BAB 2

LANDASAN TEORI

Bab ini memuat landasan teori yang mendasari penelitian yaitu *data mining*, *clustering*, 2 algoritma *clustering* yang diimplementasi yaitu *Cobweb* dan *Iterate*, dan aplikasi WEKA. Penjelasan landasan teori dilakukan berurutan mulai dari *data mining* hingga aplikasi WEKA. Khusus algoritma *Cobweb* merupakan algoritma yang populer dalam teknik *clustering*. Sedangkan *Iterate* merupakan algoritma yang baru dalam *clustering*. Selain itu, dijelaskan mengenai aplikasi WEKA sebagai aplikasi acuan. Tugas Akhir ini melakukan penelitian, analisa dan implementasi kedua algoritma, serta uji coba hasil implementasi.

2.1 *Data Mining*

Data mining adalah suatu proses yang bertujuan mendapatkan informasi-informasi baru, bernilai dan informasi *non-trivial* dari berbagai sumber data yang besar secara iteratif baik secara manual maupun otomatis (Kantardzic, 2003).

Berdasarkan aktifitasnya, *data mining* digolongkan menjadi 2 kategori yaitu *predictive* dan *descriptive* (Kantardzic, 2003). *Predictive data mining* menghasilkan suatu model dari sebuah sistem yang dideskripsikan oleh *dataset*, biasanya digunakan untuk suatu klasifikasi, memprediksi sesuatu atau melakukan estimasi. Sedangkan *descriptive data mining* menghasilkan suatu informasi yang baru dan *non-trivial* berdasarkan data yang tersedia, biasanya digunakan untuk menganalisis suatu sistem terhadap bentuk dan hubungan dalam kumpulan data tidak diperkirakan sebelumnya. Kedua aktifitas tersebut dapat dicapai dengan menggunakan teknik-teknik dalam *data mining*. *Data mining* dibagi dalam 6 teknik (Kantardzic, 2003), antara lain

- *Classification*

  Classification adalah teknik menemukan fungsi untuk memprediksikan suatu grup yang beranggotakan kumpulan data yang telah

Implementasi algoritma clustering..., Adolf Pandapotan

didefinikan terlebih dahulu. Contohnya memprediksi cuaca suatu hari apakah akan cerah, hujan, atau berawan.

- *Clustering*

  Clustering adalah suatu teknik dalam *data mining* yang digunakan untuk memasukan data ke dalam grup yang bersesuaian tanpa pengetahuan yang mendalam tentang grup tersebut. Contohnya mencari hubungan antara penjualan kopi dengan gula.

- *Regression*

  Regression adalah teknik *data mining* dengan menemukan fungsi dengan cara memetakan suatu data ke dalam nilai variabel sebenarnya yang diprediksi. Contohnya penggunaan standar deviasi untuk mengetahui berapa jumlah penjualan jika jumlah produksi ditingkatkan.

- *Summarization*

  Summarization adalah suatu teknik *data mining* dengan langkah tambahan yang melibatkan metode-metode untuk menemukan deskripsi lengkap untuk suatu kumpulan data. Contohnya menambah penjualan tiap hari untuk meningkatkan penjualan per bulan.

- *Dependency Modeling*

  Dependency Modeling adalah teknik *data mining* dengan menemukan model yang menggambarkan secara jelas keterkaitan antara variabel atau antara nilai dari suatu *attribute* dalam himpunan data atau sebagian dari himpunan data. Contohnya adanya hubungan linier jika terjadi peningkatan penjualan pasta gigi, maka terjadi juga peningkatan penjualan sikat gigi.

- *Change and Deviation Detection*

  Change and Deviation Detection adalah teknik *data mining* dengan menemukan perubahan yang signifikan dalam suatu himpunan data.

Contohnya jika terjadi kecurangan transaksi pada bank, maka kemungkinan terjadi ditempat yang jauh dari bank.

2.2 *Clustering*

Seperti yang sudah dijelaskan sebelumnya, *clustering* mengklasifikasikan data ke dalam grup-grup. Pengelompokan ini dilakukan menggunakan suatu fungsi sehingga dalam suatu grup berisikan data-data yang mirip satu dengan yang lain, sebaliknya data dari suatu grup tidak mirip dengan data dari grup yang lain. *Input* dari *clustering* yaitu *dataset* dan fungsi yang mengukur kemiripan dari antara 2 buah data. Sedangkan *output*nya adalah suatu himpunan grup yang berisikan data-data yang telah dikelompokan dan setiap grup memiliki deskripsinya masing-masing.

Suatu data dalam *clustering* memiliki *attribute-attribute* dan suatu nilai yang dapat diukur. Data yang valid memiliki kemiripan terhadap data dalam satu grup dibandingkan dengan data pada grup yang lain. Metode yang digunakan dalam *clustering* memungkinkan suatu data dieksplorasi untuk mengetahui di grup mana data tersebut akan dimasukan. Untuk data yang memiliki jumlah *attribute* satu, dua atau tiga bisa saja dilakukan pengelompokan secara manual, namun jika memiliki *attribute* lebih dari tiga dan jumlah datanya besar akan sangat sulit dan membutuhkan waktu yang banyak untuk mengelompokannya.

Grup yang dihasilkan dari pengelompokan data-data dianggap sebagai sebuah *concept* (*cluster*/kelas). Suatu *concept* memiliki deskripsi yang mewakilinya. Sebagai contoh diberikan data kambing dan sapi, keduanya berada dalam grup yang sama. Grup tersebut memiliki deskripsi hewan mamalia atau bisa saja herbivora.

