Diplomamunka

Hingyi György

Debrecen

2008

Debreceni Egyetem

Informatika Kar

Párhuzamos feldolgozás alkalmazási lehetőségei a képfeldolgozásban

Témavezető:
Dr. habil. Fazekas Attila
Egyetemi docens

Készítette:
Hingyi György
PTM

Debrecen

2008

Tartalomjegyzék

Tartalomjegyzék....3
Bevezetés....1
Célok....2
Tárgyalás....4
Fogalmak....4
Megközelítések....5
Globális megközelítések....6
Lokális komponens alapú megközelítések....8
Gépi tanulással történő klasszifikáció....9
Neurális hálózatok....9
Tartó vektor gépek (Support vector machines – SVM)....11
A rendszer....15
Az adatbázis....15
Egyedtípusok....16
Image....16
ImageInformation....16
VectorSet....17
Model....17
Pyramid....17
Parameter....18
Filter....18
Az adatbázis feltöltése....18
Képek feltöltése 18
Vektor generálás....21
Piramis építés....23

Teszteredmények és anomáliák....33
Validáció....33
Példa riportok....33
Az alkalmazásprogramozási interfész (API)....40
A detektálás megközelítése....41
Pásztázás....41
Az API....45
Az API elemei....45
Az API rejtett része – Task API....48
A pásztázási feladat, ScanTask....53
Összefoglalás....56
Továbbfejlesztési lehetőségek....57
Irodalomjegyzék....58
Ábrajegyzék....58
Függelék....60
Az adatbázis ORM modellje....60
Az API osztálydiagramja....61

Bevezetés

Szándékosan vagy sem, az emberek közötti érintkezésekben a résztvevők érzelmi állapotának ismerete jelentős szerepet játszik. Az információk megosztásának módja a különféle szituációkban kissé eltérhet attól függően, hogy partnerünk boldog, semleges, meglepett, szomorú vagy éppen dühös. Sokszor vesszük hasznát annak a képességnek, hogy olvassuk partnerünk arci gesztusait – például egy hirtelen meglepődött arckifejezés igazolhatja hipotézisünket partnerünk aktuális ismereteire vonatkozóan, vagy jobban megismerhetjük őt a világ történéseire adott első igazán őszinte reakcióin keresztül – tesszük mindezt legtöbbször tudattalan. Ezt az információforrást kiaknázva következtethetünk arra, hogy a körülöttünk lévő emberek hogyan vélekednek dolgokról, de ennél sokkal operatívabb jelentősége is van: metakommunikációjuk felhasználásával helyesebben értelmezhetjük a verbálisan vagy más módon megosztott információkat. Ha azt olvassuk „Igen, nagyon boldog voltam miatta." anélkül, hogy annak tudatában lennénk, hogy aki ezt közölte velünk valójában szomorú, elfogadnánk a mondat egy nem helyes elsődleges jelentését és nem ismernénk fel a mögötte rejlő szarkazmust. Hasonló ismeretek segíthetnek az emberi humor visszafejtésében [(1)].

Az emberek és gépek közötti érintkezés pontatlan. Az emberek gyakran kezelnek tárgyakat, mint számítógépeket - vagy adott esetben jóval nehezebben megfogható eszközöket - mint szoftvereket - úgy, mintha azok személyiséggel rendelkeznének. Ha valami nem úgy megy, mint ahogyan azt elgondoltuk, adott esetben még dühöt is képesek vagyunk érezni ezekkel a tárgyakkal szemben, noha egyértelmű, hogy az összetettségük tényleges meg nem értéséből fakadó hibánk nyomán kellett a várakozásainktól eltérő eredménnyel szembesülnünk. Természetesen a számítógépeknek és a rajtuk futó szoftvereknek nincs személyiségük – nem szándékosan fagynak le mielőtt elmentenénk munkánkat és nem szándékosan akadályoznak bennünket, mikor sokáig kell várni egy eredményre – csupán tökéletlen eszközök.

Másfelől, egy esztétikus, megfelelő színvilágot felvonultató felhasználói felület megnyugtató és elégedettség érzetét keltheti bennünk. Néhány megfelelő pillanatban bekövetkező

humoros viselkedés (például egy tréfás súgó avatar) oldhatja a stresszt és kellemesebbé teheti a számítógépekkel való munkától idegenkedők életét.

Sajnos a jelenleg használatos felhasználói interfészek – már csak a kezdeti lehetőségekből kiindulva is – a felhasználó teljes mértékű alkalmazkodását követelik meg a felhasznált szoftverhez. A nagyobb számítási kapacitások térnyerésének eredményeképpen azonban egyre inkább a középpontba kerül eme hiányosság megszüntetése. Ma még futurisztikus, de idővel az ember-gép kapcsolatok minősége bizonyosan megközelíti az emberek közti kapcsolatok minőségét, lehetővé téve a kisebb emberi alkalmazkodást igénylő emberi adatbevitelt (pl. szóbeli vezérlés) és javítva az idősek, gyermekek, illetve fogyatékkal élők számítógép használati feltételeit.

Célok

A fejlesztés célja olyan eszközrendszer felépítése, mellyel az ember-gép kapcsolatok minősége javítható azáltal, hogy lehetővé teszi szoftverrendszerek számára felhasználóik aktuális érzelmének becslését a róluk készülő videó folyam alapján.

Az állóképeken jól teljesítő technikák általában nem sikeresek videó folyamokkal változtatások nélkül. Videó folyamokkal való munka eltér, mert:

- Korlátozott idő áll rendelkezésre egy képkocka feldolgozására.
- A képminőség egy videófolyam esetén jóval rosszabb:
    - A horizontális és vertikális felbontás rendszerint jóval kisebb, mint egy fotó esetében.
    - A fotó készítője precízen megválaszthatja a fotó elkészítésének körülményeit, mint nézőpont, megvilágítás és háttér.
    - A megvilágítás és árnyékolás nem rögzíthető, mert mind a környezet, mind az alany elmozdulhat.
    - A videó folyamok tömörítéséből adódóan az effektív felbontás egyik képkockáról a soron következőre az elméletinél is rosszabb és zajosabb.

- Plusz információ nyerhető az egymást követő eredmények együttes elemzéséből.

Jelent projekt egy a Debreceni egyetemen folyó fejlesztés része, melynek távlati célja egy arci gesztusokon (például alapérzelmek kifejezése) és arcról felismerhető tulajdonságokon (nem, kor, etnikum) alapuló ember-gép rendszer kialakítása.

Mindezen célok elérésének kritikus pontja a videó folyamon történő precíz és valós idejű arc lokalizáció, a projekt felvállalt célja is elsősorban ez. A projekt másodlagos célja az SVM alapú globális arc detektálási megközelítés több párhuzamosan futó maggal rendelkező rendszerekhez illesztése, figyelembe véve a hardverfejlesztésnél megfigyelhető trendeket.

A fejlesztés kezdete óta az elérhető legnagyobb hardverteljesítmény 10-12x-esére nőtt, így a kitűzött másodlagos célok súlya csökkent. A grafikus processzorok flexibilisen programozhatóvá válása nyomán is megjelent néhány jó teljesítményű kereskedelmi rendszer. A projekt célja ily módon továbbra is nemes – a szükséges algoritmusok párhuzamosítása – ám téves irányválasztás áldozata. Másfelől, az alkalmazott gépi tanulási eljárás karakterisztikájának kiismerése nyomán szintén kérdőjelek merültek fel az intuíciós alapon kezdetben jónak tűnő megközelítésről. A technológia következetes és remek potenciált tartogat számos felhasználási területen, azonban néhány később kifejtett tulajdonsága miatt a detektálási probléma megoldására csak speciális esetekben alkalmas megnyugtatóan. Ugyanezen tulajdonságai miatt azonban kiválóan alkalmasnak gondolom az arci gesztusok, illetve arcról felismerhető tulajdonságok precíz osztályozására.

Tárgyalás

Fogalmak

Az alábbiakban tegyünk egy kísérletet a rendszer ismertetése során használatos fogalmak egy megfelelően pontos definiálására. A pontosabb megfogalmazás megköveteli az arcnak és képeinek megkülönböztetését, mivel egy alanyról több különböző kép is rendelkezésünkre állhat.

Arc lokalizáció: adott számú arc képének keresése egy bemeneti képen. Egy arc akkor minősül lokalizáltnak, ha az arc képének helye és mérete ismert.

Arcdetektálás: az arc lokalizáció olyan általánosított esete, mikor a megtalálandó arcok száma nem ismert.

Arckövetés: lokalizált arcok közötti megfeleltetés folyamata a videó folyam két egymás követő képkockáján.

Objektum-osztályozás: azon folyamat, melynek során adott halmazban található objektumok közös tulajdonságaik alapján diszjunkt részhalmazokba sorolódnak.

Bináris objektum-osztályozás: az objektum osztály meghatározás olyan speciális esete, mikor egy adott halmazban található objektumok pontosan két diszjunkt részhalmazba sorolhatóak aszerint, hogy rendelkeznek adott tulajdonságokkal vagy sem.

Adatbázis: különféle osztályba tartozó objektumokról készült képek gyűjteménye, amelyeket felhasználunk a gépi tanító halmaz előállítása során. Bináris objektum-osztályozás esetén beszélhetünk pozitív példa képekről, negatív példa képekről és validációs képekről.

- *Pozitív kép*: olyan kép, melyhez tartozó objektum a lokalizáció tárgyát képezheti.
- *Negatív kép*: olyan kép, melyhez tartozó objektum(ok) nem képezik a lokalizáció tárgyát.

- *Validációs kép*: olyan kép, melyhez további információként adott annak pozitív illetve negatív volta, de nem eleme a gépi tanulásnál felhasznált tanító halmaznak.

Model: az adatbázis része; a gépi tanulás során felhasznált adat halmazra, az úgynevezett tanító halmazra utal.

Gépi tanulás: a mesterséges intelligencia azon részterülete, amely olyan technikák tervezésével és fejlesztésével foglalkozik, amelyek lehetővé teszik a „tanulást" számítógépek számára. Induktív gépi tanulási módszereknek nevezzük azokat, amelyek nagyméretű adathalmazokból autonóm módon vonják ki a meghatározó szabályokat és mintázatokat számítási és statisztikai módszerek felhasználásával. [(2)]

Gép kapacitása: Egy gép kapacitása alatt azon modell méretét értjük, melyet az még éppen képes hibás osztályozás nélkül elsajátítani.

Jellemző vektor (feature vector): egy olyan vektor, melynek minden komponense egy objektum jellemzőt reprezentál egy kötött tartományból hozzárendelt skaláris értékkel. (A gépi tanulás függvényközelítés alapú definíciójából eredeteztethető a fogalom vektorként való értelmezése.)

Jellemző kiemelés (feature exctraction): a fontos tulajdonságok (jellemzők) lényegtelenektől (zaj) való elkülönítésének, illetve ezen információk *jellemző vektorrá* transzformálásának folyamata.

Osztályozás (classification): az a döntési folyamat, melynek során egy jellemző vektorról eldöntjük, hogy milyen osztályba tartozó objektumot ír le.

Megközelítések

Attól függetlenül milyen megközelítést alkalmazunk, az arcfelismerés számításigényes probléma. Nincs univerzálisan tökéletes megközelítés – a hatékonyság nagyban függ attól, hogy mi a felismerés tárgya és milyen a környezete. Minél kevésbé pontosan definiálhatóak a felismerés tárgyának tulajdonságai környezetéhez képest, annál kevésbé jók felismerés várható eredményei.

A különféle megközelítések rendszerint abban térnek el, hogy másként definiálják a felismerés tárgyának tulajdonságait. Az eltérés adódhat abból, hogy mely tulajdonságokat definiáljuk mérvadónak, vagy abból, hogy a tulajdonságokat milyen tartományokban ábrázoljuk.

A különféle megközelítések közös vonása, hogy elkülöníthető a detektálás két fázisa: *jellemző kiemelés* (*feature extraction*) és *osztályozás* (*classification*).

Az arcfelismerés területén alkalmazott technikák egy lehetséges csoportosítása történhet a *jellemző vektor* alapján.

- globális
- lokális komponens alapú

Más csoportosítás alapját az osztályozás technikája jelentheti. Ez alapján a két legnagyobb csoport:

- gépi tanulás (illetve statisztikai tanulás) alapú (SVM, neurális hálók)
- megerősítéses tanulás és dimenzió redukció alapú (Turk és Pentland féle Eugenface módszer [4], Fisher féle diszkrimináns analízis [5], stb.)

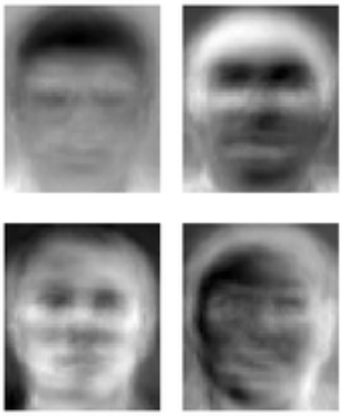
Globális megközelítések

Globális megközelítések esetén az objektum teljes, osztatlan képéből alkotott vektor adja a *jellemző vektort* a *jellemző kiemelést* követően. A *jellemző kiemelés* itt különféle normalizációkat, zajszűréseket, illetve színtér transzformációkat jelenthet – de a jellemző vektor mindenképpen a kiinduló kép egészéhez köthető.

A projektben alkalmazott gépi tanulási eljárásban használt *jellemző vektorok* lent látható szürkeskálás ábrázolása nyilvánvalóan összekapcsolható az eredeti példával.

1 Nagyfelbontású, jellemző kiemelésen már átesett pozitív példakép

Az alábbi ábrán viszont az Eugenface technika által használatos mátrixok láthatóak szürkeárnyalatos képként. Ezek tulajdonképpen a fenti képből előálló *jellemző vektorok* további feldolgozásának eredményei, a *jellemző vektorok* kovariancia mátrixának sajátvektoraiból összeálló mátrixok. Érdekes, hogy az arcok kivehetőek.



2 Eugenface képek az AT&T kutató intézetéből

A globális technikákon alapuló megoldások jól teljesíthetnek adott szögből megfigyelt, takarásmentes vagy alig takart arcok keresése esetén, de érzékenyek a helyzet transzformációkra, illetve megvilágítási körülmények változásaira.

A tisztán osztályozás célzatú globális technika felhasználások esetén – mint az arci gesztusok, illetve arcról felismerhető tulajdonságok felismerése - a helyzetváltoztatások hatása némileg csökkenthető a bemeneti képek egy referencia képhez való illesztésével. Az

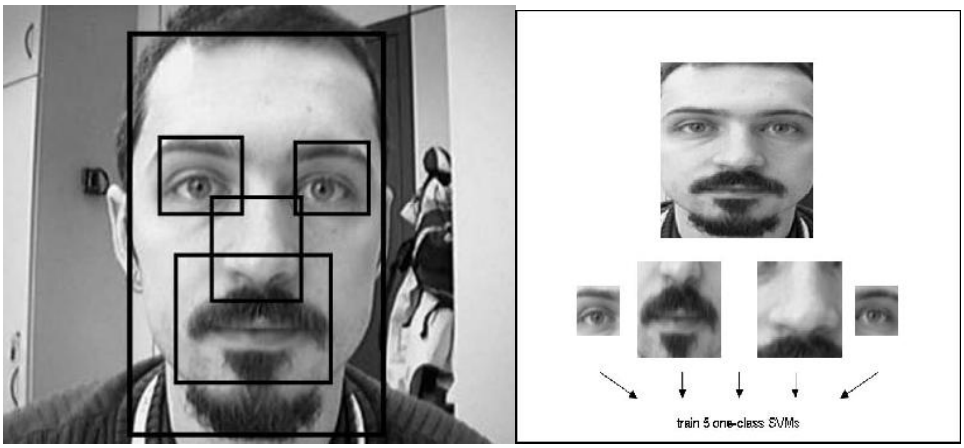
illesztés alapját képezhetik rész-jellemzők, mint a szemek középpontja, a száj illetve az orr; ezek pozícióját, méretét megvizsgálva a bemeneti és a referencia képen, meghatározható az illesztéshez szükséges transzformáció [3] – ezzel azonban már egy másik megközelítési család felé kalandozunk.

Lokális komponens alapú megközelítések

A komponens alapú módszerek az arc részeinek/komponenseinek függetlenül vagy függően történő detektálásán alapulnak (aszerint, hogy korábbi sikeres komponens detektálások információját felhasználják-e).

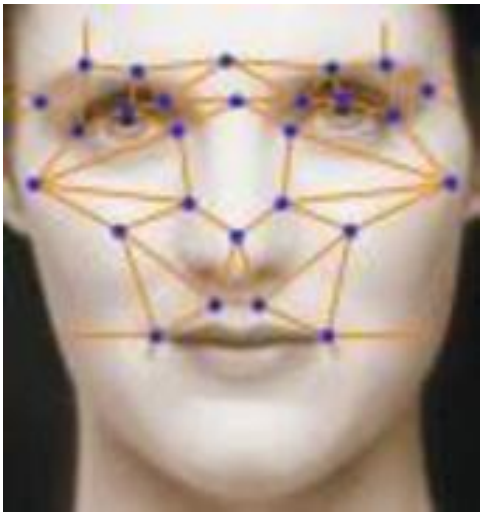
A *jellemző kiemelés* folyamata komponens alapú módszerek esetén a komponensek globális módszerekkel történő detektálását és a kapott találatok csoportosítását jelenti. Az előállított *jellemző vektor* elemei a komponensek egymáshoz képest tekintett helyükre/helyzetükre vonatkozó információi lesznek valamilyen reprezentációval.

