egy karakter szerepel. Bináris kép létrehozásának az egyik legegyszerűbb módszere, ha egy fix érték alatt a kép szürkeárnyaltos értékét feketének a



2.1. ábra. Négy és nyolc szomszédság

	→	↘	↓	↙	←	↖	↑	↗
→	0	1	2	3	4	3	2	1
↘	1	0	1	2	3	4	3	2
↓	2	1	0	1	2	3	4	3
↙	3	2	1	0	1	2	3	4
←	4	3	2	1	0	1	2	3
↖	3	4	3	2	1	0	1	2
↑	2	3	4	3	2	1	0	1
↗	1	2	3	4	3	2	1	0

2.1. táblázat. Iránykódok távolságai

fölött pedig fehérnek vesszük:

$$f_K(x, y) = \begin{cases} 0, & \text{ha } x \leq K, \\ 1, & \text{egyébként.} \end{cases}$$

Második feltétel esetén csak a legegyszerűbb módszerrel foglalkozunk, mivel nem egy teljes OCR elkészítése a feladat. A tesztadatokat egy lapra írjuk, táblázatos formában. A táblázatok elemei a számjegyek. Úgy szerepeljenek, hogy a táblázat sorait és oszlopait el lehessen vágni anélkül, hogy számjegyet kettévágtunk volna. Így először vízszintesen kivághatjuk a sorokat, azokon belül pedig függőleges vonalakkal kivághatjuk a tesztadatokat.

2.1. Vázkijelölés

Leírás folyamatának becsléséhez le kell vékonyítani a karaktereket. Alakzatok vázának a meghatározását a Hilditch algoritmussal tehetjük meg. A vázkijelölési vagy élvékonyító módszerek lényege, hogy a görbét határoló pixeleket lépésről lépésre bontjuk le. Három feltételt kell kielégítenie az eredményül kapott görbének:

- képpontnyi vastag legyen,
- eredeti alakzat közepén haladjon,
- az eredetileg összetartozó részeket ne válassza el.

Ilyen, a feltételeket kielégítő görbét úgy kapunk, ha mindig csak a határpontokat távolítjuk el, és akkor állunk meg, ha már nem tudunk pontokat törölni. Jelöljük 1-el az alakzat, 0-val a háttér és -1-el a törlésre jelölt pixeleket. Akkor ítélünk egy p_1 pixelt törlésre, ha teljesülnek az alábbi feltételek:

1. $p_1 = 1$,
2. $2 \leq B(p_1) \leq 6$,
3. $A(p_1) = 1$,
4. $p_2 \cdot p_4 \cdot p_8 = 0$ vagy $A(p_2) \neq 1$,
5. $p_2 \cdot p_4 \cdot p_6 = 0$ vagy $A(p_4) \neq 1$,

ahol $A(p_1)$ a $p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$ sorozatban az fekete-fehér illetve fekete-szürke átmenetek száma. $B(p_1)$ pedig a p_1 pont fekete színű szomszédainak száma. p_1 pont szomszédjait az alábbi módon indexeljük:

p_3	p_2	p_9
p_4	p_1	p_8
p_5	p_6	p_7

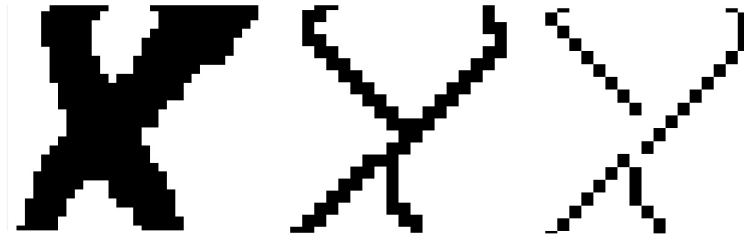
Az első feltétel egyértelmű, hogy törölni lehessen az alakzataból akkor annak még eleme kell hogy legyen.

A második feltételt célszerű szét bontani. $B(p_1) = 0$ akkor nincs szomszédja, p_1 pixel egy zaj, mellyel nem foglalkozunk most, nem az élvékonyítás feladata. $B(p_1) = 1$ akkor nincs szomszédja, végpont, nem kell törölni, ha pedig $B(p_1) > 6$ akkor még nem határpont. Harmadik feltétel az összefüggőség megőrzésére vonatkozik. Az eredeti alakzat összefüggő és $A(p_1) > 1$ teljesül és törölnénk p_1 pontot akkor nem marad összefüggő.

Az utolsó két feltétel azt biztosítja, hogy a 2 pixel széles függőleges illetve vízszintes vonalak ne törlődjenek teljesen.

Az algoritmus véget ér, ha az egyik lépés után nem sikerül módosítunk. A struktúra megőrzése miatt a megjelölt pontokat az egyes lépések után állítjuk át.

Az algoritmus jellegzetessége, hogy olyan görbét kapunk melyen végig lehet menni úgy, hogy nem lépünk átlósan, a bemeneti alakzatokat úgy alakítja át, hogy az eredmény 4 szomszédság szerint legyen összefüggő. Mivel szeretnénk használni átlós irányokat is a későbbi problémák elkerülése végett az eredményül kapott görbét átalakítjuk a 8 szomszédosság szerint legyen folytonos. Egyszerűen áttérhetünk, ha töröljük azon pontokat melyeknek van függőlegesen és vízszintesen szomszédja. Azonban, ha ez alapján térünk át, akkor az eredményül kapott kép nem biztos, hogy teljesen összefüggő lesz. Legrosszabb esetben az eredetileg összefüggő alakzat egyes részei közt egy pixel távolság lesz, például a 8-as két körre szakad, ehhez hasonló probléma több vonal találkozásakor fordul elő (2.2. ábra).



2.2. ábra. Nyolcas, mikor élvékonyítás során a „középén” elszakad

2.2. Kontúr meghatározása

Kontúrt legegyszerűbben élkereséssel határozhatjuk meg. A kontúr nem más mint az eredeti és az élkeresés után kapott képeken az azonos pozícióban lévő fekete pixelek halmaza. Vagyis, azoknak a fekete pixeleknek a halmaza amelyeket az élkeresés nem változtat fehérre.

Az élkeresésési módszerek az $f(x, y)$ kép első, illetve második parciális deriváltját használják. A Laplace-operátort a második parciális deriváltjaiból származtatjuk:

$$L(x, y) = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y}.$$

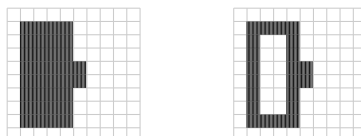
A közelítését általában 3×3 -as, illetve 5 , illetve 9 pontos maszkolással végzik:

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

Élkereső maszkoknak két fő tulajdonsága van: középpontosan szimmetrikusak és elemeinek összege 0. A képen melyeken a megtalált élek szerepelnek úgy kapjuk, hogy a maszk által lefedett képrészlet egyes pixeleinek a szürkeárnyaltos színét megszorozzuk a maszk azonos pozíciójában szereplő értékkel, majd összeadjuk, és a középső elem helyére írjuk. Ezt az összeget maszkválasznak hívják. A kép szélén lévő éleket így nem tudjuk megtalálni. Abban az esetben mikor bináris képekre alkalmazzuk élkeresést csak akkor kapunk 0 maszkválaszt, ha a maszk (nem 0 értékei) által lefedett képrészen csak fekete vagy csak fehér pixelek vannak.

Alkalmazzuk az 5 és 9 pontos Laplace szűrőket. Legyen olyan az élkereső módszerünk, hogy nem 0 maszkválasz esetén feketére egyébként fehérre módosítjuk a pixel színét. Az így kapott képen lesznek olyan fekete pixelek melyek eredetileg fehérek voltak, ezért összehasonlítjuk az eredeti képpel. Csak azokat a pixeleket hagyjuk meg, melyek eredetileg is feketék voltak. Az így kapott él (kontúr) általában egy pixel vastag lesz. Néhány esetben azonban nem, például 2×2 fekete négyzet esetén a 9 pontos Laplace szűrőt alkalmazzuk, vagy mikor egy fekete-fehér részt elválasztó függőleges szakaszra 1 pixel vastag 2 pixel hosszú fekete szakaszt rajzolunk.



2.3. Élkövetés

Az előző két módszer alapján előállított kép pixeleit már csak egy megfelelő kezdőpontból kiindulva be kell járnunk. Első lépésként meg kell határoznunk a végpontokat. Végpontot úgy választjuk ki, hogy megnézzük a P pontoknak az $2n + 1$ távolságú környezetét és megszámláljuk, hogy hány fekete pont van. Ha $n + 1$ -től nem több, akkor azt végpontnak tekintjük. Azért $n + 1$, mert az átló hosszától nem lehet több fekete. Próbálgatás alapján n értéke háromra adódott. Ezután a kép bal alsó pontjához képest növekvő sorrendbe rendezem a végpontokat. A végpont sorozat első eleme legyen $P(x, y)$, mely a legelső részlánckód kezdőpontja.

$P(x, y)$ ponton állunk akkor el próbálunk lépni rendre az 5, 4, 3, 2, 1, 0, 7, 6 iránykódokkal jelölt irányba, amerre először sikerül oda lépünk. Előző tartózkodási helyünk színét megváltoztatjuk, majd folytatjuk, addig míg tovább nem tudunk lépni. Kapunk egy részlánckódot, és a $P(\hat{x}, \hat{y})$ helyen álltunk meg. A részlánckód hosszát megvizsgáljuk, ha túl rövid nem vesszük figyelembe, a végpontok sorozatában a következő elemre lépünk. Amennyiben kellően hosszú akkor megvizsgáljuk van-e más fekete pont a $P(\hat{x}, \hat{y})$ pont N távolságú környezetében, ha van akkor a $P(\hat{x}, \hat{y})$ ponthoz legközelebbit választjuk. Ha nem találunk egyet sem, akkor a következő részlánckód kezdőelemének a kezdőpontot sorozat következő elemét kell választani. Ezen élkövető módszer hátránya, hogy egy csomóponton, két egyenes találkozásánál nem megy át, elkerüli azt.

