Tartalomjegyzék

Ábrák jegyzéke

1. fejezet

Bevezető

Napjainkban a személyi számítógépek, az internet mindennapi életünk szerves részét alkotják, amit 50 évvel ezelőtt elképzelhetetlennek tartottak olyannyira, hogy haszontalan és bukásra ítélt ötletnek bélyegezték. Az internet térhódításával még érzékelhetőbb lett a számítógép nélkülönözhetettlensége, és az internetes alkalmazások nélkül elképzelhetetlen a XXI. század. Azonban az internet „erejét" a mögötte álló hatalmas mennyiségű információ, adat jelenti.

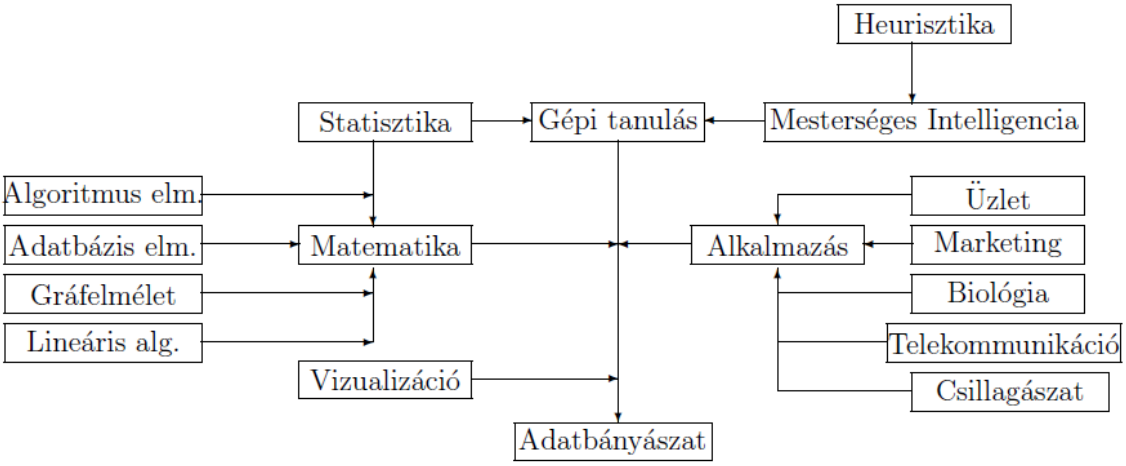
Diplomamunkánk célja, hogy az adatbányászat és ezen belül is a webes adatbányászat területén alkalmazott eljárásokat és algoritmusokat tanulmányozzuk, továbbá a webes adatbányászaton belül, a web használat, web struktúra bányászattal kapcsolatban készített webes alkalmazásunk ismertetése.

A 90-es években a tárolókapacitások méretének igen erőteljes növekedése, valamint az árak nagymértékű csökkenése miatt az elektronikus eszközök és adatbázisok a hétköznapi életben is mind inkább elterjedtek. A tárolókapacitás növekedése még Moore jóslatát[1] is jócskán felülmúlja, az utóbbi 15 év alapján ugyanis a tárolókapacitás 9 hónaponként duplázódik meg. Az egyszerű és olcsó tárolók a nyers, feldolgozatlan adatok tömeges méretű felhalmozását eredményezték, ezek azonban az egyszerű visszakeresésen és ellenőrzésen kívül nem sok mindenre használhatóak. A ritkán látogatott adatokból így "adat temetők" (data tombs) alakultak ki, amelyek tárolása haszon helyett költségekkel jártak. Ekkor még nem álltak rendelkezésre olyan algoritmusok, amelyekkel ezen adatokból hasznos információkat tudtak kibányászni.

Egyre több területen merült fel az igény, hogy hasznos információkat nyerjenek ki az adathalmazokból. A hagyományos adatbázis-kezelő rendszerek –a közvetlen kereső-

[1] 1965. április 19-én az Electronics Magazine-ben jelent meg egy Gordon Moore által írt cikk, amely megjósolta, hogy az integrált áramkörök száma évente megduplázódik majd, folyamatos költségcsökkenéssel.

kérdéseken kívül, illetve az alapvető statisztikai funkciókon túl (átlag, szórás,maximális és minimális értékek meghatározása) – komplexebb feladatokat egyáltalán nem tudtak megoldani, vagy az eredmény kiszámítása elfogadhatatlanul hosszú időbe telt. A szükség egy új tudományterületet keltett életre, az adatbányászatot[2], amelynek célja :„hasznos, látens információ kinyerése az adatokból". Az adatbányászatot az üzleti élet hívta életre és ma is ez a fő mozgató rugója. Az adatbányászati algoritmusokat az adatbázisból történő tudásfeltárás során alkalmazzák. A tudásfeltárás az adatbázisból érvényes, hasznos és érthető minták feltárását jelenti. Egyes esetekben ez megoldható különféle lekérdezések vizsgálatának segítségével, ám ez túlnyomó részben hosszadalmas és drága eljárás ugyanakkor nem elég átfogó. Fontos szerepet kap a vizsgálók szubjektivitása, ami nem minden esetben helyes.



1.1. ábra. Az adatbányászat területei

Az adatbányászat számos területet foglal magába (1.1 ábra) és mára összetett ágazattá nőte ki magát. [1]

[2]az adatbányászat, mint önálló tudományterület a 90-es évek elején jelent meg

2. fejezet

Adatbányászati alapfogalmak

2.1. Mi is az adatbányászat?

Az adatbányászat bár egy új kutatási- illetve tudományterület, ennek ellenére számos olyan problémát magába foglal, amely nem új keletű. Az osztályozás tekinthető függvény approximációnak, a klaszterező algoritmusokkal már az 1950es évek végétől foglakoztak. Az adatbányászatban gyakran nem a probléma az új, hanem a feldolgozandó adatok mennyisége (TB) valamint a feldolgozási sebesség. Cél skálázható és hatékony algoritmus készítése, amelyek nagyméretű adatbázisok esetén is számunkra elfogadható idő alatt futnak. Az adatbázis mérete megköveteli sokszor, hogy az algoritmus párhuzamosítható legyen. Az adatbázisok valamilyen objektumok tulajdonságait, attribútumait írják le és az adatbányászat e tulajdonságok rejtett kapcsolatainak feltárásával foglakozik. Számos matematikai és informatikai területet felhasznál ezek kutatása során, ilyenek:

1. Halmazok, relációk, függvények, sorozatok,
2. Valószínűség számítás, statisztika (Hipotézisvizsgálat, F-próba, Függetlenségvizsgálat, Student t-próba),
3. Matematikai logika,
4. Algoritmus-elmélet,
5. Adatstruktúrák (Szófák, Piros-fekete fák).

A statisztika és az adatbányászat közötti határ nagyon vékony, nehéz elkülöníteni a két területet egymástól. A statisztika a hipotézisek vizsgálatára helyezi a fő hangsúlyt, míg az adatbányászat a hipotézisek megtalálásának módjára.

Általánosabban kétféle adatbányászati tevékenységet különböztetünk meg:

Feltárás : A feltárás során az adatbázisban található mintákat keressük meg. A minták legtöbbször az általános trendeket/szokásokat/jellemzőket írják le, de vannak olyan alkalmazások is (például csalásfelderítés), ahol éppen az általánostól eltérő/nem várt mintákat keressük.

Előrejelzés : Az előrejelzésnél a feltárt minták alapján próbálunk következtetni a jövőre. Például egy elem ismeretlen értékeit próbáljuk előre jelezni az ismert értékek és a feltárt tudás alapján.

2.2. Adatbányászatot használunk...

Az „adat bányászata" eredetileg statisztikusok által használt kifejezés, az adatok nem kellőképpen megalapozott alkalmazására, amely során valaki nem megfelelő következtetést von le. Ugyanis tetszőleges adatbázisra igaz, hogy ha kellő ideig figyeljük az adatokat, fellelünk benne egy struktúrát, amely lehet haszontalan vagy esetleg téves következtetések oka.

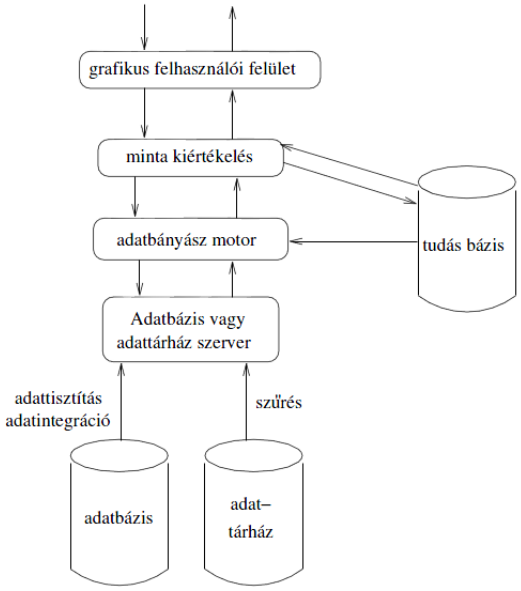
Az adatbányászatot a kialakulásának elején Davide Rhine az 50-es években végzett parapszichológiai kísérlete tette hírhetté. A kísérlet célja diákok parapszichológiai képességének mérése. 10 lefordított kártya színét kellett megmondaniuk. A kísérlet eredményeként a diákok 0,1%-a rendelkezik ilyen képességgel. Miután felvilágosították a diákokat képességükről ismét megismételték a kísérletet, de ekkor már átlagosan teljesítettek. A magyarázat az volt, hogy aki rendelkezik valamilyen parapszichológiai képességgel, miután megtudja, elveszti képességét. A következtetés nyilvánvalóan nevetséges. Annak statisztikai esélye, hogy valaki eltalálja mind a 10 lap színét nagyjából 0,1%. Mára számos helyen alkalmazzák kiemelkedő sikerrel[1]. Néhány példa:

1. A vásárlói szokások felderítése nagy hipermarketekben hasznos lehet reklámok, akciók, leárazások esetén.
2. Géntechnológia területén, a cukorbetegséget okozó géncsoportok felderítése.
3. Csillagászatban az égitestek felismerésére.
4. A vírusölő programok az új, ismeretlen vírusokat a már ismert vírusokat leíró osztályozás alapján találják meg.

5. A mesterséges megtermékenyítés során a legjobb esélyekkel induló embriók kiválasztása, stb.

2.3. Adatbányászati rendszer architektúrája

Az adatbányászati rendszernek kapcsolatban kell lennie elsősorban egy adatbázissal, egy felhasználóval és esetleg egy tudásalapú rendszerrel. Egy ilyen architektúrát ír le a 2.1 ábra.



2.1. ábra. Példa klaszterezésre

Nagyon sok projekt nem váltotta be a hozzá fűzött reményeket, ennek egyik oka a jó adatbányász szakember hiánya, másrészt nem teljesített alapkövetelmények. Egyik legfontosabb a terület szakértőjének és az adatbányász szakember együttműködése, valamint a következők:

Nagy mennyiségű adat: A szabályok statisztikai jelentőségét növelik, minél nagyobb az adatmennyiség annál inkább kizárható a véletlen eredmény. Hátrány, hogy sok adatot sokáig tart feldolgozni, valamint a nem párhuzamosítható algoritmusok érzékenyek arra, hogy az adatmennyiség elfér-e a memóriába. Az utóbbi ritkán szokott igazán gond lenni a mai memóriák mérete miatt.

Sok attribútum: A kevés attribútum maga után vonja a kevés kinyerhető tudást. Az adatbányászat akkor tud igazán érvényesülni, ha sok attribútumok jellemzik az

objektumokat, és a hagyományos módszereknek nincs esélyük (grafikonok, egyszerű ábrák és táblázatok).

Tiszta adat: Az adatok jó minősége nagyon fontos. A zajos adatok jó esetben csak nehezítik az adatbányászatot, rosszabb esetben hibás eredményt adhatnak.

2.4. Előfeldolgozás

2.4.1. Hasonlósági függvények

Először is a mérési skálák 4 típusáról beszélhetünk:

1. kategória típus, az attribútumok között csak egyenlőséget tudunk vizsgálni, speciális esete a bináris attribútum
2. sorrend típus, az attribútum értéken teljes rendezést tudunk megadni
3. intervallum típus, értelmezzünk egy + függvényt, amivel az elemek csoportot alkotnak
4. valós attribútum vagy arány skála, ha megadunk az intervallum típusú attribútumnak egy zérus elemet, azaz az elemek gyűrűt alkotnak

Ezeken a típusokon különböző hasonlósági mértékeket definiálhatunk. Minél több attribútumuk egyezik meg annál jobban hasonlítsanak az elemek. A gyakorlatban a hasonlóságot nehezebb mérni, mint a különbözőségeket, ezért különbözőségi mértéket használunk(, vagyis minél jobban hasonlóak annál kevésbé különböznek). A különbözőséget távolságként definiáljuk (d(x,y)≥), ez a távolság metrika kell legyen, tehát

- önmagától ne különbözzön $d(x, x) = 0$
- szimmetrikus legyen $d(x, y) = d(y, x)$
- teljesítse a háromszög egyenlőtlenséget $d(x, y) \leq d(x, z) + d(y, z)$

A típusokon definiált néhány metrika:

Bináris attribútum. Definiálhatunk variáns és invariáns hasonlóságot. Az invariáns hasonlóságot akkor használják, amikor a bináris attribútum két értéke ugyanolyan fontos. Ilyen például egy ember neme. A távolságot a

$$d(x, y) = \frac{r+s}{m}$$

képlettel számoljuk. Az aszimmetrikus hasonlóságot a

$$d(x, y) = \frac{r+s}{m-t}$$

képlettel, ahol m darab bináris attribútumunk van, r és s azon attribútumok száma ahol az x és y eltérő értéket vesznek fel, t ahol mind a ketten 0-t

Kategória típusú attribútumok. Nem csak kettő, hanem véges sok különböző értéket vehetnek fel, például hajszín, vallás, stb. A hasonlóságot a

$$d(x, y) = \frac{u}{m}$$

képlettel számoljuk, ahol u azt adja meg, hogy az m (összes) közül mennyi nem egyezett.

Sorrend típusú attribútum. A sorrend típusú attribútumokat általában egész számmal helyettesítik. Ha több olyan attribútum van, amely a fontos állapotokban eltérnek, akkor ezeket leképezik a [0,1] intervallumra, ezután alkalmaznak valamilyen intervallum típusú hasonlóságot.

Intervallum típusú attribútum. Ilyen egy ember magasága, súlya, éves átlaghőmérséklet, stb. Ezeket általában valós számokkal írják le. Tekinthetünk egy elemre úgy mint egy pontra az m-dimenziós vektortérben. Az elemek közötti különbséget a vektoraik közötti különbségek normáival definiáljuk. Legelterjedtebb a Minkowski-norma, de alkalmazzák ennek speciális eseteit is, a Manhattan- és Euklideszi- normát is.

Vegyes attribútumok. Ha egy objektum leírásánál vegyesen adottak a különböző típusú attribútumok, akkor csoportosítsuk őket típusuk szerint, határozzuk meg a két elem hasonlóságát minden csoportra nézve, a kapott hasonlóságot képezzük a [0,1] intervallumba. Minden attribútumhoz feleltessünk meg egy dimenziót, így két elem hasonlóságához hozzárendelhetünk egy vektort a vektortérben. A vektort feleltessük meg a vektor hosszának. Hátránya, hogy ha például egyetlen kategória típusú attribútum van, akkor az ugyanolyan súllyal fog szerepelni, mint akár tíz bináris attribútum összesen. Célszerű ezért az egyes attribútum típusok által szolgáltatott érétkeket súlyozni a hozzájuk tartozó attribútumok számával.

2.4.2. Attribútum transzformációk

Hiányzó értékek kezelése. Nagyon gyakori és sokszor végzetes hiba, amikor néhány elem attribútuma hiányzik, ugyanis az adatbányászati algoritmusok csak olyan elemekkel tudnak kalkulálni, amelyeknek az összes attribútuma meg van adva. Ilyen hibás például a magyarországi orvosi adatbázis is, de azon kívül, hogy néhány vizsgálatot fölöslegesnek ítél meg az orvos és a vizsgálat eredményének fenntartott mezőbe nem ír be semmit gyakori a hibásan kitöltött adat is. Ezeket a cellákat fel kell tölteni valamilyen értékkel, legjobb ezekre külön attribútum értéket bevezetni, de szoktak mediánt is

számolni, vagy esetleg új elemeket létrehozni. Ezek a megoldások csak kényszerűek, a legjobb, ha megadunk minden attribútumot.

Attribútumok törlése. Az adatbányász algoritmusok elvileg törlik, vagy figyelmen kívül hagyják a fölösleges attribútumokat (döntési fák csak a lényeges adatokat veszik figyelembe), de azért ha tehetjük, ne rontsunk zajokkal a rendszerünk teljesítményén, hacsak nem az a célunk, hogy teszteljük.

Diszkretizálás. A szám típusú attribútumot kategória típusúvá alakítjuk. A diszkretizáció során információt vesztünk, de segítünk az algoritmusnak. Az attribútumok értékkészletét intervallumra osztjuk és minden intervallumhoz egy kategóriát rendelünk. Diszkretizáló eljárást tartalmaz például az 1R(2.6.4) nevű osztályozó algoritmus is.

Adathalmaz csökkentése.

Mintavételezés. Vannak olyan esetek, amikor a gyorsaság sokkal fontosabb, érdekesebb számunkra, mint a pontosság. A teljes adatbázison történő feldolgozás sokszor annyira lassú, hogy az adatbázis kis szeletét dolgozzuk fel. Ez azzal járhat, hogy a kapott eredmény nem elég pontos. A hiba mértékét csak az összefüggések kinyerésének ismeretében tudjuk megmondani.

Tegyük fel, hogy elemek egy halmazából egy tetszőleges x elem előfordulásának valószínűsége p valamint m minta áll rendelkezésünkre. A mintavételezés hibája:

$$hiba(m) = \mathbf{P}(|rel.gyakorisg(x) - p| \geq \epsilon)$$

Dimenziócsökkentés. A nagy adathalmazt le lehet csökkenteni az attribútumok dimenzió-csökkentésével is. Ez arról szol, hogy az objektumok sok attribútummal való leírását szeretnénk helyettesíteni kevés attribútumot használó leírással. Az eredeti adatbázist egy $m*n$-es mátrixszal reprezentáljuk. E helyett egy $m*k$-s mátrixszal dolgozunk. Az n lehet akkora, hogy az adatbázis nem fér el a memóriában, de ha sikerül egy olyan k-t választani, amely akkorára csökkenti az adatbázist, hogy az már elférjen a memóriában, akkor sokat gyorsítottunk az eljárásunkon.

2.5. Gyakori elemhalmazok

A feladat vásárlói szokások kinyerésénél merült fel részfeladatként, ugyanis a nagy profitot a gyakran együtt vásárolt termékek jelentik. Például az online áruházak böngészése esetén a felhasználónak fel tudjuk ajánlni a termékek azon csoportját melyet a nagy többség is megvásárolt.

2.5.1. Gyakori elemhalmaz fogalma

Legyen $\mathcal{I} = i_1, i_2, ..., i_m$ elemek halmaza és $\mathcal{T} = (t_1, t_2, ..., t_n)$ az $\mathcal{I}$ halmaz felett értelmezett *bemeneti* sorozat. A $t_i$elemeket *tranzakcióknak* nevezzük. Továbbá definiálnunk kell néhány fogalmat:

Fedés: Egy $I \subseteq \mathcal{I}$ elemhalmaz fedése megegyezik azon tranzakciók halmazával, amelyeknek a részhalmaza az I.

Támogatottság: Egy I halmaz támogatottsága a fedések elemszámával egyezik meg.

Gyakori halmaz: Egy I gyakori, amennyiben támogatottsága nem kisebb egy előre megadott *min_ supp* konstansnál, ezt *támogatottsági küszöb*nek hívjuk.

A gyakori elemhalmazoknál adott egy $\mathcal{I}$ elemhalmaz egy $\mathcal{T}$ sorozat, *min_ supp* támogatottsági küszöb, feladatunk megtalálni a gyakori elemhalmazokat és azok támogatottságát. A bemeneti sorozaton tudunk egy rendezést definiálni és a továbbiakban ezzel a rendezett halmazzal dolgozunk. A keresést úgy képzeljük el mint egy irányított gráfban való keresést, amelynek csúcsai az elemhalmazok és I_1-ből él indul $I_2 - be$, ha $I_1 \subset I_2$ és $|I_1|+1 = |I_2|$. A tranzakciók egy *szűrt* halmazával dolgozunk tovább, miután töröltük belőle a ritka elemeket. Beszélhetünk:

- Horizontális: a tranzakciókat azonosítóval látjuk el és minden azonosítóhoz tároljuk a tranzakcióban található elemeket.
- Vertikális: minden elemhez tároljuk az elemet tartalmazó tranzakció azonosítóját.
- Relációs: az első elem a tranzakciót a második az attribútumot írja le. Ez a manapság leghasználatosabb.

adatbázis tárolási módról.[1]

2.5.2. Gyakori elemhalmazt kereső algoritmusok

APRIORI algoritmus

Az egyik legkorábbi algoritmus. Szélességi bejárást használ, az üres halmazból kiindulva halad szintenként előre a nagyobb méretű gyakori elemhalmazok megtalálásához. Az iterációk száma legfeljebb eggyel több, mint a legnagyobb elemhalmaz mérete. Egy iterációban csak olyan jelöltet veszünk fel, amelynek összes valódi részhalmazáról tudjuk, hogy gyakori.

