ALGORITMA APRIORI

ATURAN ASOSIASI [THRESHOLD, SUPPORT DAN CONFIDENCE]

Aturan asosiasi dalam data mining bertujuan untuk menemukan pola frekuensi, asosiasi, korelasi dan struktur hubungan antar item atau obyek dari sekumpulan item yang ada dalam transaksi database, database relasional maupun informasi dari tempat penyimpanan yang lain.

Setiap item atau obyek adalah variabel bernilai Boolean yang merepresentasikan ada tidaknya item tersebut dalam transaksi. Setiap transaksi dapat direpresentasikan oleh nilai vektor Boolean untuk setiap variabelnya. Vektor Boolean dapat dianalisis untuk menemukan pola dari item-item yang cenderung muncul bersamaan dengan nilai frekuensi tertentu.

Sebagai contoh aturan asosiasi adalah informasi tentang pelanggan yang membeli komputer juga membeli perangkat lunak keuangan, dapat ditulis dalam aturan asosiasi sebagai berikut:

Komputer ➔ perangkat lunak keuangan
[suppport = 2%, confidence = 60%]

Nilai support dan confidence pada aturan tersebut di atas adalah dua pengukuran untuk pola kecenderungan dalam transaksi tersebut. Setiap pengukuran yang obyektif untuk pola yang menarik didasarkan pada struktur dari pola itu sendiri dan nilai statistik yang mendasarinya.

Secara umum, setiap pengukuran diasosiasikan dengan sebuah nilai threshold yang nilainya dapat ditentukan sendiri oleh pengguna data mining. Aturan asosiasi yang tidak menggunakan threshold cenderung tidak menarik karena tidak merepresentasikan pengetahuan kepada pengguna data mining.

Salah satu faktor yang memberikan konstribusi untuk menentukan apakah suatu pola menarik atau tidak adalah kesederhanaannya dalam pemahaman manusia. Semakin kompleks struktur sebuah aturan, maka semakin sulit untuk diinterpretasikan sehingga pola yang dibentuk semakin tidak menarik.

Setiap pola yang disajikan harus mempunyai sebuah nilai pengukuran yang memberikan kepastian bahwa pola tersebut layak dipercaya sebagai pola yang terbaik atau tidak. Pengukuran untuk kepastian dalam aturan asosiasi dirumuskan secara singkat dalam bentuk “A=>B”, dimana A dan B adalah transaksi yang akan dianalisis. Rumus aturan asosiasi dengan tingkat kepastian disebut sebagai confidence. Confidence dari transaksi “A=>B” dari data semua transaksi yang ada dalam database dapat didefinisikan sebagai berikut:

$$confidence\ A => B = \frac{jumlah\ transaksi\ dari\ A\ dan\ B}{jumlah\ transaksi\ dari\ A}$$

Aturan asosiasi untuk pembelian komputer dan perangkat lunak keuangan dengan confidence sebesar 60% menunjukkan bahwa 60% dari pelanggan yang membeli komputer juga membeli perangkat lunak keuangan.

Salah satu kegunaan dari sebuah pola adalah faktor yang menentukan pola tersebut menarik atau tidak. Hal tersebut dapat diperkirakan dengan menggunakan fungsi tertentu yang disebut sebagai support. Support dari pola asosiasi menunjukkan besarnya persentase data yang ada dalam transaksi yang dianalisis, sehingga persentase tersebut dapat menunjukkan bahwa pola tersebut benar. Support dari aturan asosiasi transaksi “A=>B” dari data semua transaksi yang ada dalam database dapat didefinisikan sebagai berikut:

$$Support\ A => B = \frac{jumlah\ transaksi\ dari\ A\ dan\ B}{jumlah\ transaksi\ keseluruhan}$$

Aturan asosiasi untuk pembelian komputer dan perangkat lunak keuangan dengan support sebesar 2% menunjukkan bahwa 2% dari keseluruhan transaksi adalah transaksi pembelian komputer dan perangkat lunak keuangan yang selalu dibeli secara bersamaan.

KONSEP DASAR ALGORITMA APRIORI

Konsep dasar aturan apriori dijelaskan sebagai berikut: J = $\{i_1, i_2, \ldots, i_n\}$ adalah kumpulan item, sedangkan D adalah transaksi dalam database, dimana setiap transaksi dari T adalah sekumpulan item sedemikian hingga $T \subseteq J$.