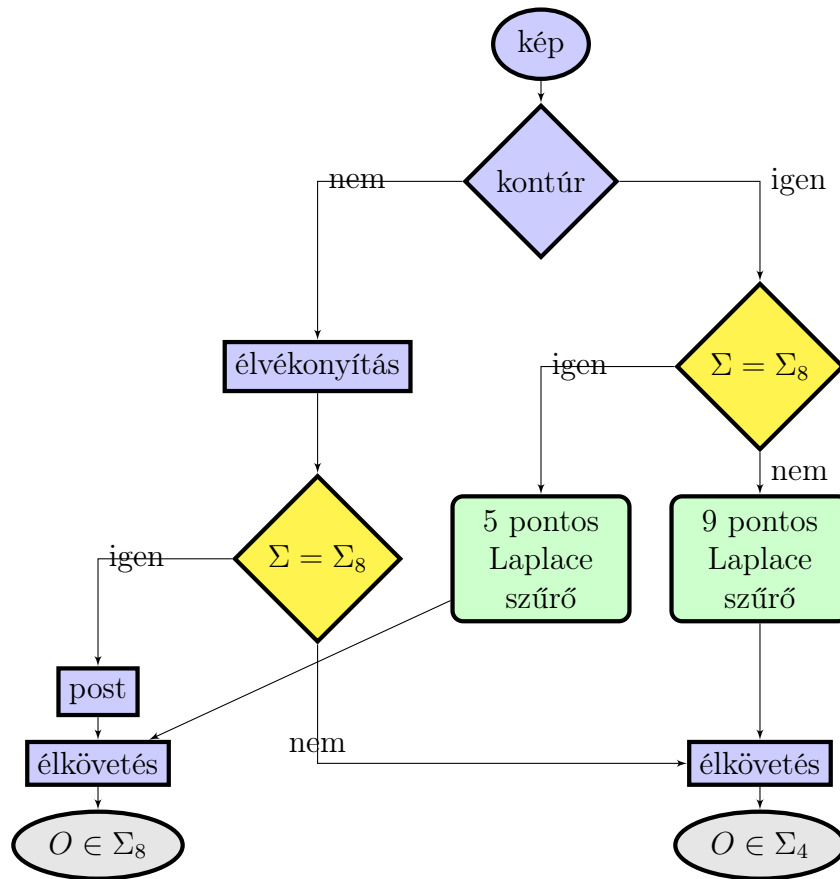
2.4. Megvalósítása

Képekkel való alapvető műveletekre célszerű egy saját adatstruktúrát létrehozni. Külön csomagokba az előfeldolgozási lényeg kiemelési és utófeldolgozási műveleteket. Először végre kell hajtánunk az előfeldolgozási műveleteket. Ide tartozik a több számjegyet tartalmazó karakterek darabolása, és a zajszűrés is.

Lényegkiemelési módszerek képből sorozatot állítanak elő. Először képből pontokat határoznak majd azokat iránykódokká transzformáljuk. A transzformálást ugyanúgy kell megvalósítani függetlenül attól, hogy szomszédságot alkalmazzuk. Pontok előállításának módja viszont az adott módszertől függ. A előállított pontok sorrendjét és a kapott láncódot célszerű megjeleníteni.

Az élkeresési, élvékonyítási feladatok képből képet állítanak elő. Ez a két módszer a képen található számjegyeknek csak a kontúrját illetve csak a vázát hagyják meg. Ezen módszerek által előállított kép tulajdonságától függ, hogy melyik szomszédságot alkalmazzuk rá. Ennek következménye képen az élkövetés a szomszédosságtól függ, kétféle lehet, de a feladat megoldása nem térnek el lényegesen, ezért általánosabb generic függvényeket célszerű alkalmazni az egységes kezelés miatt. Élkövetés estében egy adott pontról kell egy másikra eljutni, csak olyanra léphetünk amely megvan engedve. A megengedett lépéseket a szomszédság által leírt iránykódok adják meg. A szomszédságok fő tulajdonságait külön interface-ben kell megfogalmaznunk. A tényleges szomszédságok pedig megadják az irányokat és a közöttük lévő távolságokat.

2.3. ábra alapján látható, hogy az előzőekben ismertetett módszerek alapján négyféle képen is előállíthatunk sorozatokat. Elsődlegesen a karakterek leírásának követése a cél, melyhez először le kell vékonyítanunk a karaktert. Ez a művelet azonban lassú, vonal vastagságától függ az iterációk száma, mely után még utófeldolgozni is kell a képet, hogy 8 szomszédság szerint legyen folytonos. Ezzel szemben az 5 pontos Laplace szűrő már a kívánt végeredményt adja. Hátránya viszont, hogy nehezebben lehet kezel-



2.3. ábra. Lényegkiemelési módszerek kapcsolata

ni az, mikor a karakter „ketté esik” [17]. Ezen gyakorlati nehézséget kiküszöbölhetjük, ha például töltőtollal írunk a lapra.

Egyik módszer esetében sem normáljuk a különösebben a karaktereket, például nem transzformáljuk $n \times m$ -es képbe, és a geometriai torzulást sem próbáljuk kijavítani. Bináris képpé alakítás nem kifejezetten számít normálásnak, mert általában a karakter felismerő módszerek fekete-fehér képeket vizsgálnak. Normálás elhagyásának előnye, hogy gyorsabban működik, és az, hogyha egy függőleges vonalat húzunk és a fekete pontjait egy $n \times m$ méretű dobozba tesszük akkor könnyebben előfordulnak a hibák, teljesen eltorzul (2.4.ábra). Mivel a képpel nem normáljuk ezért a sorozatok elemzőjének lesz több feladata. Esetünkben több paramétere lesz majd a modellnek, mivel egy sorozatot többször kell eltárolni méretbeli vagy forgatásbeli változások miatt.



2.4. ábra. Normálás egy lehetséges hibája

3. fejezet

Markov modellek

Az iránykódok sorozatának egyik legegyszerűbb összehasonlítási módszere, hogy a sorozatokat nyújtsuk egymáshoz úgy, hogy a lehető legkisebb legyen a távolság a két sorozat között. Ezt az elgondolást valósítja meg a DTW (Dynamic Time Wrapping) algoritmus. Hátránya, hogy konkrét sorozatokat kell tárolni (karakterenként esetleg többet is), és nem a sorozatokra jellemző paramétereket pl.: gyakoriságokat. Utóbbi esetben a sorozatokban a vizsgált gyakoriságok megváltozhatnak, célszerű inkább részsorozatokra bontani és azokban vizsgálni a gyakoriságokat.

Célunk egy olyan modellt létrehozni, ami nem rendelkezik a DTW problémájával, továbbá ne csak a sorozatok különböző részében az ott a szereplő elemek gyakoriságát, hanem még a köztük lévő függőséget is képes legyen figyelembe venni. Ilyen sorozat elemző módszer a Markov-lánc.

Geometriailag hasonló alakzatok esetén az élkövetés után előálló sorozatokat felírhatnánk reguláris kifejezések segítségével. Azonban a karakterek geometriai torzulást szenvedhetnek el, alakjuk megváltozik. Kéztartástól függően globális torzulások is előfordulnak, ránézésre apró eltérések. Ezért a sorozatokra célszerű sztochasztikus nyelvtant meghatározni és ahhoz egy elemzőt készíteni. Az elemző minden sorozatot elő tud állítani adott valószínűséggel, nyilván a modellezendő sorozathoz „közel állókat” nagyobb valószínűséggel. Minden karakter esetén meg kell határozunk egy nyelvet, melynek ábécéje a Σ , az iránykódok halmaza.

Feladat nem más mint egymáshoz hasonló sorozatok egy csoportba sorolása, a sorozat tárolása nélkül, például az alábbihoz hasonlókat:

010101010101111111111110101010101.

Kétféle képen is megtehetjük:

- A sorozat elemeinek relatív gyakoriságával [16].
- Rövidítve (tömörítve) tárolom, 3 állapotban, melyek a (01), (1) és (01) jeleket bocsátják ki.

A második megoldás jobb mint az első, mivel utóbbi modell a sorozatot szűkebb halmazát fogad el, olyanokat melyek csak az egyes szakaszok hosszában tér el. Egyik fő problémája, ha a sorozat elemeit véletlenszerűen megváltoztatjuk akkor nem fogadja el. Másik sorozat, csak 1-2 eleme megváltoztatva:

0100010101011111101111110101011101.

Sorozat tömörítése ismét a feladat, melyet szintén három állapottal valósítunk meg. Most nem egyértelműen egy szimbólumot bocsáthat ki minden egyes állapot, hanem akár az összeset is. Nyilván gyakorlatban (ha a kimenet független) egy szimbólumot fog kibocsátani nagy valószínűséggel, a többit pedig csak kisebbel. Az utóbbit rejtett Markov-láncnak hívják, mivel nem látjuk a sortozatot csak a belőle előálló jeleket. Sorozat előállításánál a rejtett folyamat vezérli a kimenetet, az állapotokban lévő valószínűségek alapján amely állnak elő a kimeneti jelek, így a HMM tekinthető dinamikus rendszernek. Amennyiben véges számú kimeneti jelet bocsáthat ki, akkor másik tulajdonsága az, hogy tekinthetünk rá reguláris sztochasztikus nyelv elemzőjeként is.