Apriori (T,min_supp)
$l \leftarrow 0$
$J_l \leftarrow \{\emptyset\}$
while $J_l \neq 0$
támogatottság_meghatározás(T, J_l);
$GY_1 \leftarrow$ gyakoriak_kiválogatása(J_l, min_supp);
$J_(l+1) \leftarrow$jelölt_előállítás(GY_l);
$l \leftarrow l + 1$;
end while
return GY

A jelölt_előállítás függvény az l elemű gyakori halmazokból $l+1$ elemű jelölteket állít elő. Csak azok az elemhalmazok lesznek jelöltek, amelynek minden részhalmaza gyakori. A jelölt előállítása során olyan l elemű, gyakori I_1, I_2 elemhalmaz párokat keresünk, amelyekre igaz :

- I_1 lexikografikusan megelőzi I_2-t,
- I_1-ből a legnagyobb elem törlésével ugyanazt az elemhalmazt kapjuk mintha töröltük volna a legnagyobb elemet.

Ha a feltételnek megfelelő párt kaptuk, akkor képezzük a pár unióját, majd ellenőrizzük, hogy a kapott halmaznak minden valódi részhalmaza gyakori-e. Az alulról fölfelé építkezés miatt mindig elegendő az l+1 darab l elemű részhalmaz gyakoriságát vizsgálni.

A **támogatottság_meghatározás** függvény nagyobb elemhalmazok esetén. Adott egy tranzakció és l méretű jelöltek egy halmaza. Meg kell határozni azokat a jelölteket, amelyek a tranzakció részhalmazai. Egy egyszerű megoldás, ha a jelölteket sorra vesszük és megnézzük, hogy tartalmazza-e őket a tranzakció. Ez azonban lassú, ha sok a jelöltünk, ugyanis annyiszor kell a tranzakciókon végig haladni, amennyi jelöltünk

van. Ezen gyorsíthatunk, ha a jelölteket szófában tároljuk. A szófa éleinek címkéi elemek lesznek, minden csúcs egy elemhalmazt reprezentál, mégpedig a gyökérből az illető csúcsig vezető út címkéit. A gyorsaság abból ered, hogy a közös prefixeket csak egyszer tárolja, ezért szokták még prefixfának is nevezni. A támogatottság meghatározása mellett a jelölt előállítását is segíti, ugyanis két halmaz akkor lesz generátor, ha a legnagyobb elemük elhagyásával ugyanazt az elemhalmazt kapjuk, vagyis $l-1$ hosszú prefixei megegyeznek. Az algoritmus kezdetekor a szófa csak egy csúcsból áll, ez tartalmazza az üres halmazt, a támogatottság meghatározása után töröljük azokat a levél elemeket, amelyek számlálója kisebb min_suppnál.

Az Apriori algoritmus gyorsasága nagymértékben függ a szófa méretétől. A halmazok elemein értelmezzünk egy rendezést és ez alapján építjük fel a szófát. A szófa memóriaigény arányos a szófa pontjainak számával, így törekednünk kell a minimális szófa előállítására ez azonban egy nehéz feladat (NP). Tapasztalat szerint a gyakoriság szerinti csökkenő rendezés kisebb szófát ad mint a növekvő rendezésű vagy akár a véletlenszerűen választott rendezések. Ennek ellenére a növekvő rendezésű szófát célszerű választani, mert a csúcsok kisebbek lesznek (kevesebb él indul ki belőlük), valamint a ritka elemek közel lesznek a gyökérhez ezért a támogatottság meghatározásakor a szófa kisebb részét kell bejárni.

Az APRIORI algoritmuson tovább lehet gyorsítani néhány egyszerű technikával. Ilyen a ritka jelöltek törlése, a bemenet tárolása, tranzakció szűrése, és nyesés (equisupport és zsákutca).

Egy másik ilyen technika a Borgelt-féle támogatottsági meghatározás. Az ötlet az, hogy két tranzakció a közös prefixig ugyanazt a programfutást eredményezi a támogatottság meghatározásakor. Ha szófában tároljuk a tranzakciókat, akkor rendelkezésre áll minden szükséges információ a közös prefixről. Megoldható, hogy ugyanazokat a prefixeket csak egyszer dolgozzuk fel és ne annyiszor ahányszor előfordulna. A tranzakciófában minden csomóponthoz egy számlálót rendelünk, egy I halmaz számlálója azoknak a tranzakcióknak a számát tárolja, amelyek prefixe I.

BORGELT (n_c, n_t, l, i)
if $l = 0$ then
$n_c.szamlalo \leftarrow n_c.szamlalo + n_t.szamlalo$
else
for $j = 0$ to $n_t : elszam - 1$ do
while $i < n_c.elszam$ AND $n_c.el[i].cimke < n_t.el[j].cimke$ do
$i \leftarrow i + 1$
end while
if $i < n_c.elszam$ AND $n_c.el[i].cimke \geq n_t.el[j].cimke$ then
BORGELT $(n_c, n_t.el[j].gyermek, l, i)$
if $n_c.el[i].cimke = n_t.el[j].cimke$ then
BORGELT $(n_c.el[i].gyermek, n_t.el[j].gyermek, l - 1, 0)$
$i \leftarrow i + 1$
end if
else
break
end if
end for
end if

ECLAT algoritmus

Az ECLAT az üres mintából indul, egy rekurzív, mélységi bejárást megvalósító algoritmus. A rekurzió mélysége legfeljebb egyel több mint a legnagyobb gyakori elemhalmaz mérete. Az APRIORI-val szemben mindig egyetlen jelöltet állít elő, majd ennek azonnal meghatározza a támogatottságát. Az $l+1$ elemű, P prefixű jelölteket, P prefixű gyakori elemhalmazokból állítja elő egyszerű unióképzéssel. Egy elemhalmaz TID(Transaction IDentifier) halmazának elemei azon bemeneti sorozatok azonosítói, amelyek tartalmazzák az adott elemhalmazt.

ECLAT (T, min_supp)

támogatottság_meghatározás(T, J_1)
$GY_1 \leftarrow$ gyakoriak kiválogatása$(J_1; minsupp\ )$
for $i \leftarrow 1$ to |T| do
for all $j \in t_i \cap GY_1$ do
$j.TID \leftarrow j.TID \cup \{i\}$
end for
end for
return $GY_1 \cup ECLAT - SEGED(\emptyset; GY_1; min\_supp)$

ECLAT-SEGED(P, GY^P, min_supp)
for all $gy \in GY^P$ do
for all $gy' \in GY^P$; gy $<$ gy' do
$j \leftarrow gy \cup gy'$
$j.TID \leftarrow gy.TID \cap gy'.TID$
if $|j.TID| \geq min\_supp$ then
$GY^g y \leftarrow GY^{gy} \cup \{i\}$
end if
end for
if $|GY^{gy}| \geq 2$ then
$GY \leftarrow GY \cup GY^{gy} \cup$ ECLAT-SEGED$(gy; GY^{gy}; min_supp)$
else
$GY \leftarrow GY \cup GY^{gy}$
end if
end for
return GY

Az ECLAT jelölt előállítás megegyezik az APRIORI jelölt előállítással, azzal a különbséggel, hogy nem ellenőrzi az unióképzéssel kapott halmaznak minden részhalmazára, hogy gyakori-e, valamint egy jelölt előállítása után azonnal meghatározza a támogatottságát, mielőtt egy újabb jelöltet állítana elő. A mélységi bejárás miatt egy jelölt előállításánál nem áll rendelkezésünkre az összes részhalmaz, ezért az ECLAT legalább annyi jelöltet állít elő, mint az APRIORI. Ebből a szempontból rosszabb, mint az APRIORI, de az ECLAT erőssége a jelöltek támogatottságának meghatározásában van,

ugyanis a TID halmaz előállítása igen egyszerű és gyors. Ahogy haladunk előre a mélységi bejárás során úgy csökken a TID halmaz, ezzel szemben az APRIORI-nál ahogy haladunk az egyre nagyobb jelöltek felé, úgy nő a szófa mélysége és lesz egyre lassabb minden egyes jelölt támogatottságának meghatározása.

FP-GROWTH algoritmus

Egy mélységi bejárású, rekurzív algoritmus, amely a támogatottság meghatározását az egyelemű gyakori halmazok meghatározásával, majd a bemenet szűrésével és vetítésével valósítja meg. A szűrés a tranzakciókból a ritka elemek törlését jelenti. A T elemhalmaz P elemhalmazra történő vetítését (T|P) úgy kapjuk meg, hogy vesszük a T-ből a P-t tartalmazó tranzakciókat, majd töröljük ezekből a P-t.

Először meghatározzuk azon elemek támogatottságát, amelyek előfordulnak valamely tranzakcióban, ezekből kiválasztjuk a gyakoriakat. Ezután minden *gy* gyakori elemhez meghatározzuk a vetített bemenetet, majd meghívjuk az algoritmust a T|gy bemenetre. A jelölt előállítás a legegyszerűbb, ha az I gyakori, akkor a következő rekurziós szinten azon $I \cup j$ lesznek a jelöltek, ahol a j az I-re vetített bemenetben előforduló elem és $I \cup j$ nem volt jelölt korábban. A szűrt bemenetet egy FP-fában tároljuk, ami egy kereszt élekkel és egy fejléc táblával kibővített szófa. Ezen fa segítségével könnyen előállíthatjuk a vetített bemenetet, azokban könnyen meghatározhatjuk az elemek támogatottságát, és előállíthatjuk a vetített majd szűrt bemenetet, amit szintén egy FP-fában (vetített FP-fa) tárolhatunk. Az $FP - growth^*$ az FP-growth egy módosítása, gyorsabban állítja elő a vetített fát, ami viszont a memória rovására megy.

2.6. Osztályozás

Ismeretlen változók, attribútumok értékének előrejelzése más, ismert változókból nagyon fontos kutatási terület. Az előrejelzés folytonos értékű függvények modellezése, vagyis ismeretlen, vagy hiányzó értékek előrejelzése, míg az osztályozás tanító adatbázis alapján osztályozza az adatokat (osztály-címke) (modell-készítés), majd ezt használja fel az új adatok osztályozásában. Számos felhasználási területe van, néhány ezekből :

1. hitelbírálat
2. célpiacok meghatározása
3. egészségügyi diagnózisok, kezelések hatékonyságának elemzése

Az adatbányászatban a leggyakrabban használt előrejelző és klasszifikáló algoritmusok a következők:

1. legközelebbi szomszédok módszere
2. lineáris és logisztikus regresszió
3. mesterséges neurális hálózatok
4. döntési szabályok, sorozatok és fák
5. naiv bayesi klasszifikáció és bayesi halózatok

Ezek az eljárások kétlépcsős algoritmusok, először az adatbázison felépítjük a modellt, majd ezt alkalmazzuk olyan új adatokra, amelyen a cél változó értéke nem ismert de ismerni szeretnénk.

A modell alkotás az ismert besorolású (előzetesen osztályba sorolt) esetek tulajdonságainak leírása, osztályozási szabályok, döntési fák, vagy matematikai képletek reprezentálják a modellt. Minden eset tartozzon egy előre definiált osztályba, amelyet az osztály-címke tulajdonsággal határozunk meg, az osztályokba sorolt esetek, mint tanító adatbázis használatos a modell alkotásban.

A modell alkalmazása ismeretlen, vagy a jövőben keletkező esetek osztályozására. A tesztelésre kijelölt adatbázis osztályozásának eredményét összehasonlítjuk az esetek ismert besorolásával. A modell pontosságának becslése: pontosság mértéke a helyesen osztályozott esetek aránya. Követelmény, hogy a teszt adatbázis legyen a tanító adatbázistól független.

Amikor osztályozó algoritmust vélasztunk figyelnünk kell arra, hogy eleget tegyen néhány feltételnek:

1. gyorsaság: mennyi idő szükséges a modell felépítéséhez és alkalmazásához
2. robusztusság: mennyire érzékeny a hibás, hiányzó adatokra
3. skálázhatóság: használható-e nagy adatbázisokra is
4. értelmezhetőség: a modell által feltárt tudás érthetősége

A klasszifikáló vagy előrejelző algoritmusunk jóságát mérhetjük is. Legyen Y a cél, X pedig a magyarázó attribútumunk és f az algoritmusunk, ekkor az $E(U(Y, f(X)))$ [1] maximálása a célunk.

Az osztályozás során n-esekkel (tuple) dolgozunk, ezeket nevezzük objektumoknak (elemeknek). Ezen objektumok egy sorozatát tanító mintának, halmaznak vagy pontnak nevezzük. Egy objektum j-edik elemét az objektum j-edik attribútumának nevezzük és minden attribútum saját értékkészlettel rendelkezik.

Néhány a legismertebb osztályozó algoritmusokból:

2.6.1. k-legközelebbi szomszéd módszere

Azon a feltevésen alapul, hogy a többdimenziós adatokból kiszámolt távolságok alapján megállapítja a legközelebbi szomszédokat, és a viselkedésükből kiindulva von le következtetéseket az egyénre. Az adatok összefüggéseit nem fejti meg és nagy elemszám esetén túlságosan nagy erőforrás szükséglete van, ezért kisebb adatbázisokra vagy az adatbázisból kinyert adathalmazokra alkalmazzák.

Ez egy „lusta" eljárás, mivel nem épít modellt, alapötlet, hogy a hasonló atribútumú objektumok hasonló tulajdonsággal bírnak, a hasonlóságot távolságfüggvényekkel mérjük. A módszer tekinthető egyfajta lokális súrűségfüggvény becslő eljárásnak. A lényege, hogy a tanuló adatbázist eltároljuk és amikor egy ismeretlen objektumot kell osztályozni, akkor egy távolságfüggvény szerinti k darab legközelebbi pontot megkeressük és abba a kategóriáába soroljuk az új objektumot, amely a legtöbbször fordul elő a k szomszéd között.

Bináris osztályozás esetén a k-legközelebbi szomszéd módszerét értelmezhetjük úgy, hogy tetszőleges pontra kiszámítjuk a

$$f(x) = Med(y_i | x_i \in N_k(x)) \ ,$$

[1] U(x,y)-jelöli az y előrejelzett hasznosságát miközben a valódi érték y

ahol $N_k(x)$ az x legközelebbi szomszédját, a Med az átlagot jelöli. Ha $f(x) > 0.5$, akkor az 1-es osztályba, különben a 0-ás osztályba esik.

Tetszőleges ponthoz közel lesznek a szomszédai, ha sok tanítóponttal rendelkezünk, valamint az átlag egyre stabilabb lesz ha minnél nagyobb a k. Azonban a legtöbb esetben nem áll rendelkezésünkre sok tanítópont. A k-legközelebbi módszer egy univerzális aproximátor, tetszőleges függvényt képes reprodukálni, csak elég tanítópont kell hozzá. Csak alacsony dimenzóknál működik jól. Ha azt szeretnénk, hogy a keresési térben a tanítópontok sűrűsége rögzített legyen, akkor a tanítópontok számának exponenciálisan kell nőnie a dimenzió növelésével, a tanítópontok véges száma miatt a dimenziók száma is eléggé behatárolt. A módszer továbbá érzékeny a független attribútumokra, pl. euklideszi távolság alkalmazása esetén a független attribútum elnyomhatja a függő attribútumokat. Erre egy megoldás lehet, az osztályozni kívánt elemek alkalmazási területét ismerő szakértő bevonása, kinek segítségével eldönthetjük, hogy melyik attribútumokat vegyük számításba, a távolság meghatározásánál.

2.6.2. Lineárisan szeparálható osztályok

Egy osztályt lineárisan szeparálhatónak nevezzünk, ha egy hipersík segítségével el tudjuk különíteni a két osztály pontjait. Ennek a hípersíknak a pont dimenziójához képest eggyel kevesebb dimenziójúnak kell lennie. A hipersík képlete:

$$w_1a_1 + w_2a_2 + ... + w_na_n = 0$$

Az új elem attribútumainak w súllyal vett értékét meghatározva, attól függően, hogy ez az összeg nagyobb mint nulla, akkor az első osztályba , különben a másodikba soroljuk. Adott tanítóponthoz több hipersík is létezhet, amellyel kettéválaszthatjuk az osztályokat.
Néhány ezen a módszeren alapuló tanulási szabály:

Perceptron

Mind az n attribútumnak valósnak kell lenniük, a hipersík n+1 dimenziójú lesz, ugyanis felveszünk egy extra attribútumot, amelynek értéke mindig 1. Az algorimus hátránya, hogy könnyen végtelen ciklusba kerülhet, ha a tanítópontok nem szeparálhatóak.

Perceptron(T)

$\overrightarrow{w} = (0, 0, ..., 0)$
while van rosszul osztályozott $t \in T$ do
for all minden $\overrightarrow{t} \in T$ do
if $\overrightarrow{t}$ rosszul van osztályozva then
if $\overrightarrow{t}$ az első osztályba tartozik then
$\overrightarrow{w} = \overrightarrow{w} + \overrightarrow{t}$
else
$\overrightarrow{w} = \overrightarrow{w} - \overrightarrow{t}$
end if
end if
end for
end while

Amíg van rosszul osztályozott t pont, addig módosítjuk a hipersíkot úgy, hogy a rosszul osztályozott tanítópont közelebb kerüljön a hipersíkhoz, vagy akár a sík másik oldalára is kerülhet.

Winnow módszer

Akkor alkalmazható, ha minden attribútum bináris. Abban tér el a Perceptron tanulástól, hogy rossz osztályozás esetén a súlyvektorhoz nem hozzáadjuk a tanítópont vektorát, hanem megszorozzuk egy $\alpha > 1$ számmal.

Rocchio eljárás

Az eljárás feltételezi, hogy minden attribútum valós típusú. Minden c kategóriához megalkotunk egy prototípusvektort, amit a D_c tanulópéldák átlagaként számítjuk ki és ehez hasonlítjuk az ismeretlen dokumentum vektorát. Az osztályozandó objektum és a prototípusvektor távolságát távolságát koszinusz vagy más távolságmértékkel számolják, aminek kicsi a számításigénye, ezért nagyon gyors. Ha egy osztályba tartozó objektumok nem jellemezhetők egy vektorral, akkor rossz eredményt ad.

Lineáris és logisztikus regresszió

A fejlettebb módszerek alapja a lineáris regresszió. Feltételezzük, hogy az X magyarázó változó n dimenziós, és a magyarázandó Y változóval lineáris kapcsolatban van.

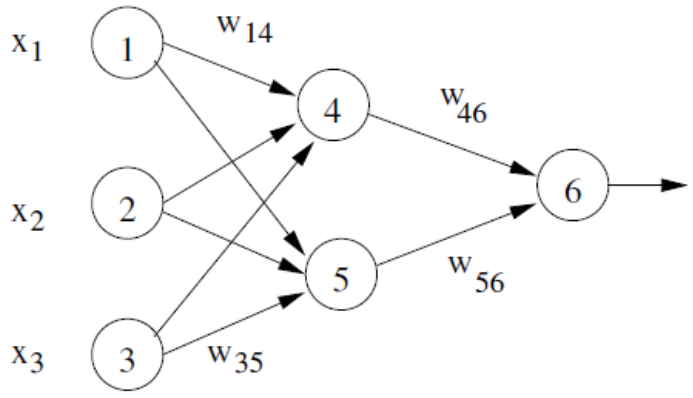
$$Y = \beta_0 + \sum_{j=1}^{n}, X_j\beta_j$$

Lineáris regresszióra visszavezethető számos nem lineáris kapcsolat is.
Ha $Y = \frac{1}{1+e}$, akkor az úgynevezett logisztikus függvényt kapjuk.

2.6.3. Mesterséges neurális hálózatok

A logisztikus regresszió meg tudja tanulni az *és, vagy, nem* logikai függvényeket, de a nem lineáris függvényekkel, mint például a *xor* nem tud mit kezdeni, ennek leírásához szükségünk van három logisztikus regresszióra. Azonban az elmondható, hogy az építőelemeket ismerve tetszőleges logikai formulát kifejezhetünk logisztikus regressziók összekapcsolásával.

A mesterséges neuronhálózatok a logisztikus regressziók kapcsolata. A legnépszerübb modell a többrétegü előrecsatolt hálózat (2.2 ábra). Az input az első réteg csomópontjaiban (neuronjaiban) (1,2,3), az output a legutolsó réteg csomópontja (6), a közbenső réteg a rejtett réteg (4,5). Egy réteg neuronjainak bemenete az előző réteg összes neuronjának kimementelével kapcsolatban van, ezen kapcsolatok szorosságát a w_{ij} súlyok jelzik.



2.2. ábra. Többrétegü előrecsatolt neurális hálózat

A súlyok megtalálása a tanítópontok alapján az ugynevezet hiba visszaterjedés eljáráson alapszik. Először az inputokon előrehaladva kiszámítjuk az outputok eredményét, majd az utolsó output rétegről rétegre visszafelé haladva a megfelelő szabály szerint módosítjuk a w_{ij} értékeket.