Az osztályozás folyamatában a globális módszerekhez hasonlóan használatosak gépi tanulási eszközök – neurális hálók illetve tartó vektor gépek egyaránt elterjedtek.



3 Paolo Abeni a Telecom Italia-tól idealizált kísérletében

Az eredmény megjeleníthető síkban ábrázolt gráfokkal, ahol a komponensek egy vagy több kontroll ponttal vannak reprezentálva, a jelentőséggel bíró távolságok pedig közöttük húzott élekként ábrázolhatóak.



4 Komponensek között definiált gráf

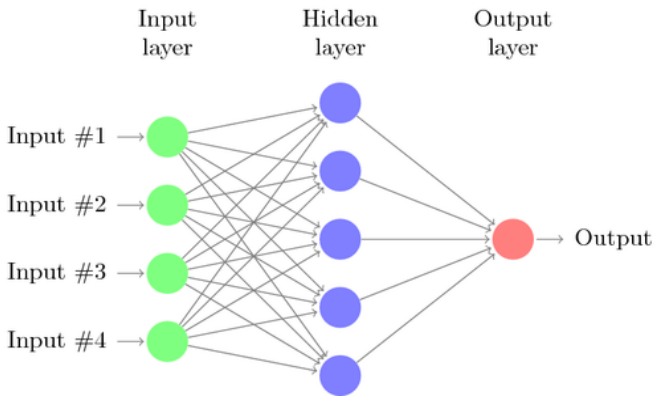
Ez a megközelítés lehetővé teszi a helyzetváltozásokra való nagyobb fokú érzéketlenség elérését, a komponensek egymáshoz képesti helye/helyzete jól kezelhető. Szinte kizárólagosan komponens alapú eljárások (angol nevén visual feature based approaches) használatosak alakzatok képből történő háromdimenziós modelljének előállításának területén.

Gépi tanulással történő klasszifikáció

Az diplomamunka tárgyát képező projekt SVM alapú globális módszert használ fel, ám kidolgozott megoldások zöme a megközelítéstől független.

Neurális hálózatok

A mesterséges neurális háló a biológiai neuronok és szinapszisok hálózatának modellje – a neuronokat függvények, a szinapszisokat részkifejezések (vagy másképpen, csúcsok és összeköttetések) reprezentálják. A neurális hálózatok rétegekre oszthatóak a neuronok közti összeköttetések szerkezetéből eredeztethetően: egy input rétegből, egy output rétegből és egy vagy több rejtett rétegből állhatnak.



5 Egy egyszerű neurális hálózat

Egy mesterséges neurális hálózat tanítása az összeköttetésekhez rendelt súlyértékek finomhangolását jelenti, amelyet a részkífejezésekhez rendelt együtthatóként lehet értelmezni. A súlyértékek olyan hangolása a cél, melyben adott számú neuron és összeköttetés, illetve bizonyos szerkezet mellett a neurális hálózat output rétegén annyi különböző input réteg konfiguráció esetén jelenik meg helyes kimenet, amennyi esetén csak lehetséges. Minél több neuronból és összetettebb összeköttetésekből áll egy neurális hálózat, annál nehezebb és lassabb a tanítási folyamat, ám a gép kapacitása növekedhet.

Arcfelismerés esetén pozitív és negatív képekből generált *jellemző vektorokat* helyezünk a bemenetre (egy komponens egy neuron, tehát összesen vízszintes felbontás*függőleges felbontás darab input réteg neuronra van szükség) és „arc” illetve „nem arc” jelentéseket reprezentáló értékeket (úgynevezett címkéket) a kimenetre. Az legelterjedtebb back-propagation algoritmus család az input réteg felé haladva a súlyok apró lépésekben történő módosításával törekszik a helyes címke értéket eredményül szolgáltató súlyértékek kialakítására.

Az osztályozás során a kérdéses *jellemző vektort* a tanítási folyamattal megegyező módon a bemenetre helyezünk és kiértékelés eredményét jelentő kimenetet vizsgálva döntünk a kapott címke érték alapján.

A neurális hálózat sokoldalú, induktív gépi tanulási eszköz, széles körben alkalmazott szakértői rendszerek, minta felismerés és az adatbányászat területén – de gyakorlatilag bárhol, ahol függvényközelítési vagy osztályozási probléma előfordulhat.

Tartó vektor gépek (Support vector machines – SVM)

Az SVM atyja V. N. Vapnik aki már 1979 környékén letette alapjait, de csak 1992-re készítette el - szintén Vapnik - az első ilyen elven működő rendszert. A 90-es évek végén került a téma középpontba, az utóbbi években számos ponton finomítottak rajta. Manapság az SVM egy teljesen új és független kutatási területté nőtte ki magát.

A tanítóhalmaz *jellemző vektorokból* és hozzájuk rendelt címkékből áll, azaz $\left(\underline{x_i}, y_i \in \{1, -1\}\right)$. Mivel minden vektort pontosan ugyanannyi, rendre azonos tartománnyal meghatározott komponens ad meg, ezeket másként ugyanazon n dimenziós vektortér különböző vektorainak tekinthetjük.

Az SVM tanulási folyamata az azonosan felcímkézett *jellemző vektorok* csoportjai közötti olyan irányított hipersíkok (úgynevezett döntési síkok, vagy szeparáló síkok) megkeresését jelenti, melyek esetében azok „margója" maximális.

A margó a döntési síkkal párhuzamos (tehát egyező normálvektorú), annak két átellenes oldalán lévő azonos irányultságú hipersík közti távolságra utal, melyek a következőképpen adhatóak meg:

$$H_1: \left(\underline{x_i} \cdot \underline{w}\right) + b = 1$$, ahol x_i-hez y_i=1

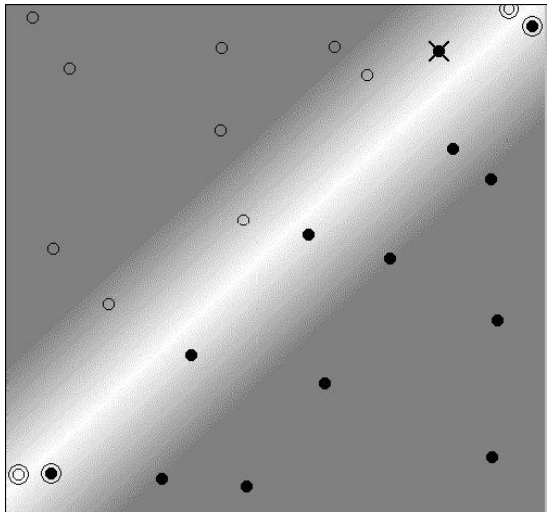
$$H_2: \left(\underline{x_i} \cdot \underline{w}\right) + b = -1$$, ahol x_i-hez y_i=-1

H_1 távolsága $\frac{|1-b|}{\|\underline{w}\|}$, míg H_2 távolsága $\frac{|-1-b|}{\|\underline{w}\|}$, amiből a margó szélességnek $\frac{2}{\|\underline{w}\|}$ adódik. Ezen H_1 és H_2 síkok közül az optimális az, amelyre $\|\underline{w}\|^2$ minimális lesz.

Tartó vektornak a vektortér azon $\underline{x_i}$ vektorait nevezzük, melyek eltávolítása a modellből megváltoztatná a szeparáló hipersíkok (tehát margók) bármelyikét. A tartó vektorok rajta vannak a margón. A margót közrefogó irányított hipersíkok között, vagy a címkéjének nem megfelelő oldalán nem fordulhat elő vektor szeparálható esetekben.

Nem lineáris szeparáció (vagy szemléletesen hiperfelülettel határolás) esetén a vektortér elemeit egy alkalmas nem lineáris leképezéssel másik Euklideszi térbe képezzük és a szeparáló hipersík keresését ott végezzük el.

Az alábbi ábrák az SVM szeparálásának feltételezett eseteit mutatják lineáris illetve polinomiális kernelfüggvény esetén. A pontok színe a hozzájuk rendelt címkékre utal. A bekarikázott pontok illusztrálják az SVM tartóvektorait, míg az „X" egy hibát okozó vektort jelez.



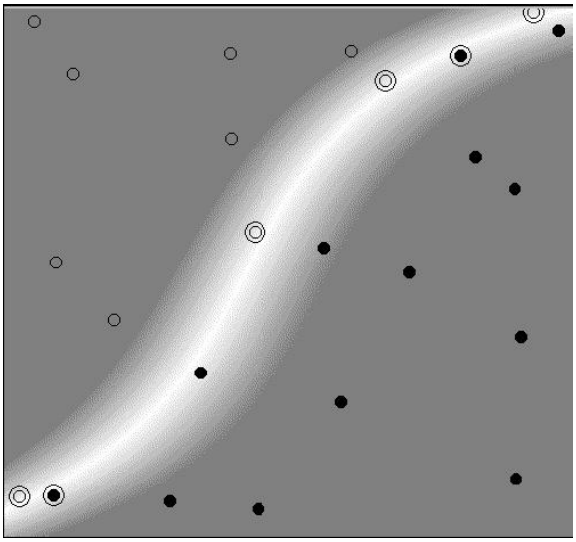
6 Szeparálás lineáris kernel függvénnyel egy nem lineárisan szeparálható esetben [7]

A fenti példa egy lineárisan nem szeparálható esetben talált megoldást mutatja. Itt az optimális döntési síknak az SVM azt a síkot választja, amely esetén maximális margó mellett a következő függvény minimális lesz.

$$Err(C,\underline{w},\underline{\varsigma}) = \frac{\|\underline{w}\|^2}{2} + C\left(\sum_i \varsigma_i\right)$$

, ahol C szabadon választható konstans, $\underline{w}$ a vizsgált sík normálvektora, $\varsigma_i >= 0$, i=1,.., l pedig a vektor helyéből számolt vektorra vonatkozó hiba mértéke.

C nagyobbra választásával a hibát okozó vektor eldobásával a „büntetés" (vagy másképpen, az adott tartóvektor választás költsége) növekszik.



7 Szeparálás 3-ad fokú polinomiális kernel fügvénnyel [7]

Ez a példa a fenti lineárisan nem szeparálható esetet illusztrálja, amely azonban szeparálható lesz polinomiális kernelfüggvény alkalmazása esetén.

$$K(\underline{x},\underline{y}) = \left(\underline{x} * \underline{y} + 1\right)^{p}$$, ahol p a kernel függvény foka.

Valójában a leképezés nem történik meg, csupán a belső szorzat függvénytől „szabadultunk meg", és alkalmazzuk helyette a fenti kernel függvényt.

Adott véges elemszámú tanítóhalmaz felhasználásával betanított gépi tanulási eljárás akkor rendelkezik a legjobb generalizálási képességgel, ha az adott tanítóhalmazon elért pontosság és a gép „kapacitása" egyensúlyban van.

Ha a gép kapacitása jóval meghaladja a modell méretét, akkor a tanításhoz felhasznált példákon kívül szinte mindent elutasít, ha viszont a modell mérete túl nagy, akkor a gép nem képes különbséget tenni két valójában különböző elem között.

Minél magasabb fokú az alkalmazott polinomiális kernelfüggvény, annál biztosabban szeparálható egy modell, azonban az osztályozás minősége romolhat, mert egyúttal a modell generalizációs képessége is csökken. A választott kernel függvény VC dimenziója [7] jó jelzőszám a gép várható kapacitásának becslésére, amely azonban még sok tényezőtől (C, epsilon, gamma paraméterek) függ. A polinomiális kernel függvény VC dimenziója a fokszám növelésével arányosan növekszik.

Az osztályozás során a kérdéses *jellemző vektor* osztályát az határozza meg, hogy az irányított hipersík mely oldalára esik.

Míg a neurális hálózatok esetén vannak struktúrák, amik egyes feladatokra jobban alkalmasak, míg mások kevésbé – az SVM semmilyen „megelőző tudást” nem használ fel a tanuláshoz, illetve klasszifikációhoz. Durva példával élve, ha a *jellemző vektorok* komponenseit egy bizonyos rögzített módon permutálva végezzük el a tanítási, illetve osztályozási folyamatot, az SVM mindezekre gyakorlatilag érzéketlen marad, míg egyes neurális hálózati struktúrák adott esetben teljesen más eredményeket adhatnak.

Az SVM alapú módszer előnye még, hogy a neurális hálózat kiértékelésénél jóval gyorsabban osztályoz.

A rendszer

A rendszer kialakítása egy hosszú folyamat eredménye, kezdetekben kitűzött célok közül sok bizonyult elérhetetlennek. Mára világosan látszanak egyes választott megoldások hátulütői.

Bármilyen nemű arci gesztus, illetve arcról felismerhető tulajdonság felismerése értelemszerűen feltételezi megbízható arcfelismerés meglétét. A megbízhatóság alatt itt a felismert arc méretének és helyzetének pontos – és videó folyam lévén – folyamatos felismerését kell érteni. Több, a globális technika sebességének növelését célzó megoldásról is kiderült, hogy magával a rendszer magját képező gépi tanulási eljárással okoz konfliktusokat, annak osztályozási minőségét rontva – ennek pedig visszahatása révén pont negatív sebességi vonzatai vannak. Általában tapasztalható volt, hogy az osztályozás pontosságát befolyásoló kisebb hibák is öngerjesztőek és komoly tényezőkké válnak.

A diplomamunka részét képező megoldás ezen okokat próbálja orvosolni, illetve a nem korrigáltakat azonosítani. Mindenképpen egy olyan rendszer kialakítása volt szempont, amelyre támaszkodva a minőségi követelmények további munkával elérhetőek valamint a magasabb felhasználói értékkel bíró alkalmazásoknak stabil alapjául szolgálhat illetve a hardver és szoftverkörnyezetet figyelembe véve is jövőbe mutató.

A következő fejezetekben megismerkedünk a rendszert alkotó két alkalmazással. Elsőként a két alkalmazás által megosztva használt adatbázis ismertetésre kerül sor, szorosan kapcsolódva az annak előállításáért felelős szoftver (továbbiakban tréningalkalmazás) leírásához, majd rátérünk a detektálást támogató alkalmazásprogramozási felületben (API) alkalmazott megoldások specifikációjára.

Az adatbázis

Az SVM gépi tanulási technika; megbízható működésének feltétele egy jó minőségű tanító halmaz kialakítása. Az SVM tanításához vektorokra és csatolt címkékre van szükség.

Mint általában a gépi tanulási eljárások, az SVM tanítás egy olyan függvényközelítési problémaként is szemlélhető, melynek során egy vektorparaméterrel rendelkező függvénytől a helyes címke érték szolgáltatását várjuk el. Helytelen, de szemléletes analógia,

ha a paramétertartomány folytonosságát pedzegetjük. A tanítás folyamán egy *jellemző vektor* halmaz elemeit vesszük sorra, és a „hasonlóakat” adott szisztéma szerint felcímkézve az SVM rendelkezésére bocsátjuk. Mely korábban ismeretlen vektor esetében várnánk el a fenti szisztéma szerinti valamely címke eredményül szolgáltatását? Minden kérdésre létezik egy egyszerű, magától értetődőnek tűnő, tömör és helytelen válasz.

Elnagyolva azt mondhatjuk, hogy a jó tanító anyag az, amelyik jól szeparálható – azaz adott címkéjű vektorok azonos térrészre esnek, míg más címkéjűek egy másikra. A későbbiekben láthatjuk, hogy vizuális képzelőerőnk a dimenzióbeli korlátok nyomán csalóka, és a jó tanító anyag problémája az egész projekt sok gondot okozó, sarkalatos kérdése.

Egyedtípusok

Alább az adatbázis ORM-je által bevezetett egyedtípusok rövid ismertetése következik. A vonatkozó ORM diagram a függelékben megtalálható.

Image

Képeket tárol. A képek reprezentációját megadó minden információ megtalálható benne, valamint képpontok sorfolytonos vektora.

Az adatbázisba integrált képek eredeti méretben, változatlan pixelformátummal kerülnek rögzítésre.

ImageInformation

Az egyes *Image* egyedek szemantikájukat az *ImageInformation* egyedeitől nyerik. Az *Image* és *ImageInformation* táblák közt N:1 számosságú kapcsolat van.

Ez az egyedtípus kulcsfontosságú a forrásanyagok rendszerezésében, egyedei meta-adat konfigurációknak tekinthetőek. Az egyes konfigurációk segítségével meghatározható a hozzájuk kapcsolt képek jelentése:

- arc esetén
    - nem; kor; etnikai hovatartozás; mutatott érzelem; szakáll, bajusz illetve szemüveg megléte; beállítás szöge
    - tetszőleges kapcsolt címke

    - validációs szerep

- negatív
    - tetszőleges kapcsolt címke
    - validációs szerep

Vector

Jellemző vektorokat tárol. Az egyedek eltérő dimenziójú vektorok lehetnek, az egyes komponenseket dupla pontosságú lebegőpontos számmal ábrázoljuk. Minden egyed tartalmazza azt a kapcsolt címkét reprezentáló értéket is, amelyet az SVM betanításakor képviselni fog.