A sztochasztikus folyamatok és a reguláris nyelvek kapcsolatának bevezetése után rátérünk a folyamatok paramétereinek becslésére, és a modell kiválasztására.

3.1. Sztochasztikus folyamatok

Sztochasztikus folyamatok elmélete az $\{X_t\}$ valószínűségi változó család struktúrájának vizsgálatával foglalkozik, ahol t egy $\mathcal{T}$ index halmazon végigfutó paraméter. A folyamatok t paramétere általában az idő, mely lehetnek folytonos, illetve diszkrét értékű. Előbbi esetben folytonos időparaméterű sztochasztikus folyamatról beszélünk, utóbbi esetben, pedig diszkrét időparaméterű sztochasztikus folyamatról, ilyenek például az idő sorok. Amíg egy adott valószínűségi változóval kapcsolatos megfigyelés lehetséges kimenetelei egy-egy konkrét értéket jelentenek, addig egy sztochasztikus folyamattal kapcsolatos megfigyelés lehetséges kimenetelei a $t \in [1; T]$ paraméter egy függvényeként jelentkeznek.

A világ folyamatai gyakran bonyolultak, csak nehezen megfogalmazhatóak, leírhatóak. Egy részüket azonban, mikor elegendő a vizsgált időpontot megelőző időpontbeli állapotot ismerni ahhoz, hogy a jelen állapotra helyes következtetést tudjunk levonni, ezek a Markov-folyamatok.

3.2. Markov-láncok

Egy folyamatot Markov-láncnak nevezünk, ha a folyamat diszkrét idejű, múltbeli kimenetele a folyamat jövőbeli alakulására csak a jelen állapoton keresztül gyakorol befolyást. Úgy is lehet mondani, hogy a folyamat múltja és jövője feltételesen független a jelen állapottára nézve:

$$P(q_1, q_2, \dots, q_t, q_{t+1}, \dots, q_{T-1}, q_T | q_t) = P(q_1, q_2, \dots, q_t | q_t) P(q_{t+1}, \dots, q_{T-1}, q_T | q_t). \quad (3.1)$$

Képzeljünk el egy olyan gráfot amelynek pontjai valószínűségi változók és a gráf élei pedig a változók közti függőségi viszonyokat szemléltetik. Ha a gráfnak véges sok pontját eltávolítjuk és ekkor két részre esik akkor a két szétesett rész az eltávolítottokra nézve feltételesen független.

A Markov-láncok definíciója úgy is értelmezhető, hogyha egy Markov-láncnak egy adott időpontban ismerjük az állapotát, akkor ezzel a folyamat jövőbeli sztochasztikus viselkedése egyértelműen meghatározott.

Tekintsünk egy olyan rendszert, ami bármikor leírható elkülönülő N állapot egyikével. Bizonyos idő elteltével a rendszerben állapotváltozás történik (ez lehet visszatérés

ugyanabba az állapotba is), az állapotokhoz tartozó átmenetvalószínűségek szerint. Az állapotváltozásokkal járó idő pillanatokat $t = 1, 2, \dots, T$ -vel, a t időpontban az aktuális állapotot q_t -vel jelöljük.

Tárolunk kell a folyamatnak az úgynevezett egy lépéses állapot átmenetvalószínűségeit, $A \in \mathbb{R}^{N \times N}$ mátrixban, ahol az a_{ij} elem azt fejezi ki, hogy az i -edik állapotból a j -edik állapotba mekkora valószínűséggel jutunk. Nyilván $a_{ij} > 0$ és $\sum_j a_{ij} = 1$. Szükségünk van még egy π eloszlásra ami azt fejezi ki, hogy a folyamat milyen valószínűséggel választja az egyes állapotait kezdőállapotnak.

Az átmenetvalószínűségek és a kezdőállapotok eloszlása meghatározza a folyamat egy adott trajektóriájának az előállításának valószínűségét:

$$P(q_1 = S_{t_1}, \dots, q_T = S_{t_T}) = \pi(S_{t_1}) \prod_{i=1}^{T-1} a_{S_{t_i} S_{t_{i+1}}}. \quad (3.2)$$

Egy adott $q_1, \dots, q_T$ trajektóriából maximum likelihood módszerrel megbecsülhetjük a modell paramétereit. A likelihood függvény (3.2) amelynek a logaritmusa:

$$\begin{aligned} \log P &= \log P(q_1 = S_{t_1}, q_2 = S_{t_2}, \dots, q_T = S_{t_T}) = \\ &= \log \pi(q_1 = S_{t_1}) \sum_{i=1}^{T-1} P(q_{i+1} = S_{t_{i+1}} | q_i = S_{t_i}). \end{aligned} \quad (3.3)$$

Számoljuk össze, hogy az állapot átmenetek $q_t = S_i, q_{t+1} = S_j$ hányszor fordulnak elő, jelöljük J_{ij} -vel. Ez alapján (3.3):

$$\log P = \log \pi(q_1 = S_{t_1}) + \sum_{i,j} J_{ij} \log a_{ij}. \quad (3.4)$$

A (3.4) egyenletet maximalizálni kell azzal a feltétellel, hogy $\forall j$ esetén $\sum_i a_{ij} = 1$. Ezek alapján a Lagrange függvény:

$$\log P - \lambda_1 \left(\sum_j \log a_{1j} - 1 \right) - \lambda_2 \left(\sum_j \log a_{2j} - 1 \right) - \dots - \lambda_N \left(\sum_j \log a_{Nj} - 1 \right) \quad (3.5)$$

Ha a (3.5) függvényt a_{ij} szerint deriváljuk, akkor minden j -re a:

$$\frac{J_{ij}}{a_{ij}} - \lambda_i = 0 \quad (3.6)$$

egyenletet kapjuk, melyből $a_{ij} = \frac{J_{ij}}{\lambda_i}$. A feltétel miatt

$$\sum_j \frac{J_{ij}}{\lambda_i} = 1, \quad (3.7)$$

melyből a Lagrange együtthatók értéke

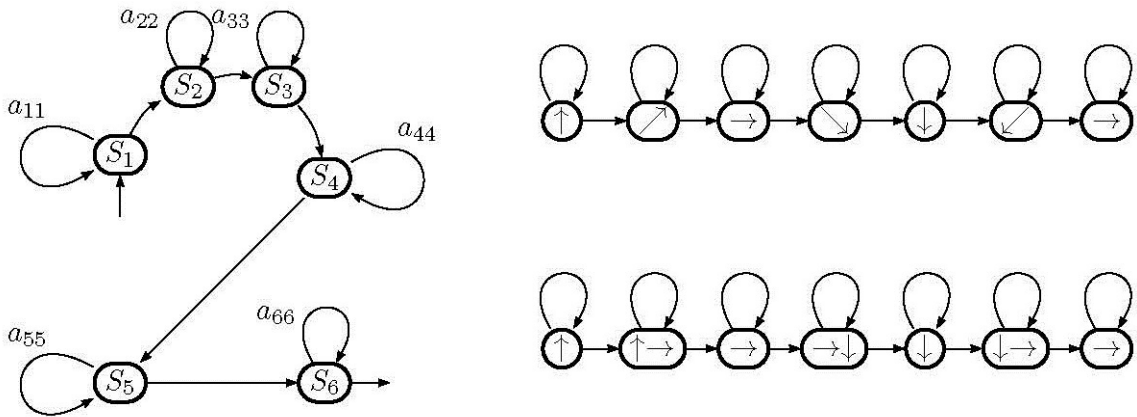
$$\lambda_i = \sum_j J_{ij}. \quad (3.8)$$

Tehát

$$a_{ij} = \frac{J_{ij}}{\sum_j J_{ij}}. \quad (3.9)$$

3.3. Rejtett Markov-láncok

Markov-lánccal csak néhány problémát tudunk jól leírni, ugyanis számos esetben az állapotok közvetlenül nem figyelhetők meg. Ebben az esetben a megfigyelhető jel nem rendelhető hozzá egy konkrét állapothoz, hanem annak valamilyen függvénye. Az így kapott modell két sztochasztikus folyamattal magyarázható, van egy nem megfigyelhető folyamat, illetve egy megfigyelhető folyamat melynek elemei a megfelelő rejtett folyamat állapotaitól függenek. Ezen elvet használja a rejtett Markov modell megértéséhez nézzük meg az alábbi példát. Olyan sorozatot szeretnénk modellezni melynek az egyes részeiben a statisztikai tulajdonságok különböznek, az egyes részek közti „váltás” pontosan nem határozható meg. Az egyes részeket alkalmas tulajdonságú részsorozatokkal közelítjük. Ehhez megadunk egy vezérlő folyamatot úgy, hogy a kimeneti, megfigyelt sorozat részsorozatai olyan tulajdonságú legyen mint a modellezendő sorozat. Legyenek a részsorozatok olyan tulajdonságúak, hogy az egyes iránykódokat nagy valószínűséggel bocsássák ki, többit kisebbel. A modellezendő sorozatban pedig megfigyelhetjük, olyan szakaszokat melyekben csak egy iránykódok fordul elő túlnyomó részt. Ilyen sorozatot kapunk például mikor a számjegyeket vázát 8 szomszédság szerint járjuk be (3.1. ábra). Abban az esetben mikor 4 szomszédság szerint járjuk be a kimenetek között lesz összefüggés, feltételezzük, hogy az egymást követők között van kapcsolat (3.2. ábra).



