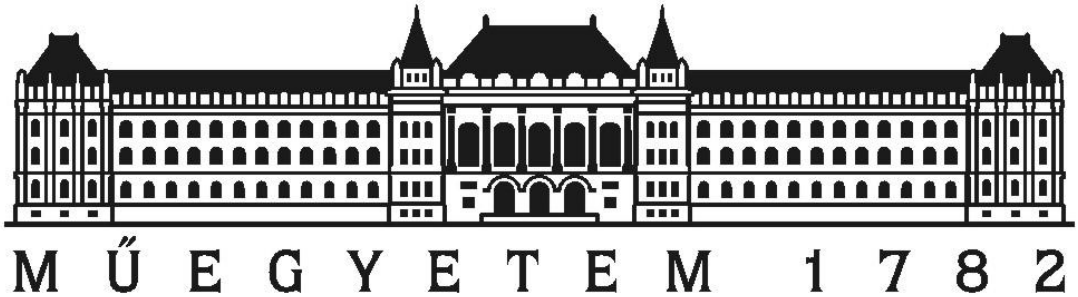


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai kar
Távközlési és Médiainformatikai Tanszék

Tudományos Diákköri Konferencia

Webshopok profitmaximalizálása adatbányászati technológiák segítségével.

Készítette:
Végh Ladislav

Konzulens:
Dr. Szűcs Gábor
Egyetemi docens

2011

Tartalom

1. Feladatleírás:

Az ajánlattevő motorok manapság egyre elterjedtebbek a webes világban, az e-shopokban hatékony ajánlatok tételére profitmaximalizálás céljából. A használt ajánlattevő algoritmusok meghatároznak egy adott terméket, vagy termékeket az eddig meglátogatott, vagy megvett termékek alapján úgy, hogy azt a felhasználó nagy valószínűséggel megnézze, vagy még jobb esetben meg is vegye. Ezzel próbálják maximalizálni a felhasználók aktivitását (hány terméket néztek meg az áruház kínálatából) és a profitot (hány terméket vásároltak meg a felhasználók). Ezért egy hatékony ajánlattevő algoritmus kifejlesztése egy fontos kutatási feladat az adatbányászat terén. Természetesen az algoritmus számos egyéb paramétert figyelembe vehet ajánlattételkor, melyeket vagy mi magunk építjük be az algoritmusunkba, vagy pedig az a tanulás folyamán magától jön rá. Ilyen lehet például termékek kategorizálása [1], népszerűségük stb.

1.1. Figyelembe vehető paraméterek:

Ajánlat tevés során rengeteg adat állhat rendelkezésünkre, melyekből a jóslást meg kell tennünk. Ezeket az adatokat ajánlatos két csoportra osztani: Első csoport, amely egy adott felhasználóhoz kapcsolódik, és a felhasználó vásárlási szándékáról biztosít számunkra információkat. Második csoportba azok az információk tartoznak, melyek termékek közti kapcsolatokat határoznak meg. A két csoport közül mindenképp sokkal fontosabb a felhasználó vásárlási szándékához tartozó információcsoport.

1.1.1. Felhasználói szándékot előrejelző információk:

Ezek közül az információk közül a legrelevánsabb egy felhasználó előzőleges aktivitása. Egy felhasználót nem nehéz azonosítani, függetlenül attól, hogy regisztrált tagja-e az oldalunknak, avagy sem, így korábbi aktivitását is könnyen nyomon tudjuk követni az oldalon. Legfontosabb adatok, amelyekhez így hozzájutunk azok a felhasználó által meglátogatott termékek, a termékek melyeket kosárba tett, valamint a termékek, melyeket megvásárolt. Természetesen az oldal felépítéséből kifolyólag további adatokat is tárolhatunk úgy, mint, hogy mennyi időt szentelt egy adott termék specifikációinak olvasásával, egy adott termékhez kapcsolódó mennyi további oldalt nézett meg. Továbbá magát a felhasználót is be lehet vonni interaktívan, ha megadjuk neki a lehetőséget, hogy egy terméket kedvencként jelöljön meg, vagy épp a felhasználóknak megengedjük egy fórumon keresztül az adott termékről történő eszmecserét. Ezek az adatok is fontos információkat szolgáltathatnak egy adott felhasználó vásárlási szándékát illetően.

1.1.2. Termékek közti kapcsolatot jelző információk:

Egy termékjavaslat meghatározásakor nagyon fontos szerepet játszik az is, hogy éppen milyen terméket néz a felhasználó illetve előzőleg mely termékeket nézett meg. Mindezen információkból - ha elegendő mennyiség áll rendelkezésünkre a korábbi felhasználók aktivitásaiból - fontos termékek közti kapcsolatokat lehet megtalálni. Ha az algoritmusunkat felkészítjük, hogy ezen adatokból tanuljon, jelentős hatásfok növekedést

érhetünk el [2]. Természetesen ezeket az adatokat tetszőlegesen kibővíthetjük további szempontok figyelembevételével, ezek azonban nagyobb adatmennyiséget feltételeznek, bonyolultabb algoritmust, ami pedig nagyobb futási időt generálhat, ami nem feltétlenül előnyös. Ilyen információ lehet például termékek kategóriákba sorolása. Kategóriákat tetszés szerint alakíthatunk ki. Lehetnek árkategóriák, termékkategóriák és egyéb tetszőlegesen kitalált kategóriák, melyekbe vagy kézzel, vagy akár automatikusan is be lehet sorolni egyes termékeket [1]. Ezen kategóriák az ajánlattevő algoritmus számára fontos információkat szolgáltathatnak. Ha egy felhasználó folyamatosan egy kategórián belül aktív, akkor valószínűleg felesleges számára más kategóriából termékajánlatot tenni, valószínűleg nem fogja érdekelni. Hogy mely kategóriák mennyire relevánsak már optimalizáció kérdése. Ugyanígy az is tőlünk függ, hogy a kategóriák fontosságát az algoritmus magától próbálja meg kitalálni, vagy mi magunk építjük bele. Mindkét esetnek vannak előnyei, hátrányai.

1.1.3. Paraméterek számossága:

Az ajánlattevéshez használt paraméterek száma nagyban befolyásolhatja algoritmusunk hatásfokát, legyen szó bármilyen algoritmusról: ez általános érvényű szabály. Több adatból sokkal több információt tudunk leszűrni, azonban a nagyszámú paramétereknek is megvannak a hátrányai. Minél több adattal dolgozunk, az algoritmusunk bonyolultsága annál nagyobb lesz. Ez a futási időt nagyban növelheti, mely ilyen alkalmazás esetén szintén fontos szempont. A bonyolultságot és ezzel a futási időt az elemezni kívánt paraméterek számán kívül a rendelkezésre álló adatmennyiség is befolyásolja. Fejlesztés során tehát ezt is figyelembe kell vennünk. Míg a paraméterek száma az algoritmus bonyolultságát befolyásolja nagymértékben, ez a futási időt befolyásolja, így valahogy meg kell határoznunk egy maximális paraméterszámot, majd a rendelkezésre álló paraméterek közül kiválasztani a számunkra fontosakat, mely újabb kihívást, optimalizációs kihívást jelent. A nagy adatmennyiség nagy memóriakapacitást is igényel, tervezés során tehát érdemes felmérnünk azt is, hogy milyen erőforrásokkal gazdálkodunk és eszerint meghatározni algoritmusunk paramétereit, valamint a felhasznált adatmennyiséget.

1.2. Az algoritmus igényei:

A megalkotott algoritmusunkkal szemben tehát különféle elvárásaink és megkötéseink vannak. Megkötésként az algoritmust futtató hardversajátosságok jelenthetnek. Az igényeink pedig a következőek lehetnek:

1.2.1. Időigény:

Az algoritmus megalkotásánál nagyon fontos szempont tehát a futási ideje. Hogy ezt optimalizálni tudjuk szükséges ismernünk az algoritmust futtató számítógépünk kapacitását, illetve a webáruházunk átlagos forgalmát. Egy ajánlattevésre egy felhasználó maximum néhány (1-2) másodpercet várakozhat ellenkező esetben lehet, hogy már teljesen más oldalra navigál, még mielőtt megjelenne az ajánlat számára. Amennyiben a webáruházunk forgalma alacsony megoldható hogy a teljes szükséges időt egy felhasználóra szenteljük, azaz

pár másodpercig történjen a jóslás. Amennyiben a felhasználók száma növekszik, azok több eseményt generálnak a rendszerben és esetlegesen egyszerre több ajánlatot is kell tennünk, melyek egyike sem várakozhat tovább a megadott határnál, azaz az időt arányosan fel kell osztanunk a felhasználók közt. Ezt mind érdemes még tervezéskor figyelembe venni.

1.2.2. Tárigény:

A rendelkezésre álló adatmennyiség, valamint az algoritmusunk bonyolultságából előre megjósolható a tárigény. Amennyiben nagyon sok adattal dolgozunk, és nagyon sok mindent el szeretnénk tárolni, problémákba ütközhetünk a tárigényt illetően. Azt szeretnénk, ha minden adatunk elférne a fizika memóriába, amellyel gyorsan tudunk dolgozni. Amennyiben ez a nagy adatmennyiség és az algoritmus komplexitása miatt nem lehetséges, három lehetőségünk akad optimalizálásra:

- Első lehetőségünk hogy azokat az adatokat, amelyeket nem használunk adott pillanatban, kiírjuk fizikai háttértárra. Ez természetesen rengeteg lassú fájlműveletet eredményez, és nagyságrendekkel nagyobb futási időt eredményez. Ez esetben jól meg kell fontolnunk rendelkezünk-e elég idővel ilyen műveletekre.
- További lehetőség, hogy korlátozzuk az adatok mennyiségét. Egy bizonyos határon túl nem törődünk további információkkal. Ez esetben az algoritmusunk pontosságát áldozzuk fel és lehetetlen megmondani mennyi adatot és mennyire fontos adatokat hagytunk figyelmen kívül. Így azt is lehetetlen megmondani, hogy ez milyen mértékben befolyásolja adott esetben az algoritmusunk hatékonyságát.
- Harmadik megoldásként pedig az jöhet szóba, ha megpróbáljuk meghatározni csupán azokat az információkat melyek számunkra fontosak és csak azokat eltárolni. Így nagyjából tudhatjuk, hogy az algoritmusunk nagy valószínűséggel, jó hatásfokkal dolgozik. Természetesen ebben az esetben sem tudhatjuk biztosra, hogy milyen mértékben térünk el adott esetben minden lehetséges információt figyelembe vevő ideális algoritmustól.