A *Vector* egyedtípus N:1 kapcsolatban áll az *Images* egyedtípussal - adott *Image* egyedből a többféle dimenziójú illetve címkéjű vektor készülhet.

VectorSet

Jellemző vektorokat csoportosít. Egyedei csupán egy azonosítóból állnak, melyre a *Vector* egyedek külső kulccsal hivatkoznak. A Vector egyedtípus N:1 kapcsolatban áll a *VectorSet* egyedtípussal.

A *VectorSet* egyedek az SVM tanítás illetve validálás egységei.

Model

VectorSet egyedeket csoportosít. A *Model* egyedtípus két kulcson keresztül is 1:1 kapcsolatban van a *VectorSet* egyedtípussal – az egyik kapcsolat az SVM tanítása során felhasználandó vektor halmazzal kapcsol, míg a másik a betanított gép minőségének ellenőrzését támogatja.

Pyramid

Model egyedeket csoportosít. A Model és a Pyramid egyedtípusok között N:1 számosságú kapcsolat van. Az egy bizonyos *Pyramid* egyedhez kapcsolt modellekhez köthető *VectorSet* egyedek rendre az ugyanazokhoz az *Image* egyedekhez tartozó, azonosan felcímkézett, de eltérő komponens számú vektorokat tartalmaznak. Az egyes *Model* egyedekhez köthető

vektor variánsok dimenziói között egy a Pyramid egyedhez kötött együttható, az úgynevezett faktor jelenti az összefüggést.

Parameter

Egyedei az SVM tanításoknál használatos paramétereket írják le. A *Pyramid* egyedtípus 1:1 számosságú kapcsolatban van a *Parameter* egyedtípussal.

Filter

Egyedei az *ImageInformation* egyedeihez hasonló meta-adat konfigurációk, azzal a különbséggel, hogy egyes meta-adatok esetén nem értéket, hanem intervallumot rögzítünk. Ebből sejthető felhasználási területe – az Image egyedek szűrésére való az egyes Image egyedekhez kötött *ImageInformation* egyedekkel szembeni megfeleltetéssel. Ha az *ImageInformation* egyed minden meta-adat értéke a *Filter* egyed megfelelő meta-adat intervallumának eleme, akkor az *ImageInformation* egyedhez tartozó Image egyed „fennakadt” a szűrőn.

A Filter egyedek további meta-adatokat is tartalmaznak:

- a leválogatott képekből készített vektorok címkéje (pozitív, illetve negatív bináris esetben)
- a leválogatott képekből készített vektorok felhasználása (validációs vagy sem)

A *Filter* egyedtípus N:1 kapcsolatban van a *Pyramid* egyedtípussal.

Az adatbázis feltöltése

Az adatbázis feltöltése egy erre a célra fejlesztett alkalmazás, a tréner alkalmazás feladata.

Képek feltöltése

A tréner alkalmazás képek feltöltésére két eltérő megoldást kínál. Mindkét megoldás kulcseleme, hogy a képek bevitele csoportonként történik és egyes csoportokhoz meta-adat konfiguráció rendelése kötelező, ami lehetővé teszi szűrők alkalmazását piramis generálásakor.

A képadatbázis filterekkel leírt szemantikai alapon történő szűrése lehetővé teszi a megadott szűrők által leválogatott képekből készített vektor halmazok intuitív

értelmezhetőségét, továbbá ugyanazon képadatbázisból eltérő célú piramisok generálhatóak (például arc-nem arc, illetve boldog-nem boldog), valamint a tanítóhalmazok által tartalmazott vektorok köre és aránya precízen beállítható.

A szűrők piramishoz kapcsolása felgyorsítja a modellekkel történő kísérletezést, a képadatbázis bővítését követően a piramis modelljeihez rendelt összes vektorhalmaz egyszerűen újragenerálható.

A szűrő a fentebb említett arci jellemzőkre vonatkozóakon felül a beállításra vonatkozó kritériumokat is rögzíthet, mint a felvétel szöge. Megfelelő filterek alkalmazásával egyszerűen vonhatunk be, vagy zárhatunk ki bizonyos képeket a modellből. Erre jó példa lehet a szemüveges illetve szakállal rendelkező alanyok kezelése, vagy a beállítás specifikus modellek készítése – profil vagy félprofil, stb.

A képcsoport és annak elemeit jellemző kép információ mentése egy tranzakcióban történik az *Image* illetve *ImageInformation* táblákba. A kép sorfolytonosan, pixel formátumának megfelelően az *Image* tábla *Pixels* nevű bináris típusú adattábla oszlopba kerül mentésre.

Importálás

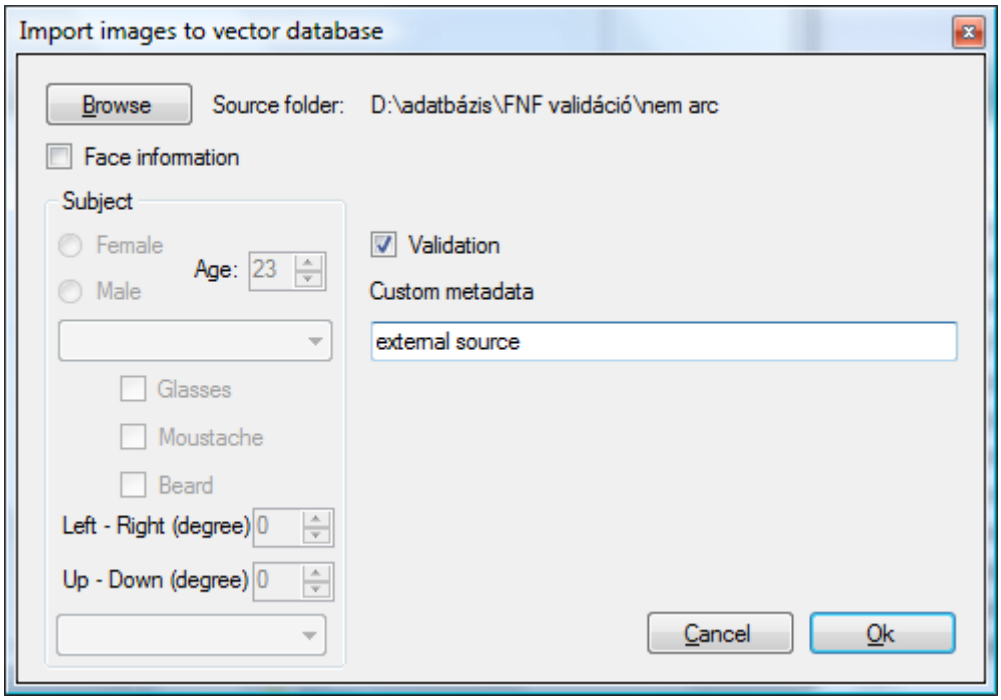
Importálással már létező képek importálhatóak az adatbázisba. Ez a funkció támogatja korábban más megoldással kezelt adatbázis elemeinek átvételét, illetve az újonnan, külső forrásból szerzett anyagok adatbázisba integrálását. Ez a dedikált funkció továbbá negatív képek adatbázisba integrálására.

Az ilyen módon az adatbázisba került pozitív képek manuális előkészítést igényelnek [(8)]. Ez a kézi szerkesztés sok szempontból korlátolt:

- a képek változatos külső forrásból származhatnak. amelyek eltérő időben, több féle körülmények közt készültek
- a pozitív képrészletek felbontása változó, gyakran túl kis részletességű

- egy alanyról kisszámú pozitív kép érhető el (különösen érzelem és nem/kor felismerés tekintetében)
- a képekhez rendelt nem minden meta-adat pontosan elérhető (kor, etnikum, tényleges pillanatnyi hangulat)

Az következő ábra a tréningalkalmazás külső forrásból történő integrálásához nyújtott felületét mutatja.



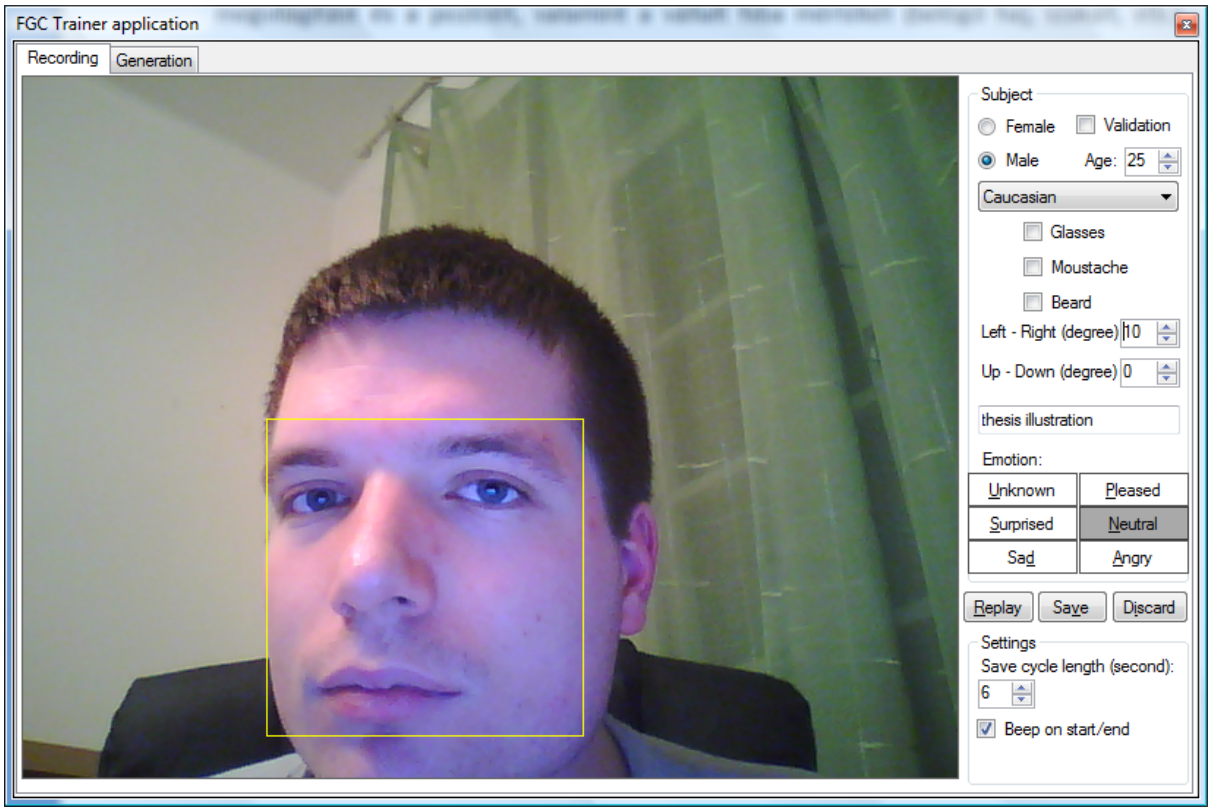
8 A tréneralkalmazás import ablaka

Felvétel

A másik megoldás csak pozitív képek bevitelére alkalmas – egészen pontosan pozitív képek elkészítésére. A kész képek preparálásához és meta-adat konfigurációval való ellátásához képest fordított logikát követ: a pozitív képek előállításához az alany kooperációját igényli, azonban cserében nagy mennyiségű, kontrollált körülmények között készült anyaghoz jutunk.

Az így előállított képeken szabadon megszabhatjuk a megvilágítást és a pozíciót, valamint a vállalt hiba mértékét (belógó haj, szakáll, stb.). Mindezekre a preparálandó forrásból

szerzett anyag esetében is van lehetőségünk, ám az a megközelítés munka és időigénye miatt állandó gátja az agilis fejlesztésnek [8].



9 A tréneralkalmazás felvételi mód közben

A felvétel ciklusokban történik. A ciklus hossza másodpercben értendő. A video folyam forrásától függően 12-30 felvétel készül, amely ideiglenesen egy memória pufferben kerül tárolásra. A puffer visszanézhető, ahol a rosszabbul sikerült képkockák törölhetőek, illetve az egész puffer egyben eldobható. Ha elégedettek vagyunk a felvétellel, akkor az összes képet a meta-adat konfiguráció csatolásával az adatbázisba menthetjük.

A tréneralkalmazás pozitív képek automatizált készítésére vonatkozó képességei nagy segítségére lehetnek olyan globális osztályozás centrikus felhasználásra irányuló projekteknek, mint amilyen az érzelem felismerés vagy a felhasználó authentikáció.

Vektor generálás

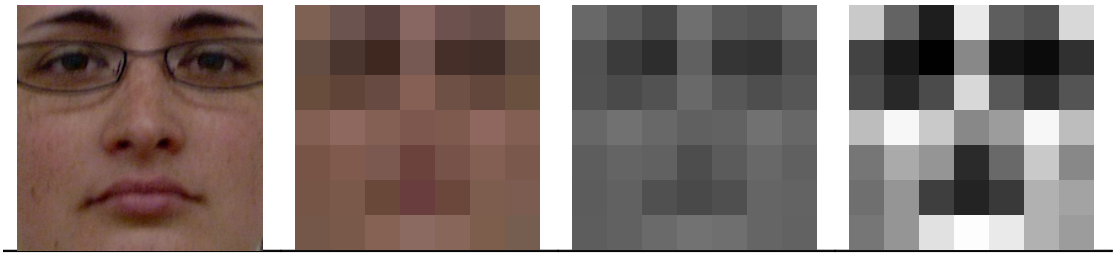
Az adatbázisba integrált képek felhasználását vektorok generálása és csoportosítása jelenti.

Egy adott képhez tartozó vektor előállításának lépései:

1. Skálázás
2. Szürkeárnyalatossá transzformálás
3. Hisztogram kiegyenlítés gyakorisághisztogram alapján

4. Vektor kialakítása a kép sorfolytonos, intenzitás értékek 0-1 intervallumba normalizálásával

Az eljárást egy pozitív képen az alábbi képsorozat szemlélteti.



Eredeti 200x200 | 7x7 | 7x7 szürkeskálás | 7x7 hisztogram kiegyenlített

1x49 sorfolytonos (a vektorkomponensek 0.0-1.0 közé normalizált lebegőpontos számként vannak ábrázolva, 1.0/255 lépcsőkkel)

10 Vektor generálás lépései

A folyamat egyszerű, viszonylag kis számítási igényű, ami fontos, mivel a detektálás folyamatában is ugyanezen eljáráson kell átesnie az éppen vizsgálat tárgyát képező képrészleteknek. *Jellemző kiemelés* szempontjából mondhatjuk, jól redukálja a probléma dimenzióját. A vektorgenerálást kimerítően specifikálja Szabó Péter 2006-os diplomamunkájában [8].

Láthatjuk, hogy a folyamat során előállított redukált vektor, ha az ilyen kisméretű, csupán alig emlékeztet az eredeti képre. Felmerül a kérdés, hogy vajon az SVM képes-e megfelelően szeparálni a vektorokat. Vajon egy 200x200-as sugárveszély piktogram hasonló vektort eredményezne?

A következő fejezet végén, a piramisokkal kapcsolatos teszteredmények és anomáliák tárgyalásánál visszatérünk a kérdésre.

Piramis építés

Az adatbázis összes eddig nem említett elemének létrehozása a *Pyramid* egyedekhez köthető. A következőkben megfigyelő-megismerő sorrendet követünk, azaz funkcionális magyarázat nélkül rögzítjük a piramis szerkezetét és felépítésének módját és csak ezt követően ismertetjük a struktúra szerepét.

A piramis

A detektálás algoritmusa az osztályozáshoz nem csupán egy SVM modellt (adott modellből betanított SVM), hanem SVM modellek egy egész családját használja fel. Ezeket az SVM modelleket egy, a továbbiakban SVM piramisnak nevezett struktúrába szervezzük.

Az egyes SVM modellek az adatbázis azonos részhalmazából készülnek, azonban a tanítóanyagot képező vektorok a képek eltérő felbontásúra skálázott változatait veszik alapul. A piramis egy SVM modelljét ezért szemléletesen rétegnek nevezzük (a legnagyobb dimenziójú vektorok által meghatározott réteget tekintve a piramis alapjának). A szomszédos rétegek dimenzióinak különbsége konstans-szoros. Ezt az értéket faktornak nevezzük.

A piramis jellemzője még, hogy az egyes SVM modellek betanításához milyen SVM paramétereket használunk fel.

A fentiek alapján egy SVM piramis meghatározása:

- A piramist alkotó modellek elemeinek meghatározása, ami az elemeket kiválasztó szűrők meghatározásával történik.
- A piramis szerkezetének meghatározása, ami magában foglalja a legkisebb réteg méretének, a faktornak és a rétegek számának megadását.
- SVM paraméterek csatolása.

Minden a piramisra vonatkozó információ egy *Pyramid* egyedhez rendelve kerül rögzítésre.

Rétegek

Egy SVM modell tanítása egy *VectorSet* egyedből történik, amely tanítóhalmazként csoportosítja a piramishoz rendelt azonos dimenziójú vektorokat. Ebből adódóan minden réteghez egy *VectorSet* egyed tartozna, ám az adatbázis két vektor csoportot értelmez rétegenként; a másikból SVM modell sosem készül, elemei validációs vektorok. Jelentőségük az SVM piramis rétegek minőségének objektív mérhetőségében van.