A neuronhálók hátránya, hogy a súlyokat nem tudja az ember értelmezni, nem tudja, hogy mi alapján hozta a rendszer a döntést, ezért szokás fekete dobozként is tekinteni.

Ezért például a pénzügyi befektetések során nem szokták alkalmazni még akkor sem, ha jól szerepelne, mivel az ügyfeleknek el kell magyarázni, hogy hogyan oszályoz, ez azonban nem megy. Egy példa a '80-as évekből, az amerikai hadsereg azt akarta elérni, hogy a tankjaik mesterséges intelligencia segítségével ismerjék fel az ellenséges tankokat egy közeli erdőben. Ennek megvalósításához 100 olyan képet tanítottak meg a tanknak ahol rejtőzik ellenség az erdőben és 100 olyat, amikor nem. A hálózat tökéletesen osztályozta a képeket, azonban a tesztelésnél ezzen programmal ellátott tankok nem reagáltak az ellenségre, potossabban nem haladta meg egy teljesen véletlenül tippelő osztályozó pontosságát. A hiba oka állítólag az volt, hogy azokon a képeken, ahol szerepelt tank borus volt az idő, amelyeken nem pedig sütött a nap. Ezzel a példával azt akartuk érzékeltetni, hogy a neurális hálók többnyire jól osztályoznak, de nem adnak magyarázatot az osztályozás okára.[1]

2.6.4. Döntési szabályok

Az $\mathcal{A}$ attributumhalmaz felett értelmezett döntési szabály alatt olyan $R: \Phi(\mathcal{A}) \to Y$ logikai implikációt értünk, amelyek feltételrészében attributumokra vonatkozó feltételek logikai kapcsolatai állnak, a következményrészben pedig az osztályattribútumokra vonatkozó ítélet.

Egy ilyen Φ szabályra illeszkedik a t objektum, ha a feltételrész attribútumváltozóiba a t megfelelő értékeit helyettesítjük, akkor igaz értéket kapunk.

Szabályhalmazok és szabálysorozatok

Szabályhalmazok esetén a szabályok függetlenek egymástól, a szabályhalmaz egyértelmű, ha egy objektum csak egy szabályra illeszkedik. Sorozat esetén az új objektum jóslásánál sorra vesszük a szabályokat, addig amíg nem illeszkedik a szabály az objektumra. Sorozat esetén az egyértelműség a sorba haladással definiálva van.

Egy szabályrendszer teljes, ha tetszőleges objektum illeszthető egy szabályra.

Döntési táblázatok

A döntési táblázat minden oszlopa egy attribútumnak felel meg, az utolsó az osztályattribútumnak. A táblázat egy sora egy döntési szabályt rögzít. Ha az attribútumok a sorban szereplő szabályt kielégítik, akkor az osztályattribútum értéke megegyezzik a sor utolsó értékével.

1R algoritmus

Kiválaszt egy attribútumot és az osztályozásban csak ezt használja. Annyi szabályt állít elő, ahány értéket felvesz a kiválasztott attribútum a tanítóhalmazban. A nevében szereplő 1-es azt fejezi ki, hogy az osztályozás során hány attribútumot használ. Létezik 0R algoritmus is, ekkor az osztályozó egy feltétel nélküli szabály, amely ítéletrészében a leggyakoribb osztály áll.

2.6.5. Döntési fák

A döntési fák bonyolult összefüggéseket egyszerű döntések sorozatára vezetnek vissza. Automatikusan felismerik a lényegtelen változókat, ezért a zajok nem rontják a fa teljesítményét, valamint a problémamegértésünket is segítik, ha megtudjuk melyik változók fontosak. Egy ismeretlen minta klasszifikálásakor a fa gyökeréből kiindulva a csomópontokban feltett válaszok szerint addig lépkedünk, amíg egy levélbe nem érünk.

A fát a tanító adatbázisból rekurzívan állítjuk elő. Kiindulunk a teljes tanulóhalmazból és egy olyan kérdést keresünk ami szerint a halmaz jól vágható szét. A különváló részekre rekurzívan alkalmazzuk ezt az eljárást, egészen addig, amíg a következőkből valamelyik be nem következik:

- a csomópont elemei ugyanabba az osztályba tartoznak
- nincs több attribútum ami alapján az elemeket tovább osztanánk
- nem tartozik az adott csomóponthoz tanítópont
- az adott mélység elért egy korlátot
- nincs olyan vágás ami javítani tudna az aktuális osztályozáson.

Minden levélhez hozzá kell rendelni a magyarázandó változó egy értékét, az lesz a döntés, amely kategóriába a legtöbb minta tartozik. A döntési fák előállítására a legnépszerűbb algoritmusok az IDE3, CART, CHAID.

2.7. Klaszterezés

Klaszterezésen elemek csoportosítását értjük, az adatbányászat legrégebben és leggyakrabban alkalmazott része, számos esetben alkalmazzák: csoportosítanak weboldalakat, betegségeket, géneket, stb. A klaszterezés attól különbözik az osztályozástól, hogy itt nincsenek megadva, hogy melyik elem melyik osztályba tartozik, ezt nekünk kell meghatároznunk. Ezért szokták a klaszterezést felügyelet nélküli tanításnak is nevezni. A klaszterezéskor meg kell adnunk az adathalmazt és egy hasonlósági definíciót, ami szerint értelmezzük az elemek hasonlóságát, valamint, hogy mi alapján osztályozunk. Ezt a „jóság" definíciót nehéz egzaktan megadni. Továbbá problémát jelent még az óriási keresési tér is, ha n pontot akarunk k csoportba sorolni, akkor a lehetséges csoportosítások száma:

$$S_n^k = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-1} \left(\begin{array}{c} n \\ r \end{array} \right) i^n.$$

2.7.1. Klaszterező algoritmusok

Nem létezik olyan távolságalapú eljárás, amely minden adatbázis esetén jól teljesít, mondja ki Jon Kleinber az „An Impossibility Theorem for Clustering (A Klaszterezés Lehetetlenség-tétele)" című könyvében.

A klaszterező algoritmusokat 5 csoportba sorolhatjuk:

Particiós módszer A pontokat k diszjunkt csoportra osztják úgy, hogy minden csoportba legalább egy elem kerüljön. A csoportok a klaszthereknek felelnek meg. Ezt a particionálást addig hajtjuk egymás után végre, amíg az elemek mozognak.

Hierarchikus módszer A klaszterekből egy hierarchikus szerkezetet épít fel, ez az esetek nagy részében egy fa.

Spektrál módszer Az adathalmazt reprezentáló mátrix sajátértékeit és sajátvektorait használják fel.

Sűrűség-alapú módszerek Egy klasztert addig növesztenek, amíg a sűrűség a szomszédságban meghalad egy bizonyos korlátot. A klaszterezés mellett kivételek, kívülálló elemek meghatározására is használják, ugyanis igaz, hogy adott sugarú körön belül mindig megtalálható bizonyos számú elem.

Grid-alapú módszerek Az elemeket rácspontokba képezik le. Ezek az algoritmusok ott használatosak, ahol a gyorsaság a fő szempont.

Szinte bármely klaszterező algoritmushoz tudunk kreálni olyan adatbázist, amit ő fog legjobban csoportosítani. Ahoz, hogy egy algoritmust jónak minősítsünk a végeredményen kívül még több feltételnek is eleget kell tennie:

1. Skálázhatóság: Egyes algoritmus csak akkor hatékony, ha az elemek elférnek a memóriában.
2. Adattípus: Vannak algoritmusok, amik csak az intervallum típusú attribútummal megadott elemekkel működnek.
3. Előzetes ismeretek: Néhány algoritmusnak előre meg kell adni a várt klaszterek számát.
4. Zajos adatok kezelése: Figyelni kell arra, hogy a távol eső adatok ne befolyásolják nagymértékben az algoritmust.
5. Sorrendre való érzékenység: Az eredményként kapott klaszterek nem függhetnek az adatok feldolgozásának sorrendjétől.
6. Elemezhetőség: Az algoritmus mennyire érthető.

2.7.2. Particionáló eljárások

A particionáló algoritmusoknál a csoportok száma előre adott (k). Első részben particionáljuk az elemeket, majd a következő lépésekben csak elemeket kell mozgatni a particiók között, akkor kerül egy elem áthelyezésre, ha ezzel javítunk a klaszterezés minőségén.

k-közép algoritmus

Ez az egyik legrégebbi és legegyszerűbb klaszterező algoritmus vektortérben elhelyezkedő elemek csoportosítására. A klaszterező minőségének jellemzésére a négyzetes hibafüggvényt használja.

Az algoritmus működési elve, hogy először ki kell választani k véletlen elemet, ez reprezentálja a klasztereket, majd a többi elemet elhelyezni ezekben a klaszterekben, attól függően, hogy melyik elemhez a leginkább hasonló (besorolás). Ezután új reprezentáns pontot, általában a középpontot választva újra besorolunk. Ezt a két műveletet addig végezzük, amíg van elemmozgás. Létezik egy ehhez hasonló algoritmus, ami csak annyiban különbözik ettől, hogy nem a középpontot választjuk új reprezentánsnak, hanem a

középpont és a régi reprezentáns között elhelyezkedő pontot. Ez az algoritmus kisebb oszcillációval működik.

Ezen algoritmusok előnye, hogy egyszerű és jól skálázható, azonban hátrány is van: lehet, hogy az algoritmus megáll, azonban létezik olyan csoportosítás, ahol a négyzetes hiba kisebb; csak olyan elemekre alkalmazható, amelyek vektortérben vannak megadva, a középpont értelmezése miatt.

k-medoid algoritmus

Az előző algoritmus hibáin próbál javítani, a klasztert nem egy középpont reprezentál, hanem a leginkább középen elhelyezkedő elem, a medoid. Így nincs szükségünk, az elemek vektortérben való megadására. Két fő algoritmus létezik: a PAM és a CLARA.

A PAM eljárás menete teljesen megegyezik a k-közép algoritmus menetével, a középpontok helyett medoidot választunk, új medoidot pedig akkor választunk, ha ezzel csökken a négyzetes hiba. Az algoritmus sokkal lassabb a k-közép algoritmusnál.

A CLARA algoritmus a PAM kiterjesztése, nem vizsgál meg minden medoid párt csak az adathalmaz egy részével, mintájával dolgozik. Így gyorsul az eljárás, de nem garantált, hogy ebben a mintában benne van az optimális medoid.

2.7.3. Hierarchikus eljárások

A hierarchikus eljárások lényege, hogy az elemeket, klasztereket hierarchikusan hozzuk létre és tároljuk. Kétféle elképzelés létezik: a fentről építkező (osztó) valamint a lentről építkező (egyesítő). A lentről építkező kezdetben minden elemet külön klaszterként fog fel, majd próbálja a hasonló klasztereket egyesíteni. A fentről építkező kezdetben egy klasztert ismer, ebben szerepel az összes elem, majd ezt a klasztert próbálja részklaszterekre bontani. Az eljárások kényes pontja az egyesítendő vagy osztandó klaszterek kiválasztása. Miután egy klaszterből létrehoztunk alklasztert (egyesítés vagy osztás) a továbbiakban az alklaszteren dolgozunk.

Kapcsolási (linkage) eljárások

Egyesítő eljárás. Távolságmárixszal dolgozik, feltéve, hogy elfér a memóriában. A távolságmátrix egy felső háromszögmátrix, amelynek ij-edik eleme tárolja az i és j elem távolságát. Amennyiben két klaszter távolságát a legközelebbi pontjaik távolságával definiáljuk, akkor az eljárást *single linkage* eljárásnak nevezzük. Jól alkalmas tetszőleges alakú jól elkülönülő klaszterek felfedezésére, amennyiben a klaszterek túl közel kerülnek

egymáshoz, akkor hajlamos összekötni őket. Továbbá érzékeny a kívülálló pontokra. Ha a minimális távolság helyett a maximálisat választjuk két klaszter távolságának meghatározásakor, akkor *complet linkage*, ha pedig az átlagos hasonlóságot, akkor *average linkage* eljárásról beszélünk.

A BIRCH eljárás

Nagy adathalmazokra találták ki, csak vektortérben megadott elemeket tud klaszterezni. Egy klasztert egy $CF = (N, LS, SS)$ hármas jellemez, ahol N a klaszterben található elemek száma, $LS = \sum \overrightarrow{x_i}$ és $SS = \sum |\overrightarrow{x_i}|$. A CF egy klaszter statisztikai jellemzőit tárolja a nulladik, az első és a második momentumokat. Az algoritmus során a CF értékeket tároljuk a bennlévő elemeket nem, ebből ki tudjuk számítani egy klaszter átmérőjét vagy két klaszter átlagos távolságát. Az algoritmus egy CF fát épít fel, a levelekben CF értékek vannak, egy belső pont pedig a pontból induló alfához tartozó klaszterek egyesítéséből kapott CF-értéket tárolja. A fának két paramétere van, az első határozza meg a pontból induló ágak maximális számát, a második adja meg, hogy mekkora lehet maximálisan a levelekhez tartozó klaszterek átmérője.

A BIRCH algoritmus két lépésben működik: először az elemeket végigolvasva felépítjük a CF fát, a második lépésben minden elemet besorolunk valamely klaszterbe, majd ebből kiindulva lefuttathatunk egy particionáló algoritmust. A BIRCH algoritmus nem skálainvariáns.

CURE eljárás

A CURE (Clustering Using REpresentatives) eljárás átmenet a BIRCH és a Single linkage eljárás között (a BIRCH-ben a középpont a reprezentáns pont, a Single linkageben a klaszter összes elemét számon tartjuk). A CURE algoritmusban egy klasztert c (konstans) elem jellemez. A CURE nem ragaszkodik az elliptikus klaszterekhez. Kezdetben minden klaszter egy elemet tartalmaz, majd egyesítjük a klasztereket.

A reprezentáns pontok alapján kiválasztjuk a két legközelebbi klasztert. Ezeket egyesítjük, majd kiválasztunk c elemet, amely jól fogja reprezentálni az új klasztert. A reprezentáns pontokat „összehúzza”, azaz az általuk kijelölt középpont felé mozdulnak úgy, hogy az új távolság a középponttól az α -szorosa legyen az eredeti távolságnak. Ennek a lépésnek a célja az outlierek hatásának csökkentése. Az egyesítés akkor ér véget, ha a klaszterek száma eléri a k-t. Ha véletlen mintákkal dolgozunk, akkor nagy adatbázisokra is alkalmazható.

Az algoritmusnak számos hibája van. Az elemek vektortérben adottaknak kell lenniük. Minden klasztert azonos számú reprezentáns pont jellemez, holott létezhetnek kisebb és jóval nagyobb klaszterek is. A reprezentáns pont kiválasztása sem kifinomult, nem a klaszter alakját fogja meghatározni, hanem egy konvex burkot. Klaszter egyesítése után a reprezentáns pontokat összehúzuk a középpont felé. Valamint rosszul kezeli az eltérő sűrűségű pontokat.

2.7.4. Sűrűség-alapú módszerek

A sűrűség-alapú módszerek szerint egy klaszteren belül jóval nagyobb az elemek sűrűsége, mint a klaszterek között. Ez alapján lehet elválasztani a klasztereket és az outliereket.

DBSCAN algoritmus

Ez a legelső sűrűség-alapú eljárás. A sűrűség meghatározásához két paramétert használ, egy sugár jellegű mértéket(eps) és egy elemszám küszöböt (minpts). A p elem szomszédjai ($N_{eps}(p)$) azok az elemek, amelyek p-től legfeljebb eps távolságra vannak. A q elem p-ből sűrűség alapon közvetlen elérhető, ha $q \in N_{eps}(p)$ és $|N_{eps}(p)| > minpts$. A q elem sűrűség alapon elérhető p-ből, ha léteznek $p_1 = p, p_2, ..., p_n = q$ elemek úgy, hogy p_{i+1} sűrűség alapon közvetlen elérhető p_i-ből. A p és q sűrűség alapon összekötöttek, ha létezik olyan o elem, amelyből p és q sűrűség alapon elérhető. Az elemek egy C részhalmaza klaszter, ha:

1. Ha $p \in C$ és q sűrűség-alapon elérhető p-ből, akkor $q \in C$
2. Ha $p, q \in C$, akkor p és q sűrűség alapon összekötöttek.

Egy elemet zajnak hívunk, ha nem tartozik egyetlen klaszterbe sem. Legyen a C klaszter egy p eleme olyan, hogy $|N_{eps}(p)| > minpts$. Ekkor könnyű belátni, hogy C megegyezik azoknak az elemeknek a halmazával, amelyek p-ből sűrűség-alapján elérhetők. E tulajdonságot használja az algoritmus. Válasszunk egy tetszőleges elemet (p) és határozzuk meg a sűrűség alapján elérhető elemeket. Amennyiben $|N_{eps}(p)| > minpts$ feltétel teljesül, akkor meghatároztunk egy klasztert. A feltétel nem teljesülése nem jelenti azt, hogy az elem zaj, lehet, hogy a klaszter határán helyezkedik el. $|N_{eps}(p)| > minpts$ esetén válasszunk egy új elemet. Ha már nem tudunk új elemet választani az algoritmus véget ért. A DBSCAN algoritmus előnye, hogy tetszőleges alakú klasztert képes felfedezni, és ehhez csak az elemek távolságát használja. Hátránya, hogy rendkívül érzékeny a két

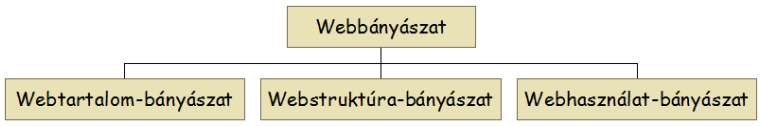
paraméterre (eps,minpts). Sőt amennyiben a klaszterekben található elemek sűrűsége eltérő, akkor nem biztos, hogy lehet olyan paramétereket adni, amivel a DBSCAN jó eredményt ad.

3. fejezet

Webes Adatbányászat

A webmining azt a gépesített dokumentum feldolgozó szakterületet jelenti, amely az internethez kapcsolódóan a nagymennyiségű weboldalon található képi, szöveges és egyéb alakú adatok feldolgozhatóvá való átalakításával foglalkozik. Az adatok átalakításának a célja a többnyire elektronikus dokumentum gyűjtőnévvel ellátott, különböző modalitású adathordozók megfelelő „finomított" vagy szűrt input kialakítása további, a felhasználó szempontjából értékes adatok kinyerésére. Ezt speciális OCR[1] és MI[2] alakfelismerő programok végzik, míg a megrendelők általában a kereskedelmi és egyéb hírszerzés területéről való cégek. [8]

A webes adatbányászat három alaptípusa a 3.1 ábrán látható.



3.1. ábra. A Webmining területei

A web-tartalom bányászat (web content mining) körébe a web szövegbányászat, az intelligens keresőügynökök, az információ-szűrés és kategorizálás, valamint a web lekérdező rendszerek sorolhatók. (Példa Google Page Rank)

A web-struktúra bányászat (web structure mining) a látogatási struktúra elemzésével, a klikkelés-sorozatok elemzésével és a web site-ok tervezési stratégiájával foglalkozik.

A web-használat bányászat (web usage mining) tárgya a forgalom elemzése, az ügy-

[1] Optikai karakterfelismerés-optical character recognition
[2] Mesterséges Inteligencia

felek szokásainak vizsgálata és az interaktivitás növelése.[2]

Web Crawlers(spiders, robots). A Weben való keresgetés egyik jó módja annak, hogy megtaláljuk a kívánt web dokumentumot. Ez a módszer működik addig, ameddig tudunk egy jó kiindulópontot (weblapok URL-t amelyek a keresett témához kapcsolódnak). Ezért ha specifikus témához keresünk web tartalmat nem túl hatékony az előbb említett módszer. Ettől jobb megközelítés, ha a tartalmak témakörönként vannak csoportosítva vagy kulcsszavak szerint kereshetőek. Témákat tartalom szerint könyvtárakba (linktárakba) gyűjtik, a kulcsszavas kereshetőséget pedig a kereső motorok teszik lehetővé. [7]

3.1. Web-struktúra bányászat

3.1.1. Információ letöltés és a webes keresés

Mióta Tim Berners-Lee megálmodta a webet sok változáson ment keresztül. A közelmúltban az Internet hatalmas és hihetetlenül gyorsan nő. Körülbelül 10 évvel Berners-Lee javaslata után, a web becslések szerint magába foglalt 150 millió csomópontot (page) és 1,7 milliárd linket. Jelenleg már több mint 4 milliárd csomópont van, ami körülbelül 1 millió csomóponttal bővül minden nap. A világháló ma már a legnagyobb, legnyitottabb és legdemokratikusabb publikációs rendszer a világon. Ugyanakkor a legnagyobb tudástár is egyben. A kérdés már csak az, hogy a keresett információt hogyan lehet megtalálni.