1.3. Algoritmusok:

Konkrét megvalósítás esetén, amennyiben csak valamilyen speciális körülmények másként meg nem követelik a leghatékonyabb módszer, ha az ajánlattételt két részre szedjük szét. A két rész bár teljesen különállóan működik, mégis szorosan összekapcsolódnak, hiszen ugyanazon adathalmazokon dolgoznak, kompatibiliseknek kell lenniük egymással. A két részre bontás optimalizációs célokat szolgál. Mind időbeli, mind pedig tárhelybeli optimalizációt biztosít számunkra bármiféle veszteség nélkül. Természetesen nem követelmény, hogy pontosan két részre bontsuk a problémát, lehet többre is, amennyiben a technikai megvalósításból nem származik hátrányunk, ezzel további optimalizációkat érhetünk el. A két legfontosabb rész a modellépítés, valamint az konkrét ajánlat meghatározása a modellből.

1.3.1. Modellépítés:

Az első rész a modellépítés, amikor is a rendelkezésünkre álló összes adatból megpróbálunk egy sémát, modellt készíteni egyes termékek közti kapcsolatokról [3]. Optimalizációt a teljes algoritmus terén ez ott hoz számunkra, hogy a modellépítésnek nem kell valós időben történnie. Azaz modellépítésre akár több napot is szánhatunk, hogy minél pontosabb képet kapjunk, minél jobb modell alapján tudjunk keresni ajánlattétel során. Ennél a résznél merül fel azonban a tárigény. Ha a modellünk meghaladja tárolókapacitásunk, vagy annak nagy közelségében van a predikciós algoritmusnak már nehezebb jó hatékonyságot elérni. Erre jelenthet megoldást a modell háttértárra történő kiírása. Ezzel azonban jelentős mértékben lelassíthatjuk az ajánlattételt. Optimalizációt jelenthet az is, ha egy bizonyos modellméret elérésekor nem építjük azt tovább. Ekkor azonban információvesztés léphet fel. Akár az is, hogy adott termékek kapcsolatát egyáltalán nem határozzuk meg másokkal és így azon termékekhez ellehetetlenítjük az ajánlattételt. Erre a problémára megoldást jelenthet, ha a rendelkezésre álló adatokat nem sorrendben, hanem látszólag véletlenszerűen kezdjük el elemezni. Modellépítés esetén fontos hogy nagyszámú adat álljon rendelkezésünkre. Az egész modellépítés alapja a statisztikai elemzés. A modellépítés folyamatát is több részre lehet természetesen bontani. Ezek a részek lehetnek vagy párhuzamosan, vagy pedig egy megadott sorrendben futtathatóak [8]. Párhuzamos futtatás lehet végrehajtani például két különböző adatállományból történő különböző adatok nyers kinyerésére. A nyers adatok kinyerése után lehet végrehajtani a nyers adatok feldolgozását, adatok fontosságának meghatározását, termékek közti kapcsolatok skálázását, mindezt már nem párhuzamos módon tehetjük meg.

1.3.2. Ajánlattétel:

A második fő fázis: a predikció konkrét ajánlattétellel. Ennél a résznél a legfontosabb szempont, az az ajánlattevő algoritmus futási ideje. Real-time futásúnak kell lennie, hiszen egy felhasználónak szinte azonnal ajánlatot kell tennünk, az nem fog ránk várakozni. Ami előny, hogy a két részre bontás miatt itt a tárigénnyel csak kis mértékben kell foglalkoznunk. Ha megvan az előzőleg felépített modellünk a termékek közti kapcsolatok jellemzéséről, az algoritmusunk nem fog jelentős mennyiségű további adatot generálni. Amire szüksége van, az csupán az aktuális felhasználó, akinek épp ajánlatot teszünk, korábbi aktivitásai, ami elhanyagolható mennyiség. Az algoritmusnak a modellből és a korábbi aktivitásból kell megjósolnia mely az a termék, vagy melyek azok a termékek melyek a leginkább kapcsolódnak az adott felhasználóhoz, amelyeket leginkább hajlamos megvenni, vagy legalább megnézni. Ez tehát csupán egy komplex keresési folyamat a modelltérben az előző aktivitással felparaméterezve. Ennek az optimalizációja természetesen fontos kérdés, és érdemes már tervezéskor elkezdeni, a modell megtervezésénél. Legfontosabb, hogy a predikció gyors legyen, még akkor is, ha ez extra tárigénnyel jár, vagy az építése lesz időigényesebb. Ha a predikciós szakasz adott időn belül nem tudná felhasználni a teljes felépített modellt, akkor érdemes bevezetnünk egy időlimitet a futásra. Az időlimit lejárta

után az aktuálisan legmegfelelőbb terméket ajánljuk a felhasználónak. Ez természetesen lehet, hogy eltér attól, amit a teljes futás végén kapnánk, ellenben így is elég jó eredményt érhetünk el, valamint nem nullázzuk ezzel az esélyeinket ellentétben azzal, amikor ajánlatot sem teszünk. Az, hogy ezt a részt tudjuk-e több különálló részre szétszedni, csupán attól függ, hogy az eljárásunk párhuzamosítható, avagy sem. Ha minden egyes a felhasználó által korábban megnézett termékhez a többitől függetlenül végzünk keresést, és a keresés a modellben nem okoz változást, úgy nagymértékben párhuzamosítható az algoritmusunk, kihasználhatja a több proceszoros rendszerek előnyeit. Amennyiben az algoritmusunk megvalósítása megköveteli, hogy az egyes termékekhez tartozó kereséseket szekvenciálisan végezzük, vagy épp minden egyes keresés módosíthat a modellben úgy a párhuzamosítás lehetetlenné válik.

1.4. Versenyfeladat:

A Data Mining Cup (DMC) verseny keretében egy ilyen hatékony ajánlattevő algoritmus kifejlesztése a cél, amely megpróbálja maximalizálni mind az aktivitást, mind a profitot, de a profitra nagyobb hangsúlyt fektet. Egy adott felhasználóhoz tehát megpróbálja meghatározni azt a terméket, melyet valószínűleg megvásárol. Mindezt teszi az adott felhasználó, valamint más felhasználók korábbi aktivitását figyelembe véve. Ehhez két alfeladat tartozik: először a tanulás és kiértékelés külön van választva (statikus tanulás), másodszor pedig folyamatosan tanulunk és egyidejűleg javaslatokat is kell tennünk (dinamikus tanulás). Minden egyes munkaszakaszban három különféle tranzakciótípus szerepelhet. Ezek a következők: Egy termék leírásának megnyitása, egy termék kosárba tétele, valamint egy termék megvásárlása. Egy munkamenet általában a következőképpen épül fel: Egy vásárló böngészi a webshopot megnézve különböző termékek leírását. Ahogy halad azokat a termékeket, melyek tetszettek neki beteheti a kosárba. A munkamenet végén a kosárra kattintva véglegesítheti a vásárlást, illetve előtte még módosíthatja annak tartalmát.

1.4.1. A DMC 2011 verseny feladatleírása:

A feladat egy statisztikai elemző algoritmus megalkotása. Az algoritmus előre megadott adathalmazból tanulhat, statisztikát készíthet már lejátszódott munkamenetekről, azaz a tanulóállományból. Hogy ki lehessen értékelni az algoritmust ezután egy előre elkészített adathalmazon jóslást kell adnia a tanultak alapján a következő termékekre. Ez úgy néz ki, hogy szintén meg vannak adva már lejátszódott munkamenetek, melyből az utolsó három termék törölve lett. Ez a tesztállomány. Az algoritmus kielemezheti a munkameneten belül eddigi aktivitásokat, majd ezek alapján, valamint a tanulóállományból tanultak alapján jóslást kell adnia az utolsó három termékre. A cél, hogy minél nagyobb találati aránya legyen az algoritmusunknak.

1.4.2. A statikus tanulás:

A feladat egyidejűleg két részre osztották. Az első részben megkapunk minden

adatot, amiből mi tetszőlegesen tanulhatunk, majd miután megalkottuk a predikciós algoritmusunk egy kiértékelő adaton jóslatokat kell tennünk [6][7]. Ebben a szakaszban tehát a tanulás és a felhalmozott tudás felhasználása elkülönül egymástól, első részben a statikus tanulást kell megvalósítanunk, azaz olyan modellt építeni a rendelkezésre álló adatokból, mely gyorsan kereshető és reprezentatív is egyben.

1.4.3. A dinamikus tanulás:

Második részben szimulálva egy valamelyest valós webshop működését a tanulás és a felhalmozott tudásból történő jóslás egyidejűleg történik. Folyamatosan kapjuk a felhasználók aktivitását, valamint bizonyos időközönként jóslatokat kell tennünk számukra. Ebben az esetben is különböző problémákkal szembesülhetünk. Amikor még nincs elég tudás a birtokunkban, nem derítettünk fényt a termékek közti kapcsolatokra az ajánlattevés nehéz, szinte véletlenszerűnek mondható. Miután egyre több adattal rendelkezünk, úgy annál inkább kialakulnak a termékek közti kapcsolatok, ugyanígy viszont egy új aktivitás hatása a már kialakult kapcsolatokat befolyásolhatja esetlegesen nagy számítási kapacitást igényelhet, amely valós időben szintén megengedhetetlen. Nagy adatmennyiség esetén is gyors algoritmusunknak kell lennie, tehát nem feltétlenül vehetjük figyelembe az összes termék közti kapcsolatokat. Ezzel egy időben a jóslattevő algoritmusunknak is szinte azonnal rendelkezésre kell, hogy álljon az ideális termékajánlat, hiszen egy idő múltán már csak az adatok közti keresés is rengeteg időt elvehet, nem lenne időnk további számításokat végezni. Ez gondos tervezést igényel az adatok bevitele, tárolása terén. Könnyen karbantarthatónak kell lennie az adatoknak. Gyorsan kereshetőnek kell lennie, valamint új adat felvétele is gyorsan kell, hogy történjen. Ami ebben az esetben nem fontos, az nem más, mint az adatok törlése, hiszen ilyen műveletet nem fogunk végezni soha.