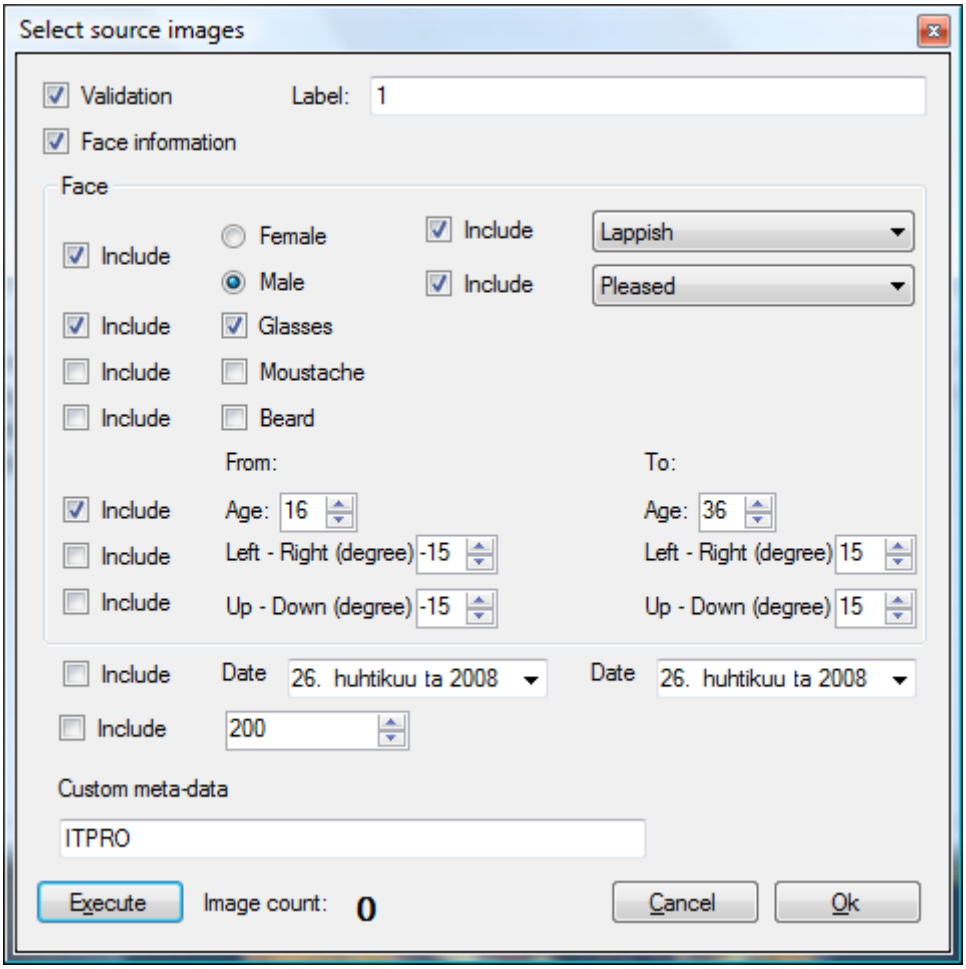
Egy piramis felépítése rétegről rétegre történik. Az egyes rétegekhez új *Model* egyed jön létre, amelyhez pedig két *VectorSet* egyed – egy a tanító halmaz, egy pedig a validációs halmaz számára. Ezen szerkezet feltöltése a *jellemző vektorok* generálását jelenti.

Az *Image* táblából szűrőkkel képeket válogatnak le, amelyekből a korábban vázolt módon vektorokat generálunk és beszúrunk a *Vector* táblába. Az egyes *Vector* egyedeket külső kulccsal a feltöltendő réteghez tartozó valamely *VectorSet* egyedhez kapcsoljuk (attól függően, hogy a szűrő validációs célzattal szűr vektorokat, vagy sem).

A *Vector* egyedeket aszerinti címke értékkel társítjuk, hogy milyen címkéjű szűrő választotta be a vektorhalmaz forrásképei közé. A címke pozitívnak tekintett képből generált vektor esetén +1.0, egyébként pedig tetszőleges eltérő érték. Amennyiben nem kívánjuk a projekt részét képező alkalmazásfejlesztési interfészt felhasználni, a címkeválasztás teljesen tetszőleges.

Forrásképek kiválasztása

A modell elemeinek meghatározása szűrők megadásával történik.



11 A tréneralkalmazásból szűrő beállítási ablaka

Fent egy olyan szűrő megadását láthatjuk, amely a validációs célzattal adatbázisba integrált képek közt fog keresni egy szemüveges lapp etnikumba tartozó 16-36 éves férfi ITPRO hallgatót, aki az adott képen boldog. A rétegek validációs vektorhalmazához lesznek kapcsolva a beszúrt *Vector* egyedek, amelyek +1 címkét fognak kapni.

Jelenleg az adatbázisban nincs ilyen kép, de ennek ellenére a szűrőt a piramishoz kapcsolhatjuk, így ha a későbbiek során az adatbázisba integrálnánk egy megfelelő képet, a piramis újragenerálásánál a kép a modell részévé válna.

SVM paraméterek

Új piramis létrehozásakor a szerkezetét leíró adatokon felül az SVM modellek készítéséhez szükséges további paramétereket is meg kell adnunk, amelyeket azonban módosíthatunk piramis feltöltésekor.

Az implementáció a Thorsten Joachims féle SVM^{light} [9] .NET-re portolt változatára épül. Az alábbiakban vegyük sorba az SVM^{light} által támogatott kernelfüggvényeket és a C paraméteres klasszifikációs SVM (CSVC) esetén szükséges paramétereket.

Kernel	Kernel függvény	Megnevezés
LINEAR	$K(x,y) = (x \cdot y)$	Lineáris
POLY	$K(x,y) = (\gamma \cdot x \cdot y + Coeff0)^d$	Inhomogén polinomiális (Coeff0 = 0 választása esetén beszélhetünk homogén polinomiális kernelfüggvényről)
RBF	$K(x,y) = e^{\frac{-\|x-y\|^2}{2\sigma^2}}$	Gauss-i radiál alapú
SIGMONOID	$K(x,y) = \tanh(\kappa x \cdot y - \delta)$, nem minden κ>0 és δ<0-ra	Szigmonoid

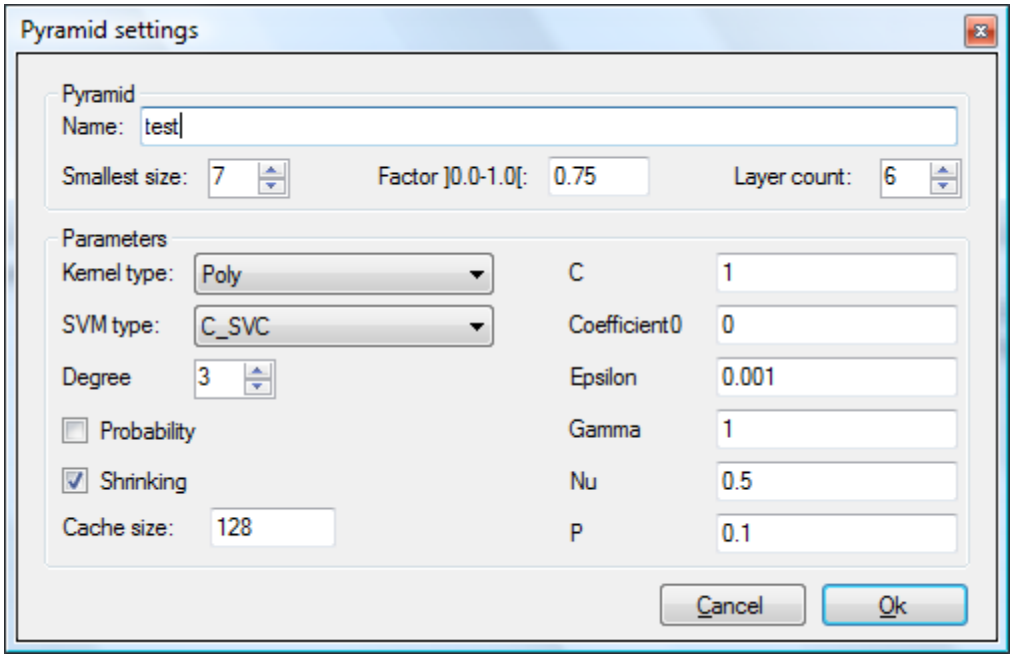
1 Támogatott kernelfüggvények

A projekt fejlesztése során jellemzően C paraméteres SVC SVM-et használtam, néhány egy osztályos klasszifikációs kísérlettől eltekintve, amellyet az érzelemfelismerés roppant asszimetrikus mennyiségi mutatókkal bíró adatbázisa motivált. A C paraméteres SVC tanítás azon klasszifikációs célzatú maximális margójú szeparáló hipersík(ok) keresését jelenti, melyek esetén az SVM technika rövid ismertetőjében említett C-t paraméteres hibafüggvény minimális lesz.

C	Nincs intuitív jelentése, bár szokás büntetés paraméternek nevezni hibafüggvénybeli szerepe miatt (vagy költségnek, innen a neve is).
γ, Coeff0	Olyan szabadon választott paraméterek, amelyek megválasztása eseti alapon történik és befolyásolják a tartó vektorok számát és a választott margót.
d	A polinomiális kernel fokszáma.

2 C SVC paraméterek

Az következő ábra a tréner alkalmazás lehetőségeit mutatja a piramis szerkezetének és modellhez csatolt paramétereinek beállítására. Tetszőleges SVM^{light} által támogatott SVM és kernel típus felparaméterezhető és használható a rendszer által.



12 Piramis paraméterek megadása

C SVC paraméterekről bővebben

C a felhasználó által szabadon megválasztott paraméter, de jól megválasztott értéke kritikus lehet a modell teljesítményére nézve. Hogy mi a jól megválasztott érték, az sajnos leginkább az aktuális tanítóhalmaztól függ.

C alacsony értékre választása megengedőbb az adott margó esetén fellépő hibákkal szemben, míg C növelése egyre keményebb margót eredményez. A kemény margó arra utal, hogy a nehezen szeparálható tartó vektorok elvesztése kevésbé megengedett, ami nagyszámú margót meghatározó tartóvektorhoz vezet, ami pedig könnyen túl illesztő SVM létrejöttét okozza.

Paraméter optimalizáció

A paraméterek megválasztása döntő az SVM hatékonyságára vonatkozólag, ezért az SVM technika kutatási területén belül a paraméterek felügyelet nélküli iteratív, illetve analitikai alapon történő optimalizációja az egyik legfontosabb és egyben leginkább nyitott, aktív

terület. A számos különböző megközelítés vizsgálatát, ebből adódóan az optimális és automatikus paraméterválasztás problémáját jelen projekt nem vállalja fel, de mindenképpen javasolt, hogy továbbfejlesztés tárgya legyen.

Felületesen megemlítve a következő két iteratív módszer nevezhető elterjedtnek:

- grid optimization – az egyes paraméterekhez intervallumokat jelölünk ki és megadjuk az osztáspontok számát. A módszer a lehetséges kombinációkat vizsgálva választja ki a cross validation alapján legígéretesebb paraméter megválasztást. Könnyen belátható, hogy már kis számú paraméter esetében is, az osztáspontok számának függvényében igen sok lehetőség van ezért módszer rendkívül időigényes.

- pattern optimization – legegyszerűbben az grid módszer olyan természetes továbbfejlesztésének tekinthető, ahol a cross validation eredményére alkalmazott, bináris keresést műveletként használó hegymászó algoritmust vezetünk be a nyilvánvalóan rosszabb paraméter kombinációk kizárására.

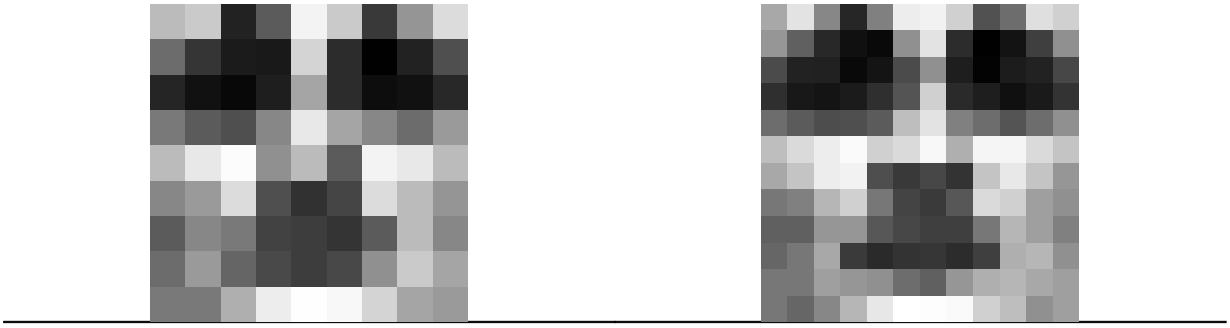
Az iteratív módszerekkel történő paraméterkeresés egyik legnagyobb problémája időigénye. Az SVM betanítása gépi tanulási szempontból gyors, de abszolút értelemben így is időigényes. Például egy teljes piramis betanítása a később ismertetett legsikeresebb heterogén modellel több, mint fél perc alacsony C értékre, és több mint 10 perc is lehet C nagyon magas értékeire nézve. Az egyéb paraméterekre is kiterjesztett léptetéssel az akár több ezer kombináció esetén a paraméter optimalizáció folyamata könnyen hetekben is mérhető időigényűvé válhat.

Az SVM piramis funkciója

A gépi tanulással történő objektum klasszifikáció rendszerint támaszkodik egy előzetes probléma dimenzió redukcióra. Ez a *jellemző kiemelés* során történik. A vektorok előállításánál láthattuk, hogy ez egyfelől színtér transzformációt, másfelől kisebb felbontásúra skálázást jelent esetünkben – a gyakorisághisztogram alapján történő hisztogram kiegyenlítés tovább már nem redukál, csupán normalizál.

Az interneten is jelenlévő globális megközelítést alkalmazó rendszerek rendszerint a 16x16 -23x23 tartományba eső méretűre skálázzák a pozitív képeket a *jellemző kiemelés* során. A teszteredmények ismertetése során kiderül, miért.

Az SVM piramis célja a klasszifikáció gyorsítása, technikai megoldás.



9x9 méretűre skálázott pozitív kép 12x12 méretűre skálázott verzió

13 Egy SVM piramis két alsóbb rétegéhez illeszkedő vektor képként

A fenti két kép ugyanazon piramis két szomszédos, alacsony dimenziójú rétege. A két réteg közötti faktor 0.75, azaz a jobboldali réteg $\left(\frac{1}{\frac{3}{4}}\right)^2 = \frac{16}{9} \cong 1.78$-szor nagyobb területű.

Ugyanezt adja persze a generált vektorok dimenzióinak aránya, 144/81. Egységnyi darabszámú SVM klasszifikációhoz szükséges idő a vektor dimenziójával lineárisan arányosan nő.

A piramis alkalmazása a probléma szubjektív megközelítéséből ered: egy nyilvánvalóan eltérő képről meg tudjuk állapítani annak eltérő voltát az alacsonyabb dimenziójú képről is, tehát megspórolhatjuk a magasabb dimenziójú osztályozás plusz időköltségét. Ha nem tudjuk eldönteni a kép egyértelműen negatív voltát, akkor talán majd egy magasabb dimenziójú réteg segítségével. A legnagyobb réteg számára való megfelelés esetén a képet pozitívnak fogadjuk el (tehát a legnagyobb rétegnél kisebb találatok lehetőségét eleve kizárjuk).

A piramis alkalmazása olyan problémák esetén, ahol az osztályozás kimenetele inkább pozitív, a piramis alkalmazása nem merül fel, de negatív túlsúlyú problémák esetén, mint amilyen a detektálás, egyértelműen időt takarít meg.

A következőkben T a legnagyobb SVM piramis réteggel történő klasszifikáció idejét jelenti, $F \in]0,1[$ a nagyobb és kisebb szomszédos rétegek közötti faktort (egy élt tekintve, területre

a négyzete), N a rétegek számát jelenti. A rétegek indexelése 1-tól történik, N a legkisebb dimenziójú réteget jelenti.

Ezek alapján egy pozitív vektor osztályozásához szükséges idő, T_p a következőképpen adódik:

$$T_p = \sum_{i=1}^{N} T \cdot F^{2(i-1)}$$

Tehát egy pozitív klasszifikáció időköltsége 1-nél nagyobb, de N-nél kisebb. A klasszifikáció az N indexű rétegen kezdődik. Ha az M-dik rétegen megszakad, akkor a döntéshez szükséges idő:

$$T_d = \sum_{j=M}^{N} T \cdot F^{2(j-1)}$$

A piramis csak akkor lehet hasznos, ha T_d a detektálás teljes folyamatára nézve átlagosan kisebb, mint T (a piramis minőségre vonatkozó negatív hatásait nem figyelembe véve). A modell javulásával $M \to N$-re lehet számítani.

Ez jó iránymutatás, de nem mond semmit arról, hogy adott M esetén mennyire kell negatív túlsúlyosnak lennie a problémának.