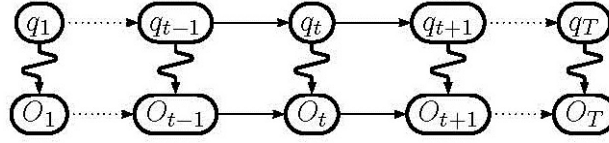
3.1. ábra. Kettes, stacionárius részfolyamatokkal felírva

Ezek alapján a HMM-nek legalább három paramétere van. A vezérlő Markov-lánc paraméterei (π, A) és az egyes állapotokhoz rendelt kimeneti eloszlások $B \in \mathbb{R}^{N \times M}$, ahol M a kimeneti jelek száma, és b_{ij} elem azt jelenti, hogy a j -edik kimeneti jelet az i -edik állapot mekkora valószínűséggel bocsássa ki. Ezt a valószínűséget $b_i(j)$ -vel fogjuk jelölni.

Bevezetünk egy $C \in \mathbb{R}^{N \times M \times M}$ mátrixot is, mely leírja azt a valószínűséget, hogy j -edik állapot által kibocsátott jel k legyen, azzal a feltétellel, hogy az öt megelőző kimeneti jel pedig l volt. Ezt a $P(O_t = k | q_t = S_i, O_{t-1} = j)$ valószínűséget $c_i(k, l)$ -el fogjuk jelölni. C paramétert akkor használjuk, ha a modellezendő sorozat kimenetei között van összefüggés. Ekkor a B paraméterei mást jelentenek, a kibocsátott első jel valószínűségeit adják meg állapotonként. A modell összes paraméterét λ -val jelöljük. Amennyiben a modellezendő sorozat kimenete összefügg $\lambda = \langle \pi, A, B, C \rangle$, ha független akkor $\lambda = \langle \pi, A, B \rangle$.

HMM-ek elméletében alapvetően az alábbi három feladatra keressük a megoldást:

1. Adott egy megfigyelési sorozat és egy HMM akkor a megfigyelt sorozatot mekkora valószínűséggel generálta a modell, mekkora a $P(O|\lambda)$ valószínűség.
2. Adott egy megfigyelési sorozat és egy HMM akkor a megfigyelt sorozatot melyik rejtett állapotsorozat (Q) magyarázza a legjobban, milyen Q sorozat esetén maximális a $P(O, Q|\lambda)$ valószínűség.
3. Hogyan válasszuk ki a modell paramétereit egy adott megfigyelési sorozat esetén, hogy a modell a legjobban magyarázza a megfigyelési sorozatot, mikor maximális a $P(O|\lambda)$



3.2. ábra. Kimeneti értékeiben összefüggő modell

3.3.1. Az első probléma megoldása

Határozzuk meg adott sorozat előállításának valószínűségét adott modell esetén, vagyis a $P(O) = P(O_1, O_2, \dots, O_T|\lambda)$ valószínűséget. Ha ismerjük a az összes állapot esetén és egy t időpillanatban a $\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = S_i|\lambda)$ valószínűségeket, akkor meg tudjuk határozni a $P(O_1, O_2, \dots, O_{t+1}, q_{t+1} = S_j|\lambda)$ valószínűséget ($\alpha_{t+1}(j)$) úgy, hogy az $\alpha_t(i)$ értéket megszorozzuk a_{ij} és $c_j(O_{t+1}, O_t)$ valószínűséggel, mivel az i -edik állapotban lévő a folyamattal ellépünk a j -edik állapotba a_{ij} valószínűséggel, ami kibocsátja a $c_j(O_{t+1}, O_t)$ valószínűséggel a következő jelet. Mivel az bármelyik állapotban lehetünk, ezért állapotonként kiszámoljuk ezeket az értékeket majd összeadjuk:

$$\alpha_{t+1}(j) = \sum_i \alpha_t(i) a_{ij} c_j(O_{t+1}, O_t). \quad (3.10)$$

Amennyiben az $\alpha_T(i)$ értékeket összegezzük i -ben, akkor megkapjuk a keresett $P(O)$ valószínűséget. Az α változók értékeit kiszámító módszert forward algoritmusnak nevezzük.

Számoljuk ki ugyanezt a valószínűséget úgy, hogy előírjuk a t -edik időpillanatban a folyamat az i -edik állapotban legyen, majd ezeket a valószínűségeket összegezzük az i -ben. Ekkor keressük a $P(O_1, \dots, O_T|q_t = S_i)$ értékeket. A 3.2. ábrából látszik, hogy

$$P(O_1, \dots, O_T|q_t = S_i, O_t) = P(O_1, \dots, O_t|q_t = S_i, O_t) P(O_{t+1}, \dots, O_T|q_t = S_i, O_t), \quad (3.11)$$

melyből:

$$\begin{aligned} P(O_1, \dots, O_T, q_t = S_i) &= P(O_1, \dots, O_t, q_t = S_i) P(O_{t+1}, \dots, O_T|q_t = S_i, O_t) \\ &= \alpha_t(i) \beta_t(i). \end{aligned} \quad (3.12)$$

A β változó értékeit hasonlóan az α változó értékeihez ki tudjuk számítani, backward algoritmusnak nevezzük. Ebben az esetben nem a $t + 1$ -től a T -ig tartó részsorozatot valószínűségét számoljuk ki, ha tudjuk hogy a t -edik állapot éppen S_i . A $\beta_T(i)$ változók

értékei 1-el egyeznek meg bármely állapot esetén. Többi állapotra és $t = T - 1, T - 2, \dots, 2, 1$ időpontok esetén az alábbi összefüggéssel határozható meg:

$$\beta_t(i) = \sum_j a_{ij} c_j(O_{t+1}, O_t) \beta_{t+1}(j). \quad (3.13)$$

Így már ki tudjuk számítani a $P(O)$ valószínűséget, és meg tudjuk határozni a $P(q_t = S_i | O)$ valószínűségeket is, melyre a későbbiekben még szükségünk lesz:

$$\gamma_t(i) = P(q_t = S_i | O) = \frac{\alpha_t(i) \beta_t(i)}{\sum_i \alpha_t(i) \beta_t(i)}. \quad (3.14)$$

Későbbiekben azonban még szükségünk lesz a $\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j | O)$ értékekre is. Előzőhöz hasonlóan vezethetjük be, a folyamat két egymást követő időpillanatában írjuk elő, hogy milyen állapotban legyen.

$$\begin{aligned} P(O, q_t = S_i, q_{t+1} = S_j) &= \\ &= P(O_1, O_2 \dots O_t, q_t = S_i, q_{t+1} = S_j) P(O_{t+1}, O_{t+2} \dots O_T | q_t = S_i, q_{t+1} = S_j, O_t) = \\ &= \alpha_t(i) a_{ij} c_j(O_{t+1}, O_t) \beta_{t+1}(j). \end{aligned} \quad (3.15)$$

A keresett ξ érték pedig:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} c_j(O_{t+1}, O_t) \beta_{t+1}(j)}{P(O)}. \quad (3.16)$$

Összevetve a [15] és az általunk bevezetett változókat jelentésük és kiszámítási módjaikat rendkívül hasonlítanak egymáshoz. A független kimenetelű modell algoritmusainak, módszereinek értelemszerű módosításával kapjuk meg az összefüggő kimenetű modell esetére alkalmazhatóakat.

3.3.2. A második probléma megoldása

Célunk megkeresni azt az állapotsorozatot amely, adott modell és kimeneti sorozat esetén a legnagyobb valószínűséggel állíthatta elő a kimeneti sorozatot. Két lehetőségünk van, az egyik a posterior módszer. Úgy állítja elő az állapotsorozatot, hogy az egyes időpillanatban veszi a maximális valószínűségeket, mellyel a folyamat ott tartózkodhatott, vagyis

$$q_t = \arg \max_i (\gamma_t(i)). \quad (3.17)$$

Módszer azonban olyan állapotsorozatot is kaphatunk amelyet a modell elő sem tud állítani. Nem veszi figyelembe hogy két egymást követő állapot átmenetvalószínűség értékei nagyobbak, mint 0.

Másik megoldás a Viterbi-algoritmus, mely a dinamikus programozáson alapszik. A módszer a Bellman-féle optimalitási elven alapul, mely szerint az optimális sorozat bármely része is optimális.

Az optimális sorozatot jelölje $Q_{opt} = q_1, \dots, q_T$ és bármely t_1 és t_2 időpontok közti részsorozata optimális. Ha ez nem lenne így, akkor van ezen két időpont közt egy jobb, $\hat{q}_{t_1}, \dots, \hat{q}_{t_2}$ állapot sorozat, melyre

$$P(\hat{q}_{t_1}, \dots, \hat{q}_{t_2} | O_{t_1}, \dots, O_{t_2}) > P(q_{t_1}, \dots, q_{t_2} | O_{t_1}, \dots, O_{t_2}). \quad (3.18)$$

Viszont ha ezt a „középső” részt kicseréljük, akkor $\hat{Q}_{opt}$ sorozatot kapjuk és $P(Q_{opt}|O) \geq P(\hat{Q}_{opt}|O)$.

Írjuk fel az optimális sorozat előállításának a valószínűségét:

$$P(q_1, \dots, q_{t_1-1} | O_1 \dots O_{t_1-1}) \times \\ a_{q_{t_1-1}, q_{t_1}} c_{q_{t_1}}(O_{t_1}, O_{t_1-1}) P(q_{t_2}, \dots, q_{t_2-1} | O_{t_1}, \dots, O_{t_2-1}) a_{q_{t_2-1}, q_{t_2}} c_{q_{t_2}}(O_{t_2}, O_{t_2-1}) \\ \times P(q_{t_2}, \dots, q_T | O_{t_2+1} \dots O_T).$$

Látható, hogy eltérés csak a középső részben van, amit a $P(q_{t_1}, \dots, q_{t_2} | O_{t_1}, \dots, O_{t_2})$ valószínűséggel jelöltünk. Feltevésünk szerint ettől van jobb. Ez ellentmondással van azzal, hogy a Q sorozat optimális, mivel ekkor $P(\hat{Q}|O) \geq P(Q|O)$ lenne.