2. Általános algoritmusok:

A következő általános algoritmusok jó alapot nyújthatnak ezen feladatok megoldására [4][11]. Természetesen specializációjuk ajánlott, ha nem kötelező konkrét feladatok megoldása során. Általános kiindulásként az előző fejezetben említett két részre bontás ajánlott. Első lépésként a modellépítés, majd pedig az ajánlattevő algoritmus megtervezése ajánlott. Mindezek előtt természetesen, vagy ezekkel párhuzamosan érdemes a modellt is alaposan megtervezni. Milyen igényeink lesznek vele szemben, azok milyen adatszerkezetekben valósíthatóak meg, és mik az egyes adatszerkezetek előnyei, vagy hátrányai.

2.1. Modell megtervezése:

A szoros tár és időigények miatt tudnunk kell, hogy ilyen feladatok megoldása esetén nem használhatunk olyan programozási nyelveket, melyek felügyelt környezetben futnak. Továbbá fontos az is, hogy objektumorientált nyelven programozzunk hisz a modell elemeit így sokkal könnyebb kezelni. Ajánlott tehát a C++ nyelv használata, de természetesen nem kötelező [5]. A modellel szemben támasztott legfőbb elvárásunk hogy gyorsan lehessen benne keresni [9]. Felvetődhet tehát az igény, hogy az egyes elemeket sorrendezett tömbökben tároljuk, hiszen így a keresés vagy logaritmikus időben történik, vagy akár konstans időben, ha ismerjük az értékkészletet amiben keresünk. Ennek az elrendezésnek viszont hatalmas hátránya, hogy az építése exponenciális idejű, azaz egy ilyen modell felépítése emberi időn belül szinte lehetetlen. Hatalmas adatmennyiségek esetén minden egyes új adat felvételekor keresést kéne végeznünk a tömbön, beszúrást, valamint ezeket a lépéseket akár több szinten is végigiterálnunk, amennyiben többszintes modellt használunk. Ehelyett a szerkezet helyett sokkal inkább ajánlott használni a bináris fákat, vagy annak változatát a piros-fekete fákat [4]. Ezek előnye, hogy építésük O(n*logn) lépésből történhet, valamint a keresés bennük logaritmikus időben történik.

2.1.1. Piros-fekete fák tulajdonságai:

A piros-fekete fa egy bináris fa, amiben:

- minden belső csúcsnak pontosan két fia van
- elemeket csak belső csúcsokban tárolunk
- teljesül a keresőfa tulajdonság
- a fa minden csúcsa piros, vagy fekete
- a gyökér fekete
- a levelek feketék
- piros csúcsnak csak fekete fia lehet
- minden V csúcsra igaz, hogy minden a V csúcsból egy levélbe vezető úton, ugyanannyi fekete csúcs van.

Egy piros-fekete fa elemenként építhető. Minden új elem beszúrása O(logn) lépésben történik. Így egy fa építés n elem esetén O(n*logn) lépés. Ami számunkra fontos eltekintve a matematikai háttérismeretektől, az az, hogy elemeket csak belső csúcsokban tárolunk, és

egy elem megtalálása O(logn) lépésben történik. Természetesen nem elég egy egyszerű piros-fekete fát alkalmaznunk még a legegyszerűbb esetekben sem. Hiszen amit a modellünknek tartalmaznia kell, az valamilyen módon két termék kapcsolatát kifejező érték. Ebből következően, egy érték meghatározásához két paraméterre van szükségünk, mégpedig a két termékre, amelyek kapcsolatát vizsgálni szeretnénk. Tehát ilyen esetekben a piros-fekete fa, minden egyes értéket tároló eleme, azaz nem levélelemek egy újabb piros-fekete fa gyökerét tartalmazzák. A külső fában megkeressük a vizsgálni kívánt két termék közül az egyiket, melyhez szintén egy piros-fekete fa tartozik melyben a kulcsok szintén termékazonosítók, azonban a tárolt érték már a két termék kapcsolatát jellemző szám-, vagy egyéb érték. Ebben a belső fában pedig rákeresve a második termékazonosítóra megkapjuk a számunkra fontos értéket. Ez a bonyolultság további paraméterek figyelembe vétele esetén csak tovább nőhet. Azonban ezekkel vigyáznunk kell. Már két paraméter esetén is jelentkezik a redundancia, azaz két termék kapcsolatát két helyen is tároljuk. Egyik esetben, amikor a külső fában az A termékre keresünk, majd a belső fában B termékre keresünk, megkapjuk A és B termékek kapcsolatát. Ugyanez a helyzet, ha a külső fában keresünk először B termékre, majd a belső fában az A termékre. A kapott érték ebben az esetben is A és B termékek kapcsolatát jellemzi, azaz két helyen is tároljuk az értékeket. Ez néhány esetben akár különböző is lehet, néhány esetben viszont megegyező értékeket is kaphatunk. Mindez attól függ milyen statisztikai elemzés alapján építjük a modellünk. Ez gondos megfontolásokkal és a modell építésénél, valamint a keresés esetén ennek figyelembevételével eltüntethető két paraméter esetén, ellenben több paraméter esetén a redundancia eltávolítása már nagy kihívás. Ugyanígy, ha a redundanciát megengedjük mind az építés, mind pedig a keresés gyorsabb lesz, de a tárigény megsokszorozódhat a sok feleslegesen tárolt adat miatt. Gondosan kell tehát eljárnunk, amikor megválasztjuk a tárolni kívánt értékek mennyiségét, valamint hogy azok hány paramétertől függnek.

2.1.2. Konkrét Modellmegvalósítás:

C++ programnyelv használata esetén a nyelv alapstruktúrai közt megtaláljuk a piros-fekete fákat. A nyelv ezeket Mapként azonosítja és használható bármilyen adattípussal [9][10]. A map a következő számunkra fontos szolgáltatásokat nyújtja:

- Új elem beszúrása
- Elem törlése
- Kulcs szerinti keresés
- Iterátorral történő bejárás

Két paramétert kell megadni:

- A kulcs típusát
- Az érték típusát

Természetesen mindkét típus lehet további összetett típus is, így könnyen tudunk egymásba ágyazást létrehozni, vagy több paraméteres kulcsokkal dolgozni. Esetünkben két kulcshoz tartozik egy érték, vagyis a második típus, az érték szintén egy piros-fekete fa lesz, vagyis egy map, egy összetett típus, melyben a második kulcsunk felhasználásával megkapjuk az

értéket. Ebben az esetben a két kulcs a két termék, melyek közti kapcsolatra kíváncsiak vagyunk, az érték pedig azok kapcsolatát fejezi ki valamilyen előre jól definiált módon.
Konkrét megvalósításban egy tárolót így definiálhatunk:
map<int,map<int,float>> valtozonev;
Az első paraméter, az int, a külső fa kulcsának típusát adja, mely ebben az esetben egy egész érték, majd a vessző utáni map<int,float> a kulcshoz tartozó érték típusát definiálja. Itt látható az egymásba ágyazás, vagyis az érték egy újabb piros-fekete fa, melynek kulcsa egy egész érték, értékének típusa pedig egy lebegőpontos szám, amely kifejezi két termék kapcsolatát.

2.1.3. Adatok és a modell kapcsolata:

A rendelkezésre álló adatok nagyban befolyásolják a modellünket, minél több és sokrétűbb adat áll rendelkezésünkre, annál sokrétűbb, többkulcsos lekérdezéseket hajthatunk végre, melyek mindegyike külön adatot, információt szolgáltathat számunkra. Alapesetben a legfontosabb adatok, amik mindenképp a rendelkezésünkre állnak, azok a felhasználók előzőleges tevékenységei. Ezek az adatok szolgáltatják számunkra talán a legtöbb információt és ezekkel az adatokkal mindenképp rendelkezünk is. Ekkor természetesen csak két termék közti összefüggést tudjuk megállapítani a felhasználók által, így a modell is egyszerű, kétkulcsos-érték párokból épül fel. Amennyiben további információkkal is rendelkezünk, például termékek kategóriákba tartozásáról is vannak információink, úgy azokat a modellünkbe is beépíthetjük. Nagy segítség lehet adott esetben, ha tudjuk, hogy egy termék egy adott kategóriába tartozik, akkor megpróbálhatunk csak abból a kategóriából terméket felajánlani, ami így jelentős mértékben lecsökkentheti a keresési teret. Ellenben ha nem vigyázunk, a modellünk roppant mértékben elbonyolódhat. A keresést a több kulcs nagyban meghosszabbítja, a modell méretét növeli, míg a szolgáltatott információ esetlegesen nagyon kevés lehet, így ezt nem éri meg a modellünkbe beépíteni. Ilyen döntéseket érdemes még a tervezés során meghatározni. Néhány esetben az adatok esetlegesen olyan formában állhatnak rendelkezésünkre, melyből vagy nem lehet, vagy csak nagyon nehezen lehet kinyerni a számunkra fontos információkat, amelyek viszont tudjuk, hogy tartalmaznak. Ekkor jelenthet megoldást, ha elő feldolgozó algoritmusokkal az adatokat olyan alakra hozzuk, hogy információvesztés ne történjen, ám a tanulás sokkal könnyebb legyen.

2.2. Esettanulmány:

A korábban említett verseny keretében a rendelkezésre álló adathalmaz csupán a webshop látogatói által generált forgalom. Ehhez kell egy modellt megterveznünk. Milyen szempontok alapján csoportosítunk, valamint, hogy milyen értékeket tárolunk.