A megtakarítás egy negatívnak klasszifikált vektor esetében:

$$T_s = T_p - T_d = \sum_{i=1}^{N} T \cdot F^{2(i-1)} - \sum_{j=M}^{N} T \cdot F^{2(j-1)} = \sum_{i=1}^{M} T \cdot F^{2(i-1)}$$

Ebből adódik, hogy egy pozitívan osztályozott vektorra legalább

$$x = \frac{T_p - T}{T_p - T_d} = \frac{\sum_{i=2}^{N} T \cdot F^{2(i-1)}}{\sum_{i=1}^{M} T \cdot F^{2(i-1)}} = \frac{\sum_{i=2}^{N} F^{2(i-1)}}{\sum_{i=1}^{M} F^{2(i-1)}}$$

darab M-dik rétegen negatívnak osztályozott vektorra van szükség, hogy a módszer éppen megérje időben, rögzített N és F mellett. M becslése érdekes kérdés, mivel függ az egyes rétegek minőségétől, illetve az egymás követő rétegek szeparációja során megtartott

tartóvektoroktól. Az egyes rétegek minősége cross validation-nal mérhető, így T_d becsülhető az alábbi módon.

Vezessük be a következő jelöléseket:

$P_p^x \in [0,1] \subset R$	annak valószínűsége, hogy a probléma egy tetszőlegesen kiválasztott klasszifikálandó vektora ténylegesen pozitív
$P_n^p: N \to [0,1] \subset R$	annak valószínűsége, hogy adott indexű réteg egy pozitív vektort rosszul klasszifikál
$P_n^n: N \to [0,1] \subset R$	annak valószínűsége, hogy adott indexű réteg egy negatív vektort helyesen klasszifikál

Ekkor annak valószínűsége, hogy az k-ik réteg előtt még nem állt meg a klasszifikáció:

$$P^s(N) = 1$$

$$P^s(k) = \prod_{i=k}^{N} \left(1 - \frac{\left(1 - P_p^x\right) \cdot P_n^n(i+1) + P_p^x \cdot P_n^p(i+1)}{2}\right), \qquad k \in [1, N-1] \subset N$$

Ennek segítségével T_d várható értéke a rétegek minőségeinek ismeretében:

$$T_d = \sum_{i=1}^{N} P^s(i) \cdot T \cdot F^{2(i-1)}$$

Ez a becslés azonban durván optimista, a gyakorlatban mért értékek ettől negatív irányban eltérnek (de még mindig jócskán a megtérülő intervallumban maradva). Az eltérés különböző rétegek azonos tanítóhalmazból való betanítottságából fakad.

Létezik egy (legalábbis ilyen kis dimenziójú és ilyen tartalmú modellből betanított SVM rétegek esetén) r=0,02 körüli konstans, mely annak valószínűsége, hogy a klasszifikáció megáll, ha a korábbi szomszédos rétegen nem állt meg. Ez rámutat arra, hogy a piramis közbülső rétegeinél való klasszifikáció felesleges, sőt, valójában időpazarló.

$$T_d \approx T \cdot F^{2(N-1)} + \sum_{i=1}^{N-1} P^s(N-1) \cdot (1-r)^{N-i} \cdot T \cdot F^{2(i-1)}$$

Teszteredmények és anomáliák

Ebben a fejezetben röviden áttekintjük a projekt során az SVM piramisok készítésével kapcsolatosan szerzett tapasztalatokat.

Validáció

A validáció Model egyedről Model egyedre történik. Az egyes egyedekhez kötött tanító halmaz funkciójú vektorhalmazból a piramishoz rendelt paraméterek felhasználásával SVM modellt képzünk. Ezt az SVM modellt a szintén a réteghez kapcsolt validációs vektorhalmaz elemeit sorra véve osztályozásra használjuk. Mivel a validációs vektorok mindegyékéről pontosan tudjuk, hogy mi a helyes vektor címke, az osztályozás eredményének helyes vagy helytelen volta megállapítható. A validációs vektorokat címke szerint csoportosítva az elsőfajú és másodfajú hibák aránya megállapítható (azaz a tévesen negatívként, illetve tévesen pozitívként osztályozott esetek aránya).

Hasonló mérést a validációs vektorhalmazon felül a tanítóhalmazra is elvégezzük, hogy a tartóvektorokra nézve további ismeretekhez juthassunk.

Példa riportok

A következőkben a fejlesztés során előkerült jellegzetes esetekre nézünk példákat és megismerjük a kapcsolatos anomáliákat.

A legjobb heterogén piramis

Az alább riportolt piramis az eddigiek során kialakított legjobb minőségű heterogén modellből készült. A heterogén jellemző arra utal, hogy egynél több alany képeiből áll a forrásmodell. Ilyen adatbázis a fejlesztéshez csak külső forrásból állt rendelkezésre, ezért az anyagra a korábban említett összes külső forrásra jellemző tulajdonság fennáll.

Az anyag manuálisan több alkalommal is szelekción esett át a hipersíkot várhatóan rosszul befolyásoló képek kiszűrésére. Az adatbázis így 403 darab arcot ábrázoló képet és 13906 darab negatív képet tartalmaz (melyből a modellbe csak 800 db-t vontam be). A validációs adatbázis 291 darab (rosszabb minőségű) pozitív példát és 4635 darab negatív példát tartalmaz.

SV dimension	Label	SV count	Correct cross %	Wrong cross %	Time (ms)	Correct SV %	Wrong SV %
49	1	229	82.48847926...	17.51152073...	140	100	0
49	-1	303	89.62243797...	10.37756202...	2152	100	0
81	1	216	85.71428571...	14.28571428...	141	100	0
81	-1	277	92.90183387...	7.098166127...	2980	100	0
144	1	205	89.40092165...	10.59907834...	218	100	0
144	-1	257	93.87270765...	6.127292340...	4649	100	0
256	1	193	86.63594470...	13.36405529...	343	100	0
256	-1	232	95.31823085...	4.681769147...	7207	100	0
484	1	203	88.01843317...	11.98156682...	686	100	0
484	-1	238	95.46925566...	4.530744336...	14477	100	0
841	1	230	87.09677419...	12.90322580...	1233	100	0
841	-1	228	96.15965480...	3.840345199...	26349	100	0

14 A legjobb heterogén piramis riportja 3-ad fokú polinomiális kernelfüggvény esetén

Az oszlopok jelentései rendre a következők:

SV dimension	használt modell vektorainak dimenziója
Label	adott sorban vizsgált vektorok címkéje
SV count	adott címkéjű tartóvektorok száma
Correct cross %	a validációs halmazban adott címkével rendelkező vektorokra elért helyes klasszifikációk aránya
Wrong cross %	a validációs halmazban adott címkével rendelkező vektorokra elért helytelen klasszifikációk aránya
Time (ms)	a validációs halmazban adott címkével rendelkező összes vektor klasszifikációjának teljes időtartama milliszekundumban
Correct SV %	a tartóvektorok közötti adott címkével rendelkező vektorokra elért helyes klasszifikációk aránya
Wrong SV %	a tartóvektorok közötti adott címkével rendelkező vektorokra elért helytelen klasszifikációk aránya

A piramis egyes rétegei 7x7, 9x9, 12x12, 16x16, 22x22, 29x29 méretűek. A klasszifikációhoz szükséges időn látható, hogy a korábbi állítással ellentétben nem teljesen lineáris – ez nem az SVM implementációból fakad, hanem a számítások során megmozgatandó memória mennyiségéből következő eltérő memórialapozási igényekből.

A legnagyobb réteg mérete

A probléma dimenziójának növekedésével a minőség, ha nem is monoton, de tendenciózusan nő, míg a tartóvektorok száma inkább csökken. Kijelenthető, hogy azonos számú tanító vektor egy magasabb dimenziós vektortérben könnyebben szeparálható és a gép generalizációs képessége a kapacitással és a minőséggel együtt növekszik. Egy darabig.

Ha ugyanis a vektorok száma a probléma dimenziójához képest túl alacsony, akkor a gép túlgeneralizál, az osztályozás minősége romlani kezd. A határ a tapasztalatok szerint ekkora tanítóhalmaz méretnél 29x29 körül van, ez a magyarázat a választott legnagyobb réteg dimenzióra.

A legkisebb réteg mérete

A minimális réteg méret magyarázata egy korábban felvetődött kérdéshez köthető – az SVM vajon képes-e megkülönböztetni a sugárveszély piktogramot egy arctól?

7x7-es méret alatt már nem jellemzően, de még ez a legkisebb réteg méret is rizikós. A minőségi mutatókból látszik, hogy 7x7, 9x9 és 12x12 között egyértelmű minőségi javulás tapasztalható, ám a vektor dimenziója és így a detektálás időigénye is minden alkalommal 1,78-szorosára növekszik az eggyel nagyobb réteg kezdő rétegnek választásával.

A korábbiakban megmutattuk, hogy a legkisebb réteg osztályozásának minősége és a detektálás sebessége között egyértelmű összefüggés van.

A legkisebb réteg mérete akkor optimális a detektálás sebességének szempontjából, ha annak eggyel való növelésével a minőség olyan mértékben már nem növekszik, hogy a piramis alkalmazásából fakadó relatív idő megtakarítás már elmaradna a vektor méretének növekedéséből adódó többletszámítási idő igénytől.

De mi a helyzet a minőségi optimalitással?

Minőségi szempontból az ideális persze a 3-ad fokú polinomiális kernel tartóvektorokra mutatott arányának validációs vektorokra mutatott volta lenne (ez a piramis funkciójánál mutatott összefüggések alapján a detektálás sebességére is igencsak jó hatással lenne).

Említettem, hogy a piramis egyes rétegei közötti egyetértés igencsak magas – ez azt eredményezi, hogy tévesen pozitív osztályba sorolt negatív vektorok nem nagyon esnek ki a

magasabb rétegekkel való osztályozás során (r kicsi). Ez másképpen úgy közelíthető meg, hogy a piramisra jellemző első és másodfajú hibák krónikusak.

A piramis alkalmazási módjából adódóan a detektáláskor rögtön elveszítjük a pozitív találati lehetőségek 17,5%-át a legkisebb rétegen és további kb. 2%-ot a magasabb rétegeken. Ez - tekintve, hogy videófolyamon detektálunk - csupán annyit jelent, hogy minden 5-ik alkalommal sikertelenül záródik a detektálási folyamat, ideális sebesség mellett másodpercenként 3-4-szer. Meglepő módon tehát az elsőfajú hibák ilyen magas aránya még elfogadhatónak tekinthető.

Azonban a minimális másodfajú hiba arány 3,8% körül van a legnagyobb rétegen, így a korábbi rétegeken kihulló téves pozitív találatok levonása után körülbelül 3% a végső osztályozás szerint is pozitív. A detektálás folyamatának specifikációjából majd láthatjuk, hogy ez nagyon komoly hiba.

Kezdetnek azonban gondoljunk bele, hogy egy 640x480-as felbontású videófolyam estében a legnagyobb réteg, amire az SVM piramis legkisebb rétegével még illesztést kísérlünk meg, 113x85 méretű. Ezen 7x7 méretű képrészletekből készített vektorokat vizsgálunk. Ilyen vektorból 8268 darab van, aminek a 3%-a 248 darab. A videófolyam kockáiból képzett multi-rezolúciós képpiramis alacsonyabb felbontású rétegeire is illesztünk, onnan további körülbelül 50-60 darab téves, de minden SVM piramis rétegen pozitívnak elfogadott illesztés is várható.

Miért lehetséges ennek ellenére jóindulattal elfogadható minőségű detektálást elérni egy ilyen SVM piramis segítségével? A magyarázat az, hogy a validációs vektorhalmaz tartalma jóval speciálisabb negatív esteket is tartalmaz, mint amilyenek egy szokványos körülmények között készített videófolyam esetén a detektálandó alany arcának hátterében megtalálhatóak. Másfelől a detektálás folyamatában az illesztésre szánt vektorok sorrendjét olyan módon választjuk meg az úgynevezett pásztázás során, hogy a ténylegesen pozitív találatot lehetőleg ne előzze meg krónikus másodfajú hiba.

A példák számának növelése

A példák számának aszimmetrikus növelése a jobban növelt osztály hatékonyabb felismeréséhez vezet a kevésbé növelt elemszámú osztály rovására. Ez az SVM tanítása során kialakuló hipersíkok margójának csökkenése miatt következik be.

Az következő riport a fenti forrásmodell annyiban eltérő változatából készült, hogy 800 negatív példavektor helyett 1600-at vontam be.

SV dimension	Label	SV count	Correct cross %	Wrong cross %	Time (ms)	Correct SV %	Wrong SV %
49	1	238	74.19354838...	25.80645161...	124	100	0
49	-1	387	93.67853290...	6.321467098...	2527	100	0
81	1	247	81.56682027...	18.43317972...	187	100	0
81	-1	407	96.31067961...	3.689320388...	3932	100	0
144	1	217	87.09677419...	12.90322580...	281	100	0
144	-1	367	96.35382955...	3.646170442...	5835	100	0
256	1	221	82.48847926...	17.51152073...	468	100	0
256	-1	339	97.51887810...	2.481121898...	10203	100	0
484	1	229	81.56682027...	18.43317972...	842	100	0
484	-1	303	97.95037756...	2.049622437...	17628	100	0
841	1	247	85.71428571...	14.28571428...	1482	100	0
841	-1	299	98.16612729...	1.833872707...	31528	100	0

15 A legjobb heterogén piramis riportja aszimmetrikus tanítóhalmaz bővítést követően

Itt már a ténylegesen pozitív találatok 25%-át dobjuk el a legkisebb rétegnél, de másodfajú hibák minimum aránya 1,8%-ra csökkent. A 9x9-es rétegtől indítva a detektálást alapvetően nem lenne rossz választás ez a modell, de két helyen is sebességet veszítünk vele: egyfelől az 1,78-szer nagyobb területű legkisebb réteg méret miatt; másfelől a tartóvektorok száma nőtt az előző SVM piramishoz képest, így a klasszifikáció időigénye is nőtt valamelyest. Nehéz megtalálni az egyensúlyt, de valójában a projekt során soha nem is sikerült.

A példák számának szimmetrikus növelésével kismértékben együtt csökken az elsőfajú és másodfajú hibák aránya, de egyfelől ez a tartóvektorok számának drasztikus növekedésével jár, másodsorban számottevően nagyobb jó minőségű heterogén tanítóanyag nem állt rendelkezésre a fejlesztés során.

Paraméterek beállítása

Mind C, mind a szabadon választott kernel- és hibafüggvény paraméterek megállapítása próbálgatással történik. Ennek oka a kalkulatív eszköztár hiánya, a tanítóhalmaz összetételének eseti jellege, illetve az felügyelet nélküli paraméter optimalizáció hiánya.

A paraméterek beállítására nem sok iránymutatás adható, nagyban próba-hiba alapú a folyamat. Polinomiális kernelfüggvénnyel az SVM-nek láthatóan nem okoz gondot a modell hibamentes elsajátítása, így a hibás vektor választásának költségével próbálkozásnak kevés haszna lehet. Jelentős csökkentésével a tartóvektorok száma tovább csökkenthető, hiszen a margó felpuhul és így a maximális margó szélesség más tartóvektorok választásával érhető el - persze a minőség egyértelmű rovására, mivel az SVM modell túl fogja generalizálni a problémát. Azonban egy nehezebben szeparálható osztályozás centrikus problémánál (mint az érzelem felismerés) a paraméterek beállításának képessége még hasznos lehet.

Érdekességképpen tekintsük meg a legjobb heterogén modell lineáris kernellel történő SVM tanulásának eredményét a következő ábrán.

SV dimension	Label	SV count	Correct cross %	Wrong cross %	Time (ms)	Correct SV %	Wrong SV %
49	1	218	78.34101382...	21.65898617...	62	48.623853...	51.3761467...
49	-1	223	90.80906148...	9.190938511...	1576	78.475336...	21.5246636...
81	1	176	82.48847926...	17.51152073...	94	60.795454...	39.2045454...
81	-1	182	92.16828478...	7.831715210...	1965	73.626373...	26.3736263...
144	1	144	89.40092165...	10.59907834...	125	70.833333...	29.1666666...
144	-1	158	91.49946062...	8.500539374...	2886	79.113924...	20.8860759...
256	1	116	85.25345622...	14.74654377...	188	91.379310...	8.62068965...
256	-1	125	90.74433656...	9.255663430...	3931	94.4	5.6
484	1	144	85.71428571...	14.28571428...	390	100	0
484	-1	124	92.75080906...	7.249190938...	8471	100	0
841	1	173	84.79262672...	15.20737327...	842	100	0
841	-1	137	93.57065803...	6.429341963...	17628	100	0

16 A legjobb heterogén piramis riportja lineáris kernelfüggvénnyel esetén

A lineárisan betanított SVM piramis esetén az alacsonyabb dimenziójú rétegeknek a tartóvektorokra való visszaellenőrzés során láthatóan komoly gondot okozott az osztályozás. Érdekes, hogy a cross-validation eredménye még így sem tér el annyira a polinomiális kernelétől, mint amennyire várnánk.