Tehát a sorozat első elemétől számítsuk ki az optimális q_1 ; q_1, q_2 ; $q_1, q_2, \dots, q_t$ rész-sorozatokot, míg $q_1, q_2, \dots, q_T$ optimális állapotsorozatot meg nem kapjuk. Szükséges bevezetnünk egy további változót,

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1, q_2, \dots, q_T = S_i, O_1, O_2, \dots, O_t). \quad (3.19)$$

$\delta_t(i)$ a t hosszú állapotsorozathoz tartozó legnagyobb valószínűség, ami figyelembe veszi az első t megfigyelést, és az i -edik állapotban fejeződik be. Induktívan adódik $\delta_{t+1}(j)$:

$$\delta_{t+1}(j) = \max_i (\delta_t(i) a_{ij}) c_j(O_{t+1}, O_t). \quad (3.20)$$

Ha csak a δ értékeket tároljuk akkor csak a $\max_q P(O, Q = q)$ valószínűséget kapjuk meg. Azért, hogy megkapjuk az állapotsorozatot, az argumentum nyomát, amit (3.20) maximalizált, meg kell tartanunk minden t -re és j -re, amit $\psi_t(j)$ -vel jelöljük.

Ezek alapján az α változók kiszámítására szolgáló módszerhez hasonló algoritmust kapunk:

A Viterbi-módszer algoritmus:

Kezdés:

$$\delta_1(i) = \pi_i b_i(o_1) \\ \Psi_1(t) = 0$$

Rekurzió:

$$\delta_{t+1}(j) = \max_{1 \leq i \leq N} (\delta_t(i) a_{i,j}) c_j(o_{t+1}, o_t) \\ \Psi_t(j) = \arg \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{i,j})$$

Befejezés:

$$p^* = \max_{1 \leq i \leq N} (\delta_T(i)) \\ q_T^* = \arg \max_{1 \leq i \leq N} (\delta_T(i))$$

Az útvonal meghatározása:

$$q_t^* = \Psi_{t+1}(q_{t+1}^*)$$

Nyilván az utolsó elemétől is kezdhethetnénk a részsorozatok meghatározni, ekkor a számítás módja a β változók kiszámításához hasonlít.

3.3.3. A harmadik probléma megoldása

A legjobb modellt mely egy adott megfigyelési sorozatot magyaráz nem egyszerű feladat megtalálni. Általában több lehetséges paraméterrel célszerű kipróbálni, és ezekből a

legjobb lesz a végleges modell. Ezeknek a modelleknek nem szükséges tudnunk a pontos értékeit. Elegendő, ha csak a paraméterek nagyságrendjét meg tudjuk határozni, mivel léteznek olyan iteratív módszerek melyekkel pontosítani tudjuk a modellt úgy, hogy az iterációk egyes eredményeiként kapott modellek egyre jobban magyarázzák a megfigyelt sorozatokat. Az egyik ilyen módszer az úgynevezett EM (más néven Baum-Welch) módszer, amely nem más mint egy maximum likelihood módszer hiányos adatokra ([5]). Megpróbálja rekonstruálni a hiányzó adatokat, miközben a paraméterre is egyre jobb becslést ad. Ezeket a törekvéseit egy iterációban két lépésben valósul meg:

1. **E**-lépés (Expectation): az előző iterációban kapott paraméter értékek alapján rekonstruáljuk a hiányzó adatokat feltételes várható érték képzéssel.
2. **M**-lépés (Maximization): a kiegészített adatokból meghatározzuk a λ új paramétert úgy, hogy likelihood függvény minél nagyobb legyen.

Első lépésben a

$$\mathcal{Q}(\hat{\lambda}, \lambda) = E(\log P(Q, O|\hat{\lambda}) | O = o_1 \dots, o_T, \lambda) = \sum_q P(Q|O, \lambda) \log P(Q, O|\hat{\lambda}) \quad (3.21)$$

feltételes várható értéket kell meghatároznunk, ahol $\hat{\lambda} = \langle \pi^*, A^*, B^*, C^* \rangle$ jelöli az új, $\lambda = \langle \pi, A, B, C \rangle$ pedig az előző iterációban eredményül kapott paramétert. Második lépésben pedig meghatározzuk az új paramétert,

$$\hat{\lambda} = \arg \max_{\hat{\lambda}} \mathcal{Q}(\hat{\lambda}, \lambda) \quad (3.22)$$

alapján.

Az, hogy az iterációkban kapott modellek egyre jobban magyarázzák a sorozatot egyszerűen belátható, mivel fennáll az alábbi összefüggés a Q függvény, az entrópia és a modell magyarázóerejének a logaritmusai között:

$$\mathcal{Q}(\hat{\lambda}, \lambda) + H(\lambda, \hat{\lambda}) = \log P(O|\hat{\lambda}). \quad (3.23)$$

Ha ebbe az összefüggésbe az új modell helyére a régít helyettesítjük és kivonjuk a két egyenlőséget egymásból akkor:

$$\mathcal{Q}(\hat{\lambda}, \lambda) - \mathcal{Q}(\lambda, \lambda) + H(\lambda, \hat{\lambda}) - H(\lambda, \lambda) = \log P(O|\hat{\lambda}) - \log P(O|\lambda). \quad (3.24)$$

Mivel az I-divergencia nemnegatív szám, ezért ha $\mathcal{Q}$ függvény értékének növelésével nő a hiányos adat loglikelihood függvény értéke is, az iterációval kapott modellek egyre jobban magyarázzák a sorozatot.

Jelölje $\chi_t^b(j, k)$ indikátor változót, melyre

$$\chi_t^b(j, k) = \begin{cases} 1, & \text{ha } q_t = S_j, \ o_t = k, \\ 0, & \text{egyébként.} \end{cases}$$

Korábban a $\gamma_t(i)$ változót annak a valószínűségként definiáltuk, hogy t időpillanatban a folyamat az S_i állapotban van adott megfigyelési sorozat és modell ismeretében. Ha a $\gamma_t(i)$ -ket a t -ken keresztül szummázzuk, akkor ezt az összeget az S_i állapotba lépések várható számának értelmezhetünk. Hasonlóan a $\sum_t \xi(i, j)$ az S_i állapotból az S_j állapotba

történő átmenetek várható számát határozza meg. Ezen összefüggések felhasználásával megadhatjuk a modell paramétereinek újrabecslésére szolgáló módszert:

$$\begin{aligned}
 \pi^*(i) &= \text{az } S_i \text{ állapot előfordulásának várható gyakorisága ha } t = 1 = \gamma_1(i), \\
 a_{ij}^* &= \frac{S_i \text{ állapotból az } S_j \text{ állapotba való átmenetek várható száma}}{S_i \text{ állapotok várható száma}} \\
 &= \frac{\sum_t \xi_t(i, j)}{\sum_t \gamma_t(i)}, \\
 b_j^*(k) &= j \text{ állapot és a } k \text{ szimbólum megfigyeléseinek várható száma, ha } t = 1 \\
 &= \chi_1^b(j, k) \gamma_1(j), \\
 c_j^*(k, l) &= \frac{j \text{ állapot a } k \text{ szimbólumot bocsássa ki a } k\text{-t az } l \text{ szimbólum előzi meg}}{\text{várható száma, hogy } j\text{-edik állapot a } k \text{ jelet bocsássa ki}} \\
 &= \frac{\sum_t \chi_t^b(j, k) \chi_{t-1}^b(j, l) \gamma_t(j)}{\sum_t \chi_t^b(j, k) \gamma_t(j)}.
 \end{aligned}$$

A másik modell pontosítására alkalmas módszer az optimális állapot sorozaton alapszik. Először meghatározzuk az állapotsorozatot, amelyből meghatározzuk az állapot átmeneteket, és a kezdő állapot meghatározására szolgáló eloszlást, mintha egyszerű Markov-lánc lenne. Ezek után a kimenetet összehasonlítjuk az állapotsorozat elemeivel, megszámloljuk hogy a folyamat a j állapotban hányszor tartózkodik miközben k szimbólumot bocsátja ki, és k -t megelőző szimbólum j , jelöljük ezt az értéket $I(j, k, l)$ -el. Ezeket elosztjuk $\sum_l I(j, k, l)$ -el. A kezdő kimeneti értéket pedig egyszerűen megbecsülhetjük, ha az állapotsorozat első eleme S_i és az általa kibocsátott jel k , akkor a $b^*j(k) = 1$, a többi pedig 0.