2.2.1. A rendelkezésre álló adatállományok:

A verseny során a statikus tanulás feladatrészhez két fájlt biztosítanak számunkra, a training data állományt, valamint a test data állományt. Mindkét esetben egy strukturált

szövegfájlról van szó. Egy sor egy tranzakciót reprezentál és a következőképpen épül fel: MUNKAMENETSZÁM|TERMÉKAZONOSÍTÓ|TRANZAKCIÓ
Ahol a munkamenetszám a munkamenetet azonosítja, monoton növekvő. A termékazonosító szintén egy szám, és egy terméket azonosít. További információnk nincs a termékről. A tranzakció típusa lehet 0, 1, vagy 2 attól függően, hogy a terméket megnézték, kosárba tették, vagy megvásárolták. Mind a két fájl fejlécében megtalálhatóak ezek az információk. Minden sor le van zárva újsor karakterrel. A TRAINING DATA esetén az egy munkamenethez tartozó tranzakciók elő feldolgozás nélkül szerepelnek, tehát egy terméket akár többször is megnézhettek, többször kosárba tehettek stb. TEST DATA esetén az adatokat elő feldolgozták. Egy termék egy munkameneten belül egyszer szerepelhet. Szerepelni az első előfordulási helyén szerepel. A hozzátartozó tranzakciótípus a munkameneten belül összes előfordulása közül kerül kiválasztásra úgy, hogy ha megvették, akkor 2 kerül a tranzakció típusához, ha nem vették meg de kosárba tették, akkor 1, egyébként 0. Mindez független attól, hogy mit végeztek el vele először, vagy épp utoljára. Ezen felül minden egyes munkamenet utolsó három termékét törölték a munkamenetből. Ezeket egy külön fájlban tárolják, melyet a versenyzőknek nem tettek közzé. Ez szolgál a kiértékelésre. Amennyiben egy munkameneten belül nem szerepel legalább 4 különböző termék, úgy az egész munkamenet törlésre kerül a TEST DATA állományból és nem számít bele a végső kiértékelésbe.
A tranzakciókra továbbá a következő szabályozás áll fenn: Ha egy terméket megvásároltak, biztos, hogy előtte kosárba tették, ugyanígy ha egy terméket kosárba tettek, akkor biztos, hogy előtte megnézték a leírását.
Így tehát a feladat egy algoritmus megalkotása, mely a TRAINING DATA állományból tanultak alapján a TEST DATA állományban szereplő munkamenetekre megjósolja mik lehettek a további termékek az előzőek alapján.

2.2.2. A training data állomány statisztikája:

Munkamenetek:	1196485
Termékek:	17349
Megnézett:	8052961(84.49%)
Kosárba tett:	1012895(10.62%)
Megvett:	465268 (4.88%)
Összes tranzakció:	9531124

A statisztikából látni, hogy egy felhasználó átlagosan 7 terméket látogat meg. Természetesen a szórás ebben az esetben elég nagy. Ettől lényegesen kevesebb termék került kosárba. Felhasználónként mindössze 0.85 termék került kosárba, azaz nem mindenki tett kosárba terméket, valamint aki tett, az se sokat. Körülbelül minden 8-adik megnézett termék után került 1 termék a kosárba. Vásárlásokat tekintve egy felhasználó csupán 0.38 terméket vett átlagosan. Kevésnek tűnik, ellenben figyelembe véve a kosárba tett termékek számát megfigyelhető egy viszonylag szoros kapcsolat a két aktivitás között, hiszen a kosárba tett termékek felét meg is vásárolták.

2.2.3. A teszt állomány statisztikája:

Munkamenetek: 61809
Termékek: 17349
Megnézett: 473084(87.75%)
Kosárba tett: 35047(6.5%)
Megvett: 30960(5.74%)
Összes tranzakció: 539091

A teszt állományban már jóval kevesebb felhasználó aktivitását kapjuk meg. Már csak a kiértékelésre, algoritmusunk tesztelésére szolgál az állomány. Termékek tekintetében természetesen szintén tartalmazza az összes terméket, hogy hitelesen tudjunk tesztelni, ne lehessen figyelmen kívül hagyni egyetlen terméket sem. Ezen állomány esetén is, az elő feldolgozás ellenére is egy felhasználó átlagosan 7 terméket látogatott meg. Nem számítva ide természetesen azon három terméket, melyet a mi algoritmusunknak kell megjósolnia. Ebbe továbbá az is figyelembe van véve, hogy egy termék egy munkamenetben mindössze egyszer szerepel, a rajta végzett legmagasabb aktivitással. Ezek mind arra utalnak, hogy eredetileg ezek nagy aktivitást mutató munkamenetek voltak, ellenben nem vonhatjuk le a következtetést, hogy a Training állományban rengeteg a kis aktivitású munkamenet, hiszen lehet, hogy a Test állományba szándékosan lettek ezek beválogatva. Ezen állomány esetén már szinte elhanyagolható a különbség a megvett, valamint a kosárba tett termékek között. Ez valószínűleg annak tudható be, hogy eredetileg is aktív felhasználókkal dolgozunk, ami arra utal, hogy vásárlási szándékkal keresték fel a boltot. Az összaktivitást tekintve egy felhasználó átlagosan majdnem 9 aktivitást végzett, nem számítva a törölt hármat, melyet nekünk kell megjósolnunk, valamint a törléseket.

2.2.4. Az állományok előfeldolgozása:

Az adatokon nem tiltott elő feldolgozást végezni. Ez azt jelenti, hogy a számunkra szolgáltatott Training_Data állományt saját elképzeléseink szerint alakíthatjuk, törölhetünk belőle adatokat, módosíthatjuk azokat, esetlegesen akár ki is egészíthetjük azokat. Ezekkel természetesen óvatosan kell bánni, hiszen értékes információkat veszthetünk gondatlan törlésekkel.

Az előfeldolgozás elsődleges célja, hogy a tanulóalgoritmusunk segítsük, gyorsítsuk. Ez természetesen a tanulóalgoritmus minőségétől és precedenciáitól függ, éppen milyen elő feldolgozást végezhetünk az állományon. Futási időileg nagyságrendekkel gyorsabbnak kell, hogy legyen, mint amennyit a tanulóalgoritmus idején javítani tud. Konkrét algoritmusokról később lesz szó.

Mindez természetesen csak a statikus tanulás feladatban működik, amikor először tanulunk egy teljes adatállományból. A dinamikus tanulási feladat esetén ilyen elő feldolgozó algoritmus használata értelmetlen és egyben lehetetlen is.

2.3. Modellépítés, tanulás:

Miután elkészítettük a modellünk vázlatát, valamint az adatainkat is megfelelő formára hoztuk elkezdhetjük a tanulás, a modellépítés folyamatát. Mivel különböző tanulóalgoritmusok különböző adatrepzentációkból tudnak hatékonyan és gyorsan tanulni, ezért az adatok elő feldolgozása mindenképpen úgy kell, hogy történjen, hogy az a saját algoritmusunk számára megfelelő legyen. Hogy a legáltalánosabb példánál maradjunk ezeket az algoritmusokat is vizsgáljuk meg akkor, ha csupán a webshopot meglátogató vendégek korábbi aktivitásai állnak rendelkezésünkre, valamint a termékek kategorizáltságát használhatjuk extra információként.

2.3.1. Adatok információtartalmának vizsgálata:

Mielőtt bármilyen algoritmust is elkezdenénk megtervezni, érdemes elgondolkodni azon, hogy a rendelkezésünkre álló adatokból vajon milyen információkat is tudunk leszűrni. Melyek azok az információk, amelyek ajánlat tételkor számunkra hasznosak és ezekből az adatokból jól kiszűrhetőek. Alapvetően statisztikai elemző algoritmusokról beszélünk [7]. Ezek tehát nagymértékű ám jól definiálható számításokat végeznek, majd az eredményeket eltárolják a megalkotott modellben. Egyes információk, mint például egy termékre vett statisztika, azaz hogy hányszor vették meg, hányszor tették kosárba, valamint hányszor kérték le az adatait könnyen és gyorsan számítható. Ezzel ellentétben vannak olyan információk, melyek szinte elengedhetetlenek egy jó ajánlat megtalálásához, ám számításuk roppant időigényes. Ilyen például két termék kapcsolatának jellemzése összehasonlítva más termékek kapcsolatával. Ilyen jellemzőt rengeteg módon számíthatunk, melyekre később példákat is látunk. Ezek közül megtalálni a legideálisabbat egy nehéz feladat. Egyik ilyen algoritmus lehet, ha meghatározzuk, hogy egy adott termék után mely másik termékek és hányszor szerepeltek. Ez természetesen egy nagyon pongyola algoritmus melytől sokkal jobb hatásfokúak is vannak, de érezhető, hogy már ennek a futási ideje is nagyságrendekkel nagyobb. Amit ezekből az adatokból nem tudunk leszűrni, az a termékek kategóriákba soroltsága. Ezeket vagy kézzel kell megadnunk minden egyes termékre, hogy rendelkezésünkre álljon, vagy klaszterező algoritmusok felhasználásával megpróbálhatjuk őket csoportosítani. Amennyiben a termékek kategorizáltsága már el van készítve számunkra, lényegtelen milyen módon, úgy ezt nagyon jól fel lehet használni modellépítés esetén. Két termék közti kapcsolat ebben az esetben sokkal erősebbnek feltételezhető, ha egy kategóriába vannak besorolva. Ennek a módszernek is rengeteg megvalósítása létezhet, attól függően milyen szabályokat vezetünk be a kategorizálásra. Fontos szempontok, hogy megengedjük-e egy termék több kategóriába tartozását, tehát a kategóriáknak vannak-e metszetei, esetleg hierarchikus kategorizálást alkalmazunk, ezzel többféle kapcsolódást engedélyezve melyek mindegyikét külön kell kezelnünk. Kategorizálás nagy előnyünkre válhat belegondolva egy felhasználó alapvetően vagy vásárlási szándékkal látogatja meg oldalunkat, vagy pedig adott termékről próbál információt szerezni. Esetlegesen költségcsökkentést is elérhetünk, azaz gyorsabban tudunk jobb reprezentációs modellt építeni, ha külön kategóriákba tartozó termékek kapcsolatát teljes mértékben figyelmen

kívül hagyjuk. Ezzel több részre bontva a teljes keresési teret, bár több, de összességében sokkal kisebb keresési teret kapunk. Figyelembe véve pedig azt is, hogy egy adott keresés esetén nem is kell az összes részben keresnünk ez időben nagyságrendekbeli javulást jelent.

2.3.2. Adatok felvétele a modellbe:

Miután megvizsgáltuk a rendelkezésre álló adatainkat, erőforrásainkat, valamint ezekhez az adatreprezentációs modellünket is megterveztük a tényleges tanulóalgoritmusunk futtathatóvá válik. Fontos feladata, hogy megkeresse, és adott módon eltárolja a modellben a számunkra fontos adatokat, jellemzőket. Legfontosabb és egyben legnehezebb feladata természetesen a kapcsolatok milyenségét feltárni, és reprezentatívan jellemezni azt. Amennyiben rendelkezésünkre állnak a termékek kategorizáltsága, úgy az algoritmusunkat felkészítve arra, hogy egyszerre csak egy adott kategóriába tartozó termékeket elemezzen jelentős javulást érhetünk el mind időben, mind pedig a modell hatékonyságában. Amennyiben ez nem áll rendelkezésünkre, úgy az algoritmusnak minden termék minden más termékhez való kapcsolatát meg kell határoznia. Ez időben jóval nagyobb kihívást jelent. Amit a korábbi aktivitásokból figyelembe tud venni, figyelembe kell vennie az algoritmusnak az, az hogy két termék egy adott felhasználó aktivitása esetén milyen sorrendben, milyen módokon szerepeltek. Itt az algoritmus megvalósításától függ csupán, hogy milyen konkrét szempontokat analizálunk. Legfontosabb szempontok, hogy egy adott termék hányszor szerepelt együtt egy másik termékkel egy felhasználó aktivitása során, hogy egy adott termék után közvetlenül hányszor néztek meg egy adott másik terméket. Egy adott termék után szerepelte egy másik termék egy adott felhasználó aktivitása esetén. Ezeken kívül persze rengeteg szempont szerepelhet. Mindezekhez ismernünk kell alap statisztikai elemzéseket, rendelkeznünk kell matematikai alapokkal, hogy megvalósítsuk algoritmusaink. Mind statisztikát kell ismernünk egy alapszinten, mind pedig a valószínűség számítás néhány fogalmával nem árt tisztában lennünk [6].