A csoportok osztályozása jóval gyorsabban megtörténik, de ennek ellenére sem érdemes a lineáris kernelfüggvényt használni. Annyira tovább gyorsítani a detektálást már nem szükséges, hogy akár a legkisebb mértékben is romoljon a minőség, márpedig látható, hogy a minimális másodfajú hiba arány 6,4%, azaz a polinomiális duplája. A lineáris kernel ráadásul a nehezebben szeparálható esetekben látványosan rosszabb minőségi mutatókkal rendelkezne.

Azonban a kis kitérő célja a margó keményítésének, azaz C növelésének hatásának bemutatása. A következő riport egy olyan SVM piramisról készült, amelynél C 10-ről 1000-re változott, de egyébként azonos tanítóhalmazból lettek generálva.

SV dimension	Label	SV count	Correct cross %	Wrong cross %	Time (ms)	Correct SV %	Wrong SV %
49	1	200	79.26267281...	20.73732718...	78	52	48
49	-1	211	90.03236245...	9.967637540...	1388	73.933649...	26.06635071...
81	1	152	83.41013824...	16.58986175...	78	61.184210...	38.81578947...
81	-1	159	91.24056094...	8.759439050...	1731	74.842767...	25.15723270...
144	1	100	88.94009216...	11.05990783...	94	70	30
144	-1	127	90.24811218...	9.751887810...	2075	82.677165...	17.32283464...
256	1	109	82.02764976...	17.97235023...	171	100	0
256	-1	108	89.23408845...	10.76591154...	3525	100	0
484	1	159	83.41013824...	16.58986175...	437	100	0
484	-1	134	94.32578209...	5.674217907...	9048	100	0
841	1	186	81.56682027...	18.43317972...	842	100	0
841	-1	133	95.33980582...	4.660194174...	17800	100	0

17 A legjobb heterogén piramis riportja lineáris kernelfüggvénnyel esetén C=1000-re

A piramis minősége valamelyest jobb, ami a tanítóhalmaz olyan elemeinek eldobásának eredménye, amelyek címkének megfelelő csoportból „viszonylag kilógnak” a másik címkebeli csoporthoz közelebb. A tartóvektorok száma is csökkent valamelyest, de ez C növelésével inkább csak a lineáris kernelfüggvénnyel betanított SVM sajátosságának tudható be, polinomiális kernelfüggvénnyel a tartóvektorok számának növekedése felé fejtene ki hatást.

Összegzés

Láthattuk, hogy másodfajú hibák az elsőfajúaknál jóval több gondot okoznak. A piramis struktúrát is figyelembe véve logikusnak hat, hogy a piramis egyes rétegeit eltérő tanító anyagot (és paramétereket) figyelembe véve készíttessük el. Az első réteg tanítóhalmazának pozitív túlsúlyos állapotába helyezésével a tévesen visszautasított pozitív találatok száma csökkenthető lenne, míg az legnagyobb réteg negatív túlsúlyba helyezése a másodfajú hibákat eredményesen szűrné ki, mindeközben a magasabb dimenzióból fakadóan viszonylag kevésbé növelné az elsőfajú hibák számát. A közbülső rétegek használata feleslegesnek tűnik és kismértékben káros a pozitív találatok helyes felismerésére nézve.

Érdekes kísérletek

A rendszer fejlesztése során többször is mellékszálakat jelentő kísérletekbe kezdtem. A két legjelentősebbet alább röviden ismertetem.

A tréneralkalmazás pozitív képek készítésére vonatkozó képességeit kihasználva számos homogén, azaz csak rólam készült modellel kísérleteztem. Ezen modellek minősége kiemelkedő volt – úgy a validációs vektor halmaz elemeire, mint a tartóvektorokra való ellenőrzés esetén, mind az első, mind a másodfajú hibák aránya 0%-os volt, azaz minden pozitív vektort pozitívnak és minden negatív vektort negatívnak ismert fel.

Ez nagy segítségemre volt a detektáló alkalmazás fejlesztésében. A modellben tökéletesen megbízva az implementáció számos hibáját megtaláltam és kijavítottam – az implementáció hibái által a multi-rezolúciós képpiramisban felmerülő vakfoltok nyilvánvalóvá váltak.

Ezek a modellek többnyire 150 pozitív példát és 35-50 válogatott negatív példát tartalmaztak. A betanított SVM modell a várakozásoknak megfelelően az egymáshoz „közel eső" pozitív példákból körülbelül 120 darabot eldobott, míg a negatív példák közül csupán csekély számút. A margó egyértelműen széles lehetett, a kisszámú negatív vektor ellenére alig fordult elő téves klasszifikáció a videófolyamon való detektálás során – a szoftver gyorsan és megbízhatóan követte az arcomat. Nem volt alkalmam más alanyon tesztelni ezen széles margó esetleges áldásos hatását, de sejtésként felmerült, hogy más alanyokkal is elfogadhatóan működhet. Amennyiben ez nem így van, a pozitív képek kontrollált készítésének képességét kihasználva más, eltérő arci jellemzőket mutató alany bevonásával az adatbázist olyan módon bővíthetőnek gondolom, hogy a tulajdonság az idealisztikustól csak kevésbé távolodjon el.

Egy másik kísérlet célja egy esetleges lokális komponens alapú megközelítés felé történő irányváltást célozta: a szemeimről készített adatbázis alapján betanított piramis mindössze egyetlen 5x5-ös méretű rétegből állt. A validáció során az SVM modell megközelítőleg ideális eredményeket adott (egy speciálisan szemekhez leválogatott validációs halmazra) és a detektálásnál visszaellenőrizve is csak kevés téves találatot fogadott el. Ezen találatok további szűrése lehetséges teljes lokális komponens alapú elemzés megvalósításával, vagy a globális megközelítés második lépcsőben alkalmazásával a lehetséges találatok osztályozására.

Az alkalmazásprogramozási interfész (API)

A rendszer másik eleme egy alkalmazásprogramozási interfész (API). Ez objektum orientált paradigma szerint absztrahált programozási eszközöket nyújt magasabb rétegbeli szoftverek

számára, amelyek segítségével azok megoldhatják a működésükhöz köthető detektálási problémákat. Itt a cél természetesen a hatékony erőforrás gazdálkodás és magától értetődő, könnyen integrálható OO rendszer kialakítása volt.

Az API .NET alapokra épül, kihasználja a 3.5 által nyújtott lehetőségek egy részét. Az API implementáció forrásnyelve C#3.0, illetve a linq C# nyelvbe integrált deklaratív lekérdezőnyelv. Az implementáció néhol úgynevezett *unsafe* blokkokat tartalmaz egyes kódrészletek gép közelibb implementálása érdekében, így a kép, illetve vektor manipulációs műveletek zömében ilyenek. A probléma számítás intenzív jellegéből adódóan más .NET filozófia idegen pontok is előfordulnak, mint például nem .NET felügyelt memóriaterületek kezelése. A .NET és C# felhasználását a felsőoktatásban népszerűbb Java környezettel szemben az indokolta, hogy a .NET viszonylag elegáns és elfogadhatóan hatékony megoldást kínál a felügyelt világ és az attól valamelyest idegen közvetlen erőforrás használat elegyítésére, amellyel így olyan képfeldolgozási feladatok megoldása is lehetővé válik, mint a videófolyamon történő arcdetektálás.

A detektálás megközelítése

Az arcdetektálás egy folyamat, inputját egy kép jelenti és egy SVM piramis, az elvárt output pedig a (potenciális) találatok bemeneti kép viszonylatában megadott helyzete.

Az SVM piramis illesztéséhez egy másik, a képből képzett multi-rezolúciós képpiramis építése szükséges, amelynek faktora az SVM piramiséval megegyezik. Ha erre a korábbi utalásnak megfelelő módon ***valahova*** illesztjük az SVM piramist és az illeszkedik, akkor a multi-rezolúciós képpiramis adott rétegéről az eredeti képre visszavetítve a találat helyét, az illesztés a keresett arc méretétől nagyban független lesz. A multi-rezolúciós piramis alkalmazásának ehhez köthető vonatkozásai más, a Debreceni Egyetemen végzett munkák kimerítő tájékoztatást adnak [8].

De hova és hogyan is illesztjük az SVM piramist?

Pásztázás

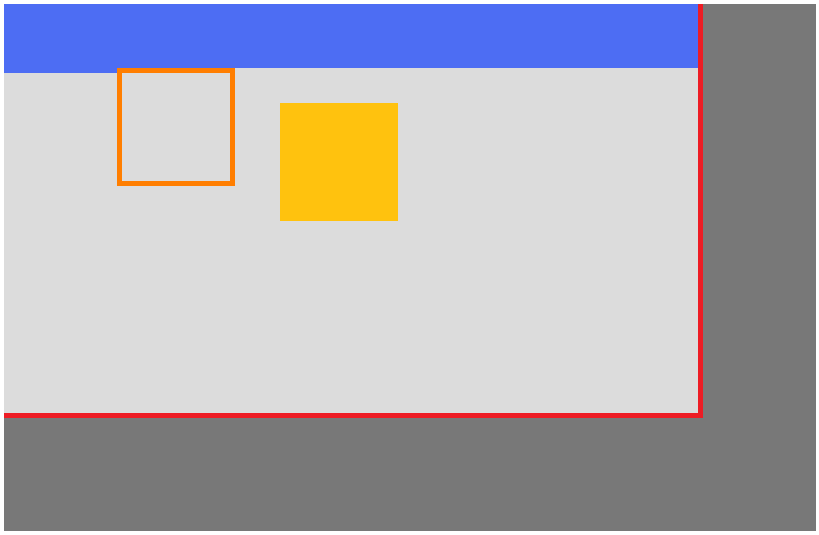
Pásztázásnak a továbbiakban azt a folyamatot fogjuk nevezni, amelynek során a multi-rezolúciós képpiramis megfelelő méretű képrészleteiből rendre vektorokat készítünk és azokra az SVM piramis legkisebb rétegét felhasználva osztályozást végzünk.

Követelmény a pásztázással szemben: minden alkalmas képrészletet csak pontosan egyszer illesszen.

A valós idejűség célkitűzésének eléréséhez a pásztázási folyamat hatékony kialakítása a döntő pont. Egy pásztázás annál hatékonyabb, minél kisebb az elutasító osztályozások várható száma egy elfogadót megelőzően – persze feltéve, hogy rögzített számú arc megtalálása a cél. További fontos követelmény volt a kialakítandó pásztázással szemben, hogy jól párhuzamosítható legyen.

Egyszerű pásztázás

Az alábbiakban tekintsük meg egy naiv pásztázás illusztrációját.



18 Egyszerű pásztázási algoritmus illusztrációja

Világosszürke színűek azok képpontok, amelyek a későbbiek során illesztési pontnak nevezett pontok. Ez egy osztályozandó képrészlet bal felső sarkára utal, inkluzív. A piros színű képrészletek szintén lehetnek illesztési pontok – ezek specialitását azért emeltem ki, mert az ilyen illesztési ponthoz tartozó képrészletek a sötétszürke sávot alkotó képpontokból állnak, melyek az osztályozandó képrészlet méretéből adódóan értelemszerűen nem lehetnek illesztési pontok.

Kékkel a már vizsgált illesztési pontokat jelöltem, a pásztázás jól láthatóan fentről lefelé, balról jobbra történik a bal felső sarokból kiindulva. A sötétnarancs színű keret az éppen feldolgozott képrészletet szimbolizálja, míg a világosnarancs színű téglalap egy feltételezett ténylegesen pozitív képrészletet szimbolizál.

A videófolyamok megkülönböztetése

Mivel a detektálás videófolyamon történik, adódott az ötlet, hogy használjuk ki a helyzet specialitásait. Egy videófolyamon jellemzően vagy nincs a felvétel célját jelentő emberi arc, vagy ha van, akkor valószínűleg központi státuszt élvez, azaz nagy méretben, fókuszban és a kép középső régiójában látható.

Adódott tehát több olyan megközelítés is, amely a pásztázást ebből a központi régióból a szélek felé haladva, a multi-rezolúciós képpiramis legkisebb rétegétől kiindulva végezte, hogy a lehető legnagyobb és leginkább központi szereppel bíró emberi arcot találjuk meg először. Ha ez az a találat, amit keresünk, akkor a találatok számára adott korlát miatt itt a pásztázást meg is szakíthatjuk és hozzákezdhetünk a további feldolgozásokhoz.

Kis kitérőként itt jegyzem meg, hogy az ilyen megközelítések alkalmazása okozza azt az anomáliát, hogy amennyiben az SVM piramis nem képes felismerni ezt a központi szerepet játszó emberi arcot, úgy a felismerés a nagyobb dimenziójú multi-rezolúciós képpiramis rétegeken folytatódik és a (téves) sikertelenség megállapítása ezért hatványozottan több ideig tart, ezzel újabb kapcsolódási pontot adva az SVM modellek minősége és a detektálás sebessége közt.

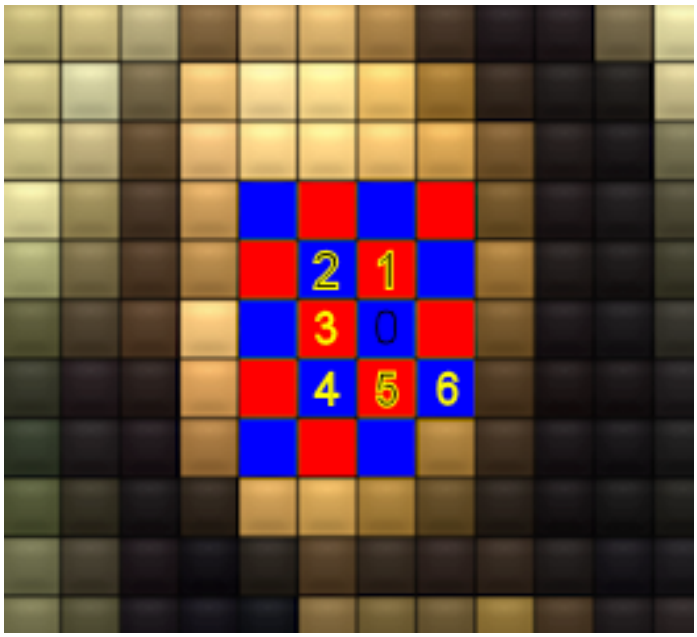
A fejlesztés során számos pásztázó algoritmus kialakult, amely a követelményeknek jól megfelelt, ám nehezen volt párhuzamosítható. A következőkben tekintsük meg az elkészült rendszerben implementált, legjobbnak tekintett megoldást.

A kurrens pásztázási megoldás

A jelenlegi processzorarchitektúrák és operációs rendszerek, vagy adott esetben az x64 és Windows környezet esetén egy feldolgozási szál indítása, leállítása illetve a szálak időosztásos alapon történő osztozása a végrehajtó egységeken viszonylag kevéssé hatékony, sok órajel ciklust elfogyasztó művelet.

Az ideális az lenne, ha a korábban vázolt központi régiótól perifériák felé haladó pásztázást szinte illesztési pontonként eloszthatnánk a végrehajtó egységek közt. Bár egyetlen illesztés a teljes SVM piramisra iszonylag időigényes, szerencsére annyira azért nem, hogy az architektúra figyelembevételével is hatékonyan működő megoldás legyen, de a gondolat közel jár: az illesztési pontokat cellák kialakításával csoportosítjuk. A végrehajtó egységek

közötti legkisebb lehetséges feladat megosztási egység a cella lesz. A következő ábra egy ilyen cella alapú csoportosítást illusztrál.



19 Illesztési pontok egy lehetséges cellás felosztása

Az egyes cellákon belüli hasonló színűnek ábrázolt képpontok lesznek az egy csoportba tartozó illesztési pontok. Figyeljük meg, hogy a cellák nem feltétlen azonos számú illesztési pontot tartalmaznak – a széleken a naiv pásztázásnál már részletezett okok miatt a cellák jellemzően csonkák.

A számozás az a sorrendet mutatja, amelyben a pásztázási megoldás feldolgozza az illesztési pontok csoportjait. Látható, hogy a következő cella megválasztásában a cellának az input kép középpontjához vett távolsága szerepet játszik.

Ha csupán egyetlen végrehajtó szállal rendelkezünk, akkor minden cellát ugyanazon végrehajtó egységen fogunk feldolgozni. Az illusztráció kék illetve piros kódolása azt a szerencsésebb esetet mutatja, mikor a futtató rendszer két végrehajtó egységgel bír. A cellák azért ilyen módon kerülnek szétosztásra, mert így bármelyik végrehajtó egység is találjon előbb megoldást, továbbra is igaz bizonyos megszorításokkal, hogy a lehető leginkább központi találatot szolgáltattuk eredményül.

Az API

A következőkben tekintsük át azt az absztrakciót, ahogyan az API megközelíti a detektálási problémát. Az API teljes osztálydiagramja megtalálható a függelékben.

Az API elemei

A Scanner és Match osztályok kivételével az osztályok példányai a detektálási folyamat során felhasznált erőforrások lesznek, ezen példányok létrehozása az API felhasználó feladata. Ezt alapvetően kétféleképpen teheti meg:

- Konstruktorhívással és a szükséges paraméterek szolgáltatásával.
- Az osztályon értelmezett gyártó osztály metódus hívásával, mely az tréneralkalmazás által felügyelt adatbázisból szerez be minden szükséges információt a példány előállításához. Ilyen módon SVM modell vagy teljes SVM piramis példány legyártására is lehetőség van.