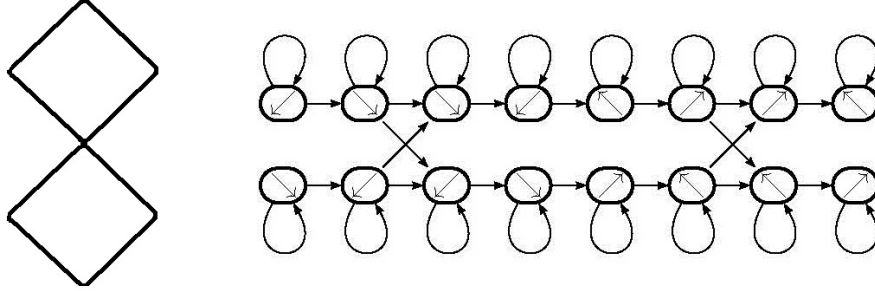
3.3.4. Vezérlések típusai

A legáltalánosabb esetben bármely S_i állapota után következhet bármely másik S_j állapot, azonban számos esetben a lehetséges átmenetek nagymértékben korlátozottak. Irodalomban általában két modell terjedt el, az úgynevezett Bakis modell, melyeknek átmenetvalószínűségei:

$$\begin{bmatrix}
 a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & \dots & 0 \\
 0 & a_{22} & a_{23} & a_{24} & 0 & 0 & \dots & 0 \\
 0 & 0 & a_{33} & a_{34} & a_{35} & 0 & \dots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
 0 & 0 & 0 & 0 & \dots & 0 & a_{n-1,n-1} & a_{n-1,n} \\
 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1
 \end{bmatrix}$$

és a „left-right” modell amely annyiban különbözik hogy az $a_{i,i+2}$ értékek helyén is 0-ák szerepelnek. Elterjedésük azzal magyarázható, hogy egyszerű szerkezetűek, ritka mátrixokkal írható le nullákat tartalmazó elemek értékei pedig a paraméterek pontosítása közben nem változnak meg (következésképpen az egyesek sem), ezáltal gyorsabbak a paraméterbecslési módszerek, illetve a $P(O|\lambda)$ valószínűség kiszámítása is. Esetünkben ez a tulajdonsága hátrány, mivel nem a gyorsaság számít, paraméterek tároláskor és az algoritmusok megvalósításakor nem lett ez a tulajdonsága figyelembe véve. Ez esetben gyorsításra egy módszer lehet, hogy úgy próbáljuk az állapotok számát csökkenteni,

hogy a folyamat egyes részeit egy korábbi hasonló részéhez visszacsatoljuk, 3.3 ábrához hasonlóan. Ennél négyféle sorozat magyarázta a nyolcast legjobban, azok melyek úgy keletkeztek, hogy a sorokból elindulva ugyanabba a sorban fejezzük be. Amennyiben nem lenne rövidítve hanem a „left-right” típusú modellt alkalmaznánk, akkor az állapot átmenetvalószínűségeket 32×32 méretű mátrix írná le, most viszont elegendő egy 16×16 -os.



3.3. ábra. Nyolcas modellje

Ezen modell hátránya, hogy azonos kimenet esetén több olyan sorozatot is viszonylag nagy valószínűséggel generál amelyet az előző kisebbel. Az utóbbit akkor alkalmaztam mikor magam határoztam meg a modell kezdeti paramétereit, az elsőt pedig akkor, mikor automatizálni próbáltam.

3.3.5. Kezdőmodell választása

Kezdőmodell választása esetén több hasonló jó megoldás lehetséges, de jó kezdőmodell meghatározásának problémájára az irodalomban nem igazán sikerül módszert találni, főleg olyat melyet egyszerűen automatizálni lehetne. Feladatra alkalmazunk kisebb megszorításokat a modell vezérlésére és a kimeneti értékeire is. A legegyszerűbb vezérléssel rendelkezők a „left-right” típusú modellek. Ez esetben a modell elemeinek nyilván olyannak kell lennie, hogy a folyamatnak az egyes állapotaiban maradásának valószínűsége nagy legyen.

Feladatból következik az is, hogy az a modell melynek a kimeneti elemei függetlenek szimmetrikus. Normális eloszlás sűrűségfüggvényéből válasszunk 4. ill 8 értéket, hogy szimmetrikus legyen majd, hogy az összegük 1 legyen az elemeket, osszuk el az elemek összeggel. Az összefüggő kimenetű modell esetében másképp kell gondolkodni. A kimeneti ábécé legyen $\{k, l, a, b\}$. Ha a részsorozatra a $(kl)^*$ kifejezés jellemző, akkor az $k \rightarrow l$ és $l \rightarrow k$ átmenetek a leggyakoribbak, azonban a $k \rightarrow k$ és $l \rightarrow l$ átmenetek is jellemzőek. A k -ból vagy l -ből k -től és l -től különböző jel csak igen ritkán következik, azonban ha bármely k -től és l -től különböző jel után k vagy l jel igen nagy valószínűséggel következik. Ebben a modellben is a kimeneti eloszlásokat az alábbiak szerint feltételeztem:

	j	k	a	b
j	q	p	$\frac{1-p-q}{2}$	$\frac{1-p-q}{2}$
k	p	q	$\frac{1-p-q}{2}$	$\frac{1-p-q}{2}$
a	$\frac{p+q}{2}$	$\frac{p+q}{2}$	$\frac{1-p-q}{2}$	$\frac{1-p-q}{2}$
b	$\frac{p+q}{2}$	$\frac{p+q}{2}$	$\frac{1-p-q}{2}$	$\frac{1-p-q}{2}$

Abban az esetben, amikor a részsorozat a $(kk)^*$ kifejezéssel közelíthető, akkor alkalmazhatjuk az előbb megismert módszert ekkor a (k, k) elem helyén lesz egy nagy szám a többi pedig egyforma. A kimeneti eloszlásokat pontosabban is megkaphatjuk, mivel a k után nem biztos, hogy egyforma valószínűséggel következnek. A független kimenet eloszlásainak leírásakor alkalmazott módszerrel meghatározott eloszlást írunk a táblázat k -hoz tartozó sorába. A k -hoz tartozó oszlopába pedig egy tetszőleges $[0, 1]$ intervallumbeli számot. A többi kimeneti érték pedig egyforma.

A lehető legjobb modell előállítása még az egyszerűbbnek tűnő esetekben sem egyértelmű. Ha kevesebb sorozatot használunk akkor pontatlanabb a modell, ha túl kevés akkor szinte használhatatlan. Ha a sorozatok számát növeljük akkor pontosabbá válik, több karaktert helyesen ismer fel, de több paramétert tartalmaz így lassabb. Továbbá adott sorozatokat és vezérlés mellett sem szükséges egyből a legjobb kimeneti értékekkel rendelkező modellt kiválasztanunk, hanem csak egy ahhoz közelít, hiszen a paramétereit lehet pontosítani. Modellek paramétereinek az értéke pedig elegendő, hogyha nagyságrendileg megegyezik azzal ami a tényleges folyamatot magyarázza. Ekkor többnyire hasonló eredményt kapunk a modell pontosítása elvégzése után.

Alapvető feladatunk egy sorozatban az egymás mellett lévő hasonló elemeknek az összevonása. Eredendően két klaszteranalízisből ismert módszerrel próbálkozhatunk agglomeratív és K-közép. Az utóbbi módszerrel nem sikerült egy sorozatot úgy szétbontani, mintha azt ember csinálta volna, túl sok apró csoport maradt. Agglomeratív módszerekkel nem próbáltam meg, mivel lassúnak tűnt a két egymás melletti elem megkeresése, összevonása. Helyette egy talán gyorsabb és talán egyszerűbben megvalósítható módszert próbáltam ki, amely feltételezi a kimeneti paraméterek jellegének ismeretét.

Első lépésként hozzunk létre egy állapotssorozatot, amely „left-right” típusú. Úgy is tekinthetünk rá mint egy sorozat mely megmutassa, hogy mettől-meddig szerepelnek a kimeneti sorozatban homogénnek tekinthető területek. Második lépésben, az állapotssorozatra tekintünk úgy, mintha az lenne az optimális állapotssorozat és mint ahogy megismertük a Viterbi-féle pontosító módszernél számoljuk ki a modell értékeit. Az igazán nagy kérdés az állapotssorozat megtalálása. Ehhez az állapotssorozat meghatározására való Viterbi-módszert használjuk, kissé átalakítva. Nem egy optimális sorozatot keresünk, hanem egy olyan sorozatot mely a kimeneti sorozatot nagy homogén területekre osztja. Az előbbieket alapján a kimeneti valószínűségeket ismerjük, Viterbi-módszerhez kellenek még az állapot átmeneteket. Vezérlésnek nem engedhetünk meg mindent, hiszen ekkor két eltérő kimeneti érték esetén állapotot váltana. Ezért ha el akar lépni bármelyik állapotból akkor büntessük meg, kis számmal szorozzuk meg azt a kimeneti értéket ahova el szeretne lépni. A keresett algoritmus az alábbi:

Kezdés:

$$\delta_1(j) = e_j(o_1)$$

$$\psi_1(t) = 0$$

Rekurzió:

$$\delta_{t+1}(j) = \max_{1 \leq i \leq N} (\delta_t(i)k(i, j))e_j(o_t)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} (\delta_{t-1}(i)k(i, j))$$

Befejezés:

$$q_T^* = \arg \max_{1 \leq i \leq N} (\delta_T(i))$$

A szűrt sorozat meghatározása:

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$

ahol $e_i(O_t)$ azt fejezi ki, hogy az i -edik kimeneti eloszlás mekkora valószínűséggel bocsássa ki a t -edik időpillanatban megfigyelt jelet, $k(i, j)$ pedig az, hogy mennyivel kell büntetni a kibocsátási valószínűséget, ha $i \neq j$ (amennyiben $i = j$ akkor $k(i, j) = 1$). A szűrt sorozatból már egyszerűen meg tudjuk becsülni a kezdő modellt, az állapot átmeneteket, mint a Markov-láncnál. A kimeneti eloszlásokat pedig hozzá kell rendelni az állapotokhoz.

Egy kimeneti sorozat alapján megtudunk határozni egy kezdőmodellt. Azonban nem magyarázza megfelelően az adatbázisunkban lévő számjegyek lánc kódjait, ezért azokat csoportosítanunk kell. A csoportokban a csoport elemeitől egyforma „messze” lévő elemet pedig az előbbi módszerrel meghatározzuk, majd a kezdőmodelleket egye-sítjük.