2.3.3. Ajánlattétel felépített modell alapján:

Miután a megszerzett tudást elmentettük az előzőleg megtervezett modellünkbe, következhet annak felhasználása. Ajánlattételkor mindössze néhány információ áll rendelkezésünkre, ez biztosítja számunkra ez esetben a gyors működést. Ezek az információk a modellben tárolt termékek közti kapcsolatok jellemzői, a felhasználó által aktuálisan megfigyelt termék, és esetlegesen időnk engedi a felhasználó előzőleges aktivitását is figyelembe vehetjük. Ezekből az adatokból kell meghatároznunk azt a terméket, amelyikre a felhasználónak szüksége lehet. Amennyiben lehetőségeink másként nem engedik a leggyorsabb megoldás az, ha csupán az aktuális termékhez a modellből kiválasztjuk a vele leginkább összefüggő terméket. Ez a megoldás egy jó kiindulópontot biztosít további fejlesztésekre, melyek során jól nyomon követhetjük a fejlesztések által okozott időtöbblet növekedését, ugyanígy a modell növekedésével járó tárigény többletről is jó képet kaphatunk a tesztelések során. Fejlesztések ként első lépésben érdemes figyelembe vennünk adott termékek statisztikáját, ezzel súlyozni a termékek közti kapcsolatot, majd ezután

választani egy maximumot [4]. Ezzel azt érjük el, hogy a súlyozás finomra hangolása esetén sokkal több terméket tudunk eladni, azaz profitot maximalizálunk, hiszen olyan terméket fogunk inkább felajánlani, amely lehet, hogy kevésbé függ össze az adott termékkel, de nagyobb valószínűséggel veszik meg, ha már egyszer ráakadtak a vásárlók. További lépésként fokozatosan elkezdhetjük vizsgálni felhasználónk előzőleges aktivitását is, ha az rendelkezésünkre áll és nem először látogat oldalunkra. Ez esetben minden korábbi termékhez is meghatározhatunk egy másik termékvonzatot. Így egy listát kapunk különböző termékekről, melyeket érdemes lehet felajánlani. Ebből valamilyen szempont szerint ismét választanunk kell, mely szintén rengeteg módon történhet. Kiválaszthatjuk azt, amelyikhez a legtöbb vonzat volt. Időarányosan súlyozhatjuk a termékeket, azaz azt a terméket, melyet egy jóval korábbi termék vonzott nem ajánlatos reklámként megjeleníteni. Összevethetjük továbbá az összes vonzott terméket az összes megnézett termékkel és ezek közül egy maximum pontszámot elért terméket kiválaszthatunk, amely úgymond összességében a legjobban összefügg a meglátogatott termékekkel. Azt is el kell döntenünk, hogy a felhasználónak mennyi terméket fogunk reklámozni. Túl sok termék esetén a lényegét elveszthetti az egész reklám és a felhasználó meg se nézi őket. Viszont ha több mint egy terméket határozunk meg és reklámozzuk a felhasználónak, úgy ezzel az esélyeinket is megnöveljük arra, hogy sikerült olyan terméket találnunk, mely esetlegesen érdekli a felhasználót. Amennyiben úgy döntünk, hogy több mint egy terméket határozunk meg egy adott felhasználó számára, az bonyolíthatja számításainkat, hiszen nem csupán maximumkereső algoritmusról van szó egy adott állapottérben, hanem a termékeket a szempontok szerint sorrendezni kell, majd a legjobbnak bizonyuló termékeket kell kiválasztanunk.

3. Megvalósítások összehasonlítása:

A munka során a különböző algoritmusok és megvalósításainak elemzésére a Data Mining Cup verseny egy jó lehetőséget biztosított. A verseny keretében így 2 darab algoritmust fejlesztettem ki és implementáltam c++ nyelven a hozzá tartozó kiegészítő algoritmusokkal együtt. A verseny során csupán korábbi felhasználók aktivitásaival dolgozhatunk és semmilyen egyéb információ nem áll rendelkezésünkre a termékünkről. Az alapvető információ, amit mi meg tudunk szerezni az a termékek kapcsolata számszerűsítve. Ez azt jelenti, hogy a megadott adatokon azt próbáljuk megkeresni statisztikai és valószínűség számítási elemzésekkel, hogy két adott termék közt milyen szoros kapcsolat van, mennyire függnek össze. Ebből következtethetünk arra, hogy ha egy felhasználó az egyik terméket nézegeti, lehet, hogy érdekelni fogja a másik termék is, és mivel annak jobb az eladási statisztikája lehet, hogy azt nagyobb valószínűséggel veszi meg.

3.1. Egytermékes sorrendezett becslés:

Az algoritmus leírásához vezessük be a következő fogalmakat:

V_i – Az i terméket a munkamenetben megvásárolták

K_i – Az i terméket a munkamenetben kosárba tették

M_i – Az i terméket a munkamenetben megnézték

Ahogy a név is sugallja, amit ezen algoritmus figyelembe vesz, az az, hogy egy adott *i* termék után egy munkameneten belül mely más *j*, *k*, *l* stb. termékek szerepelnek. Ez azért fontos, mert direkt módon, vagy akár indirekten a felhasználó a böngészés során egy adott termékből egy másik termékhez jutott, azaz a két termék kapcsolatba hozható egymással. Fontos a sorrend, hiszen nem *j* terméket követte *i*, hanem az *i* terméket követte *j*, vagyis *i* termék vonja maga után *j* terméket.

Statisztika készítéshez becsüljük meg a $P(M_i|M_j)$ feltételes valószínűséget a következőképpen [6]:

Az A halmaz legyen egyenlő azokkal a munkamenetekkel, amelyekben j terméket legalább egyszer megnézték. B halmaz legyen az A halmaz azon részhalmaza, amelyekben i terméket legalább egyszer megnézték j termék után, tehát vegyük figyelembe a sorrendet.

$$P(M_i|M_j) = \frac{|B|}{|A|}$$

Hasonlóképpen számoljuk ki a következő értékeket is:

$P(M_i|K_j)$ – Az *i* terméket megnézték, *j* terméket pedig kosárba tették;

$P(M_i|V_j)$ – Az *i* terméket megnézték, *j* terméket pedig megvásárolták;

$P(K_i|M_j)$ – Az *i* terméket kosárba tették, *j* terméket pedig megnézték;

$P(K_i|K_j)$ – Az *i* terméket kosárba tették, *j* terméket szintén kosárba tették;

$P(K_i|V_j)$ – Az *i* terméket kosárba tették, *j* terméket pedig megvásárolták;

$P(V_i|M_j)$ – Az *i* terméket megvásárolták, *j* terméket pedig megnézték;

$P(V_i|K_j)$ – Az *i* terméket megvásárolták, *j* terméket pedig kosárba tették;

$P(V_i|V_j)$ – Az *i* terméket megvásárolták, *j* terméket szintén megvásárolták!

Ezek az adatok egyben meghatározzák a modellünket is. Tárolásukat kétféleképpen valósíthatjuk meg:

- Az első megvalósításban mindegyik esetre külön adathalmazt használunk, és két kulccsal indexelünk, a két termék azonosítójával, majd a kapott érték visszaadja a feltételes valószínűség értékét.

- Másik megvalósításban tárolhatjuk az összes adatot egy struktúrán belül, ekkor azonban 2 helyett 4 kulcsot kell alkalmaznunk, amely jelentős mértékben megnöveli a keresési állapotteret, így a keresés idejét is. A négy kulcs ebben az esetben a következő kell, hogy legyen:

- Az *i* termék azonosítója
- Az *i* terméken végzett tranzakció típusa
- A *j* termék azonosítója
- A *j* terméken végzett tranzakció típusa

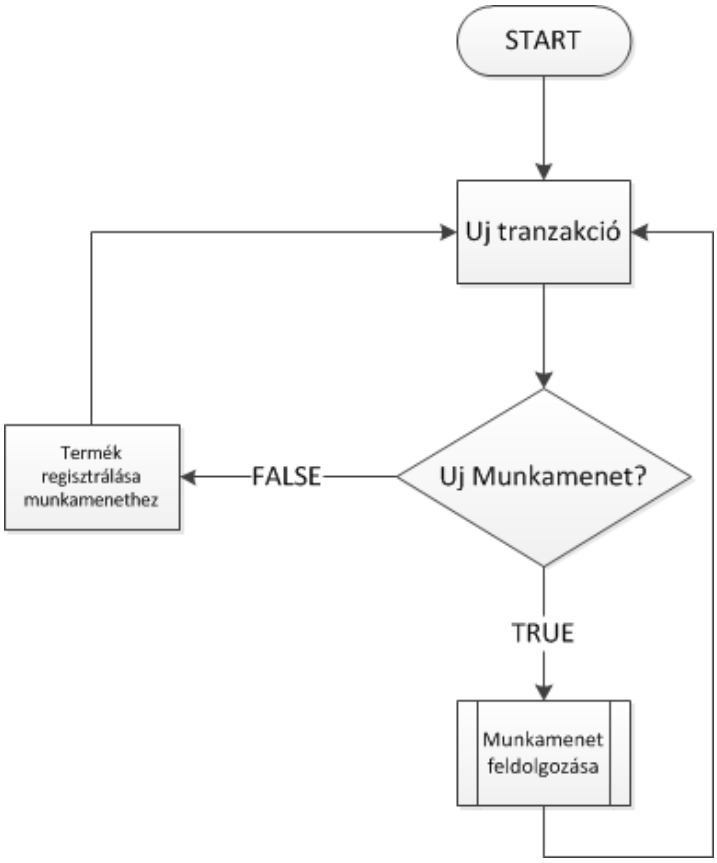
Ugyanakkor előnyhöz nem jutunk vele, hiszen a tárolt adatok mennyisége ugyanannyi, vagyis az egy adatstruktúrába történő mentés felesleges és időpazarló. Mind az építése, mind pedig a felhasználása időigényesebb, ennek az egy adatsruktúrás változatnak, ezért a 9 külön részre bontott megvalósítást választottam az implementálás során, azaz a fent említett feltételes valószínűségek mindegyikének tárolására külön struktúrát használok. C++ nyelven erre a már korábban említett map beépített struktúra biztosít lehetőséget. Ezen struktúra építésének lépésszáma O(N*logN), ahol N az termékek számának a négyzete. Keresés a struktúrában O(logN) számú lépésben történik. Egy ilyen struktúra mérete tekintve hogy két kulcsot, valamint ahhoz egy értéket tartalmaz, melyek közül a kulcsok INT, az érték pedig float típusú későbbi feldolgozásokra felkészítve - a következőképpen számítható:

$4 * N + 2 * 4 * N * N = 4 * N * (2 * N + 1)$ byte

Ahol 4*N byte a primer kulcsok mérete, hiszen egy INT típus 4 byte, valamint N darab ilyen kulcs kell, ahol N a termékek száma. A 2*4*N*N byte pedig a primer kulcshoz tartozó szekundér kulcs, valamint az ahhoz tartozó érték méretét reprezentálja, ahol N szintén a termékek számát jelöli. Ebből kiszámítható, hogy a teljes adathalmaz mérete így 9*2,5GB memóriát igényelne, amellyel egy átlagos számítógép nem rendelkezik. Itt jön tehát a képbe az optimalizálás, melyre két lehetőségünk akad. Egyik, hogy a méretet lekorlátozzuk, azaz nem veszünk figyelembe minden terméket. Ezzel bár időt spórolunk, de értékes információt veszíthetünk. Másik lehetőség, hogy az éppen nem használt struktúrákat kiírjuk háttértárra, hiszen egy mai átlagos számítógép rendelkezik elegendő tárhellyel. Ebben az esetben viszont roppant mértékben megnövekedik mind a tanulás, mind pedig a keresés folyamata, hiszen az aktuálisan használni kívánt struktúrát a háttértárról be kell olvasnunk a memóriába, az előzőt pedig a memóriából kiírni háttértárra. Ezek a műveletek pedig roppant mértékű overheadet eredményeznek, így számolnunk kell azzal, hogy ezt megengedhetjük-e magunknak, avagy inkább veszítsünk el néhány adatot a nagyságrendekkel nagyobb sebesség érdekében. Az implementáció során én a korlátozást vezettem be, melyet szabadon lehet változtatni így esetlegesen finomítani a tanuláson és kihegyezni a saját számítógépünk tulajdonságaira. Ahhoz hogy egy 4GB memóriával rendelkező számítógép

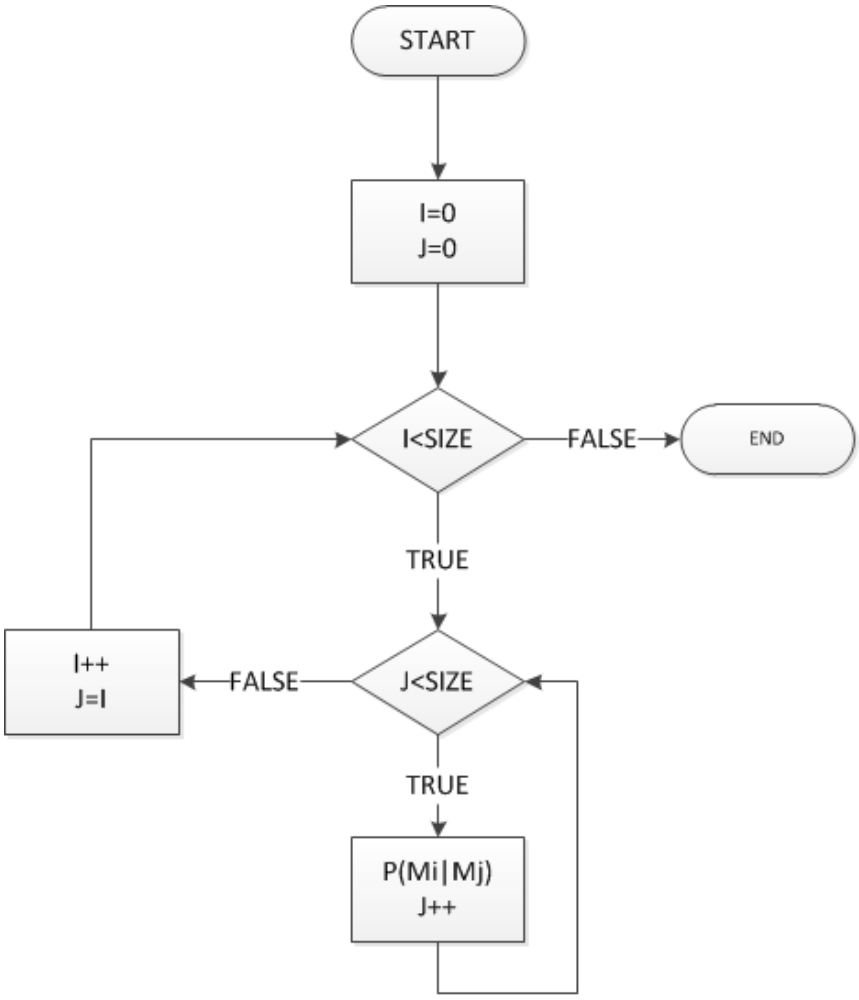
még tudja kezelni az adatokat figyelembe véve azt is, hogy az ajánlattétel során is generálódik némi adatmennyiség maximum 2-2,5 GB méretet engedhetünk meg a 9 struktúrának összesen, ami azt jelenti, hogy N értékét körülbelül 5900-ra le kell korlátoznunk. A 17000 termékhez képest érezhető ez mekkora információ veszteséget jelent, így ha tehetjük ajánlatosabb a háttértáras megvalósítást alkalmazni, hiszen ezen megvalósítás elméletben bármennyi időt megenged a tanulásra. Gyakorlatban viszont néhány napnál nem érdemes többet rááldozni, esetleg ha dinamikusan tudjuk együtt használni az ajánlattevő algoritmusunkkal.

3.1.1. Blokkdiagramok [12]:



1 Ábra: Az elemző algoritmus sematikus blokkvázlata

Legelőször beolvasunk egy új tranzakciót. Kiolvassuk belőle a munkamenetet, a termékazonosítót, valamint a végrehajtott tranzakció típusát. Ezek után megvizsgáljuk, hogy új munkamenet kezdődött-e, vagy még az előző folytatódik. Amennyiben folytatódik, úgy nincs más dolgunk, mint a munkamenethez nyilvántartásba venni a terméket a rajta végrehajtott tranzakcióval, valamint egyéb információkkal, amelyekre szükségünk van, például, hogy hányadik termék a munkameneten belül, a termék első előfordulási helyének indexe, vagy a rajta végrehajtott tranzakciók mennyisége. Amennyiben új munkamenet kezdődött, úgy a régi munkamenetet feldolgozzuk, majd az új munkamenethez regisztráljuk a beolvasott adatokat és folytatjuk a beolvasást.



2 Ábra: A munkamenet feldolgozásának részletezése

Első lépésben inicializáljuk az I és J változókat 0 értékre. A lépések során végighaladunk a meglátogatott termékek listáján, melyen a termékek időrendi sorrendben szerepelnek, azaz a listában *i* termék előrébb szerepel, mint *j* termék, ha a munkameneten belül *i* termék első előfordulási helye kisebb, mint *j* termék első előfordulási helye. Ezek után minden *i* termékhez kiszámoljuk minden utána következő *j* termék esetén a $P(M_i|M_j)$ feltételes valószínűséget. Ezt természetes minden további adathalmaz esetén elvégezzük.

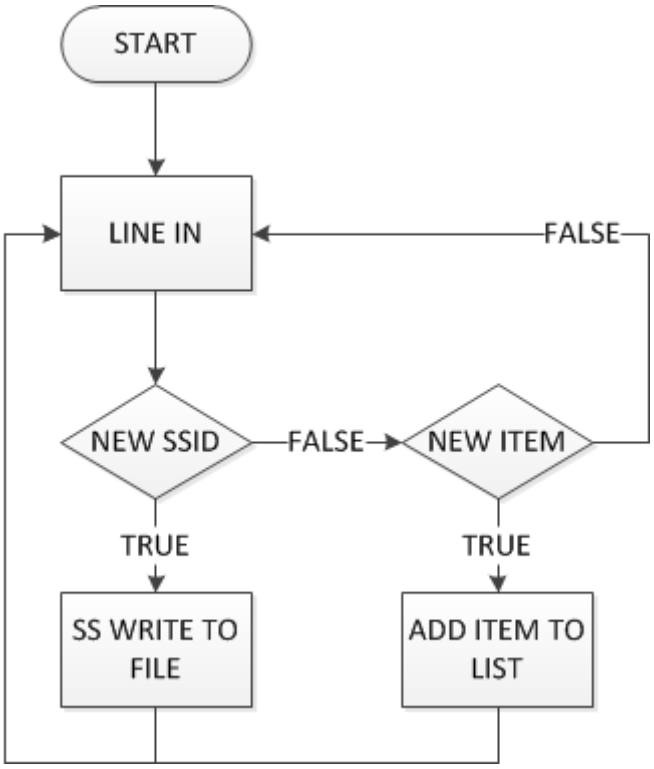
3.1.2. Adatok elő feldolgozása:

Hogy a lehető leghatékonyabban tudjuk felhasználni a rendelkezésre álló adatokat, érdemes azokat a tanulási fázis előtt előkészíteni olyan formára, hogy abból könnyen ki tudjuk nyerni a hasznos információkat. Ez ebben az esetben azt jelenti, hogy figyelmen kívül hagyhatjuk azokat a termékeket melyek egy munkameneten belül többször is előfordulnak. Helyette elég csak a legelső előfordulási pontjukat figyelembe vennünk a többi termékhez képest, mindezt a munkameneten belül előforduló, hozzá tartozó legnagyobb tranzakciós számmal. Ezen algoritmusnak nincs nagy tárigénye, hiszen az adatokat csak feldolgozza, módosít azokon, de nem tárolja őket, valamint az időigénye is O(N)-esnek tekinthető, ahol N

a tranzakciók száma. Hogy mi tekinthető egy hatékony elő feldolgozó algoritmusnak azt erősen befolyásolja a statikus tanulásra használni kívánt algoritmus.