Webes kereső. A megoldást a webes keresők adták. Ilyen a legismertebbek közül a Google, Yahoo, Live Search, stb. Az internetes keresők feltárják a web struktúráját (szemantika mentes) és megkeresik azokat a dokumentumokat, amelyek a legjobban illeszkednek a felhasználó keresési kritériumaira, azaz szemantikát viszünk a webes keresés folyamatába. Az alapvető irányelv, hogy a felhasználó megad egy szóhalmazt és megkeresi azokat a dokumentumokat, amelyek ezeket tartalmazzák. A keresés során IR[3] technikákat alkalmaznak, hogy elkerüljék a túl általános és túl specifikus találatokat. A letöltést követően a dokumentumokat rangsorolják a kulcsszóval való egyezés mértékének megfelelően, majd tovább rangsorolják egyéb fontos tulajdonságok alapján.

[3]Information Retrival - információ visszakeresés

Téma Könyvtárak. Egy másik megoldás, amely segít a keresett információ megtalálásában, a weboldalak hierarchikus struktúrába való rendezése, amely tükrözi a tartalmukat. Ezt nevezik Téma könyvtárnak (Topic Directory) vagy csak Könyvtárnak. Ezek elérhetők szinte minden webes keresőportálon. A legnagyobb a Open Directory Project (dmoz.org) és ezt használja a Google. A könyvtárakat általában manuálisan hozza létre a több ezer weboldalszerkesztő, de vannak olyan módszerek, ahol gépi tanulást alkalmaznak az osztályozásra és a klaszterezésre.

Szemantikus web. A szemantikus web egy közelmúltbeli kezdeményezés a webes konzorcium részéről (w3c.org). A Szemantikus háló célja olyan infrastruktúra létrehozása, amely lehetővé teszi a hálón lévő adatok integrálását, a közöttük levő kapcsolatok definiálását és jellemzését, illetve az adatok értelmezését. Ebben az értelmezésben az „adat" lehet egyszerűen a hálón lévő adat (például egy online címlista) vagy metaadat, vagyis adat az adatról (például egy könyvtári katalóguscédula elektronikus változata). A szemantikus web hozzáad formális ismertető anyagot a weboldalhoz, ami az ember előtt láthatatlan csak a gép látja, ezzel segítve a keresők dolgát. Ezt az infrastruktúrát egy sor alkalmazás tudja kihasználni (például intelligens ágensek) új típusú alkalmazások céljára. A Szemantikus Web a Web technológiák egyik legdinamikusabban fejlődő területe, amely jelentős szerepet fog játszani az elkövetkezendő években.

3.1.2. Indexelés és kulcsszó-keresés

Általában két típusú adatról beszélhetünk: strukturált és strukturálatlan. A strukturált adatok rendelkeznek egy kulccsal(egy attribútum), ami tükrözi az adat tartalmát, értelmét vagy használatát. A strukturált adatokkal elsősorban az összekapcsolásuk a gond, a strukturálásuk folyamata költséges. Ezért is van az, hogy az emberek által használt információk zöme strukturálatlan formában, szöveges dokumentumként(könyv, folyóirat, újságok) van megadva, illetve a strukturálás kimerül a könyvtári valamint a bibliográfiai rendszerek fogalmában. Azonban ez is sok időt és energiát vesz igénybe, mivel emberek által hajtódik végre. Vannak próbálkozások, hogy számítógépekkel(mesterséges intelligencia által történő természetes nyelvi feldolgozás) végezzék ezt a munkát, de a probléma a tartalom-alapú strukturálás, ugyanis ez megértést igényel.

Létezik olyan megoldás is, amely megkerüli a problémát, de ehhez is szükséges némi tartalom-alapú ismeret. Ez az IR területen is használt kulcsszón alapuló keresés. Egyszerűen a keresendő szóhalmaz jelenlétét kell figyelni a szövegben. Általában nem egy

dokumentumra illik a minta, ezért az eredmények között fel kell állítani egy sorrendet (relevance rank) is. Röviden a feladat, hogy releváns adatokat találjanak irreleváns kulcsok használatával. Az internetes kereső motor nagymértékben IR technológiát alkalmaz. Ezáltal egy webspiderrel (web crawler) az IR alapú kulcsszó keresés igen egyszerű, mivel a belső HTML tag szerkezete és külső internetes kapcsolat felépítése (a webes dokumentumok szerkezete) gazdagabb, mint az egyszerű szöveges dokumentumoké.

Az első lépés az, hogy letöltse a dokumentumokat az internetről, távolítsa el a HTML-címkéket, és tárolja a dokumentumokat, mint egyszerű szöveges fájlokat. A tárolandó dokumentumokon további műveleteket hajtunk végre:

1. A dokumentumot tokenekre bontjuk[4].
2. Minden karaktert vagy kis- vagy nagybetűre alakítunk.
3. A szavakat redukáljuk kannonikus[5] formájukba.
4. Az névelőket és a többi olyan szavakat, amelyek nem bírnak érdemi információval, töröljük.

Az így kapott dokumentum termeket tartalmaz, amelyeket az eredeti dokumentumokból nyertünk ki. Ezután a kulcsszó keresés megközelítést lehet alkalmazni.

Ez a keresés egyszerű és hatékony, de a dokumentumok egy halmazát téríti vissza, ahogy ezt már említettük. Általában a felhasználók két termre szoktak rákeresni, nyilvánvaló, hogy ilyen rövid lekérdezés nem pontosan adja meg azokat az információkat, amelyek kielégítik az internetes felhasználók igényeit, a válasz halmaz nagy, és ezért használhatatlan (képzeljük el, hogy kapunk egy listát egymillió dokumentummal véletlenszerű sorrendben). A megoldás a dokumentumok rangsorolása és így egy rendezett listát tudunk visszatéríteni a felhasználónak. Ehhez a rangsoroláshoz, azonban további információkra van szükségünk, mint például a term száma, helye és egyéb jellemzője a kontextusban.

A dokumentumokat, mint vektorokat a multidimenzionális Euklideszi térben kell elképzelni, ahol a termek reprezentáljak a tengelyeket, ezt nevezik *vektortér modell*nek. Ezen belül beszélhetünk három típusú term dokumentumról: Boolean (logikai), TF (term frequency), TFIDF (term frequency - inverse document frequency). A Boolean dokumentum nem rendelkezik semmilyen plusz információval a termekről. A TF dokumentumokban a termek rendelkeznek új információval, gyakoriságuk és a dokumentum

[4] eltávolítunk minden veszőt, pontot, stb, és az igy kapot szóköz nélküli karakterláncokat nevezzük tokeneknek
[5] pl. „is”-ből és az „are”-ból „be” lesz

hosszának hányadosával is. A TFIDF-ben is a termek rendelkeznek egy súllyal, ez egy statisztikai szám, arra, hogy a szó milyen fontos a dokumentumban. Ez a szám nő a szó előfordulásának növekedésével. Míg a TF egy konkrét dokumentumon belül rendel fontosságot egy termhez, addig az IDF(inverse document frequency) általánosabban vizsgálja a term fontosságát, azt nézi, hogy a term hány dokumentumban fordul elő. A TFIDF pedig a TF és az IDF súlyok szorzataként áll elő. Ezen a modellen alapszik sok kifinomultabb rangsoroló függvény.

A keresők ezen kívül még a HTML dokumentum adta előnyöket is kihasználják. A cimkékben tárolt információk igen fontosak. A cím és a metacímke tartalmazhat információt a weboldalról, pl:

```
<title>Music</title>
<meta name="Author" content="John Smith ">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
```

A headings és a font módosítások különböző szövegrész kihangsúlyozására használhatók.

A HTML tageknek két fontos használatuk van a webes keresésben. Először is a keresett szavak szerepelhetnek valamelyik tagben és ekkor ezt önállóan is indexelni lehet a gyorsabb keresés érdekében. Ezt használja a Google is, amikor opciót kínál arra, hogy megmondjuk hol forduljon elő a szó, a címben, az URL-ben, a szövegben. A másodrendben a releváns rank-nál, ha valamely fontos tagben (head, anchor, title) szerepel a keresett term, akkor átrendeződhet a TFIDF dokumentum sorrendje. Ilyen elven működtek a korai keresők, azonban ezt ki lehetett használni (sok fölösleges, láthatatlan taget létrehozva, a weblapunkat a kereső előbbre rankelte). Manapság is fontos szerepet játszik az úgynevezett anchor tag, amelyek hiperlinkeket tartalmaznak más weboldalakra.

A kulcsszón alapuló keresésnél meg kell említenünk az úgynevezett hasonlósági keresést (similar search). Ha a keresendő termnek megfelelő, releváns dokumentumot találunk, akkor ebből egy egész sor lehetséges releváns dokumentumot tudunk előállítani az eredeti dokumentumhoz hasonló dokumentumokból. A hasonló oldalak lehívás opció majdnem minden keresőbe elérhető. Ezt egyszerűen az oldalak klaszterezésével oldják meg.

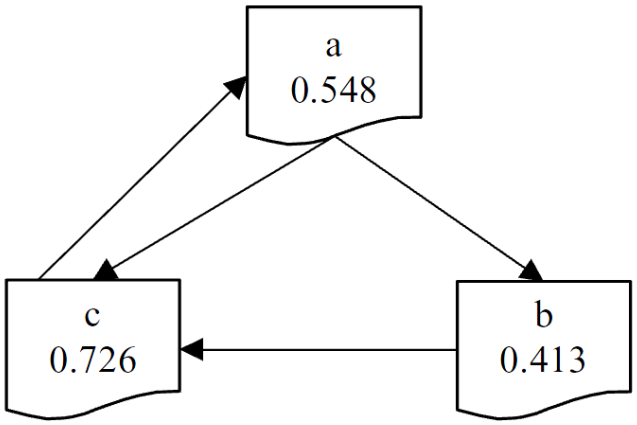
3.1.3. Hiperlinken alapuló rangsorolás

A hiperlinkeknek a megszokott funkciójukon kívül létezik egy igen fontos szerepe, külső segítséget tudnak nyújtani egy oldal page rank-jének a felállításában. Ezt úgy kell

elképzelni, mint a közismert, mindenki által használt idézeteket. Amelyik idézet sokatmondó és fontos számunkra azt szeretjük használni, így van ez a weboldalak esetében is. Azt az oldalt melyet sokan szeretnek, arra sokan is hivatkoznak. Az alapötlet az volt, hogy minden weboldalhoz a hivatkozások felépítése alapján rendeljünk hozzá egy rangot.

A social network-ot úgy lehet ábrázolni, mint egy irányított gráfot melynek élei sulyokkal vannak megadva. Ebben az állapotban a presztizs fogalom ahhoz társítható, hogy hány él érkezik egy csomópontba. De egy csomópont presztizs függ attól is milyen minőségű csomóponttal (prestig) áll kapcsolatban, így a definíciónk rekurzív.

Legyen a gráfunk szomszédsági mátrixa A a következőképpen értelmezve. $A(u, v) = 1$, ha az u dokumentumból indul él a v dokumentumba, u hivatkozik v -re $A(u, v) = 0$ máskülönben. Minden u csomópontnak van egy presztizs, száma melyet a következőképpen kapunk meg $p(u) = \sum_v A(v, u)p(v)$, melyet egy P oszlopvektorként írunk le. Egy kezdő P presztizs vektorból egy új P' presztizs vektort úgy kapunk, ha $P' = A^T * P$. Egy dokumentum presztizst (3.2 ábra) „saját" érték (λ) kiszámításával kapjuk a következő egyenlőségből. $\lambda * P = ATP$.



3.2. ábra. Dokumentum hálozat presztizs értékekkel

Page rank

A hiperlinkek nem az egyetlen módja annak, hogy egy weboldal presztizsét mérjük. Léteznek utak melyet a felhasználók bejárnak egyik weboldalról a másikra navigálva. Egy weblap népszerűségét ez adhatja, hogy egy átlagos felhasználó milyen gyakran látogat oda. Minden u weboldalon Nu darab link található, ezek közül egy, v weboldalra

mutató. Ha véletlennek tekintjük a felhasználó cselekvését, akkor annak az esélye, hogy u weboldalról a felhasználó átlinkeljen v weboldalra $\frac{1}{Nu}$. Ez a tény azt jelenti, hogy át kell gondolnunk ez előbbiekben tárgyalt presztizs mutatónk kiszámolásának módját. Az alapötlet az, hogy egy u weboldalon található v weboldalra mutat link esetén v ne kapja meg u prestig-ét, hanem annak csak $\frac{1}{Nu}$-ad részét. Ez az ötlet volt az alapja a webet rangsoroló Page Rank algoritmusnak is melyet eredetileg a Google is használt és többek között ez vezetett sikerükhöz.

Az egyszerűsített Page Rank algoritmus ciklusokkal dolgozik, ezért abban a speciális esetben, ha két oldal többszörösen is egymásra hivatkozik, modellünk nem követi egy valós felhasználó viselkedését. Valószínűleg senki nem fog két oldal között klikkelgetni. Hogy ezt a problémát megoldják bevezettek egy rank source E vektort mely az összes weboldal esetén tartalmazza annak a valószínűségi eloszlását, hogy véletlenszerűen egy adott oldalra ugrunk.

$$R(u) = \lambda[\sum_v \frac{A(v,u)R(u)}{N_v} + E(u)]$$

Az így definiált Page Rank algoritmus az olyan véletlenül böngésző felhasználót modellezi, ahol:

1. R vektor megmondja a valószínűségi eloszlását egy véletlen lépésnek a Web-en
2. kis valószínűséggel a felhasználó által választott véletlen oldal E eloszlásának megfelelő lesz
3. E olyan módon befolyásolja a véletlen böngésző modelljét, hogy ha E normája nagy, akkor egy véletlen oldalra való látogatási esélye nagyobb

A rank source alkalmazása megoldja az egymásra hivatkozás problémáját és lehetőséget ad arra hogy olyan WEB -et modellező gráffal is lehessen dolgozni melynek vannak nem összefüggő (disconnected) részei is (mint ahogy ez a valóságban is van). Továbbá arra is lehetőséget ad, hogy olyan helyzeteket is lehessen kezelni ahol manipuláció, üzleti érdekek is felmerülnek. Pld. Egy weblap page rankje növelhető, ha fontos oldalról mutat link az oldalra, vagy nagy számú link egy kevésbé fontos oldalról. Ilyen esetekben, ha E-nek megfelelő értéket állítunk be azzal minimalizálhatjuk a nem kívánt eredményt. E vektor előállítására az egyik lehetőségünk, ha E be összegyűjtjük a világ összes web szerverének gyökér (start) oldalát, ez a módszer nehézzé teszi a manipulációt. Másik lehetőség, ha kijelölünk egy zéró pontot E számára, ő megkapja a legmagasabb page rank pontszámot, majd őt követik a linkjei. Egy ilyen zéró pont lehet egy megbízható web könyvtár.

Azok a weboldalak melyeknek magas a page rank számuk fontosnak minősülnek, ide a felhasználók gyakrabban látogatnak. Ezért az ilyen oldalakat a web robotok hamarabb látogatnak meg és indexelnek be.

Authorities and Hubs (Tekintélyek és Gyűjtőlapok)

Széles körben elismert, hogy a link alapú rangsorolás és leginkább a Page rank az alapja a webes keresésnek, de hiba lenne csak ebből a megközelítésből szemlélni a témátt. A linkek sokszor nem tudnak mit kezdeni az authority-vel, másfelől a legnépszerűbb és authority oldal nem biztos, hogy a legrelevánsabb egy adott kereséshez. Más szavakkal, önmagában a relevancia és népszerűség nem oldja meg a feladatot, kell találni egy egészséges középutat. Megoldást jelent Kleinberg [10] által javasolt HITS (hyperlink induced topic search) mely keveri a tartalom alapú relevanciát és a link alapú authority rangsorolást. Az alapötlet az volt, hogy koncentráljunk első lépésben az olyan oldalakra, ahol a tartalom releváns a kereséshez, majd csak ez után számoljuk az authorityt. Ennél a megközelítésnél figyelembe vesszük a HUB oldalakat is, oldalakat melyek sok más releváns és authority oldalra mutatnak.

A HITS ellentétben Page rankel-el egy sokkal kisebb és kérés függő (query-dependent) részfájával dolgozik a web gráfnak. Először kulcsszó alapú keresés történik, majd a találatoknak elemzi a link (hivatkozás) felépítését. Ezt az egész eljárást Topic distillationnek nevezik. Megadott q lekérdezés mellett a következő lépések hajtódnak végre:

1. Standard IR rendszert használva, egy kis halmazát a releváns oldalaknak keressük, ez lesz a gyökér halmaz R_q.

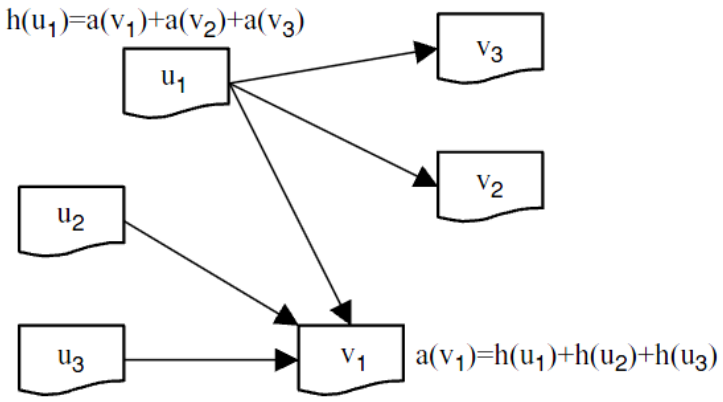
2. A gyökér halmazt kibővítjük az R_q-ra hivatkozó és hivatkozott oldalakkal, ez lesz az alap halmazunk S_q.

3. Authority és hub-okat keresünk az S_q hiperlink struktúrájának elemzésével.

Az algoritmus befejeztével visszaadja a legmegfelelőbb authority és hub találatokat. Az algoritmus hátrányai közé sorolható az, hogy a használt hub és authority értékek nem számolhatók ki előre, mivel az algoritmus mindig a kérés által visszaadott részgráfon dolgozik. Viszont ez az előnye is a Page rank-el szemben, mivel az olyan esetekben ahol nincs tartalom csak linkek sokasága az oldalon nem érnek el magas pontszámot.

Jelentős különbség a HITS és Page rank között a pontszámok szétosztásában található. HITS esetén a hub, azoktól a lapoktól gyűjti be pontjait amelyekre mutat. Ezért a

pontok szétosztása kétirányú, testvér csomópontok pontszámai is hatással vannak egymásra. Minden lépésben u1 oldal h(u1) hub pontjait azon oldalak authority pontjaiból gyűjti össze, melyekhez csatlakozik(3.3 ábra).



3.3. ábra. Authority és Hub értékek

A következő lépésben v1 oldal authority pontjait a rá mutató hub oldalak hub pontjainak összegéből kapja. Ez addig folytatódik, ameddig minden pontszám elér egy fixpontot. Ha valamelyik lépésben v2 egy nagy authority pontszámot kap, akkor az később kihatással lesz v1, v3 testvérére is. Hub pontok kiszámítása egy előny mivel a hub oldalak igen fontos információ lelőhelyek a weben. Page rank nem számol hub pontokat, de gyakorlatilag ez nem olyan nagy hátrány mivel jól működő hub oldalak hamar begyűjtik a megfelelő linkeket. HITS hátrányai közé sorolható az is, ha olyan keresést indítunk, amely többértelmű akkor a HITS algoritmus beragadhat valamilyen kevésbé releváns részébe a web gráfnak. Ez azzal magyarázható, hogy csak a nagy algráfokat tekinti authority és hub oldalaknak, csak a domináns saját vektorokat találja meg.

Link-based similarity search

HITS ötvözi a tartalom alapú keresést és link alapú rangsorolást. Azt az alapfeltevést feltételezi, hogy a gyökér halmazban lévő oldalak a kérés témájához közel esnek, továbbá az olyan oldalak melyek az alap halmazban találhatóak (egy link távolságra) tartalmuk alapján szintén közel vannak a keresés témájához. Ezt a feltevést kutatások is alátámasztják, miszerint általában az egy link távolságra található oldalak témája hasonló, megegyezik. Ez a gondolat arra enged következtetni, hogy megalkotható egy link alapú csoportosítása a hasonló oldalaknak. Ez egy alternatív megoldása lehet az

előbbiekben tárgyalt HITS algoritmusnak, melynek alapja egy tartalom alapú keresés. A különbség az lenne, hogy a gyökér halmazt nem kulcsszavas kereséssel határoznánk meg, hanem egy linkeket használnánk helyette. Megadva egy u oldalt és k paramétert az algoritmusnak, első lépésként keres k darab oldalt mely u oldalra mutat, ez a halmaz fogja alkotni a gyökér halmazt. További lépései megegyeznek az előbbiekben leírt HITS algoritmussal. [4] Egyik előnye a link alapú hasonlóság keresésnek az, hogy nem kerülnek hátrányba az olyan oldalak ahol nagyon kevés a szöveges rész, vagy egyáltalán nincs szöveg, mint például képtárak esetén.