Setiap transaksi diasosiasikan dengan sebuah identitas TID. A adalah kumpulan item. Sebuah transaksi T akan mengandung A jika dan hanya jika $A \subseteq T$. Aturan asosiasi adalah implikasi dari bentuk A=>B, dimana $A \subset J$, $B \subset I$, dan $A \cap B = \phi$ Aturan A=>B berada dalam transaksi D dengan support s, dimana s adalah persentase dari transaksi yang ada dalam D yang mengandung $A \cup B$. Konsep ini diambil dari rumus probalilitas $P(A \cup B)$. Aturan A=>B mempunyai confidence c dalam transaksi D jika c adalah persentase dari transaksi D yang mengandung A dan juga mengandung B. Konsep ini diambil dari rumus probabilitas P(A|B). Dengan demikian:

support(A=>B)=$P(A \cup B)$
confidence(A=>B)=P(B|A)

Kedua rumusan di atas yang memenuhi kedua nilai baik nilai minimum support *threshold* (min_sup) maupun *minimum confidence threshold* (min_conf) disebut sebagai *strong*. Untuk kesepakatan, nilai *support* dan *confidence* berada dalam range 0% sampai dengan 100%, lebih baik bila dibandingkan dengan range 0 sampai dengan 1.

Kumpulan dari item mengacu pada istilah *itemset*. Istilah "itemset" lebih banyak digunakan dalam literatur data mining bila dibandingkan dengan istilah "item set". Sebuah *itemset* yang mengandung k item disebut sebagai k-itemset.

Kumpulan {komputer, perangkat lunak keuangan} adalah 2-itemset. Frekuensi kejadian dari *itemset* adalah jumlah transaksi yang mengandung *itemset* tersebut, secara sederhana disebut sebagai *frequency* atau *support count* atau *count of itemset*.

Sebuah *itemset* yang memenuhi *minimum support* jika *frequency* dari *itemset* tersebut lebih besar atau sama dengan hasil kali antara *min_sup* dan jumlah total transaksi dalam D. Jumlah transaksi yang dibutuhkan *itemset* untuk memenuhi *minimum support* adalah mengacu pada *minimum support count*. Jika *itemset* memenuhi *minimum support*, maka disebut sebagai *frequent itemset*. Kumpulan dari frequent k-*itemset* umumnya dilambangkan sebagai L_k.

Dalam implementasinya dalam database berukuran besar, aturan asosiasi menggunakan 2 proses utama yaitu sebagai berikut:

1. Mencari semua *frequent itemset* yang mempunyai *support* dan *confidence* yang lebih besar daripada *minimum support* dan *minimum confidence*
2. Hasilkan aturan asosiasi yang bersifat *strong* dari semua *frequent itemset* sehingga memenuhi baik *minimum support* maupun *minimum confidence*

ALGORITMA APRIORI

Secara umum, aturan asosiasi dalam data mining terbagi menjadi beberapa bagian sesuai dengan implementasinya dalam database berukuran besar, antara lain:

1. Aturan asosiasi Boolean berdimensi tunggal dari transaksi di beberapa database
2. Aturan asosiasi multi dimensi dari database relasional dan data warehouse
3. Aturan asosiasi multilevel dari transaksi di beberapa database
4. Aturan asosiasi untuk analisis korelasi
5. Constraint dari aturan asosiasi dan pola sequential

Aturan asosiasi yang telah di bahas pada sub bab terdahulu adalah aturan asosiasi berdimensi tunggal dan merupakan aturan asosiasi Boolean. Oleh karena itu dibutuhkan suatu metode sederhana untuk mencari semua *frequent itemset* yang merupakan proses utama dari aturan asosiasi.

Algoritma apriori adalah algoritma yang sangat mendasar dalam mencari *frequent itemset* dari transaksi database berukuran besar. Nama algoritma apriori diambil dari kenyataan bahwa algoritma ini menggunakan pengetahuan sebelumnya (*prior knowledge*) dari *frequent itemset* untuk proses iterasi berikutnya.

Apriori menggunakan pendekatan iteratif yang lebih dikenal dengan istilah pencarian *level-wise*, dimana k-itemset digunakan untuk mengekplorasi (k+1)-itemset. Jika frequent dari 1-itemset sudah ditemukan, maka hasilnya dapat dilambangkan dengan L_1. L_1 kemudian digunakan untuk mencari L_2, *frequent* dari 2-itemset. L_2 digunakan untuk mencari L_3, dan seterusnya, sampai tidak ditemukan lagi *frequent* untuk k-itemset. Untuk mendapatkan hasil L_k dibutuhkan pembacaan database secara keseluruhan.

Untuk meningkatkan kinerja dan efisiensi dari pencarian *level-wise* dari *frequent itemset*, maka diperlukan reduksi dari hasil pencarian tersebut. Proses reduksi disebut sebagai *apriori property*. Semua *subset* yang tidak kosong dari *frequent itemset* harus dihitung nilai frekuensi kemunculannya.