3.1.3. A megvalósított előfeldolgozó algoritmus blokkdiagrammja:

Az általam implementált előfeldolgozó algoritmus blokkdiagrammja, mely mindkét realizált statikus tanulóalgoritmus segítségét egyaránt szolgálja [12]:



3 Ábra: Előfeldolgozó

A feldolgozni kívánt állományból beolvasunk egy tranzakciót, azaz egy új sort, majd abból a munkamenet számát, a termékazonosítót és a végrehajtott tranzakciót. Amennyiben új munkamenet kezdődött a régi munkamenet módosított adatait kiírjuk a feldolgozott adatokat tartalmazó fájlba, majd folytatjuk az olvasást. Amennyiben a régi munkamenet folytatódott, megvizsgáljuk, hogy a termék szerepelte már a munkameneten belül valamikor korábban. Ha nem szerepelt, a terméket regisztráljuk a munkamenethez, és a termékeket tartalmazó lista végére tesszük, ezzel megőrizve a termékek előfordulásának sorrendjét. Ha már szerepelt a termék az adott tranzakcióval, teljes mértékben figyelmen kívül hagyjuk az új előfordulási helyet és folytatjuk a beolvasást. Ezekkel a lépésekkel adott algoritmus esetén információvesztés nélkül értünk el adattömörítést, így hatékonyabb és gyorsabb tanulást produkáltunk.

3.1.4. Ajánlattétel:

Miután megalkottuk a modellünket, valamint a tanulóalgoritmusunk is lefuttattuk az adatokat felhasználhatjuk ajánlatok tételére. Az ajánlattétel egyik megvalósítása a következőképpen nézhet ki:

Vegyük a munkamenetben megnézett, megvásárolt és kosárba tett termékeket. Tároljuk ezeket tömbökben melyek az XM, XK, XV tömbök lesznek. Ezután minden termékre határozzuk meg, hogy a munkamenetben szereplő termékek mennyire támogatják más termékek a megvásárlását, kosárba tételét, valamint megtekintését. A támogatottság kiszámítására a következő képleteket alkalmazzuk:

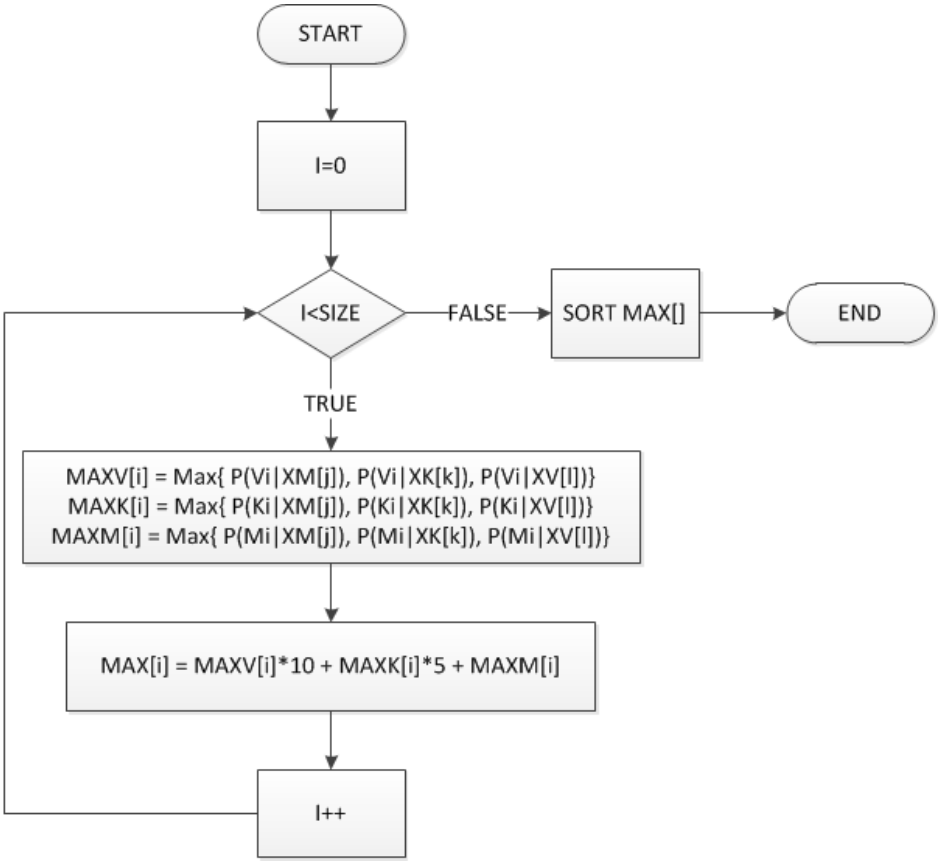
$$MAXV[i] = Max\{P(V_i|XM[j]), P(V_i|XK[j]), P(V_i|XV[j])\}$$
$$MAXK[i] = Max\{P(K_i|XM[j]), P(K_i|XK[j]), P(K_i|XV[j])\}$$
$$MAXM[i] = Max\{P(M_i|XM[j]), P(M_i|XK[j]), P(M_i|XV[j])\}$$

Itt $P(X_i|X_j)$ a tanulás során megalkotott piros fekete fák végigindexelve a munkamenetben szereplő egyes termékekkel, melyeket előzőleg a XM, XK, XV tömbökben eltároltunk. A verseny keretében azonban ezen értékekkel még nem lehetünk elégedettek, hiszen profitot szeretnénk maximalizálni, így szeretnénk minél több terméket eladni, mintsem inkább, hogy minél több terméket nézzenek meg a felhasználók. Így a versenyben kiszabott pontozási útmutató szerint bevezetek az értékekre egy súlyozást is, majd eszerint keresem meg a leginkább támogatott terméket. Erre a képlet a következő:

$$MAX[i] = MAXV[i] * 10 + MAXK[i] * 5 + MAXM[i]$$

Ezután egy maximumkeresést végrehajtva a MAX tömbön megkapjuk, hogy mely az a három termék, mely a legnagyobb valószínűséggel szerepel a munkamenet folytatásában. A tömböt sorrendezve pedig tetszőleges számú ajánlatot tehetünk a leginkább támogatott termékek köréből. Amit érdemes figyelembe vennünk, hogy a tömb sorrendezése O(N*logN) lépésből történik ahol N a tömb elemeinek száma, ezután pedig O(1) lépés kiolvasni a számunkra elegendő mennyiséget. Amennyiben a tömb nem sorrendezett, úgy O(N) lépésben végig kell olvasnunk a tömböt, ennyi viszont elég is ahhoz, hogy meghatározzuk a maximumokat bármennyire is legyen szükségünk.

3.1.5. Predikció blokkdiagramm:



4 Ábra: Predikciós algoritmus

3.2. Egytermékes, sorrend nélküli becslésű algoritmus:

Az előző részben leírt algoritmustól mindösszesen a tanulási fázis különbözik, az ajánlattétel, valamint a modell szerkezete teljesen megegyezik. Így az algoritmusok tárigényében nincs különbség, valamint az ajánlattevő algoritmus lefutása is azonos idő alatt történik, viszont a modellépítőnek futási ideje kismértékben megnövekedik a bevezetett különbség miatt.

A tanulásban bevezetett különbség pedig mindössze egy nagyon apró finomítás, mégpedig, hogy nem csak egy adott termék után előforduló termékekkel hozzuk kapcsolatba, hanem visszamenőleg is, azokkal, amelyek előtte szerepeltek, de még ugyanabban a munkamenetben.

Formalizálva a különbség a halmazok definíciójában jelenik meg:

A halmaz: Azon munkamenetek, melyekben *j* terméket megnézték

B halmaz: Azon munkamenetek, melyekben *i* és *j* terméket is megnézték.

$$P(Mi|Mj) = \frac{|B|}{|A|}$$

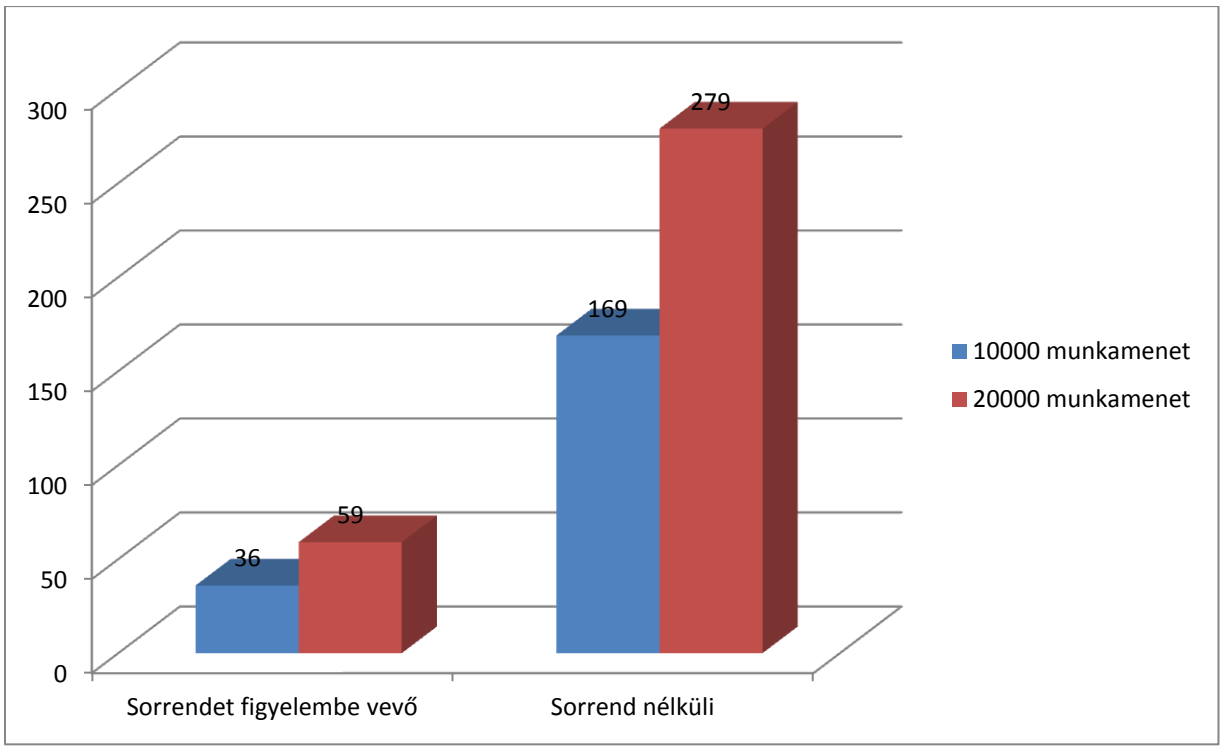
Ezek után pedig minden megegyezik az előző algoritmussal.