Enhanced techniques for page ranking

Mind a Page Rank mind a HITS alapja az az alap feltevés, hogy a linkekkel összekapcsolt oldalak tartalma hasonló vagy megegyezik. Ha ezen oldalakhoz további linkeket adunk, előfordulhat, hogy olyan tartalmakra lesz hivatkozás melyek nem kapcsolódnak a keresés témájához. Ezt a folyamatot téma általánosításnak nevezzük (topic generalization). Egy másik nem várt helyzet alakulhat ki, ha egy adott témához kapcsolódó oldalhalmaz egy oldala hivatkozik egy másik nagy oldalhalmazra egy másik témakörből. Ekkor az alaphalmaz kibővítése esetén belekerülhetnek olyan weboldalak is a kibővített halmazba melyek a nem kívánt témából valók. Ezt téma csúszásnak (topic drift) nevezzük és page rank és HITS esetén is fennáll a veszélye.

A fent említett problémák (topic generalization and drift) leginkább ez egyszeres (single) rangsoroló és csak a web gráfra építő rendszereket fenyegeti. Megoldás az lehet, ha többszörös rangsorolókat használunk. Ilyen lehet a tartalom alapú és link alapú rangsoroló módszerek. Végeredményeiket pedig bizonyos súlyokkal átlagoljuk így kapjuk majd a végleges rangsort.

3.2. Web-tartalom bányászat

A web content mining az a folyamat, amikor hasznos információt fedeznek fel szövegekben, képekben, audio és video fileokban az interneten. A tartalom bányászatot szokták szövegbányászatnak is hívni, mivel a szövegtartalom a legjobban kutatott terület. Az általában használt technológiák az NLP (natural language processing - természetes nyelvi feldolgozás) az IR amiről már beszéltünk, valamint az objektumok osztályozása és klaszterezése.
Az Osztályozás és a Klaszterezés fejezetben erről részletesen írunk.

3.3. Web-használat bányászat

Web használat bányászat a következőképpen lett definiálva Srivastava [11] által. Web használat bányászat az adatbányászó technikákat használó program, melynek az a feladata, hogy használati mintákat fedezzen fel a Webes adathalmazban. Mindezt azért, hogy megértsük és jobban kielégítsük a webes alapú alkalmazásokkal szemben támasztott igényeket. A bányászat ezen módja különbözik a többi fajta bányászattól abban, hogy itt visszatükröződik az emberi interakció, a viselkedésmód amit az interneten tanúsít. Ez az információ hasznos lehet a tartalmat karbantartó és az oldalt fejlesztők számára. Például egy filmes adatbázist üzemeltető személy miután elvégzett egy web használat elemzést az oldalán, kiderült, hogy a látogatók nagy része miután horror kategóriában böngésztek utána tudományos fantasztikus filmek kategóriát is megnézték. Ennek az információnak a tudatában a fejlesztő a horror kategóriából direkt linket épített az weboldalba a tudományos fantasztikus kategóriához, ezzel megkönnyítve a böngészést.

Több szintje lehet a web használat bányászatnak. Megfigyelhetjük különálló személyek viselkedését és az ő szokásaikhoz igazíthatjuk weboldalunkat vagy egy bizonyos időintervallumban az oldalunkat látogató személyeket, ekkor a weboldalon való navigálással kapcsolatos problémákat deríthetjük ki.

3.3.1. ClickStream analysis

Web használat bányászatra sokszor hivatkoznak úgy, mint klikk sorozat (clickstream) elemzés. Clickstream-nek nevezünk minden olyan klikk sorozatot, ami egy felhasználó a weboldalon való navigálás alatt elkövet. Az oldallehívásokon (request) kívül a clickstream elemzéskor felhasznált adatok web szerver logokból, cookie-ből (sütik), metatagokból és még más adatokból áll. Legegyszerűbb esetben a felhasználó böngészője segítségével egy kérést intéz valamely web szerverhez az URL beírásával. Ekkor több folyamattól eltekintve egy bejegyzés történik a webszerver log-ban a kéréssel kapcsolatban. Egy weboldal betöltésekor a böngésző a weboldalon belül található objektumok (képek) esetén is külön lekérdezés indít a szerver irányába. Előfordulhat az is, hogy gépi tevékenység hatására kerül bejegyzés a log-ba, ilyen eset fordulhat elő ha a WWW -et fáradhatatlanul kutató WEB Robot látogat oldalunkra. Ezért előfeldolgozás folyamán az ilyen valóban egy web oldalt jelentő több lekérdezést összesíteni kell. A felhasználó által összesen meglátogatott oldalakat pedig session-be kell gyűjteni. Miután az adat-

halmazunkat megfelelően megtisztítottuk feltehetjük kérdéseinket:

1. Melyik a leggyakoribb pont ahol a felhasználók megérkeznek oldalunkra?
2. Milyen sorrendben lettek a weblapok látogatva?
3. Milyen más oldalról van hivatkozás az oldalunkra?
4. Egy átlagos látogatás alatt hány weblapot néztek meg a felhasználók?
5. Milyen hosszú ideig tartózkodnak egy oldalon a felhasználók?
6. Melyik a leggyakoribb pont ahol a felhasználók elhagyják oldalunkat?

3.3.2. Web Server Log Files

Mielőtt klikk sorozat elemzéssel foglalkoznánk, nézzük meg mi is az a web szerver log. Minden a szerverhez beérkező kérés (request) után automatikusan eltárolódik egy bejegyzés. Ez általában egyszerű ASCI text-ként, falhasználó számára olvasható formában történik. Pl.
141.243.1.172 [09/Jun/1988:03:27:00 -0500] "GET /Hallgato.html HTTP/1.0" 200 1325

Első része a remote host mező, itt a kérést feladó IP címe vagy neve található, ha az elérhető (DNS címfeloldás). Dátum mező követi [DD/Mon/YYYY:HH:MM:SS offset] formában. Harmadik eleme a HTTP kérés mező (HTTP request), ez további 4 részből áll:

1. kérésre használt metódus (GET, POST)
2. URI (uniform resource identifier) tartalmazza a dokumentum nevét és a szerveren belüli elérésének az útvonalát
3. header, fej rész mely opcionális információt tartalmazhat, pld. Milyen kulcsszavakat használt a felhasználó a keresése során, melynek eredménye a mi oldalunkra mutat
4. Protokoll, általában HTTP és emelett még a verzió is szerepel HTTP 1.1.

Negyedik elemként a státusz kód (satus code) szerepel a weblogban. Nem minden lekérdezés sikeres. Ha a kódunk 2XX alakú, akkor sikeres választ feltételezünk, ha 4XX akkor valamilyen hiba történt. A legismertebb a 200-as kód amely sikeres lekérdezést

jelöl. Hibakódok közül a 404-es hiba a legtöbbet látott, ő a szerveren a dokumentum nem található (not found) hibát jelenti. Legutolsó elemként az átküldött adat volumenét (transfer volume) írjuk a log-ba byte-okban megadva. Csak a siker (státusz=200) esetén található pozitív érték itt.

A fentebb felsorolt mezők a leg általánosabbak. Ezen kívül a log még tartalmazhat authentikációról, hivatkozó félről (referrer), felhasználó böngészőjéről (user agent), a válasz idejéről is információt.

Amint azt már említettük a weblog-ban tárolt adathalmaz nem használható közvetlen feldolgozásra. Ezt az adatmennyiséget meg kell tisztítani. Ezt a folyamatot hívjuk előfeldolgozásnak[6].
Miből is áll az előfeldolgozás: Adatok megtisztítása. Ki kell szűrni az automatikusan létrehozott kéréseket, melyeket a böngésző hozot lére pld. képadatok lekérésénél. Ezt általában a képadatok kiterjesztése (jpg, png, gif) alapján szűrjük ki a log-ból. Nem emberi kérés után bekerült adatok. Ilyenek lehetnek a World Wide Web-et elemző, elemző Spider-ek. Legegyszerűbb dolgunk akkor van, ha ismert spider látogatta az oldalunkat, őket host nevük alapján könnyen kiszűrhetjük. Meg kell különböztetni minden egyedi felhasználót. Az Internet szabad formájából adódóan a felhasználó látogatása egy weboldalra névtelen. Ezért a web használat bányászat esetén minden kérést IP címmel, sütiben (cookie) tárolt adatokkal illetve regisztráció esetén a felhasználónévvel egészítünk ki, hogy a felhasználók elkülöníthetőek legyenek. Ennek a menete általában a következő lehet:

1. Rendezzük a log sorait IP cím alapján majd idő alapján.

2. Minden azonos IP cím esetén válasszuk szét a különböző böngészőt használókat (tegyük fel, hogy ők különböző felhasználók).

3. Minden a második lépésben elkülönített felhasználó esetén, felhasználjuk az útvonal információt a hivatkozó mezőből (referrer) és a weboldalunk felépítését, hogy el tudjuk dönteni az adott viselkedési mód (navigálás) leginkább egy vagy több felhasználót takar.

4. Hogy minden felhasználót el tudjunk különíteni, ahhoz össze kell fésülnünk az 1-3. Lépések eredményeit a sütikkel (cookie) és regisztrációs információkkal.

Meg kell különböztetni az adott felhasználóhoz tartozó munkamenetet. Milyen lapokat nézett meg, átlagosan mennyi ideig tartózkodott ott és azt is próbáljuk megbecsülni,

[6] Bővebben az Előfeldolgozás részben

hogy mikor hagyta el a weboldalt. Munkamenet (session) azonosítására az eljárás a következő:

1. Minden a fenti eljárással elkülönített egyedi felhasználóhoz egy ID-t rendelünk.
2. Megszabunk egy tétlenségi időkorlátot (ha ennél a korlátnál hosszabb ideig nem érkezett a felhasználótól kérés, akkor a munkamenetet lezárjuk)
3. Minden felhasználó esetén, találjuk meg az időbeni különbséget két kérése között. Ha ez az idő kívül esik a 2. Pontban megadott korláton akkor rendeljünk egy új ID-t a későbbi bejegyzéshez, hiszen ő egy másik felhasználó.
4. Rendezzük a bejegyzéseket ID alapján

Útvonal kiegészítést kell végrehajtanunk. Sokan a weboldalon való navigálás közben nem az arra hivatott linkeket használják, hanem a gyorsaság kedvéért a „Back" gombot megnyomva visszaugranak ez előző oldalra. Ez a művelet nem indít kérést a web szerver irányába, hanem a böngésző memóriájából töltődik be, így nem marad nyoma a szerveren. Weboldalunk struktúráját ismerve ezeket az úgynevezett lyukakat kell betömni.

3.3.3. Felderítő adatelemzés

Miután megszabadítottuk az adathalmazunkat a fölösleges adatoktól és mielőtt elkezdenénk a web használatot modellezni hasznos lehet néhány feltáró elemzést elvégezni. Elsősorban arra szolgál ez az elemzés, hogy mélyebben bele tudjunk látni az adathalmazban található kapcsolatokba.

Number of visit actions

Megmondja, hogy egy munkamenet (session) során hány oldallekérés történt. Érdekelhet minket az is, hogy átlagosan hány oldalt töltöttek le a felhasználók látogatásukkor.

Session duration

A felhasználók klikkelései mellett, másik fontos mutató a munkamenet hossza. Ennek a mutatónak a számolása megfontolást igényel, mivel az adott oldalon töltött időt nem tudjuk közvetlenül mérni. Egyetlen módja, ha két interakció (weblog bejegyzés) közt eltelt időt mérjük. Ekkor viszont ki kell szűrnünk az olyan felhasználókat, akik csak egy oldalt néztek meg. Illetve azokat, akik túl sokáig tartózkodnak egy oldalon, azaz tétlené

váltak. Minden más felhasználó esetén vesszük az első és utolsónak megnézett oldal log bejegyzését és kivonjuk a két időt egymásból.

Number of visit actions és a Session duration között kapcsolat fedezhető fel, méghozzá ha nő az egy adott látogatás alatt megnézett oldalak száma, nő a weboldalon eltöltött idő is.

Average time per page

Az oldalon átlagosan eltöltött idő = Session duration / number of visit actions -1 . Azért vonunk ki 1 et a látogatási tevékenységből, mert az utolsó oldalletöltést nem tekintjük a munkamenet részének. A kapott értékeket sorba rendezzük. Itt kereshetünk kimagaslóan alacsony értékeket (1s alatt 10 oldal letöltése), ezek valószínűleg nem emberi kérések voltak. Ilyen esetben célszerű az 1 es lépést (adathalmaz megtisztítása) megismételni ez újonnan felfedezett gépi szereplőre. Ez a példa is mutatja, hogy mennyire szükséges az interakció az elemzés teljes folyamán. A nagytömegű adatfeldolgozás miatt szükséges bizonyos műveletek automatizálása, de ha pontos eredményt szeretnénk elérni, mindenképpen szükséges az emberi tevékenységet is bevonni az elemzésbe.

3.3.4. Web használat bányászat modellezése: clustering, association rules and classification (klaszterezés, asszociációs szabályok, osztályozás)

Miután adatunkat megtisztítottuk a nem kívánt elemektől, néhány mérőszám alapján tisztább képet kaptunk a látogatóink viselkedéséről. Következhet az adatok modellezése. Három fő modellezési technikát fogunk megvizsgálni: klaszterezés, kapcsolat és osztályozás. Ezen módszereket és használt algoritmusokat előbbi fejezetek már tárgyalják. Ezért itt csak a problémához szorosan kapcsolódóan írunk róla, általánosságban nem elemezzük.

Először a klaszterezés áttekintésével kezdjük mivel ez a legegyszerűbb modellező metódus, így a később használt számításigényesebb elemzéseket nem kell már a NEM hasonló elemekre elvégezni. Klaszterezés célja az, hogy az adathalmazunkat lehetőleg homogén csoportokra vagy fürtökre bontsuk szét, ahol a fürtön belüli hasonlóság maximális viszont fürtön kívüli elemek közt minimális legyen a hasonlóság. Ehhez használhatjuk az előző fejezetekben tárgyalt algoritmusok közül BIRCH, DBSCAN-t melyek jól skálázhatók nagy adatmennyiség esetén is. Az algoritmust az előbbiekben kiszámított munkamenet szintű mérőszámokra alkalmazzuk, ilyenek pld. munkamenet ideje, munkamenet tevékenység száma, átlagos tartózkodási idő oldalanként stb.

Kapcsolati(hasonlósági, asszociációs) szabályok (association rules) az adathalmaz attribútumai közötti kapcsolatot írják le. Egy ilyen szabály alakja „Ha előzmény, akkor következmény" alakúak. Kapcsolati/ hasonlósági szabályok (association rules) bányászata arra használható, hogy megtaláljuk a kapcsolatot az olyan oldalak között melyek leggyakrabban egy munkamentben együtt voltak hivatkozva. Ez a mi esetünkben azt jelenti, hogy a szabályok olyan web oldal halmazokat jelölnek ki melyek esetén egy segédváltozó (mutató) meghalad egy előre meghatározott küszöbértéket. Ezek a web oldalak nem feltétlenül kell, összekötve legyenek hiperlinkek segítségével.

Minden ilyen szabályokat kutató algoritmus abba a problémába ütközik, hogy az attribútumok növekedésével a lehetséges szabályok száma exponenciálisan nő. Egy Agrawal [12] által kifejlesztett úgynevezett apriori algoritmust használunk asszociációs szabályok (association rule) bányászására, mely algoritmus kihasználja a szabályok belső szerkezetét (szabályok másságát) azért, hogy a keresési problémát egy sokkal kezelhetőbb méretre hozza. Az apriori algoritmus bemenetként és kimenetként nem tud numerikus értékeket kezelni, ezért az értékeket sávokba sorolva (diszkrét értékként) kell megadni és a kapott eredménytől is ezt várjuk el.

Algoritmusunk ezután szabályokat fog generálni. Minden hasonlósági szabályhoz tartozik egy előzmény egy következmény és két mérőszám, az egyik a szabály támogatottsága (rule support), a másik a hitelessége (confidence). Előzménynek nevezzük azt, ha valaki meglátogat egy bizonyos oldalt, az ehhez tartozó következmény pedig az, hogy ezzel azt váltja ki hogy a 2. es számú klaszterbe fog tartozni. A támogatottságot úgy kapjuk meg, hogy megnézzük hány adatelemre lesz igaz a szabály. Hitelesség pedig azt mutatja, hogy az előzmény alapján kiszűrt soroknak hány százaléka a következmény által megkapott sorok (adatelemek). Az így kapott sorokat támogatottság és hitelesség alapján csökkenő sorrendbe rendezzük és megnézzük, hogy melyik az a szabály melynek mutatója elég magas és a szabálynak olyan értelme van, amit fel tudunk használni. Példának okáért az, hogy a látogatók 90%-a meglátogat egy bizonyos web oldalt és ez az oldal látogatói mind a 2. fürtbe tartoznak, ez tény, de nem tudjuk felhasználni semmire, hasztalan. Érdekesebb lenne pld olyan sort keresni ahol a bizonyos web oldalon eltöltött idő rövid és maga a munkament ideje is alacsony.

Valószínűleg egy ilyen szabálynak csak a látogatók kis % felel meg, mondjuk 1/3. De gondoljunk csak bele, ha sikerül a megfelelő lépéseket eszközölni annak érdekében, hogy a látogatóink 30% többet maradjon web oldalunkon, már sokat nyertünk. De kereshetünk céltudatosan adott problémára megoldást. Azt szeretnénk megtudni, hogy melyik

az a felhasználócsoport, amelyik kevés időt tölt az oldalunkon. Hogy használhatnánk fel a meglévő szabályainkat ezen csoport azonosítására? A feladatunk az, hogy olyan felhasználókat találjunk, akik kevés időt töltenek el az oldalon. Ezért vesszük az olyan szabályokat melyeknek előzménye az, hogy a látogatási idejük kicsi, utána megnézzük, hogy milyen következmények társulnak ezen sorokhoz és hol a legnagyobb az érintettség (rule support). Ha az illetékes (marketinges) csoportnak sikerül megoldást találni arra, hogy növeljék a web oldalanként átlagosan eltöltött időt, ez növelni fogja az egy munkamenetben eltöltött időt.

Classification and regression trees (CART)

A legelterjedtebb adatbányászati feladat az osztályozás. Osztályozás esetén mindig van egy kitűzött célváltozó (pld. Adott oldal látogatási ideje), melynek az értéke előre meghatározott osztályokba van sorolva. Ilyen osztályok lehetnek alacsony, közepes, magas (időtartam). Az adatbányászati modell egy tanuló adathalmaznak nevezett, nagy adathalmaz rekordjait elemzi, melynek minden sora (rekordja) tartalmaz információt a célváltozóról, és a célváltozót előrejelző változókról. Ezután ha egy új értéket adunk meg, ahol a célváltozónk ismeretlen, a tanuló részben begyűjtött információ (minták) alapján el tudja dönteni, hogy melyik osztályba tartozik.

Egyik lehetséges osztályozó metódus lehet az, ha döntési fát építünk. A döntési fa, döntési csomópontokból áll melyeket élek kötnek össze egészen a levél elemekig. Minden döntési csomópontban attribútum értékek értékelődnek ki. Egy csomópontból annyi kiinduló él van ahány lehetséges kiértékelési módja van a tesztelt attribútumnak. Minden levél elem egy adott osztályba sorolja a megadott elemet.