Két lánc kód távolságának meghatározására a dinamikus programozáson alapuló DTW algoritmust használjuk. Az algoritmus egy L_1 és L_2 , T_1 illetve T_2 hosszúságú sorozatot illeszt egymáshoz. Az illesztés során a $(0, 0)$ kezdőpontból a (T_1, T_2) végpont-ba kell eljutni. (i, j) azt fejezi ki, hogy az L_1 sorozatban az i -edik elemet hozzáillesztet-tük az L_2 sorozat j -edik eleméhez. A hozzárendelés nem egyértelmű. Definiálnunk kell egy előre haladási szabályt, mely azt mutatja meg, hogy egy (i, j) illesztésből milyen illesztés következhet, általában $(i + 1, j)$, $(i, j + 1)$ és $(i + 1, j + 1)$. Feladat az illesztések összegét minimalizálni:

$$D(L_1, L_2) = \min_{i,j} \sum_{i=0}^{T_1} \sum_{j=0}^{T_2} d(i, j). \quad (3.25)$$

A DTW ezzel szemben rekurzívan számolja ki a minimum értékét, algoritmus:

Kezdés:

$$\begin{aligned} D(0, 0) &= 0 \\ D(i, 0) &= D(i - 1, 0) + d(i, 0) \\ D(0, j) &= D(0, j - 1) + d(0, j) \end{aligned}$$

Rekurzió:

$$D(i, j) = \min[D(i, j - 1), D(i - 1, j - 1), D(i - 1, j)] + d(i, j)$$

Végezetül a sorozatok távolsága a $D(T_1, T_2)$ érték.

Határozzunk meg egy d értéket amely felett külön csoportba teszünk két elemet. Létre-hozunk egy csoportot belerakunk egy tetszőleges elemet. Csoportosítás ezek után nem más mint meghatározunk az egyes elemekre, hogy melyik csoportba tartozzon. Abban az esetben mikor egyikhez se tudjuk hozzá tenni új csoportba helyezzük. Csoporthoz miután hozzáadunk egy elemet újraszámoljuk a középső elemet. Egy csoport középső eleme, az az elem amelytől a csoport többi eleme legkevésbé tér el. Ennél a módszernél előfordulhat ránézésre viszonylag különböző lánc kódok is egy csoportba kerülnek.

3.4. Megvalósítása

A három különböző modellre, Markov-lánc, HMM, HMM melynek kimeneti elemei összefüggnek (jelöljük HMM2-vel) célszerű külön adatstruktúrákat létrehozni, melyek-ke a paramétereiket módosíthatjuk, lekérdezhajjuk. Az összefüggő kimenettel rendel-kező modell pontosítása a HMM-nek, technikailag csupán az állapotonkénti kimenet jelátmeneti mátrixokkal van több paramétere. A Markov-lánc specializálása pedig az a modell, melynek a HMM. Olyan struktúrát is célszerű bevezetni, melynek elemei

HMM-ek. Ezeket karakter csoportoknak hívjuk, segítségükkel könnyebb tárolni a modelleket, továbbá visszaadják azt, hogy egy adott megfigyelési sorozatot melyik modell generálta a legnagyobb valószínűséggel.

Nyilván a HMM algoritmusok számára teljesen mindegy, hogy egy sorozat hogyan állt elő (kontúr vagy írás modellezésével), csupán a benne szereplő iránykódokra kell odafigyelnünk, hogy a megfelelő modell kapja a megfelelő sorozatot. Módszerek hasonlítanak egymásra, különösen az EM módszerrel történő pontosítása is, ezeket öröklődéssel oldottam meg, a HMM Baum-Welch algoritmusának kiterjesztése a HMM2-é.

EM módszer lassú, sok iteráció szükséges az optimális paraméterek megtalálásához, ezért a felhasználónak valamilyen visszajelzést kell adni. Megkülönböztetünk olyan feladatokat melyek eléggé számításigényesek, ilyen esetekben egy külön ablakban jelenik meg egy részfeladat eredménye. Paraméterbecslések esetén az a valószínűség melyet az aktuális modell generált. Számítások során rendkívül kis valószínűségekkel kell dolgoznunk, hogy ne legyen olyan skálázni kell az értékeket. Erre a problémára találunk megoldást [14]-ban, az α, β, γ és δ változók valószínűségeinek a logaritmusával számol. Skálázás által azonban rendkívül számításigényessé válik már a forward algoritmus is, ezért csak egy helyen, egy adott rajzolt karakter esetén számítom ki így a $-\log P(O|\lambda)$ értéket. Annak érdekében hogy viszonylag gyorsan kapjuk meg az eredményt a változók skálázását kihagytam.

4. fejezet

Eredmények

Az eddigi fejezetekben egy élkövető módszeren alapuló OCR megvalósításához szükséges ismereteket foglaltuk össze, melyekből összesen négy eljárást lehet létrehozni. Ezen módszerek kipróbálását három nagyobb részre lehet bontani. Az első nem más mint amikor rajzolunk egy számot és a leírásának a folyamatát eltároljuk. Szinte csak a módszer hatékonyságának vizsgálatára jó. Feltételeztem, ha már ezekre a kifogástalan adatokra sem kapunk jó eredményt akkor javítani kell a felismerési módszeren, esetleg mást kell alkalmazni. Második lépésben 8 iránykóddal próbáltam a számjegyeket modellezni, feltételeztem, hogy a kimeneti sorozat elemei között nincsen összefüggés. Harmadik lépésben pedig 4 szomszédsággal közelítettem. A próbálgatások eredményeképpen három kis alkalmazás készült el.

4.1. Online módszer

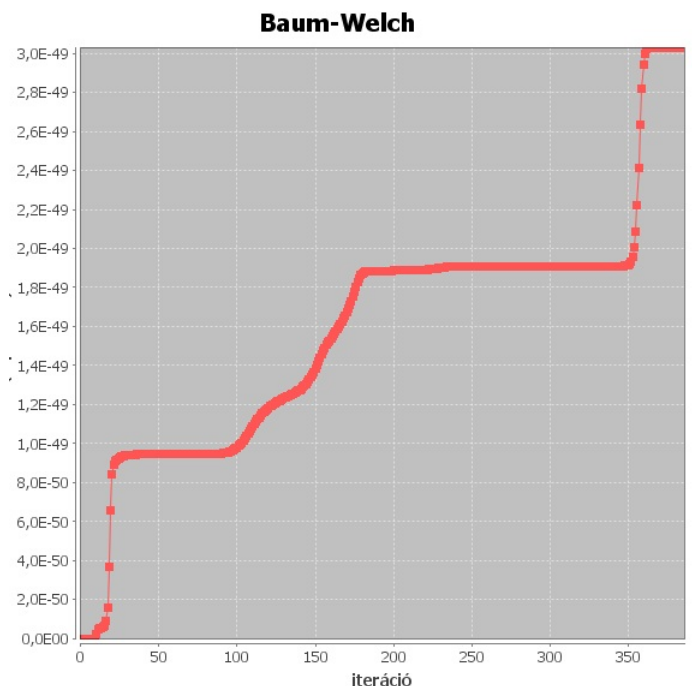
Az online módszerre készített alkalmazás igazán egyszerű felépítésű. Bal oldali képre rajzolunk egy számjegyet, és a **mouseDragged** metódus eltárolja a leírása közben a képernyőn keletkező pontok helyét. Felismerés után pedig az ablak jobb oldalán pedig azokat a valószínűségeket láthatjuk amekkorával a modell generálja az adott pontsorozatot. Lehetőségünk van megvizsgálni, hogy a legnagyobb valószínűséggel modell és a valószínűség és a legvalószínűbb sorozat előállításának a valószínűsége mekkora. Ez általában nem tér el néhány nagyságrenddel többel (maximum 5). Jobb alsó sarokban kiválasztatjuk a pontosítandó karaktert. Először rajzolni kell karaktereket majd elmenteni a lánckódját. Kellően sok adat esetén pontosíthatjuk EM módszerrel. A modellek pontosítása után az összes modellt célszerű lehet elmenteni, a tíz különböző számjegy modellje egy bináris fájlba kerül. A modellből számjegyenként a lehető legjobb előállítása a cél.

4.2. Offline módszerek és eredményeik

Offline esetben a képekből kell vizsgálandó sorozatot előállítani. Első lépésben szkennelt lapon lévő karaktereket kell kiválogatni. A lapokat osszuk két részre, egy 123 és egy 43 elemű mini adatbázisra. Az előző paraméterek pontosítására az utóbbi pedig az eredmények tesztelésére szolgál. Ezután válasszuk ki, hogy kontúr, vagy a leírásának közelítését akarjuk-e. Mindkét esetben a középső képen kapjuk meg, hogy merre mentünk végig a karakterben vagy kontúrján.

Kezdőmodellünkön többféleképpen tudunk pontosítani: EM, Viterbi, posterior és véletlenszerűen választott paraméterekkel. Utóbbi esetben beállíthatjuk, van-e a kimeneti paraméterére ötletünk illetve, hogy hány darab véletlenszerűen megválasztott értékből kapjunk eredményt.

A Viterbi és EM módszereket is össze hasonlíthatjuk, leginkább az iterációk közötti különbségben van eltérés. Jól látható, hogy az EM esetében nagyobb ugrások is előfordulhatnak (4.1. ábra). Az az eset amikor a kimeneti sorozatban két egymást



4.1. ábra. EM és Viterbi-módszer összehasonlítása

követő elem összefügg megjelenésében hasonlít az előzőre, csak kevesebb funkció van megvalósítva.