Ennek a változtatásnak következtében tehát minden terméket minden termékkel kapcsolatba hozunk egy adott munkameneten belül, míg előzőleg ez nem történt meg.

Ebben az esetben, ha N az átlagosan előforduló termékek száma egy munkamenetben, és K munkamenet száma, akkor K*N*N kapcsolatot vizsgálunk, szemben az előző példával, ahol csak K*N*(N/2). Ez tehát kétszer annyi vizsgálatot jelent, így később érdemes lesz megfontolnunk, hogy ha egyáltalán jobb eredményeket produkál az algoritmus megéri-e ezt használnunk a megnövekedett időigény miatt.

3.3. Algoritmusok összehasonlítása

A két implementált algoritmust több szempont szerint is elemeztem. Első esetben a modellépítés, azaz a statikus tanulás tár és időigényét vizsgáltam. Az első, azaz a sorrendet figyelembe vevő algoritmus futási ideje minden esetben nagyságrendekkel kisebb volt, mint a második algoritmus futási ideje. Tárigényt illetően a két algoritmus megegyezett, ez logikus is, hiszen ugyanazon mennyiségű adatot tárol mindkét algoritmus ugyanarra a bementre, viszont különbség a két algoritmus közt, hogy a fontos számértékek számítása esetén, a második algoritmus sokkal több paraméterrel dolgozik, mint a sorrendet figyelembe vevő algoritmus. Számszerűsítve ezek az értékek:



5 Ábra: Időigények

Sorrendet figyelembe vevő algoritmus 10000 munkamenet figyelembe vételével: 36 másodperc.
Sorrendet figyelembe vevő algoritmus 20000 munkamenet figyelembe vételével: 59 másodperc.
Sorrend nélküli algoritmus 10000 munkamenet figyelembe vételével: 169 másodperc.
Sorrend nélküli algoritmus 20000 munkamenet figyelembe vételével: 279 másodperc.
Ez körülbelül ötszörös növekedést jelent időigényt illetően. A tárigény mindkét esetben 10000 munkamenet esetén 85MB volt, míg 20000 munkamenet esetén 127MB-re

növekedett. A nem lineáris növekedés oka a beolvasás optimalizációjával magyarázható. Mivel az IO műveletek időigényesebbek, mintha memóriából dolgoznánk, ezért a teljes adatállományt először beolvasom a memóriába, majd onnan kezdem el a modellépítést, a statikus tanulást. Mindez némi memóriát vesz igénybe, mely minden esetben jelentkezik, viszont jelentős mértékben, nagyságrendekkel csökkenthető ezzel a futási idő, így megéri ezt a módszert alkalmazni. Mivel nem a teljes adatállományt olvastam be ebben az esetben, csupán annak egy töredékét ez a statikus méret ebben az esetben 31.2 MB memóriát foglalt. Ebből könnyen megkaphatjuk tehát hogy a tényleges modell 10000 munkamenet esetén 53.8 MB, míg 20000 munkamenet esetén 95.8 MB. Itt már jobban megfigyelhető a linearitás. A minimális eltérés a lineáris növekedéstől azzal magyarázható, hogy a bővített esetben nem feltétlenül szerepelnek csupán új termékek, termékkapcsolatok, hanem esetlegesen olyanok is előfordulhatnak, melyek a modellben már szerepelnek, így új értéket nem kell felvennünk, csupán változtatnunk.
A predikciós algoritmus már jóval lassabb mindkét modellépítőtől. Ennek oka a nagyszámú keresés a modelltérben. Amennyiben száz munkamenethez szeretnénk meghatározni a további legvalószínűbb három terméket, előre megadott három termékből, úgy ez 99 másodpercig tartott mindkét esetben, hiszen a predikciós algoritmusok azonosak a két megvalósított esetben. A használt számítógép paraméterei:
- Cpu: Intel® Core™ 2 Duo P7550 2,27GHz
- RAM: 4GB
- Windows 7 Home Premium 64bit

Ez azt jelenti, hogy egy munkamenet kiértékelése körülbelül 1 másodpercig tart, ami elfogadhatónak tekinthető, a való életben ennyi késleltetést megengedhetünk. Ezt az értéket természetesen jelentős mértékben lecsökkenthetjük, ha nem három, hanem csupán egy termékből végezzük a predikciót, ezzel viszont persze pontosságot veszíthetünk. Az eredményességet tekintve, mint ahogy az várható is volt, a sorrend nélküli algoritmus lényegesen hatékonyabb. Mindösszesen 10000 munkamenetből tanulva, valamint 1000 munkamenetre becslést adva az első esetben 87 munkamenet esetén volt legalább egy találat, összesen pedig 103 helyes találat volt, míg a második esetben 193 munkamenet esetén történt 262 találat, amely több mint kétszeres javulást jelent. Az eredmények kiértékeléséhez a keresztvalidációs technikát alkalmaztam.

3.4. Keresztvalidáció:

A keresztvalidáció során a tanulóállomány 3 lehetőleg azonos méretű adathalmazra bontjuk szét. Ezekután a tanulást a háromból csak két állományon végezzük el, majd kiértékeljük a harmadik állományre a predikciós algoritmust úgy, hogy számára azokat az adatokat, melyek csak a tanulóállományban szerepelhetnek kitakarjuk. A predikciós algoritmus lefutása után az eredményt összehasonlíthatjuk az eredeti állománnyal és így egy képet kaphatunk a hatékonyságról. Ezeket a lépéseket háromszor kell elvégeznünk, azokban az esetekben, amikor a predikciós algoritmust az első, majd második, végül pedig a harmadik részállományra futtatjuk.

4. További lehetőségek, és azok értékelései

További algoritmusokat gyárthatunk hasonló megfontolások és apró változtatások útján. Ilyen változtatások lehetnek, hogy adott munkameneten belül két terméket csak akkor hozzunk kapcsolatba, ha *i* termék után rögtön *j* termékre navigált a felhasználó *k* köztes termék nélkül.

Előre jelezni ezek hatékonyságát szinte lehetetlen, viszont kisebb adathalmazokon könnyen tesztelhetőek. Tesztelésre érdemes olyan méretű adathalmazt választani, mely esetén az algoritmusok még nagyon gyorsan lefutnak, ellenben már nagyon jó reprezentációnak számíthatnak valós körülmények leképzésére. Ha nagyon kis adatokon teszteljük az algoritmusainkat, könnyen előfordulhat olyan eset, amikor nem a valóságnak megfelelő eredményeket kapunk és így kevésbé hatékony megoldást kezdünk el alkalmazni.

Az előző két algoritmushoz hasonló algoritmusok, amikor nem csak egy termékkel hozunk kapcsolatba egy adott terméket, hanem kettő, vagy több termékkel. Azaz a feltételes valószínűségek, melyeket számolunk a következőképpen nézzenek ki:

$P(V_i | M_j M_k)$

Az A halmaz tartalmazza azon munkameneteket, amelyekben mind a *j*, mind pedig a *k* terméket megnézték.

A B halmaz legyen az A halmaz azon részhalmaza, melyekben az *i* terméket megvásárolták a *j* és *k* termékek előfordulása után.

Ekkor tehát:

$$P(V_i | M_j M_k) = \frac{|A|}{|B|}$$

További algoritmus pedig lehet, ha a sorrendet ezekben az esetekben is figyelmen kívül hagyjuk, azaz nem lényeges, hogy az egyes termékek milyen sorrendben szerepeltek egymás után.

Ezen algoritmusok implementálására nem kerítettem sort, hiszen a futási idő, valamint a tárigény már a két implementált esetben is jelentős volt, ezen algoritmusok igényei még nagyobbak, így érdemleges tesztelést sem tudtam volna rajtuk végrehajtani. A sejtés viszont az, hogy valamelyest jobb eredményt lennének képesek ezen algoritmusok produkálni.

5. Irodalomjegyzék

[1] A. K. Jain, R. C. Dubes: “Algorithms for Clustering Data”, Prentice Hall Advanced Reference Series, 1988.

[2] Bodon Ferenc: Adatbányászati algoritmusok, tanulmány, 2010. http://www.cs.bme.hu/~bodon/magyar/adatbanyaszat/tanulmany/adatbanyaszat.pdf

[3] Jiawei Han, Michelin Kamber: Adatbányászat – Koncepciók és technikák, 2004, Panem KFT.

[4] Marton László – Fehérvári Arnold: “Algortimusok és adatstruktúrák“, Győr, 2001. http://eki.sze.hu/ejegyzet/ejegyzet/drmarton/algo.pdf

[5] Benedek Zoltán, Levendovszky Tihamér: “Szoftverfejlesztés c++ nyelven”, 2007. Szak Kiadó KFT.

[6] Ketskeméty László: “Valószínűségszámítás tömören”, 2007. Aula kiadó

[7] Ketskeméty László – Pintér Márta: “Bevezetés a matematikai statisztikába”, 1999. http://www.szit.bme.hu/~kela/stat.pdf

[8] Joshua Bloch, Joseph Bowbeer, Brian Goetz, David Holmes, Doug Lea, Tim Peierls: “Párhuzamos Java-programozás a gyakorlatban”, 2009. Kiskapu kiadó

[9] Michael T. Goodrich, Roberto Tamassia, David M. Mout: Data structures and algorithms in C++, 1999.

[10] Mark Allen Weiss: Algorithms, Data structures, and problem solving with C++

[11] Iványi Antal (szerk.): Informatikai algoritmusok II., ELTE Eötvös Kiadó, Budapest, 2005.

[12] Sike Sándor, Dr. Varga László: Szoftvertechnológia és UML, 2003, ELTE Eötvös Kiadó