A CART rendszert először Breiman [13] javasolta 1984-ben. Ez a döntési fa szigorúan bináris, tehát minden döntési csomópontnak pontosan két ága (éle) van. CART előre megadott szabályok és a vágás jóságának vizsgálata alapján épít döntési fát. A fa építése a gyökér csomópont felosztásával történik egy nagyon egyszerű alakú „X<d?” szabály alapján. Ahol X az adathalmazunk egy eleme, „d” egy magadott konstans. Kezdetben minden észrevételt feljegyzünk a gyökér csomópontba. Ez a csomópont nem „tiszta”, heterogén mivel különböző osztályokba tartozó észrevételek vannak benne. Célunk az, hogy úgy bontsuk szét a gyökér csomópontot, hogy a gyerekeiben lévő észrevételek kevésbé legyenek heterogén állapotban. A tanuló adathalmaz rekordjait a CART rekurzívan osztja fel alhalmazokra, melyekben a cél attribútumok értékei hasonlóak, egyre inkább egy osztályba tartoznak. A CART algoritmus egy mélyreható keresést végezve

az összes elérhető attribútum és az összes lehetséges vágás közül optimálist választva, minden csomópontban növeli a fát.[14]

$$\Theta(s|t) = 2P_L P_R \sum_{j=1}^{no.classes} |P(j|t_L) - P(j|t_R)|$$

$\Theta(s|t)$ jelöli a Gini indexet, mely egy vágás jóságát mondja meg, ahol „s" a vágás, „t" a csomópont, továbbá:
t_L a bal gyereke a t csomópontnak
t_R a jobb gyereke a t csomópontnak
$P_L = \frac{t_L rekordokszama}{azsszesrekordszama}$
$P_R = \frac{t_R rekordokszama}{azsszesrekordszama}$
$P(j|t_L) = \frac{josztlybelielemekt_L-ben}{t-belielemekszama}$
$P(j|t_R) = \frac{josztlybelielemekt_R-ben}{t-belielemekszama}$

Az optimális vágás ott lesz a t csomópontban ahol a Gini index maximális az összes lehetséges vágások közül. Minél közelebb van a vágás topológiai helye a gyökérhez, annál fontosabb (jelentősebb) a vágás. A fa építésének a következő a menete:

1. CART felosztja ez első csomópont adathalmazát az összes lehetséges módon. Minden vágás esetén a kérdésre teljesülő adatok balra, a nem teljesülők jobbra kerülnek.
2. CART a vágás jóságát ellenőrzi a GINI index alapján. (használhatna más indexet is)
3. Azt a vágást választja ki, amelyik a legjobban csökkenti az adatok heterogén entitását (szennyezettségét).
4. 1-3 lépések ismétlődnek a gyökér csomópontban megmaradt változókra.
5. CART rangsorolja a legjobb vágásokat minden változóra.
6. CART kiválasztja az a változót és a hozzá tartozó vágást, amely a legjobban csökkenti a szennyezettséget
7. CART ezután osztályokat rendel ezekhez a csomópontokhoz figyelembe véve azt is, hogy a hibás osztályozás költsége minimális legyen. A CART tartalmaz egy beépített algoritmust, amely figyelembe veszi a felhasználó által megadott téves osztályozás költségeit. (pl sokkal nagyobb hiba lehetősége, olyan esetben, ha valamelyik elemet besorolunk egy olyan halmazba mely súlyosabb hibának minősül mintha nem megfelelő, de más osztályba soroltuk volna)

8. Mivel a fa építése rekurzív ezért az 1-7 lépések ismétlődnek minden nem terminális gyerek csomópontra.

9. A vágás addig folytatódik, ameddig minden észrevétel terminális csomópontot nem alkot.

Ha szeretnénk beazonosítani a rövid munkameneteket, akkor alkalmazzuk a CART-ot a web log adathalmazunkon, session idejét választva cél változónak. Az előbbiekben a priori algoritmussal felfedeztük, hogy szoros kapcsolat létezik a session ideje és az egyes weblapokon eltöltött idő között. Hogy kiszűrjük ezt a kapcsolatot ezért az egy oldalon eltöltött átlagos időt nem vesszük be az előrejelző változók közé a munkamenet időtartamát osztályozó döntési fánkba, mivel azt szeretnénk megtudni, hogy milyen eddig nem ismert tényezők befolyásolják még a rövid munkamenetet. Minden terminális csúcs egyedien azonosítja egy szabálysorozatot, a gyökértől a levél elembe vezető úton található szabályok sorozata.

Osztályozás segítségével érdekes szabályokat fedezhetünk fel, pl. A felhasználóink 30% melyek online rendeltek 18-25 év közötti korosztályba tartoznak és nyugat Magyarországon élnek.

4. fejezet

Program leírása

4.1. ClickHeat

Ebben a fejezetben az általunk létrehozott, ClickHeat-nek elnevezett webes alkalmazást ismertetjük. Leírjuk milyen problémára jelent megoldást, érveket hozunk ezen módszer mellett és ellene. Bemutatjuk hátterét, használati módját.

Az internet világa folyamatosan bővül, az információ nagy sebességgel terjed, a szokások változnak. Eszközeinktől elvárjuk, hogy alkalmazkodjanak ehhez a komplex és változó helyzethez. Ebben nyújt támaszt a ClickHeat alkalmazás, weboldal üzemeltetőknek. Leegyszerűsítve az alkalmazás lényegét, nem más, mint egy klikk sorozatot gyűjtő és hő kép formájában visszaadó alkalmazás. ClickHeat kliens oldalon Javascriptet, szerver oldalon PHP alapú alkalmazás, mely PHP script futtatására alkalmas web szerveren fut, az adatokat MYSQL adatbázisban tárolja. Implementációt tekintve, Joe Stump [15] által megírt Model 2 (MVC) tervezési mintát követő keretrendszerét használtuk alapnak.

ClickHeat abban nyújt segítséget, hogy weboldalunk szerkezetét ezen változó igényekhez tudjuk igazítani. Dióhéjban nézve ClickHeat-et a következőképpen kell használni: web böngésző segítségével meglátogatjuk a ClickHeat adminisztrátori oldalát, az adminisztrátori oldalon lehetőségünk van regisztrálni. Regisztráció után beléphetünk adminisztrátori felületünkre. Az adminisztrátori felületen található három soros javascript kódrészletet beillesztjük a kívánt oldalaink fejlécébe. Ennek következtében, ha valaki oldalunkra látogat és ott klikkel a klikkelés helye és ideje automatikusan a megfelelő oldalhoz hozzárendelődik. Klikkelések helyét hő térkép formájában megnézhetjük és levonhatjuk a következtetéseinket.

Miért előnyös alkalmazásunk és milyen további előnyt nyújt az, hogy web alkalmazásként használhatjuk?

- analitikus módszerekkel csak egy statikus értelmezését kapjuk meg a felhasználók tevékenységének. Meg tudjuk mondani, hogy hány látogató látogatott a weboldalunkra, vagy hányan néztek meg egy bizonyos weboldalt, de nem leszünk képesek dizájn elemek, link elrendezések, oldalaink népszerű részeinek hatékonyságát mérni. Ha azt akarjuk megtudni, hogy a felhasználók pontosan hova klikkeltek a weboldalon és melyik részeket lehet tökéletesíteni akkor a ClickHeat a megfelelő eszköz.

- Segít megtalálni, hogy a felhasználók hova klikkeltek, vagy még fontosabb a weboldal azon részeit, melyeket a felhasználók figyelmen kívül hagynak.

- Sok analitikus program létezik, melyek információt gyűjtenek a felhasználókról, de nem képesek viselkedéssel kapcsolatos adatot szolgáltatni, amit felhasználhatunk weblapunk optimalizálására.

- Marketinges kampány esetén, kipróbálhatunk több fajta elrendezési módot hirdetéseinknek. ClickHeat segítségével megvizsgálhatjuk, hogy melyik bizonyul a legjobbnak, a hirdetés teljesítmény szempontjából.

- HTML oldalakban szükséges változtatás minimális. Nem szükséges oldalanként különböző javascript kódot megadni, köszönhető mindez az intelligens algoritmusnak, így a hiba lehetősége minimális.

- Nem URL alapján dönti el, hogy melyik weboldal elemre volt klikkelve, hanem magát a klikk helyét küldi el a szerver oldalra. Így nem fordulhat elő rosszul értelmezett nem megfelelő helyen visszaadott hő térkép.

- Alkalmazásunk minimális számú metódust használ a nyomon követéshez, így használata minimális plusz terhelést jelent web szerverünkre.

- Böngészőben való futás lehetővé teszi, hogy platform független legyen az alkalmazás. Nincs szükség semmilyen előzetes konfigurálásra.

- Szoftver frissítése esetén nincs szükség újratelepítésre, mindig a legfrissebb változat futtatható.

- A világ bármely részéről elérhető.

- Kevesebb memóriát és CPU időt követel, mint egy asztali alkalmazás, mivel az munka oroszlán része szerver oldalon történik.
- Kevesebb hiba: web alapú alkalmazások kevésbé hajlamosak összeomlani, más lokális szoftverekkel nem kerül összeütközésbe. Mindenki ugyanazt a verziót használja, minden hibajavítás azonnal használható.

Mik lehetnek alkalmazásunk hátrányai?

- Alkalmazásunk nem tudja a különálló felhasználók szokásait megfigyelni. A nagy átlag szokásai lesznek a mérvadóak és ezek fogják dominálni a visszaadott képet.
- Nincs folyamatos interakció a felhasználó és az alkalmazás között.
- Kevésbé biztonságos, mint egy asztali alkalmazás, mivel web alkalmazásokat egy széles kör elérheti.
- Ha a felhasználó kikapcsolja a Javascript támogatást a böngészőjében, mert megteheti, akkor adminisztrátori felületünk nagy része használhatatlanná válik.

4.1.1. Fejlesztői környezet

Kliens oldali kód

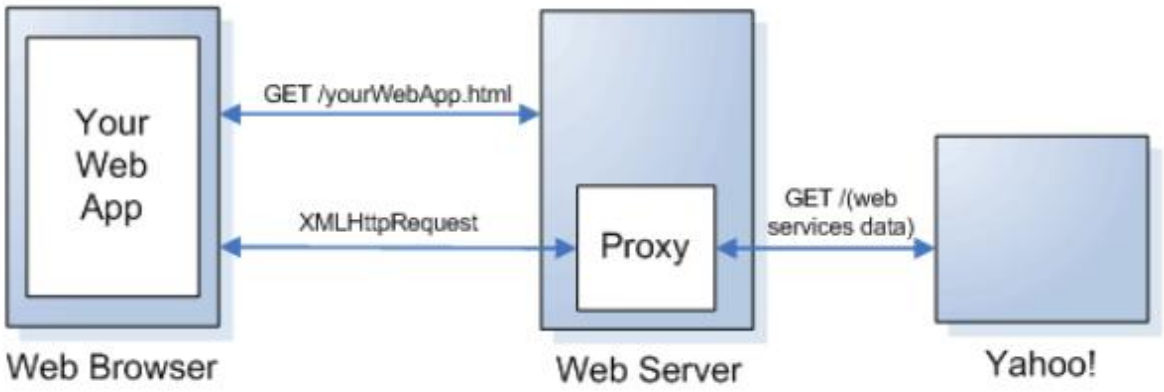
Kliens oldalon az adminisztrátori oldal bizonyos elemeinél, továbbá a klikk sorozatok elküldése esetén javascriptet használunk. Mi is a javascript? Javascript az egy objektumalapú szkript nyelv, böngészőben futtatható, melyet weboldalakon elterjedten használnak. Segítségével programozás-technikailag elérhetünk objektumokat más alkalmazásokon belül. Leggyakrabban ezt a böngészőben található objektumokra tesszük. Sok nyelv befolyással volt a Javascriptre és úgy lett megtervezve, hogy a Java-ra hasonlítson.

Adminisztrátori weboldalon használt Javascript csupán a használat megkönnyítését szolgálja és a megjelenítésben van szerepe. Ezért ezt nem is részletezzük.

A weboldalak nyomon követése (tacking) esetén használt javascript kód már érdekesebb lehet. Az alap probléma az, hogy a felhasználó által klikkelt hely koordinátáit el tudjuk juttatni szerver oldalra. Erre létezik egy beépített megoldás XMLHttpRequest API formájában. Viszont van egy hátulütője a mi szempontunkból, használata korlátozva van az azonos származás politika miatt (same origin policy). Ez a gyakorlatba azt jelenti, hogy csak a szerver HOST neve alatt található oldalakat tudnánk

nyomon követni. Alkalmazásunk viszont úgy épül fel, hogy egy központi szerver korlátlan (amennyit a fizikai tároló kapacitása megenged) számú különböző HOST név alatt lévő weboldalt tudjon nyomon követni.

Erre a problémára egyik megoldás lehet az, ha web szerverre egy proxy-t telepítünk, mely fogadja a böngészőből érkező kérést és azt a megfelelő helyre továbbítja. Így kérésünk eljut a célhoz és a böngésző is meg fogja engedni a kérdés indítását mivel a kérés a „saját" web szerverre érkezik be. A Proxy a hívott web szolgáltatástól (web service) visszakapott adatot továbbítja a böngésző felé, mintha csak tőle kapta volna (Fig: 4.1).



4.1. ábra. Proxy

Ez egy működő megoldás, de biztonsági okokból korlátoznunk kell használatát. Egy nyitott Proxy, mely kapcsolatot továbbit bármely web oldal URL-hez veszélyes lehet. Viszont a proxy kapcsolatot csak a saját alkalmazásunkra korlátozni elég nehéz, meg kell adnunk a szervereket melyekhez engedélyezni szeretnénk a kapcsolódást. Ez folyamatos karbantartást igényel, amit nem szeretnénk.[16]

Az előbbiekben említett megoldás nem úgy hangzik mint egy igazi AJAX, amit elképzeltünk. Mivel mi úgy akarjuk használni mint ahogy az XMLHttpRequest működik, ahol a böngészőből tudunk URL-t hívni, azzal a különbséggel, hogy nem csak azonos domain névre.

```
<script type="text/javascript">
    function xss_ajax(url) {
        var script_id = null;
        var script = document.createElement('script');
        script.setAttribute('type', 'text/javascript');
```

```
        script.setAttribute('src', url);
        script.setAttribute('id', 'script_id');

        script_id = document.getElementById('script_id');
        if(script_id){
            document.getElementsByTagName('head')
[0].removeChild(script_id);
        }

        // Insert <script> into DOM
        document.getElementsByTagName('head')
[0].appendChild(script);
    }

    function callback(data) {
        var txt = '';
        for(var key in data) {
            txt += key + " = " + data[key];
            txt += "\n";
        }
        alert(txt);
    }

    var url = "http://www.test.org/callback_json.php";

</script>
```

A fenti Javascript kódrészletben két függvényt láthatunk: xss_ajax() és callback(). Az első, xss_ajax(), intézi a http kérést a megfelelő URL-hez. Működését tekintve utánozza az XMLHttpRequest open() és send() metódusainak kombinációját. Paraméterként megadott URL-t nem korlátozza a közös-ős politika, lehet bármilyen. A második függvény, callback() fogja feldolgozni a http kérés után visszaadott adatokat.

Egyetlen dolog amiben különbözik az XMLHttpRequest API-tól, hogy a http kérésre kapott válasz nem lehet valós JSON vagy XML. Helyette egy sztring kezelő függvény hívásával kell felépíteni, ahol a paraméterként kapott sztring mag az objektum valós for-

mája, amit várunk visszatérési értékként. Egy http kérésre kapott válasz lehet az alábbi:
callback("firstname":"John","lastname":"Smith","email":"john.smith@johnsmith.com");

A varázslat az egész mögött a <script> DOM (Document Object Model) elem használatában van. A web böngésző képes betölteni JavaScript forráskódot idegen domain név alól is. Ha ezt megtehetjük akármilyen *.js fájlokra, akkor megtehetjük bármilyen más kiterjesztésre is pld. *.php, *.xml stb.

```
    <script type="text/javascript"
src="http://external-domain.com/mypage.html">
</script>
    <script type="text/javascript"
src="http://external-domain.com/mypage.cgi">
</script>
    <script type="text/javascript"
src="http://external-domain.com/mypage.php">
</script>
    <script type="text/javascript"
src="http://external-domain.com/mypage.xml">
</script>
```

Erre alapozva, hogy utánozzuk az XMLHttpRequest féle hívást, nincs más teendőnk, minthogy futás közben minden http kérés esetén dinamikusan létrehozzunk (vagy módosítunk) egy script elemet. Mivel a script tag elem JavaScript kódot vár, ezért az URl-ről kapott válasz kötelezően futtatható kell legyen, mint JavaScript kód. Ezért is használunk egy callback() függvényt, amelyet script tag létrejötte után a böngésző futat. Ha kérésünk esetén szükséges paramétereket is megadnunk, csak egyszerűen egy „?” után URL-nél megszokott módon utána írjuk név=érték formában, & elválasztva akár többet is.[17]

```
<script type="text/javascript"
   src="http://otherdomain.com/mypage.php?name=JOHN&password=secret">
</script>
```

Szerveroldali kód

MVC. Ebben a részben azt mutatjuk be, hogyan építünk fel egy MVC alapú keretrendszert PHP 5-öt használva. Az új PHP 5-nek és a benne található OOP funkcióknak köszönhetően egy sokkal stabilabb API-ról és egy komplexebb MVC keretrendszerről beszélhetünk. Ez eddig is megvalósítható volt, de sokkal több bonyodalommal járt.

Az MVC, vagyis Model-View-Controller egy szoftvermérnöki munkában használt szerkezeti minta. Összetett, sok adatot a felhasználó elé táró számítógépes alkalmazásokban gyakori fejlesztői kívánalom az adathoz (modell) és a felhasználói felülethez (nézet) tartozó dolgok szétválasztása, hogy a felhasználói felület ne befolyásolja az adatkezelést, és az adatok átszervezhetők legyenek a felhasználói felület változtatása nélkül. A **M**odell-**N**ézet-**V**ezérlő (**MNV**) ezt úgy éri el, hogy elkülöníti az adatok elérését és az üzleti logikát az adatok megjelenítésétől és a felhasználói interakciótól egy közbülső összetevő, a vezérlő bevezetésével. A mintát Trygve Reenskaug írta le először 1979-ben, miután a Smalltalkon dolgozott a Xerox kutatói laboránál. Az eredeti megvalósítás részletesen a nagyhatású Applications Programming in Smalltalk-80: How to use Model-View-Controller című tanulmányban olvasható.

Gyakori egy alkalmazás több rétegre való felbontása: megjelenítés (felhasználói felület), tartománylogika és adatelérés. Az MNV-ben a megjelenítés tovább bomlik nézetre és vezérlőre. Az MNV sokkal inkább meghatározza egy alkalmazás szerkezetét, mint az egy programtervezési mintára jellemző.

Modell Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A tartománylogika jelentést ad a puszta adatnak (pl. kiszámolja, hogy a mai nap a felhasználó születésnapja-e, vagy az összeget, adókat és szállítási költségeket egy vásárlói kosár elemeihez). Sok alkalmazás használ állandó tároló eljárásokat (mint mondjuk egy adatbázis) adatok tárolásához. Az MNV nem említi külön az adatelérési réteget, mert ezt beleérti a modellbe.

Nézet Megjeleníti a modellt egy megfelelő alakban, mely alkalmas a felhasználói interakcióra, jellemzően egy felhasználói felületi elem képében. Különböző célokra különböző nézetek létezhetnek ugyanahhoz a modellhez.

Vezérlő Az eseményeket, jellemzően felhasználói műveleteket dolgozza fel és válaszol rájuk, illetve a modellben történő változásokat is kiválthat.

Az MNV gyakran látható webalkalmazásokban, ahol a nézet az aktuális HTML oldal, a vezérlő pedig a kód, ami összegyűjti a dinamikus adatokat és létrehozza a HTML-ben

a tartalmat. Végül a modellt a tartalom képviseli, ami általában adatbázisban vagy XML állományokban van tárolva.

Habár az MNV-nek sok értelmezése létezik, a vezérlés menete általánosságban a következőképp működik:

1. A felhasználó valamilyen hatást gyakorol a felhasználói felületre (pl. megnyom egy gombot).

2. A vezérlő átveszi a bejövő eseményt a felhasználói felülettől, gyakran egy bejegyzett eseménykezelő vagy visszahívás útján.

3. A vezérlő kapcsolatot teremt a modellel, esetleg frissíti azt a felhasználó tevékenységének megfelelő módon (pl. a vezérlő frissíti a felhasználó kosarát). Az összetett vezérlőket gyakran alakítják ki az utasítás mintának megfelelően, a műveletek egységbezárásáért és a bővítés egyszerűsítéséért.

4. A nézet (közvetve) a modell alapján megfelelő felhasználói felületet hoz létre (pl. a nézet hozza létre a kosár tartalmát felsoroló képernyőt). A nézet a modellből nyeri az adatait. A modellnek nincs közvetlen tudomása a nézetről.

5. A felhasználói felület újabb eseményre vár, mely az elejéről kezdi a kört.

A modell és a nézet kettéválasztásával az MNV csökkenti a szerkezeti bonyolultságot, és megnöveli a rugalmasságot és a felhasználhatóságot.

Az ötlet az MVC keretrendszer mögött nagyon egyszerű és rendkívül rugalmas. Az elképzelés az, hogy van egy vezérlő (mint az index.php), ami ellenőrzi az alkalmazások indítását a kérések argumentumaira alapozva. A kérés általában magában foglal legalább egy argumentumot, amely azt definiálja, melyik modellt kell meghívni, egy eseményt, és a szokásos GET argumentumokat. Onnan az adatkezelő érvényesíti a kérést (hitelesítés, érvényes modell, stb.) és futtatja a kért eseményt. Például egy kérés: /index.php?module=foo&event=bar betölt egy foo nevű osztályt és futtatja a foo::bar()-t. Ezen módszer előnyei közé tartozik:

1. egy belépési pont minden alkalmazáshoz

2. megszünteti a számos script - mindegyik saját relatív útvonalakkal, adatbázis kapcsolatokkal, hitelesítéssel, stb. - fenntartását

3. lehetővé teszi a konszolidációt és a kód újrahasznosítást.

Keretrendszerünk. Felhasználjuk a PEAR-t[1], ami egy keretrendszer újrafelhasználható PHP komponensekkel.
Az osztályhierarchiánk a következőképpen néz ki:
-FR_Object az ősosztály az alap jellemzőkkel amiket minden objektum igényel: login, setForm(), toArray(); Ebből származik két osztály:

- FR_Object_DB: ez a réteg az adatbázissal teremt kapcsolatot
  - FR_Object_Web user információkat és session-eket tartalmaz
    - FR_Module : ebből az osztályból származtatjuk az összes alkalmazást
      - FR_Auth : authentikációs osztály
      - FR_User
      - Fr_Admin az alkalmazáson belül létező két felhasználói szint szétválasztásáért.
      - FR_NoAuth : osztály azért ha bizonyos modulokat el szeretnénk érni authentikáció nélkül is
- FR_Presenter : ez az alap nézet osztály, ez kezeli a betöltést és a megjelenítést
  - FR_Presenter_smarty: a prezentációs réteg képes különböző templatek betöltésére.
  - FR_Presenter_debug: megjelenítési mód kiegészitve az adatbázis kérések, futási idő megjelenítésével.
  - FR_Presenter_raw: a böngészőben futó JavaScript kérésekre ezen a osztályon keresztül adunk válaszokat. Annyiban tér el az elöbbiektől, hogy nincsenek html tagek használva megjelenítés (html kód generálás) esetén, nincs egy alap html oldalt felépítő template benne.