Pattern

Pattern, azaz minta. Az API által használt terminológia az SVM modellre, de azonfelül méreteinek tárolásával logikailag összerendel egy téglalapot egy alkalmas dimenziójú vektorral. A *Pattern* egyedei által leírt téglalap dimenziói annak azon képek dimenziójára utalnak, amelyből a korábban tárgyalt módon képzett vektorok a példányhoz rendelt SVM modellel klasszifikálhatóak lesznek.

A *Pattern* osztály rendelkezésre bocsát osztály szintű gyártó metódust a példány előállítására adatbázisból, ennek neve *LoadFromDatabase*, és paraméterül az adatbázis csatlakozás, az *Pyramid* egyedhez társított szöveges azonosítót és az egyedhez kötött azon *Model* egyed indexét várja, amelyhez rendelt tanító vektorhalmazból az SVM modell tanítását el kívánjuk végezni.

PatternPyramid

A *PatternPyramid* az API által használt terminológia egy teljes SVM modellre. A hozzá köthető adatbázis egyedtípusnak a *Pyramid* egyedtípus tekintendő.

A *PatternPyramid* példányok létrehozása történhet konstruktoron keresztül a piramist alkotó *Pattern* példányok és az egyes *Pattern* példányok dimenziói közötti faktor megadásával, vagy adatbázisból osztály szintű gyártó metódussal. Utóbbi esetben csupán az

adatbázis kapcsolat és a forrásként szolgáló *Pyramid* egyedhez társított azonosító paraméterül szolgáltatása szükséges.

A *PatternPyramid* a rendelkezésére bocsátott *Pattern* példányokat méret szerint rendezi és ellenőrzi, hogy a paraméterül kapott faktor értékkel ténylegesen összeegyeztethetőek-e.

A *PatternPyramid* csupán egy erőforrás a detektálási probléma megoldása során, indexer és property révén rendelkezésre bocsátja a példány által egy egységbe foglalt *Pattern* példányokat.

MatchLayer

MatchLayer logikailag hasonló kapcsolatban van a *MatchPyramid*-dal, mint a *Pattern* a *PatternPyramid*-dal.

A *MatchLayer* példányai egy a MatchPyramid által egységbe zárt multi-rezolúciós képpiramis egyetlen rétegének reprezentációját adják. A réteg tartalma a *Fill* metódussal adható meg.

A kép réteg reprezentáció nem felügyelt memóriaterületen történik az eredeti kép rétegnek megfelelő méretre skálázott és szürkeskálássá alakított verziójának sorfolytonos tárolásával. Ez két dologban is hasznos. Egyfelől a képmanipuláló algoritmusok *unsafe*-k, így egy pixel érték elérése metódus illetve *indexer* hívása helyett nagyságrenddel gyorsabban mutató aritmetikával történik. Másfelől egy nem felügyelt memóriaterület eltér egy felügyelt objektumtól azon a nagyon fontos ponton, hogy míg egy *fixed* blokkal ideiglenesen rögzített felügyelt objektum a *fixed* blokkon kívül a .NET *garbage collector* által szabadon áthelyezhető a memóriafragmentáció csökkentésére, addig egy nem felügyelt memóriablokkal ez nem történhet meg. Ezt kihasználva minden korábbi verziónál hatékonyabb és egyszerűbb pásztázási algoritmus implementációt tudtam elérni.

Cellák kialakítása

A képpontokat tároló nem felügyelt memóriaterület foglalása egyszer történik meg egy *MatchLayer* példány számára, és csak a példány destruktorának meghívásakor szabadul fel. Új képen való detektáláshoz a korábbi réteg memóriaterülete lesz újrafelhasználva.

A réteg fentiek szerinti cellára bontása is csak egyszer, a memóriafoglalást követően történik meg. Mivel a nem felügyelt memóriaterület a fenti rögzített tulajdonsággal bír, itt kihasználjuk, hogy az egyes képpontok memóriabeli címe sem változik (legfeljebb az érték

komponensek, ha új képpel írjuk felül). Minden cella számára egy szintén nem felügyelt memóriablokkot foglalunk, amelynek elemei a cellába sorolt egyes illesztési pontok 32bites címei lesznek és egy lezáró 0 strázsaérték.

Minél kisebb egy cella, annál inkább jó elvben a központi régióhoz közelítés. Hogy a feldolgozási szálak miatt fellépő gyakorlati vonatkozások ellenére is megérje, célszerű egy bizonyos minimális cellaméret felett maradni. Egy SVM modellel történő illesztés meglehetősen lassú, nagyjából fél milliszekundum azon 2Ghz-es Core2Duo magos rendszeren, amelyen dolgozom. Ebből kiindulva a Windows 20 milliszekundumos ütemezési hagyományaira tekintettel általában 6x6-os cellaméretet alkalmaztam. Ez a *MatchLayer* konstruktora számára szabadon megadható.

MatchPyramid

Funkciója és szolgáltatásai teljesen szimmetrikusak a *PatternPyramid*-nál leírtakra, persze annak kivételével, hogy az adatbázisból való létrehozás nem értelmezhető.

A *MatchPyramid* számára értelmezett a *Fill* metódus, amely az összes a multi-rezolúciós képpiramishoz tartozó réteget feltölti a metódus számára paraméterül szolgáltatott képpel.

Match

A *Match* osztály példányai a detektálási folyamat közben azonosított egyes találatokat reprezentálják. A példány tárolja, hogy az SVM piramis legkisebb rétegét a multi-rezolúciós képpiramis melyik rétegére és azon belül pontosan hova illesztve döntött a teljes SVM piramis a képrészlet pozitív volta felől. A találat helyének megadása a forrás képrészlet bal felső sarkának rétegen belüli abszolút koordinátáival és a legkisebb SVM modell dimenzióinak rögzítésével történik, hiszen az illesztett képrészlet akkora volt.

A *Match* osztály példányain elérhető a *GetMatchRectangle* metódus, amely a találathoz tartozó képrészlet helyének és méretének eredeti képre vetített értékeiből példányosított *RectangleF* .NET osztály példányával tér vissza.

Scanner

A *Scanner* osztály implementálja a detektálás folyamatában felhasznált erőforrások felhasználását.

Scanner példány előállítása a detektálás során használandó *PatternPyramid* és *MatchPyramid* példányok konstruktornak átadásával történhet. A két példányhoz megadott faktor értékeknek egyezniük kell, továbbá a multi-rezolúciós képpiramis nem állhat kevesebb rétegből, mint amennyi rétegből az SVM piramis áll.

A *Scanner* példányokon egyetlen metódus, a *Scan* értelmezett, amely a detektálási folyamat belépési pontja. Paraméterül a multi-rezolúciós képpiramis egyes rétegein belül vizsgálandó részterületet meghatározó téglalap normalizált koordinátáit és a detektálás folyamán legfeljebb megtalálni kívánt találatok számát várja. A részterületet leíró téglalap koordinátáinak normalizációja azt jelenti, hogy mind a bal felső sarok koordinátái, mind a szélesség és magasság értékek lebegőpontosan vannak megadva a [0,1] intervallumban, ahol (0,0) koordinátapár az aktuálisan vizsgált réteg bal felső sarkát jelenti, (1,1) a jobb alsót, a kettő közötti értékek pedig a réteg valamely közbenső pontját adja lineáris interpolációval. Hasonlóan értelmezendő a szélesség és magasság.

A könnyebb érthetőségért a *Scan* metódus pszeudókódja a *ScanTask* osztálynál van tárgyalva.

Az API rejtett része – Task API

A következő három API elem az API felhasználó elöl rejtve marad. Az API implementációja által a detektálási probléma párhuzamos megoldásához használt belső API-nak tekintendő. A párhuzamos számítási problémák egy általános, a problémakörön kívül is értelmezhető absztrakcióját nyújtják.

Pszeudó kódok

Az algoritmusokat pszeudókódokkal adom meg. A pszeudókódokban a nyelvi és eszközkörnyezetre utaló azonosítók és foglalt szavak angol nyelvűek, az API-hoz köthető eszközök azonosítói magyar nyelvűek. Az alábbiakban felsorolom a speciálisabb, illetve több algoritmus közt megosztva használatos pszeudókód kiterjesztéseket.

lock(erőforrás név) … end lock	Monitorral védett kritikus régió.
set(szemafor változó)	Szemafor engedélyezett állapotba billentése.
reset(szemafor változó)	Szemafor tiltott állapotba billentése.
waitreset(szemafor változó)	Feldolgozási szál altatása a szemafor engedélyezett állapotáig és a szemafor tiltása továbbhaladás előtt. Az utasítás szálütemezés

	szempontjából egységnyi.
begin(feladat)	Megkezdi az adott feladat végrehajtását egy új szálon.
tulajdonság(objektum)	Tulajdonság értékének elérése.
kollekció(index)	Kollekció elem elérése.
metódus(objektum vagy osztály, paraméterek)	Példányszintű metódus hívása.

ITask

Az *ITask* interfész leírja a párhuzamosan elvégezhető feladatosztályokkal szembeni olyan funkcionális követelményeket, amelyeket egy adott feladat implementációjának teljesítenie kell ahhoz, hogy a *TaskEngine*, azaz a párhuzamos *erőforrásgazdálkodást* vezérlő API által kezelhető legyen.

Egy ilyen feladatosztály ismérvei:

- Az adott feladat osztályba tartozó feladat példányok, azaz a konkrét paraméterekkel társított és megkezdett számítási folyamatok egyedi azonosítóval rendelkeznek.
- Adott feladatról egyértelműen eldönthető, hogy a folyamat egy adott pillanatában lehetséges-e a feladat részfeladatokra bontása. Ez kissé talán ködösen hangzik, de valójában csupán arról van szó, hogy a *TaskEngine* lehetőséget nyújt arra, hogy az olyan feladatok, amelyek inputként egy nagy elemszámú halmazt kapnak és azok elemeire rendre, függetlenül ugyan azt a részfolyamatot ismételik, két vagy több párhuzamos feladat példányra bonthatóak legyenek a még fel nem dolgozott input halmazbeli elemek szétosztásával. A detektálási feladat tipikusan egy ilyen feladat – az input halmaz alatt itt a forráskép illesztésre váró területeit értjük.
- Az adott feladat implementálja a paraméter nélküli *Do* metódust, amelynek végrehajtása a feladat végrehajtását jelenti.
- Az adott feladat implementálja a *Distribute* metódust, amely az input halmaz szétosztását valósítja meg, amennyiben a folyamat részfeladatokra osztható.

TaskEngine

A *TaskEngine* statikus osztályként implementálja a párhuzamos feldolgozási feladatok erőforrás hatékony ütemezését. A *TaskEngine* két kívülről is elérhető metódusai az *Add*, illetve *Ready* metódusok.

Az *Add* metódus használható új feladat példány ütemezésre felkínálására.

```
sub Add(feladat)
   lock (feladatok)
      aktuális := head(feladatok)
      while aktuális <> end(feladatok) and
            prioritás(aktuális) < prioritás(feladat) do
         aktuális := next(aktuális)
      end while

      if aktuális = end(feladatok)
         addtail(feladatok, feladat)
      else
         insertbefore(aktuális, feladat)
      end if

      TryActivate()

   end lock
end sub
```

A metódus az új feladatot egy társított prioritás alapján a már ütemezésre kínált feladatok láncolt listájába szúrja. A kisebb prioritás érték jelöli a fontosabb feladatokat. A láncolt lista bővítését követően meghívja a *TaskEngine* később tárgyalt *TryActivate* privát metódusát, amely a tényleges ütemezésre kísérletet tesz.

A *Ready* metódus használható annak jelzésére, hogy egy feladat feldolgozása véget ért.

```
sub Ready(feladat)
   aktívfeladatN := aktívfeladatN - 1

   lock (feladatok)
      remove(feladatok, feladat)

      TryActivate()

   end lock
end sub
```

Következzék az ütemezés lelke, amely valójában egyszerűbb, mint amilyennek látszik.

A *TryActivate* metódus meghívása a feladatok listájának *Add* illetve *Ready* metódusok által történő manipulációját követi. A *feladatok* feladat példányokat tartalmazó kollekción felül a végrehajtása során felhasználja a *TaskEngine* osztályon értelmezett volatilis *aktívfeladatN* és *maxaktívfeladatN* adattagokat. *maxaktívfeladatN* rendszer által meghatározott kezdeti értéke a futtató rendszerben található párhuzamos feldolgozást végezni képes végrehajtó egységek száma, míg *aktívfeladatN* egy kisebb vagy egyenlő, éppen végrehajtott feladatok számát tartalmazó érték, amelyet *TryActivate Ready*-vel párban tart karban.

A kódban sok a szimmetria, ezek felismerése segíthet, illetve színkódokkal is támogatom a kódot követő magyarázatot – a logikailag egy egységet alkotó részek azonos színűek.

```
sub TryActivate()
   if containselement(feladatok) and
      aktívfeladatN < maxaktívfeladatN
      then
      lock (feladatok)
         if containselement(feladatok) then
            while aktívfeladatN < maxaktívfeladatN
               aktuális := head(feladatok)
               while aktuális<>end(feladatok) and
                     aktív(aktuális) or
                     not osztható(aktuális) do
                  aktuális := next(aktuális)
               end while

               if aktuális<>end(feladatok) then
                  if aktív(aktuális) then
                     prioritás(újfeladat) := prioritás(aktuális)
                     újfeladat := feloszt(aktuális)
                     insertafter(aktuális, újfeladat)
                     begin(újfeladat)
                     aktív(újfeladat) := true
                  else
                     if osztható(aktuális) and
                        maxaktívfeladatN > aktívfeladatN + 1
                        then
                        létrehozandóN := maxaktívfeladatN - aktívfeladatN
                        addfirst(újfeladatok, aktuális)

                        while true do
                           for i := 1 to létrehozandóN do
                              if osztható(újfeladatok(i)) then
                                 addlast(úfeladatok, feloszt(újfeladatok(i)))
                              else
                                 break while
                              end if
                           end for
                        end while

                        for i := 2 to length(újfeladatok) do
                           prioritás(újfeladatok(i)) := prioritás(aktuális)
                           insertbefore(aktuális, újfeladatok(i))
                           begin(újfeladatok(i))
                           aktív(újfeladatok(i)) := true
                           aktívfeladatN := aktívfeladatN + 1
                        end for
                     end if

                     begin(aktuális)
                     aktív(aktuális) := true
                  end if

                  aktívfeladatN := aktívfeladatN + 1
               else
                  break while
               end if
            end while
         end if
      end lock
   end if
end sub
```

A sárgával jelölt rész azt dönti el, hogy elvben lehetséges-e új feladat ütemezése. Ehhez nyílván az kell, hogy kevesebb éppen végrehajtott feladat fusson, mint amennyire lehetőség van, valamint, hogy legyen ütemezésre felkínált feladat példány a *feladatok* kollekcióban.

A *feladatok* kollekció példány szintű metódusai nem szálbiztosak, ráadásul annak tartalmának változását egy felhasználó által másik szálról hívott *Add* vagy egy befejeződő feladat által hívott *Ready* hívás is megkísérelheti, ezért az ütemezési logika (valamint *Add* illetve *Ready* metódusok kollekció manipulációja) során a *feladatok* kollekciót monitorral védjük.

Érdekes lehet, hogy a *lock* blokk előtt, illetve abban is megvizsgáljuk, hogy van-e feladat példány a *feladatok* kollekcióban. Ezt a monitor jellege miatt szükséges – megeshet, hogy a *lock* blokkba való belépés előtt a monitor altatta a szálat, miközben éppen egy olyan *Ready* hívás zajlott, amely kivette az utolsó elemet a feladatok kollekcióból.

A piros kódrészletek egy olyan ciklus élettartamát szabályozzák, amelynek célja, hogy az ütemezési logika adott feladat példányokra nézve mindaddig ismétlődjön, amíg el nem értük párhuzamos futtatási lehetőségeink határát. Ez vagy akkor történik meg, ha nincs több ütemezendő feladat, vagy ha nincs több szabad végrehajtó egység (illetve .NET szemszögből szál, amelyek számát a végrehajtó egységek számában határoztunk meg alapból).

A kék rész már az ütemezési logika központi része. Az *aktuális* nevű lokális változónak meghatározó szerepe van. Értékként a *feladatok* kollekcióba helyezett valamely feladat példányt veheti fel. Hogy éppen melyiket veszi fel, azt egy ciklus határozza meg. A ciklus kilépésekor aktuális értéke vagy a *feladatok* kollekció végjele (.NET implementációban *null*), vagy a feladatokhoz rendelt prioritás értékek alapján a legsürgősebb olyan feladat, amely még nincs aktiválva, vagy aktiválva van, de tovább osztható. Ez a feladatok elvégzését mohóvá teszi. Ilyen módon lehetséges lenne tetszőleges számú képkockára vonatkozó detektálási feladatot ütemezésre kínálni, de mindig a legkorábban átadott kapná a nagyobb figyelmet, feltéve, hogy azonos prioritás értékeket rendeltünk a detektálási feladatokhoz. A megközelítés egymásra utalt heterogén feladatoknál is örvendetes hatású.