Konkrét elfogadási eredményeket először arra az esetre készült, mikor írást 8 iránykóddal modelleztem a kezdőmodell pedig magamtól határoztam meg. (4.1.táblázat). Eredmények nem igazán térnek el egymástól, azonban a számítási igényben igazán nagy

tévedések száma		0	1	2	3	4	5	6	7	8	9
EM	amiből pontosítottuk	16	26	27	7	35	22	10	9	3	1
	tesztadatokra	6	6	3	3	11	2	3	6	4	1
Viterbi	amiből pontosítottuk	14	21	0	15	35	10	9	11	7	1
	tesztadatokra	5	6	0	5	10	1	7	11	9	3

4.1. táblázat. Módszerek tévedései

különbség van köztük. A Viterbi-módszer jóval gyorsabb. Abban az esetben amikor a kimenet összefüggő az eredmények hasonlítanak a 4.1. táblázat eredményeihez.

Posterior módszer esetén paraméterbecslés után van olyan számjegy, amelyek esetén egész jó eredményt adott. Hasonlóan magas a felismerés aránya mint a Viterbi vagy EM

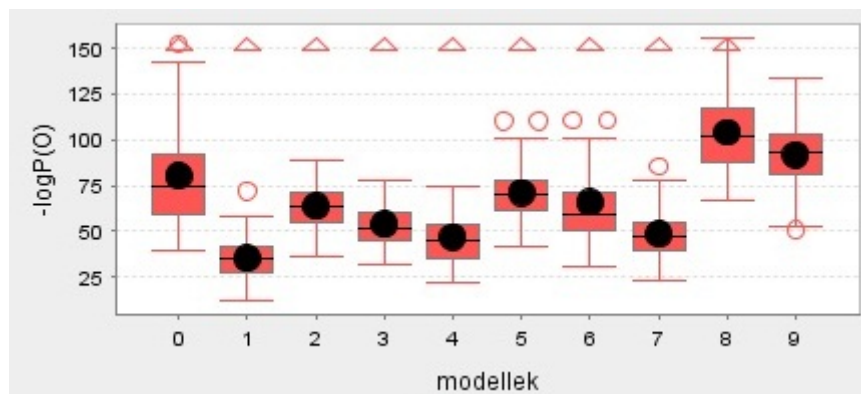
módszernél, viszont egyes számjegyekre teljesen rossz eredményt ad. Például a 4-est az esetek 70%-ában nézi 7-esnek, a 9-est pedig több mint felében 0-ának. A 8-ast és a 0-át fele-fele arányban téveszti el. A többi karakter esetén viszont elég jó eredményeket adott.

Ezzel szemben a modellek állapotátmenet mátrixát véletlenszerűen is megválaszthatjuk. Ekkor természetes több modellt is el kell készítenünk, és azokból összeválogatni azt a tíz modellt, amelyre összességében a legjobb eredményt kapjuk. A 3-as a leggyengébb, azt 66%-ban ismeri fel, de nem teljesen rossz valószínűleg tovább kellett volna próbálkozni. Átlag 86,9%-ban ismeri fel helyesen a karaktereket.

A legjobb eredményt a generált modellek adták, azokra az adatokra, melyekből becsültem a modell paramétereit. Esetükben a legrosszabb eredmény is 85% fölött volt, a modelleket Viterbi-módszerrel pontosítottam. Viszont a tesztadatokra rosszabb eredményt ad, van amikor a 43-ból 10-et is eltéveszt.

A kontúrkövető módszert csak generált modellekre készítettem, ezért ezek is rendkívül jó eredményeket adtak, legrosszabb esetben 123-ból 4-et téveszt el. A tesztadatokra itt is vissza eset a helyes felismerés.

A számjegyek felismerési aránya mellett célszerű megvizsgálni a felismert karakterekhez tartozó valószínűségek eloszlását. Bármelyik módszer esetén a box & whiskers ábrák hasonlítanak (4.2. ábra). A dobozok helyzete a karakterek hosszával hozható kapcsolatba. Az 1-eshez nagyobb valószínűségek tartoznak, rövidebb az iránykód sorozat, a 0-ás, 8-as és 9-es számjegyekhez viszont több elem kell. A generálásának valószínűsége csökken, mivel a $(0, 1)$ intervallumbeli számokat kell összeszoroznunk.



4.2. ábra. A helyesen felismert karakterekhez tartozó valószínűségek

Box & whiskers ábrából ezenkívül meg lehet határozni minden egyes modellre egy konfidenciaintervallumot az elfogadás valószínűségére. Ezt csak az írott karakterekre 8 láncóddal való közelítés esetén, a Viterbi és EM paraméterbecslés eredményére, határoztam meg. Konfidencia szintet úgy igyekeztem meghatározni, hogy a másodfajú hiba minél kisebb legyen, és elfogadható mennyiségű számot azonosítson helyesen. Ekkor a módszer „erejét” is be lehet vezetni. Azt mutassa meg, hogy egy adott számjegynek hiányozna a modellje, akkor tudja-e helyettesíteni azt valamely más modell. Sajnos a 0-ás, 6-os, 8-as és 9-es kivételével legalább három-négy modell is tudja, így nem mondhatni erősnek.

Kontúrkövető és írást visszaadó módszer hasonlóan bizonyult a pici adatbázisomra, így felismerés szempontjából egyforma jó a két megközelítés, ha a kimenetek függetlenek egymástól. Amennyiben összefügg a kimenet ott a kontúrkövető nem próbáltam ki, viszont a másik esetben hasonló eredmények születtek, így azt is célszerű kipróbálni.

5. fejezet

Összefoglalás

Dolgozatomban igyekeztem bemutatni egy rendkívül egyszerű elven működő karakter felismerő módszert. Először bemutattam néhány algorimust, melyekkel képekből sorozatokat lehet készíteni. A dolgozat legnagyobb részét a sorozatok elemzésére használt rejtett Markov-láncok leírása adta. Kétféle modellt kellett használnunk, mivel az esetek egyik részében a kimeneti értékek nem voltak összefüggőek. A modellekkel kapcsolatban négy fő problémát próbáltam megoldani: adott modell esetén egy sorozatot mekkora valószínűséggel állítja elő, a legvalószínűbb vezérlés előállítás, egy sorozat esetén a kezdőmodell meghatározásának és pontosításának módja. Ezekre többé kevésbé sikerült választ adnom. Nem igazán kaptam jó eredményeket, egy OCR-nek a karaktereknek több mint 95%-át fel kell ismernie, hogy hatékornak tekinthető legyen. Ezt azonban kézírás esetén általában nem szokták tudni a programok. OCR-eket mindig tovább lehet fejleszteni, mivel általános megoldás még nem született a problémára. Sorozat elemzésében még ki lehet kötni, hogy egy részsorozat milyen hosszú legyen. További fejlesztést jelenthet egy olyan karakterfelismerő módszer kidolgozása, amely szürkeárnyaltos képeket is kezel. Képekből sorozatokat készítettem, mivel a rejtett Markov-láncok sorozat elemzésre alkalmasak. A két dimenziós általánosítását pedig képfeldolgozásban is gyakran alkalmazzák, karakterfelismerésre is.

Irodalomjegyzék

- [1] Y. Al-Onaizan et. al.: *Statistical machine translation*,
http://www.clsp.jhu.edu/ws99/projects/mt/final_report/mt-final-report.ps.
- [2] L. Baum, T. Petrie: *Statistical inference for probabilistic functions of finite state Markov chains*, Ann. Math. Statist. vol. 37, pp. 1554–1563, 1966.
- [3] C. M. Bishop: *Pattern Recognition and Machine Learning*, Springer, 2006.
- [4] H. Coetzer: *Off-line signature verification*,
dip.sun.ac.za/~herbst/research/PhD_theses/hanno_coetzer_thesis.pdf
- [5] A. P. Dempster; N. M. Laird; D. B. Rubin: *Maximum Likelihood from Incomplete Data via the EM Algorithm*, Journal of the Royal Statistical Society. Series B (Methodological), vol. 39, pp. 1–38, 1977.
- [6] R. Durbin, S. Eddy, A. Krogh, G. Mitchison: *Biological Sequence Analysis*. Cambridge University Press, 1998.
- [7] Fazekas G., Hajdu A. : *Képfeldolgozási módszerek*,
http://www.inf.unideb.hu/~hajdua/km_main.pdf
- [8] G. A. Fink: *Markov Models for Pattern Recognition*, Springer, 2008.
- [9] Gerencsér L. et al.: *Rejtett Markov Modellek I.*,
<http://www.sztaki.hu/~gerencs/download/HMMjegyzet1.ps>
- [10] A. G. Ghuneim: *Contour Tracing*,
http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/index.html
- [11] Gyurkovics É. : *Optimális irányítások*,
<http://tankonyvtar.ttk.bme.hu/pdf/13.pdf>
- [12] B. H. Juang, L. R. Rabiner: *A Probabilistic Distance Measure for Hidden Markov Models*, AT&T Technical Journal, vol. 64, no. 2, pp. 391–408, 1985.
- [13] S. Karlin, H. M. Taylor : *Sztochasztikus folyamatok*, Gondolat Könyvkiadó, Budapest, 1985.
- [14] T. P. Mann: *Numerically Stable Hidden Markov Model Implementation*,
http://bozeman.genome.washington.edu/complib/mbt599_2006/hmm_scaling_revised.pdf.

-
- [15] L. R. Rabiner: *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, In Proc. IEEE, vol. 77, no. 2, pp. 257–285, 1989.
- [16] C. E. Shannon: *A Mathematical Theory of Communication*, The Bell System Technical Journal, vol. 27, pp. 379–425, July, 1948.
- [17] R. Smith: *An Overview of the Tesseract OCR Engine*, 9th Int. Conference on Document Analysis and Recognition (ICDAR), 2007, pp. 629-633.
- [18] Tusnády E. G.: *Sztochasztikus modellek a fehérjekutatásban*
www.enzim.hu/media/uploads/0f6d012b_phd.pdf
- [19] S. V. Vaseghi: *Advanced Digital Signal Processing and Noise Reduction*, Wiley, 2006.