Kontroller Kezeli a beérkező kéréseket. Az URI-ből olvassuk ki azt az információt, hogy melyik modult, osztály és eseményt kell betölteni.
Pl. http://example.com/index.php?module=users&class=login
-module : megmondja, hogy melyik modult kell betölteni
-class: melyik osztályt kell betölteni, ha nem adunk meg osztályt akkor a kontroller megpróbálja azt az osztályt betölteni, amelynek neve megegyezik a module-ban leírtakkal

[1] Jelenleg 40 kategóriában 405 csomag áll a fejlesztők rendelkezésére, 215 fejlesztő dolgozik a csomagok naprakészen tartásán és továbbfejlesztésén, és a csomagokat együttesen eddig közel 12 milliószor töltötték le.

-event: megmondja, melyik metódust kell futtatni miután autentikáltuk a felhasználót

View Alapértelmezetten a Smarty osztályt értelmezi. Miután a Model fut, a keretrendszer FR_Presenter::factory() metódust használja a presenter megkreálásához.

Model Ebben a részben helyezkedik el a logika legnagyobb része. Itt hajtódnak végre a lekérdezések az adatbázison és a bemeneten a számítások. Egy jó példa a login script. A login script megadja a felhasználó adatait a formról, validálja azt az adatbázisban, majd belépteti a felhasználót.

4.2. Logikai háttér

A web alkalmazás használatba vételéhez be kell jelentkezni. Két szint létezik az alkalmazáson belül, egyszerű felhasználói szint és adminisztrátori szint. Továbbiakban a felhasználó az egyszerű felhasználót és adminisztrátort együtt jelenti, minden más esetben egyszerű felhasználót használok. Az egyszerű felhasználó bárki lehet, aki regisztrációs űrlapot kitöltve regisztrál a weboldalra. Egyszerű felhasználóból adminisztrátor csak úgy lehet, ha egy már meglévő adminisztrátor kinevezi őt annak, viszont ekkor elveszti jogait, mint egyszerű felhasználó. Igen, az egyszerű felhasználó szintet nem tartalmazza egészében az adminisztrátor szint. Adminisztrátor nem láthatja az egyszerű felhasználó nyomon követési időbeosztását, pontjainak csoportosítását, beszámolóit.

Milyen pontokról is beszélünk? Minden egyszerű felhasználó regisztrációkor kap 1000 pontot, amely alapértelmezetten 1000 klikk nyomon követését teszi lehetővé. Ezt a pontszámot az alkalmazást üzemeltető ajándékának kell tekinteni és a termék ingyenes kipróbálását teszi lehetővé. Viszont minden ezen felüli pontot valamilyen ellenszolgáltatás fejében kaphat csak meg az egyszerű felhasználó. Ezért is létezik két szint az alkalmazáson belül. Adminisztrátorként az egyszerű felhasználó adatait és pontjait menedzseljük, egyszerű felhasználóként ezeket a pontokat költhetjük el.
Az alkalmazás a következő funkciókat biztosítja miután a felhasználó bejelentkezett:
Az egyszerű felhasználó:

- megváltoztathatja regisztrációs adatait
- szabályokat adhat pontjai elköltésére (kampány nyomon követésére)

- megnézheti a kívánt időszakok jelentéseit hő térkép formájában.

Adminisztrátor:

- módosíthatja az egyszerű felhasználó regisztrációs adatait
- státuszát megváltoztathatja (active, disabled) a felhasználónak
- kinevezhet adminisztrátorrá egy egyszerű felhasználót
- pontokat adhat egy egyszerű felhasználó meglévő pontjaihoz (ez lehet negatív értékű is) a befizetései alapján
- megnézheti egy egyszerű felhasználó eddigi számláit

4.2.1. Adatbázis táblái és kapcsolatuk

Az adatbázis nyolc táblából áll:

- FR_CLICK_ACCOUNT_ARH
- FR_CLICK_ACCOUNT_TMP
- FR_TRACK_CLICKS
- FR_USERS
- FR_USERS_HOSTS
- FR_USERS_IMAGES
- FR_USERS_PAGES
- FR_USERS_TRACK_SESSIONS

Az FR_CLICK_ACCOUNT_ARH táblát az egyszerű felhasználó számlájához hozzáadott pontok archiválására használjuk. Ebbe a táblába az alkalmazás csak ír, minden az adminisztrátor által hozzáadott vagy levont pontot itt regisztrálunk. Így a későbbiekben nyomon követhető a számlatörténet. Az FR_CLICK_ACCOUNT_ARH tábla oszlopai:

1. userID(int(11)) : A felhasználó egyedi azonosítója
2. amount (bigint(20)) : A regisztrált pontok száma egész számként megadva

3. date (datetime) : A pontok regisztrációjának ideje

4. notice (varchar(100)) : Megjegyzés rész, ha szükséges valamit hozzáfűzni

Mind az adminisztrátor mind az egyszerű felhasználó ugyanazon bejelentkezési ablakon kell megadniuk adataikat (email, jelszó), bejelentkezés után adminisztrátor az általános adminisztrátori oldalra kerül, az egyszerű felhasználó pedig, a saját személyre szabott felhasználói oldalára. Az FR_CLICK_ACCOUNT_TMP táblát az egyszerű felhasználó aktuális pontjainak az adminisztrálására használom. Minden pontlevonás ebben a táblában rögzül. Az adminisztrátor által hozzáadott vagy levont pontszám a FR_CLICK_ACCOUNT_ARH táblán kívül ebbe a táblába jegyződik be. Azért használok két különböző oszlopot (available_clicks, spent_click) látszólag egy oszloppal is megadható értékre, mivel havi záráskor (melyet a rendszer automatikusan végez) a spent_click értéke rögzül az FR_CLICK_ACCOUNT_ARH táblába, mint a múlt havi elköltött pontszám, majd ezután a spent_click nullázódik a következő hónapot előkészítve. Az FR_CLICK_ACCOUNT_TMP tábla oszlopai:

1. userID (int(11)) : A felhasználó egyedi azonosítója

2. available_clicks (bigint(20)) : A rendelkezésére álló pontszám

3. spent_clicks (bigint(20)) : Két zárás között elköltött pontszám

Az FR_TRACK_CLICKS táblát a nyomon követett oldalt böngésző klikkeléseit rögzíti. A short_session és long_session-t az alkalmazás jelen verziója nem használja, de későbbi változatának igen hasznosnak bizonyulhat, mivel ez lehet az alapja sok adatbányászati algoritmusnak. A short_session és a long_session is cookie formájában tárolódik el a böngészőben. Short_session élete munkamenet hosszáig tart, míg long_session egy évig. Megemlítendő még a width oszlop, amely azért szükséges, mivel a weboldalak különböző felbontásban különböző helyen jelennek meg (alkalmazásunk azt feltételezi, hogy a weboldal rögzített szélességűek, így különböző felbontás esetén csak a weboldalon kívüli rész mérete változik). Így különböző felbontások esetén ki tudjuk számolni a felbontások közötti különbséget, tehát egységes jelentésben nézhetjük a nyomon követett klikkeket. Az FR_TRACK_CLICKS tábla oszlopai:

1. pageID (bigint(20)) : a nyomon követet oldal azonosítója

2. click_x (int(11)) : a klikkelt hely x koordinátája

3. click_y (int(11)) : a klikkelt hely y koordinátája

4. width (int(11)) : a böngészőben megjelenített klikkelt weboldal szélessége, ez felbontásfüggő (800, 1024, 1280 ...)
5. date (date) : a klikkelés dátuma
6. short_session (varchar(40)) : egy munkamenethez tartozó hash kód
7. long_session (varchar(40)) : egy felhasználóhoz tartozó hash kód, böngésző bezárása után se változik, mivel cookie-ban tárolódik

Az FR_USERS táblát felhasználó adatainak a tárolására használom. A status és role oszlopok csak adminisztrátori hozzáféréssel változtathatóak. Az FR_USERS tábla oszlopai:

1. userID (int(11)) : A felhasználó egyedi azonosítója
2. email (varchar(45)) : A felhasználó email-je
3. password (varchar(30)) : A felhasználó belépéshez szükséges kódja
4. fname (varchar(30)) : A felhasználó vezetékneve
5. lname (varchar(30)) : A felhasználó keresztneve
6. address (varchar(100)) : A felhasználó teljes címe
7. tel (varchar(30)) : A felhasználó telefonszáma
8. posted (date) : A felhasználó regisztrációjának ideje
9. status (enum('active','disabled')) : A felhasználó státusza
10. role (enum('admin', 'user')) : A felhasználó jogosultsági szintje

Az FR_USERS_HOSTS táblát a felhasználók által nyomon követett weboldalak HOST neveit tartalmazza. Az FR_USERS_HOSTS tábla oszlopai:

1. id (int(11)) : Az adott host azonosítója
2. userID (int(11)) : Azon felhasználó azonosítója akihez a host tartozik
3. host (vachar(100)) : A host név (htttp://inf.unideb.hu - formában megadva)
4. notice (varchar(100)) : Megjegyzés, ha szükséges

Az FR_USERS_IMAGES táblát a felhasználók jelentéseinek megjelenítésénél (hő térkép) használt képek adminisztrálására használjuk. Mivel a sebesség növelése érdekében, a jelentések csak külön kérésre frissülnek, ezért szükséges a képeket meg kell őrizni, frissítés esetén pedig törölni azokat. Az FR_USERS_IMAGES tábla oszlopai:

1. userID (int(11)) : Azon felhasználó azonosítója, akihez a képek tartoznak
2. pageID (int(11)) : Azon oldal azonosítója, amelyhez a képek tartoznak
3. date (date) : A képek elmentésének ideje
4. width (int(4)) : Az elmentett képek szélessége
5. height (int(4)) : az elmentett képek magassága
6. path (varchar(500)) : a képek nevei kiterjesztéssel együtt veszővel elválasztva
7. calendar(vachar(1000)) : azon időszakok melyeket felhasználva előállítottuk a képeket (veszővel elválasztva)

Az FR_USERS_PAGES táblában az adott host-hoz tartozó weboldalak host rész nélküli URL részét tároljuk. Az FR_USERS_PAGES tábla oszlopai:

1. id (int(11)) : A weboldal egyedi azonosítója
2. hostID (int(11)) : Azon host azonosítója, melyhez a weboldal tartozik
3. url_path (varchar(50)) : Az URL host utáni, de kérdés (query - ?) elötti része
4. url_query (varchar(100)) : Az URL kérdés része

Az FR_USERS_TRACK_SESSIONS táblában a nyomon követési szabályokat (kampány időtartama és elköltendő pontok száma) tároljuk.
Az FR_USERS_TRACK_SESSIONS tábla oszlopai:

1. id (int(11)) : A szabály egyedi azonosítója
2. hostID (int(11)) : Azon host azonosítója melyhez a szabály tartozik
3. click (bigint(20)) : A nyomon követendő klikkek száma
4. date_from (date) : A nyomon követési idő intervallum kezdő dátuma
5. date_to (date) : A nyomon követési idő intervallum végső dátuma
6. notice (varchar(100)) : Megjegyzés, ha szükséges

Az alkalmazás tábláinak sematikus ábrája:

4.3. Felhasználói leírás

Alkalmazás telepítése. Az alkalmazás használatba vételéhez nem igényel semmilyen telepítést. Egy egyszerű JavaScript futtatására alkalmas böngészővel használható.

Alkalmazás indítása. A böngészőben meg kell adni web alkalmazásunk gyökér könyvtárának elérhetőségét, ez teszt környezetünkben a következőképpen néz ki: http://localhost/diplomamunka/framework/

Az alkalmazás használata. A webes alkalmazás indítása után az első weboldal, amit látni fogunk az egy bejelentkezésre szolgáló űrlap, email és jelszó mezővel. Ahhoz hogy be tudjunk jelentkezni meg kell adnunk mind az email címünket, mind a jelszavunkat.

4.2. ábra. Beléptetés

Bejelentkezéskor a következő hibák fordulhatnak elő:

1. A felhasználó nem jó formátumban adta meg az email címét:

4.3. ábra. Formailag hibás email cím

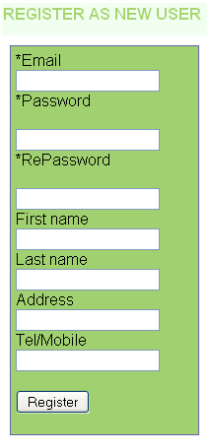
2. A felhasználó email címe nem található az adatbázisban:

4.4. ábra. Adatbázisban nem létező email cím

3. A felhasználó jelszava nem megfelelő:

4.5. ábra. A megadott email címhez tartozó jelszó hibás

4. Ha még nem vagyunk a weboldalon regisztrált tagok, akkor ezt megtehetjük a következő űrlap segítségével:



4.6. ábra. Regisztrációs űrlap

5. A *-al jelölt mezők kitöltése kötelező, ha ezt elmulasztjuk a következő hibaüzenetet kapjuk:

4.7. ábra. Kötelező mező kitöltése elmaradt

6. Ha olyan email címmel próbálunk regisztrálni amely már használatban van, a rendszer figyelmeztet.

4.8. ábra. A megadott email cím használatban van

7. Ha a regisztráció sikerrel zárul, akkor a következő üzenet jelenik meg:

4.9. ábra. Sikeresen regisztráltunk

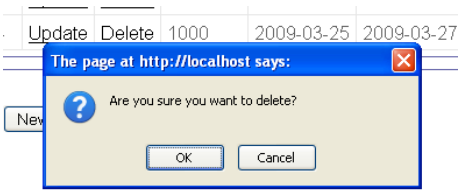
Az alkalmazás használata közben a rendszer üzeneteket jeleníthet meg a felületen. Ezek az üzenetek három kategóriába sorolhatók. Súlyos hiba, figyelmeztetés, visszaigazoló üzenet. Súlyos hiba lehet, adatbázis hiba történik, az alkalmazás tervezett futási menete megszakad. Figyelmeztetést kaphatunk, ha az alkalmazásban magában nem történik hiba, de logikailag a megadott adatok nem megfelelőek. Visszaigazoló (információ) jellegű üzenetet mentések sikeres telejésülése esetén kaphatunk. A felületen megjelenítve ehhez az üzenetek a következőképpen néznek ki:

4.10. ábra. Üzenetek

Minden tábla esetén megadjuk a tábla használatának rövid leírását, a táblában található oszlopok neveit, azok típusát (típus MYSQL adatbázis által használt típust jelent), és egy rövid megjegyzést. Ha egyszerű felhasználóként lépünk be, akkor az alábbi ábrán (Fig:4.11 látható weboldal fogad bennünket. Az elrendezést tekintve 3 fő részre van osztva. Fent található a fejrészben az üdvözlő szöveg, kijelentkezési lehetőség és a navigáláshoz szükséges menürendszer. Ezek a felületen való navigálás közben változatlanok. A középső rész egy dinamikus táblázatból áll melynek három része van. Az első az alaphelyzetben rejtett dinamikus naptár (később meglátjuk miért is dinamikus). A második maga a szabályokat host-onként megjelenítő dinamikus táblázat. A harmadik, az alsó kiemelt színű hasábban pedig új host hozzáadására alkalmas űrlap található. Itt egyszerűen beírjuk Host mezőbe az új host nevét http://host_neve alakban és a New Host gomb megnyomásával már hozzá is adtuk új host-unkta a dinamikus táblázathoz.

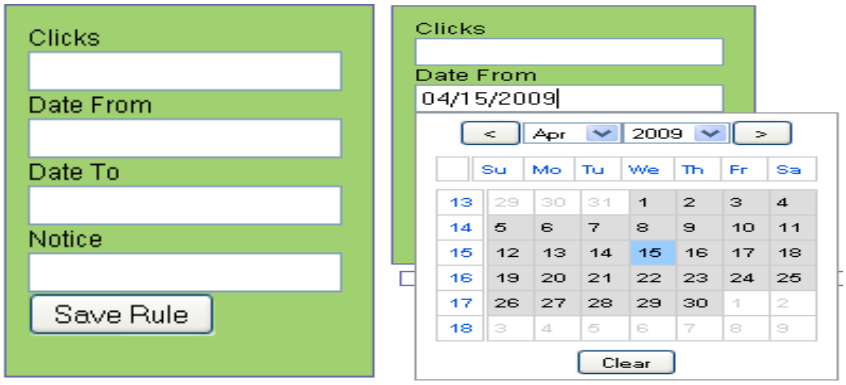
4.11. ábra. Egyszerű felhasználói oldal

A szabályokat megjelenítő dinamikus tábla tetején láthatunk egy Total clicks remained feliratot és utána egy számot. Ez a szám jelenti a még megmaradt klikkeket, adott pillanatban maximálisan még ennyi klikket fog az alkalmazás feldolgozni, rögzíteni. A dinamikus táblát függőlegesen további két részre oszthatjuk. A bal oldali részen találhatóak a host nevek, a jobb oldalon a szabályok. A host nevek és szabályok között 1:N kapcsolat áll fenn, egy host névhez több szabály tartozhat. Azokon a helyeken ahol nincs host név megadva (üres a host név rész) úgy kell értelmezni, hogy az adott sorban lévő szabály az egy fentebb lévő sorban (az első nem üres host nevű sorban) található host-hoz tartozik. Minden host esetén két műveletet tudunk végrehajtani. Ezeket a műveleteket a dinamikus tábla OP. Oszlopában találjuk. A megfelelő host név előtt található OP. Oszlopban lévő műveletek az adott host névre érendőek. Ez a két művelet a Delete, mely host törlésére szolgál és az AddRule melyre kattintva az adott host-hoz szabályt tudunk hozzáadni. Host törlésére klikkelve figyelmeztető ablak ugrik elő melyben meg kell erősítenünk törlési szándékunkat, vagy akár vissza is vonhatjuk azt:



4.12. ábra. Megerősítés

AddRule megnyomása után nyomon követési szabályt a lentebb látható űrlap kitöltésével tudunk megadni. Meg kell adnunk a kampány időszakban elkölteni akart pontok (klikkek) számát és a kampány idejét, kijelölve a kezdő és a vég dátumot.



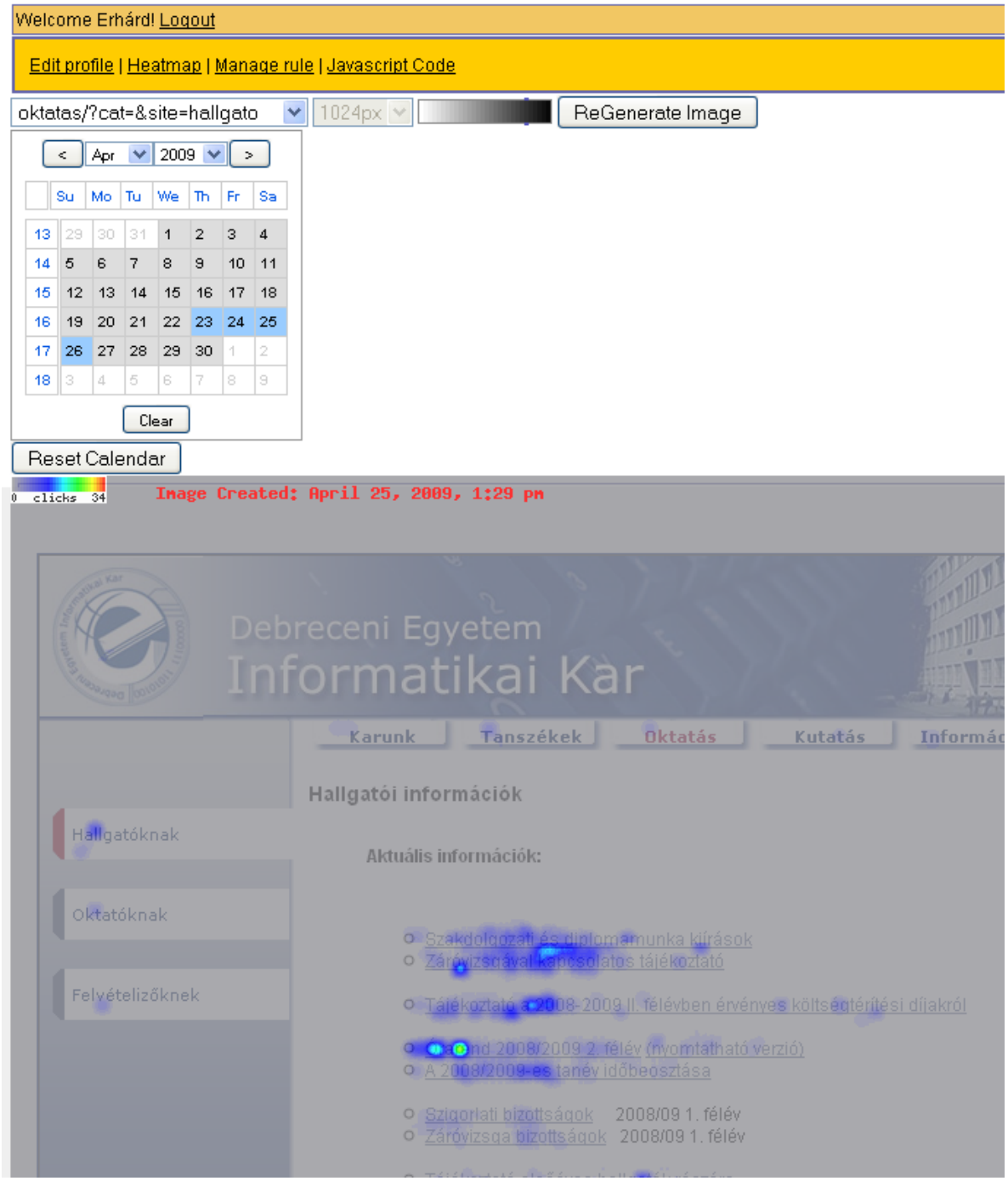
A dinamikus tábla jobb oldali részében láthatóak a szabályok. Minden szabályra az OP. Oszlopban látható két művelet értelmezett. Az első az Update mely segítségével módosíthatjuk szabályunkat, ekkor a szabály létrehozásakor már megismert űrlap jelenik meg, ahol elvégezhetjük a kívánt módosításokat. A második művelet a törlés melyre kattintva törölhetjük az adott szabályunkat, merősítés itt is szükséges. A dinamikus tábla további oszlopaiban tartalmazza a szabályokhoz tartozó klikk számot, szabály kezdetének érvényességét, végét és egy megjegyzés oszlopot, ahol emlékeztetőt írhatunk, hogy miért is fontos az adott szabály.