Példaként tekintsük azt a két feladat típust ismerő esetet, mikor a detektálási probléma feladat-alapú implementációján kívül rendelkezünk a detektálások eredményét tovább

osztályozó érzelem felismerési probléma feladat-alapú implementációjával is. A detektálási feladat a feldolgozás végére érve egy érzelem felismerési feladat példányt kínál fel ütemezésre a detektálási feladatok osztályához képest „sürgősebb” prioritással, majd befejeződik egy *Ready* hívással. Feltételezve, hogy a korábbi detektálás közben már a videófolyam következő képkockája rendelkezésre állt és a detektálási probléma fel lett kínálva ütemezésre, és, hogy az érzelem felismerési feladat nem osztható, úgy a *Ready* hívás miatt bekövetkező *TryActivate* futás eredményeként egy N végrehajtó egységgel rendelkező környezetben egy egység megkezdi az érzelem felismerést, míg N-1 már a következő detektálási probléma megoldásátnak kezd neki. A példán látható, hogy egymásra épülő többlépéses folyamatok viszonylag kényelmesen programozási modellben jól párhuzamosíthatóvá vállnak – a végrehajtó egységek tétlenkedése jól elkerülhető.

A barna kódrészletekre akkor kerül vezérlés, ha a ciklus eredményeképpen beállított *aktuális* érték aktív, de osztható. Ekkor a feladat felosztásra kerül és a végrehajtásba bevont egységek száma nő.

A zöld és szürke kódrészletekre akkor kerül vezérlés, ha a ciklus eredményeképpen beállított *aktuális* érték egy inaktív és ráadásul osztható feladat példány. A zöld kódrészlet speciális, mert jellemzően csak a rendszer indítását követő erőforrás elfoglalási időszakban fordul elő, hogy egy inaktív feladat ütemezésekor egynél több szabad végrehajtó egység van. *Ready* hívás következményeként felszabaduló végrehajtási egység esetén az eddig inaktív feladat egy befejeződő aktív feladat „helyét veszi át”. Ez utóbbit a szürke kódrészlet végzi el és ez a magyarázat arra is, hogy aktív osztható aktív feladat felosztásakor (barna) miért nem kíséreljük meg egynél több felé osztani a feladat példányt. A zöld rész tehát addig osztja az inaktív osztható feladatot és adott esetben az osztalékokat tovább, míg elegendő inaktív feladat példány rendelkezésre nem áll az összes szabad végrehajtási egység lefoglalásához, majd végrehajtási szálakhoz rendeli őket. Ez a viselkedés eredményezi a lehetőségekhez képest „agresszív” erőforrás kiosztást nyugalmi állapotból indulva.

A pásztázási feladat, ScanTask

A *ScanTask* osztály implementálja az *ITask* interfészt, ezáltal alkalmas a *TaskEngine*-val ütemezésre. A *ScanTask* példányok létrehozása és ütemezése a *Scanner* osztály *Scan* metódusának hívásával közvetett módon történik az alábbi módon:

```
func Scan(maxtalálatN)

   találatok := new list

   for i := 0 to rétegszám(képpiramis) - rétegszám(svmpiramis) do
      rendezettcellák := rendez(x(képpiramis(i)) / 2,
                                y(képpiramis(i)) / 2,
                                cellák(képpiramis(i))
                           )

      reset(pásztázásvége)

      Add(TaskEngine,
          new ScanTask(képpiramis,
                       svmpiramis,
                       i,
                       rendezettcellák,
                       találatok,
                       pásztázásvége,
                       maxtalálatokN
          )
      )

      waitreset(pásztázásvége)

      if length(találatok) >= maxtalálatN then
         break for
      end if
   end for

   return találatok
end func
```

Ez a metódus egyszerű – legfeljebb N darab találat megkeresésének megkezdésére szolgál. A Scanner példányok számára *képpiramis*, *svmpiramis* és *pásztázásvége* példány szintű adattagok. A rendez metódus *MatchLayer* példányokon értelmezett cellák olyan rendezését végzi, amelynek eredményeképpen a cellák adott koordinátáktól vett távolságuk szerint növekvőleg lesznek rendezve.

A detektálás a multi-rezolúciós képpiramis rétegeit kisebbtől a nagyobbig sorra véve, egyszerre csak egy rétegen történik. Ez amiatt a korábban említett kritérium miatt szükséges, hogy mindenképpen a legnagyobb találatot találjuk meg elösször. Ez felvet egy feladat kontrollálási problémát, amit az algoritmus a *pásztázásvége* szemafor használatával old meg.

Rétegenként csak egyetlen pásztázási feladatot osztunk ki – a feladat esetleges lehetséges továbbosztása már a *TaskEngine* problémája lesz. A feladat ütemezésre felkínálása elött a *pásztázásvége* szemafort tiltott állapotúra állítjuk, majd a feladat kiosztását követően rögtön várakozni is kezdünk rá. A *pásztázásvége* szemafor engedélyezett állapotba

billentését majd egy bizonyos pásztázási feladat fogja megtenni a feldolgozásának befejeztével.

ScanTask

A fenti, feladatot jellemző ismérvek:

- Egy *ScanTask* feladat azonosítója egy rendszer által generált128 bites *Guid*.
- Egy *ScanTask* feladat akkor osztható, ha még legalább két olyan feldolgozatlan cella van, amelyhez még nem kezdett hozzá a végrehajtó egység.
- Implementálja a paraméter nélküli *Do* metódust. A pásztázáshoz szükséges paraméterek a példány konstruktorán keresztül kapja meg.
- Implementálja a *Distribute* metódust. A *Distribute* metódus a feladat szétosztását a még fel nem dolgozott cellák sorának egyenlő mértékben, az elemeket felváltva megtartva illetve kiemelve valósítja meg. Az ily módon létrehozott kiemelt feladatokat tartalmazó sort egy új *ScanTask* példány létrehozásakor a konstruktorban annak átadja.

A következő pszeudókódok a pásztázás folyamatát, illetve a pásztázási feladat leállásakor elvégzett teendőket részletezik.

Do

```
sub Do(képpiramis, svmpiramis, i, rendezettcellák, találatok, pásztázásvége,
futópásztázásN, maxtalálatokN)

   futópásztázásN := futópásztázásN + 1

   while true do
      lock (találatok)
         if elemN(találatok) >= maxtalálatokN
            break while
         end if
      end lock
      cellamutató := dequeue(rendezettcellák)
      if cellamutató <> end(rendezettcellák) then
         while indirection(cellamutató) <> 0 do

            if osztályoz(indirection(cellamutató),
                         képpiramis(i),
                         svmpiramis(i)
               ) = pozitív then
               #illesztési pont vetítése felsőbb rétegekre
               #illesztés a nagyobb mintákkal a nagyobb
               #képpiramis rétegekre
               #csupa pozitív esetén találat mentése találatok-ba
            end if
            cellamutató := cellamutató + 1
         end while
```

```
      else
         break while
      end if
   end while

   Ready(TaskEngine, this)
   Dispose(this, pásztázásvége, futópásztázásN)
end sub
```

A pásztázás algoritmusa egyszerű a cellák kialakításának köszönhetően, ennek magja a kékkel jelölt iteráció. A cellamutató maga egy mutató – arra a nem felügyelt területre mutat, ahova ”felsoroltuk” a cellába tartozó illesztési pontok 32 bites címeit és a 0 strázsát. Nincs más teendő tehát, mint sorra venni az egyes illesztési pont címeket és meghívni az osztályozás metódust. A .NET implementációban ennek neve *PredictPoint* és a képrészlet kiemelését, vektorizációt és annak bináris osztályozását is elvégzi akorábban leírtak alapján. Erről és az SVM piramis alkalmazásáról a felsőbb rétegre való illesztéskor további részletekhez juthatunk Szabó Péter többször hivatkozott diplomamunkájából [8].

A zöld kódrészletek a pásztázási folyamat, míg a piros részek a feladat példány életciklusát szabályozzák. A pásztázást végző ciklusból akkor lépünk ki, ha vagy elegendő találat áll már rendelkezésre vagy nincs több feldolgozatlan cella. A feladat példány mindenképpen befejeződik, de vajon időközben történt-e feladat osztás, ki engedélyezi a *pásztázásvége* szemafort, amire az egész feladatot ütemezésre felkínáló *Scan* metódus azóta is várakozik?

Erre a kérdésre a *Dispose* metódus adja meg a választ, amelyhez nem szükséges további magyarázat.

Dispose

```
sub Dispose(pásztázásvége, futópásztázásN)
   futópásztázásN := futópásztázásN - 1
   if futópásztázásN = 0 then
      set(pásztázásvége)
   end if
end sub
```

Összefoglalás

A dolgozatban egy nagyobb ember-gép kapcsolatok javítására törekvő projekt általam fejlesztett részeit tárgyaltam és összefoglaltam a témával kapcsolatos tapasztalataimat, következtetéseimet, mindezek átadására törekedve. Fontosnak tartottam, hogy az egyes pontok kialakítása mögötti motivációk világosak legyenek, hogy azok továbbfejlesztésére és

a bennük rejlő hibák felfedezésére az érdeklődőknek lehetőséget adjak a kutatás folytatását támogatva.

A fejlesztés eredményeképpen elkészült egy alkalmazás programozási interfész, amely elvben képes a detektálási probléma globális módszerekkel történő jó minőségű megoldására, ám a gyakorlatban a modell kialakításával kapcsolatos részeknél tárgyaltak miatt az eredmények az elvárt szint alatt maradnak.

Azért, hogy ezt orvosolni tudjam, egy kifejezetten képadatbázis adatbázis kezelésére szolgáló alkalmazás fejlesztésébe is belefogtam, amellyel valamelyest sikerült emelni a detektálás minőségét. Mivel az adatbázis mérhetővé és vizsgálhatóvá vállt, így adott kérdésekre választ kaptam és persze számos kérdéssel is gazdagabb lettem.

Továbbfejlesztési lehetőségek

A dolgozat folyamán több ponton is ismertettem az adott fejezethez köthető meglátásaimat, esetleges kísérleteimet. Ezeket összefoglalva a továbbfejlesztésre a globális megközelítést alapulvéve a következőket tartom érdemesnek:

- asszimetrikus modellből épített piramis – alacsonyabb rétegeken pozitív túlsúlyos (és lineáris?), felsőbb rétegeken negatív túlsúllyos modell felhasználásával
- csak a piramis két szélső rétegére történjék illesztés
- az SVM paraméterek felügyelet nélküli optimalizációjának felhasználása a jobb minőségű SVM piramis rétegek kialakításához
- *jellemző kiemelés* változatása a probléma dimenziójának jobb redukciójához
- több alanyról részletes, változatos meta-adat konfigurációjú képsorozatokat tartalmazó adatbázis kiépítése, külön figyelemmel az érzelmi adatbázisra
- grafikus vektor processzorok felhasználása a párhuzamos feldolgozás előtérbe helyezése helyett, mellett

Irodalomjegyzék

Julia M. Taylor, Lawrence J. Mazlack, "Reverse Engineering Humor," Proceedings CogSci 2007, Nashville, August, 2007

Wikipedia definition

B. Moghaddam, W. Wahid, and A. Pentland. Beyond eigenfaces: probabilistic matching for face recognition. In Proc. IEEE International Conference on Automatic Face and Gesture Recognition, pages 30–35, 1998.

M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In Proc. of Computer Vision and Pattern Recognition, pages 586-591. IEEE, June 1991b.

http:/www.cs.wisc.edu/dyer/cs540/handouts/mturk-CVPR91.pdf

http://digital.library.adelaide.edu.au/coll/special/fisher/138.pdf

P. Belhumeur, P. Hespanha, and D. Kriegman. Eigenfaces vs fisherfaces: recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):711–720, 1997.

A Tutorial on Support Vector Machines for Pattern Recognition CHRISTOPHER J.C. BURGES,Bell Laboratories, Lucent Technologies

Szabó Péter, Diplomamunka, Multi-modális ember-gép kapcsolat képfeldolgozási aspektusa, 2006

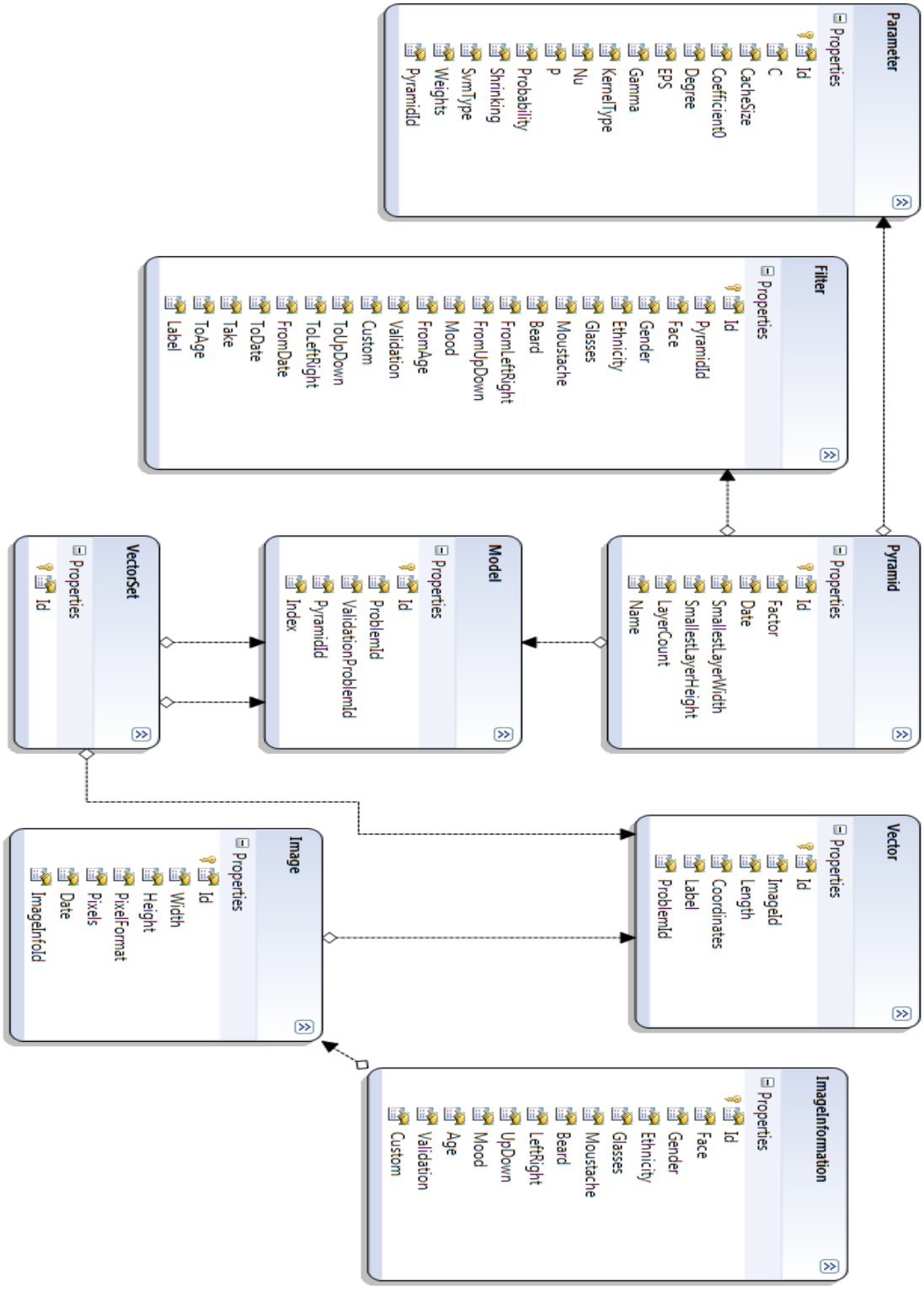
Thorsten Joachims [2004]: SVM^{light}, http://svmlight.joachims.org

Ábrajegyzék

10.o. 1 Nagyfelbontású, jellemző kiemelésen már átesett pozitív példakép
10.o. 2 Eugenface képek az AT&T kutató intézetéből
11.o. 3 Paolo Abeni a Telecom Italia-tól idealizált kísérletében
12.o. 4 Komponensek között definiált gráf
13.o. 5 Egy egyszerű neurális hálózat
15.o. 6 Szeparálás lineáris kernel függvénnyel egy nem lineárisan szeparálható esetben (7)
16.o. 7 Szeparálás 3-ad fokú polinomiális kernel fügvénnyel (7)
23.o. 8 A tréneralkalmazás import ablaka
24.o. 9 A tréneralkalmazás felvételi mód közben
25.o. 10 Vektor generálás lépései
27.o. 11 A tréneralkalmazásból szűrő beállítási ablaka
29.o. 12 Piramis paraméterek megadása
31.o. 13 Egy SVM piramis két alsóbb rétegéhez illeszkedő vektor képként
35.o. 14 A legjobb heterogén piramis riportja 3-ad fokú polinomiális kernelfüggvény esetén
38.o. 15 A legjobb heterogén piramis riportja aszimmetrikus tanítóhalmaz bővítést követően
39.o. 16 A legjobb heterogén piramis riportja lineáris kernelfüggvénnyel esetén
40.o. 17 A legjobb heterogén piramis riportja lineáris kernelfüggvénnyel esetén C=1000-re
43.o. 18 Egyszerű pásztázási algoritmus illusztrációja
45.o. 19 Illesztési pontok egy lehetséges cellás felosztása

Függelék

Az adatbázis ORM modellje



Az API osztálydiagramja