Dengan demikian, jika itemset I tidak mencapai nilai *minimum support threshold* atau *min_sup*, maka I tidak perlu dihitung nilai frekuensi kemunculannya, oleh karena P(I)<min_sup. Jika sebuah item A ditambahkan ke dalam *itemset* I, maka hasil *frequent* dari *itemset* berikutnya tidak lebih dari itemset I. Oleh karena itu, IÈA juga tidak dihitung nilai frekuensinya, karena P(I∪A)<min_sup. *Apriori property* dapat diimplementasikan dalam algoritma berikut yang menjelaskan bagaimana L_{k-1} dapat digunakan untuk mencari L_k.

Kedua proses yaitu langkah *join* atau "mengawinkan" dan *prune* atau "memangkas" dapat dijelaskan secara konseptual sebagai berikut:

1. Langkah *Join* : untuk mencari L_k, maka *candidate* dari k-*itemset* harus dihasilkan dengan cara menggabungkan L_{k-1} dengan dirinya sendiri. Candidate dari k-itemset dilambangkan sebagai C_k. I_1 dan I_2 adalah itemset dari L_{k-1}. Notasi $I_i(j)$ mengacu pada item ke-j di dalam I_i (Contoh $I_1(k-2)$ mengacu pada item kedua terakhir dari I_1). Berdasarkan kesepakatan, apriori mengasumsikan bahwa item-item dalam transaksi atau *itemset* sudah dalam keadaan terurut secara *lexicographic*. Join dari $L_{k-1} \bowtie L_{k-1}$ dapat dilakukan, dimana semua anggota dari Lk-1 dapat di-join jika terdapat anggota utama (k-2). Oleh karena itu, anggota I_1 dan I_2 dari L_{k-1} dapat di-join jika $(I_1[1]=I_2[1])$ ^ $(I_1[2]=I_2[2])$ ^ ….^ $(I_1[k-2]=I_2[k-2]$ ^ $(I_1[k-1]<I_2[k-1])$. Kondisi $I_1[k-1]<I_2[k-1]$ hanya memastikan bahwa tidak ada nilai duplikasi yang akan dihasilkan. Hasil dari item yang dibentuk dari proses join I_1 dan I_2 adalah $I_1[1]I_2[2]….I_1[k-1]I_2[k-1]$.
2. Langkah prune : C_k adalah superset dari L_k, oleh karena itu, anggota dari C_k dapat di-*frequent* maupun tidak dapat di-*frequent*, namun semua k-*itemset* yang di-*frequent* ada di dalam C_k. Proses pencarian dalam database untuk mendapatkan jumlah dari setiap item di dalam C_k dapat ditentukan dari L_k (semua *candidate*

yang jumlah-nya lebih kecil dari *minimum support count*, oleh karena itu merupakan anggota dari L_k). C_k akan mempunyai ukuran yang sangat besar sehingga waktu untuk proses komputasi menjadi lebih lama. Untuk memperkecil ukuran C_k, maka digunakan *apriori property*. Semua (k-1) itemset yang tidak di-*frequent* tidak menjadi subset dari L_{k-1} yang di-*frequent*. Oleh karena itu, jika semua subset (k-1) dari candidate k-*itemset* yang tidak ada di dalam L_{k-1}, maka candidate tidak di-*frequent* dan dapat dihapus dari C*k*.

Kedua proses join dan prune pada penjelasan di atas dapat dijabarkan dengan lebih singkat dalam bentuk psedeu-code di bawah ini:

- Langkah join: C_k dihasilkan dengan cara mengawinkan L_{k-1} dengan dirinya sendiri
- Langkah Prune: semua (k-1)-*itemset* yang tidak di-*frequent* tidak menjadi *subset* dari *frequent* k-*itemset*
- Pseudo-code:

```
Ck: Candidate itemset dari k
Lk : frequent itemset dari k
L1 = {frequent items};
for (k = 1; Lk ≠ ϕ; k++) do begin
     Ck+1 = candidate yang dihasilkan dari Lk;
    for each transaksi t dalam database do
       hitung jumlah semua candidate di dalam Ck+1
       yang terkandung dalam transaksi t
    Lk+1  = candidate di dalam Ck+1 dengan nilai min_support
end
return ∪k Lk;
```

Sedangkan untuk menghasilkan candidate, maka digunakan psedeu-code di bawah ini:

- Asumsi bahwa item dalam L_{k-1} sudah dalam keadaan terurut
- Langkah 1: join L_{k-1} dengan dirinya sendiri

```
insert into Ck
select p.item1, p.item2, …, p.itemk-1, q.itemk-1
from Lk-1 p, Lk-1 q
where p.item1=q.item1, …, p.itemk-2=q.itemk-2, p.itemk-1 < q.itemk-1
```

- Langkah 2: pruning

```
for all itemsets c in Ck do
      forall (k-1)-subsets s of c do
            if (s is not in Lk-1) then delete c from Ck
```