Táblázatunk több host és egy hosszabb időszakot lefedő szabály sorozat esetén hamar több 10 soros lehet. Így az adott szabályok esetén megadott időintervallumok könnyen átfedhetik egymást, előfordulhat, hogy úgy is többszörösen költjük el pontjainkat, hogy az nem volt eredetileg a szándékunk. Ha ezt el szeretnénk kerülni a Calendar gombra klikkelve előugrik egy dinamikus naptár. Ebben a naptárban az olyan napok esetén ahol van aktív szabályunk, kéken jelenik meg az adott nap, olyan esetben, ha átfedés történt, az adott nap pirosan jelenik meg. Így könnyen megnézhetjük, hogy az év mely napjaiban figyeljük meg felhasználóinkat és az esetleges ütközéseket is könnyen kiszűrhetjük (Fig:4.13). A Calendar gombra újból klikkelve a naptár eltűnik.

4.13. ábra. Naptár

A navigációra szolgáló menüben felfedezhetünk még három menüpontot. Az első ezek közül az Edit Profile, itt módosíthatjuk regisztrációkor megadott adatainkat. Az utolsó a Javascript Code ahol a mindig naprakész változata található annak a kis Javascript kódnak, amit a nyomon követni kívánt HTML weboldalunk <head> részébe kell beillesztenünk. A második és egyben a legfontosabb menü a Heatmap ahol magát a

kimutatásokat tudjuk megtekinteni hő térkép formájában. Ha a Heatmap menü gombra klikkelünk, akkor a következőhöz hasonló kép fogad majd bennünket:



4.14. ábra. Az Informatika Kar információs weboldaláról saját adatok alapján készített hőképe

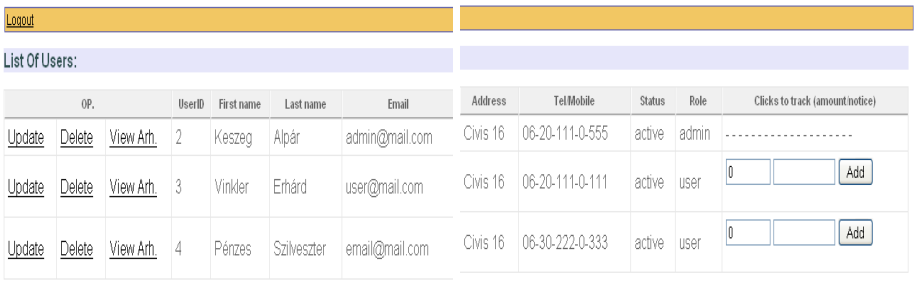
A felső sorban találunk egy legördülő menüt. A legördülő menüre klikkelve kivá-

laszthatjuk az adott weboldal adott weblapját. A Menüben található elemek weboldalanként vannak csoportosítva. Ha kiválasztunk egy elemet, akkor a neki megfelelő hő térkép töltődik be. Ha még nem volt az adott weboldalhoz hő térkép generálva, azt tapasztalhatjuk, hogy az oldalunk betöltés kicsit hosszabb ideig tart majd.

4.15. ábra. Legördülő menü

Legördülő menünk után egy jelenleg nem változtatható elem található mely 1024px -re van állítva. Ez azt jelenti, hogy attól függetlenül, hogy a felhasználó milyen felbontásban használta weboldalunkat, mi az eredményt mindig 1024px-es megjelenítéshez igazítva látjuk. A következő elem egy kis csúszka melyel azt szabályozhatjuk, hogy a hő térképünk mennyire legyen átlátszó, mennyire látszódjon a mögötte elhelyezkedő weboldal. Ebben a sorban a legutolsó elem egy ReGenerate Image feliratú gomb melynek az a funkciója, hogy az adott naptárban kijelölt időszakok mellett újragenerálja a hő térképet. A következő elem egy interaktív naptár. Itt az adott napokra, hetekre klikkelve kijelölhetjük azokat, így azok kék színűek lesznek. Hő térképünk generálása esetén azon napok eredményeit fogja megjeleníteni a hő térkép, amelyeket kijelöltük a naptárban. Naptárunkban kijelölt elemre még egyszer klikkelve a kijelöltség eltűnik, ezt megtehetjük az összes elemre, ha a ResetCalendar gombra klikkelünk. Ha nincs egyetlen naptári napunk se kijelölve, akkor figyelmeztető üzenetet kapunk. Továbbá figyelmeztető üzenetet kapunk akkor is ha, az adott időszak nem volt egyetlen klikk sem a megfigyelt oldalon. Alkalmazásunk az erőforrások megkímélése végett csak addig generál hő képet ameddig a weboldalon klikk helye található. Ezért előfordulhat, hogy weboldalunk a megjelenítettnél hosszabb (szélesebb) de nem jelenik meg az a része ahol már nem volt klikk regisztrálva. A hő kép bal felső sarkában található egy szín paletta és egy dátum. A dátum a kép generálásának idejére vonatkozik. A szín paletta és az alatta található számok jelmagyarázatként érendőek. Megmutatják, hogy a hő kép színeit hogyan kell értelmezni, melyik színárnyalat hány klikk azonos helyének felel meg. Ahogyan azt az előbbiekben is említettük az alkalmazásba kétféle felhasználói szinten lehet bejelentkezni, adminisztrátorként illetve egyszerű felhasználóként. Mindkét fel-

használó típus ugyanazon az űrlapon jelentkezik be, az alkalmazás dönti majd el, hogy melyik oldalra továbbítja az adott felhasználót jogainak megfelelően. Ez nem jelenti azt hogy magát az URL-t megváltoztatva bejelentkezés után meg tudnánk nézni akármelyik felhasználó vagy adminisztrátor weboldalát. Alkalmazásunkban már modul szinten elkülönül a két felhasználói típus, továbbá a bejelentkezett egyszerű felhasználó azonosítója munkamenet szinten van tárolva. Így felhasználói oldalon az URL bármi nemű manipulációja más adatainak megtekintésének céljából nem lehet sikeres. Az egyszerű felhasználó bejelentkezése után kapott felületet az előbbi részben kimerítően tárgyaltuk. Most az adminisztrátori felület következik. Az adminisztrátori oldalon elvégezhető teendők száma kevés, csupán a felhasználók és pontjaik menedzselése, így maga a felület is egyszerű, csupán egyetlen oldalból áll. Viszont éppen ezért az összes információ egyetlen dinamikus táblázatban található, melynek mérete meglehetősen nagy. Hogy jobban látható, értelmezhető legyen két fél képként mutatom be.



4.16. ábra. Admin oldal

A dinamikus tábla első részében az OP. Oszlopban találhatóak az egy adott felhasználóra alkalmazható műveletek. Update segítségével módosíthatjuk egy felhasználó regisztrációkor megadott adatlapját. Adminisztrátorként a felhasználó adatlapjából látott rész kiegészül két új mezővel, egy státusz és egy szerep (felhasználó szintje) mezővel. Státusz mezőben aktív felhasználót le tudunk tiltani, illetve tiltást fel tudunk oldani. Role mezőben felhasználót ki tudunk nevezni adminisztrátornak. Regisztrációkor mindenki alanyi jogon megkapja az aktív státuszt és egyszerű felhasználó szerepkört. Delete segítségével törölhetünk egy felhasználót a rendszerből, felhasználó törlését az előbbiekhez hasonlóan, mint minden törlés esetén, felugró ablakban jóvá kell hagynunk. View Arh. -ra kattintva megtekinthetjük az adott felhasználó számlatörténetét. Ez az alábbihoz hasonló időrendi sorrendbe rendezett táblázat lehet:

4.17. ábra. Beléptetés

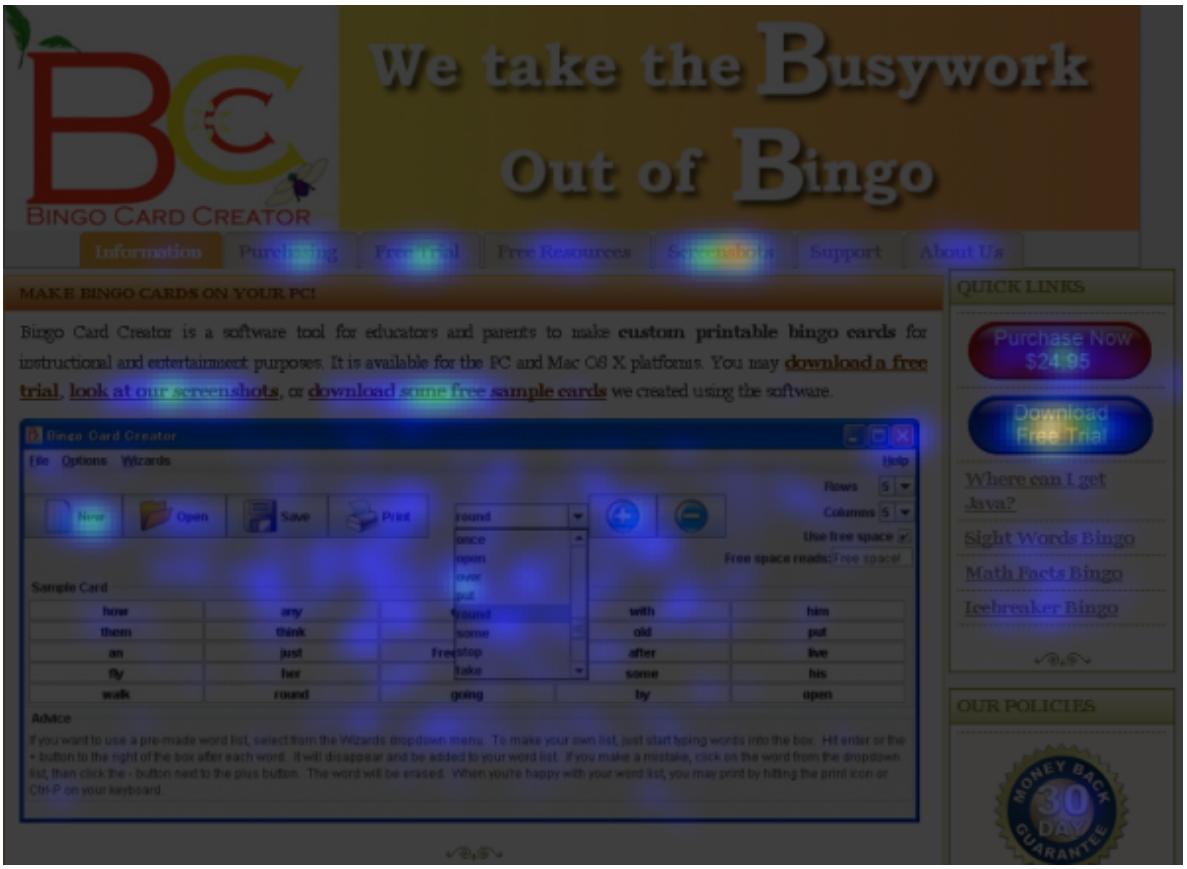
A dinamikus táblában az OP.-t követő oszlopokban a felhasználó adatlapjának megfelelő oszlopok láthatóak. Utolsó, Clicks to track feliratú oszlopban adhatunk egy felhasználó számlájához adott mennyiségű pontot, megjegyzéssel együtt, az Add gombra kattintva. Ezt a hozzáadott pontszámot az egyszerű felhasználó azonnal igénybe is veheti.

4.4. Esettanulmány

Ebben a részben szeretnénk bemutatni, az alkalmazás működését valós környezetben, valós felhasználók klikk sorozatait megjelenítve. Mivel saját alkalmazásunk beüzemeltetéséhez nem sikerült partnert találni. Továbbá saját környezetünkben ismerőseink által tesztelt weboldalakból viszont nem sikerül kellően nagy mennyiségű adatot begyűjteni, hogy abból a megfelelő következtetés levonható legyen, ezért egy nagyon hasonló alkalmazás (www.CrazyEgg.com) valós adatain alapuló hő képét mutatnánk be. Itt továbbá lehetőség nyílik arra is, hogy az igen elterjedt google analitikus eszközével összehasonlításba kerüljön a hő képes elemzési mód. Mivel az CrazyEgg -es alkalmazás részleteitől eltekintünk, csak a hő képet elemezzük, ezért a saját alkalmazásunktól a CrazyEgg-es alkalmazás adathalmazával dolgozva azonos eredményt várunk el (azonos hő képet). Esettanulmányunkat Email Patrick [19] cikke alapján írtuk, ahol a BingoCardCreator.com weboldal analitikus és hő térképes jelentéseit elemzi. Lássuk az első képet a Google analitikus eszközének jelentéséről:

4.18. ábra. BingoCardCreator analytics

Ahogy azt láthatjuk a jelentés alapján a legnépszerűbb link a kék színű Downlaod Free Trial gomb. Viszont ami aggodalomra ad okot azaz, hogy a jelentés alapján elég sokan nyomkodják az Information gombot, mely erre az oldalra irányítja őket. Ebből arra következtethetünk, hogy nem tökéletesen értik az oldalon való navigálás módját a felhasználók. De ahogy ezt láthatjuk a későbbiekben feltevésünk hibás. Második képen CrazyEgg -es hő képet láthatunk:



4.19. ábra. BingoCardCreator hőképe

Ahogyan azt észrevesszük a hő képes jelentés alapján senki nem klikkelt az Information gombra. Hogy lehet ez? Ez egy hibája a Google analitikus eszközének, ha bizonyos böngészőkben a felhasználó egy klikk helyett duplán klikkel. Továbbiakban, a középső részen található download a free trial feliratú linkre senki nem klikkelt és az is látható, hogy a ScreenShots menügomb látogatottsága is jobban kiemelkedik a többi közül, mely az első képen nem volt látható. Az is észrevehető hogy a látogatók milyen nagy száma klikkel a Bingo alkalmazás képére, megpróbálva azt interaktivitásra bírni.

A jelentést elemezve, alkalmazhatunk néhány változtatást a weboldalon. Átírhatjuk promóciós szövegünket úgy, hogy a senki által nem használt link már ne legyen benne, vagy más legyen a felirata. Az alkalmazásról készült screenshot-okat egy képnéző keretbe rakhatjuk, így senki nem téveszti majd össze a valós alkalmazással. A screenshot-ra klikkelve pedig átirányíthat bennünket a fizetés weboldalra.

De miért is kapunk különböző eredményt Google analitikus eszköze és a hő képes elemzés esetén? A válasz egyszerű, ameddig Google analitikus eszköze link alapján ké-

szíti a jelentést addig az alkalmazásunk koordináták alapján. Így nem fordulhat elő, hogy ha egy web oldalon belül ugyanazon linket használjuk több helyen is, akkor nem tudjuk megkülönböztetni azokat jelentésünkben. Továbbá az olyan helyeket is felderíthetjük, ahol nincsenek linkek mégis sokan ott linket keresve próbálják használni oldalunkat.

Tévedés ne essék, habár az előbbiekben elmondottakat úgy is lehet értelmezni, hogy a Google szolgáltatásánál jobb a mi alkalmazásunk. Nem jobb, másképpen mutatja be a látogatók tevékenységét. Mely sok esetben egyszerűbbnek, könnyebben érthetőnek bizonyul. A legjobb eredményt az analitikus és hő képes megoldás együtt való alkalmazása esetén érhetjük el.

4.4.1. Továbbfejlesztési lehetőségek

Véleményünk szerint az alkalmazást a következő módokon lehetne továbbfejleszteni:

1. Nyomon követés (tracking) esetén, ha a weboldal nem középre van igazítva, akkor a jelentésben megjelenő klikkek rosszul lesznek értelmezve. Úgy kellene megírni az alkalmazást, hogy kevésbé legyen érzékeny a különböző elrendezésből adódó hibákra.

2. A rendszer hibatűrésének javítása. Jobban értelmezhető, több esetleges hibára kiterjedő kivétel.

3. Ne lehessen egy adott oldalra egy bizonyos idő elteltén belül újból klikkelni, a rossz szándékú klikkelők kiszűrése érdekében.

4. Dinamikus táblák esetén a lapozást vagy tördelést megoldani. Ha a tábla mérete elég nagy, akkor egy görgető hasáb jelenik meg, amellyel az egész tábla tartalmát görgetéssel bejárhatjuk. Viszont így könnyedén eltévedhetünk a tábla sorai között. Ezt orvosolhatná, ha 20-30 soronként a tartalom új lapon kezdődne, vagy külön táblában jelenne meg.

5. Dinamikus tábla sorai rendezhetőek legyenek bármely oszlop szerint az oszlopra kattintva.

Köszönetnyilvánítás

Szeretnék köszönetet mondani szüleimnek, hogy támogattak tanulmányaim során, valamint tanáraimnak, akik mindent megtettek azért, hogy a szak tudásanyagát maximálisan átadják, kiemelném közülük témavezetőmet, Dr. Ispány Mártont.

Irodalomjegyzék

[1] Dr. Bodon Ferenc: Adatbányászati algoritmusok, 2009.01.11. http://www.cs.bme.hu/ bodon/magyar/adatbanyaszat/tanulmany/adatbanyaszat.pdf

[2] Nyéki Lajos: A COEDU E-LEARNING KERETRENDSZER HASZNÁLATÁNAK ELEMZÉSE, 2007.08.23. http://bmf.hu/conferences/multimedia2007/26_NyekiLajos.pdf

[3] Adatbányászat: Koncepciók és technikák, matinf.atk.u-kaposvar.hu/public/kover/adatb

[4] Soumen Chakrabarti: Mining the Web, Morgen Kaufmann Publishers, 2003

[5] Bernard J. Jansen, Amanda Spink, Isak Taksa: Web Log Analysis, 2009

[6] Adriano Moreira, Maribel Y. Santos and Sofia Carneiro: Density-based clustering algorithms - DBSCAN and SNN, 2005.07.25. http://ubicomp.algoritmi.uminho.pt/local/download/SNN&DBSCAN.pdf

[7] Zdravko Markov, Daniel T. Larose: Data Mining The Web, 2007

[8] http://en.wikipedia.org/wiki/Web_mining

[9] http://www.cs.uic.edu/ liub/WebContentMining.html

[10] Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM, 46(5): 604 - 632, 1999, http://www.cs.cornell.edu/home/kleinber/auth.ps.

[11] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan, Web usage mining: discovery and applications of usage patterns from web data, SIGKDD Explor., 1(12), Jan. 2000.

[12] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami, Mining association rules between sets of items in large databases, in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data.

[13] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone, Classification and Regression Trees, Chapman & Hall/CRC Press, Boca Raton, FL, 1984.

[14] Ruby L. Kennedy, Yuchun Lee, Benjamin Van Roy, Christopher D. Reed, and Richard P. Lippman, Solving Data Mining Problems Through Pattern Recognition, Pearson Education, Upper Saddle River, NJ, 1995.

[15] http://www.onlamp.com/pub/a/php/2005/09/15/mvc_intro.html

[16] http://developer.yahoo.com/javascript/howto-proxy.html

[17] http://alvinabad.wordpress.com/2009/02/13/feb13/

[18] http://hu.wikipedia.org/wiki/Modell-nézet-vezérlő

[19] http://www.kalzumeus.com/2007/04/20/crazyegg-vs-google-analytics/