Clustering dibagi menjadi 2 teknik yaitu *hierarchy clustering* dan *non-hierarchy clustering* (Thearling, 2008). Dalam teknik *hierarchy clustering*, *concept-concept* yang ada membentuk suatu *hirarchy*. Pada awalnya *hierarchy clustering* membuat *concept* paling atas yang menyimpan semua obyek. *Concept* tersebut merupakan *concept* yang paling umum dan membagi obyek-obyek

Implementasi algoritma clustering..., Adolf Pandapotan

tersebut dalam *cluster* yang lebih khusus atau dari banyak *concept* diatasnya kemudian digabung menjadi satu *concept*. Bentuk *hierarchy* ini ditampilkan menggunakan *tree*, sehingga disebut klasifikasi *tree*, sedangkan *non-hierarchy clustering* sebaliknya.

Pada tahun 1980, Michalski memperkenalkan *conceptual clustering* yang merupakan *hierarchy clustering* (Michalski dan Stepp, 1983). *Conceptual clustering* adalah salah satu teknik yang menghasilkan *concept-concept* dalam bentuk *hierarchy concept* dari kumpulan obyek yang terus menerus dengan cara membaginya ke dalam suatu sub-class secara iteratif (Hammouda dan Kamel, 2002). Dalam setiap tahap dari proses pembentukan *hirarchy*nya menghasilkan suatu *hierarchy* yang baik sehingga pada pengelompokan data terakhir, *hierarchy* tetap baik.

Metode tersebut menerapkan algoritma kecerdasan buatan yang disebut *hill-climbing search*. Algoritma ini terinsprasi dari seorang pendaki yang ingin mencapai puncak gunung. Agar pendaki gunung tersebut dapat mencapai puncak dengan cepat atau selamat, setiap kali pendaki gunung menemui persimpangan jalan, maka pendaki gunung tersebut memilih jalan yang terbaik dari antara pilihan. Sehingga pada saat pendaki gunung mencapai puncak gunung, ia melewati jalur yang baik, walaupun belum tentu jalan yang tercepat. Dengan kata lain, dalam setiap pemilihan yang terbaik pada setiap langkah diharapkan hasil akhirnya merupakan yang terbaik.

Dalam *conceptual clustering*, setiap data dianggap sebagai obyek yang memiliki pasangan *attribute-value*. *Attribute-value* adalah suatu pasangan yang terdiri dari *attribute* dan memiliki nilai (*value*). Contoh data seseorang dapat dianggap sebagai *attribute-value* seperti pada tabel 2.1.

Pada *conceptual clustering*, ada 2 problem yang harus dipenuhi (Fisher, 1987), yaitu

Implementasi algoritma clustering..., Adolf Pandapotan

- Problem *clustering*, meliputi cara menentukan subset-subset mana yang baik untuk suatu kumpulan obyek. Ini dilakukan untuk mengidentifikasikan kumpulan obyek dalam *class-class*.
- Problem *characterization*, menentukan *concept-concept* yang cocok untuk setiap obyek dalam *class.*

Tabel 2.1 Contoh pasangan *attribute-value.*

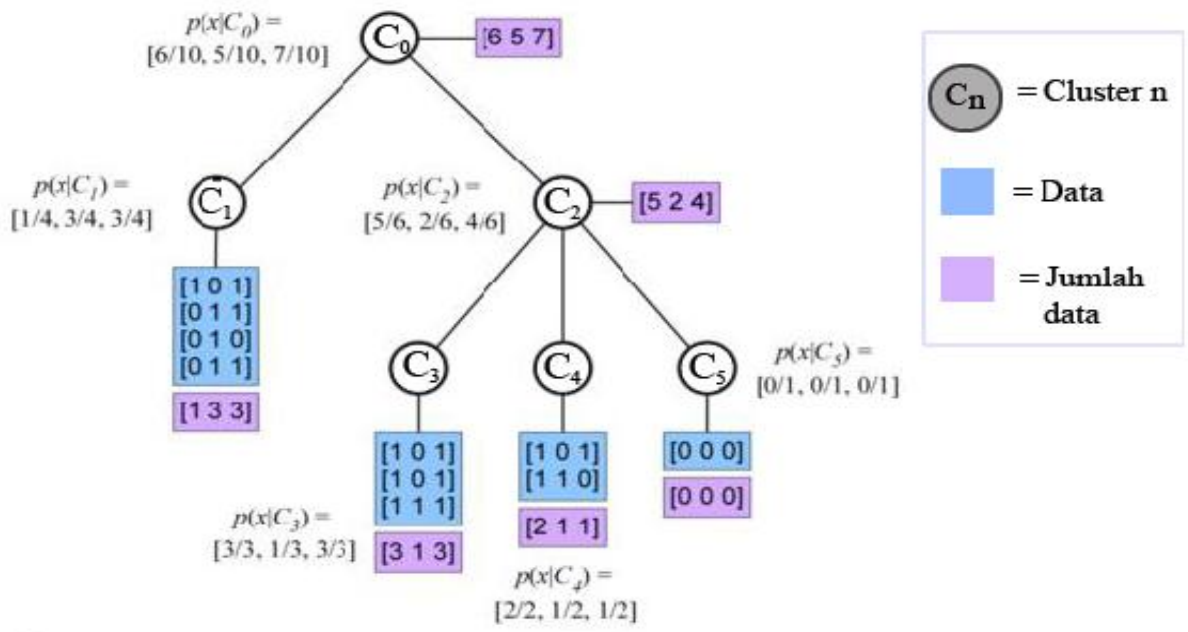
Attribute	Deskripsi	Nilai (*value*)
Nama	Nama orang	Rudi
Alamat rumah	Alamat orang	Jalan Sawo no.5
Pekerjaan	Pekerjaan orang tersebut saat ini	Wartawan

Contoh algoritma yang menggunakan metode *conceptual clustering* adalah *Cobweb*. Dalam algoritma *Cobweb*, *concept* yang digunakan adalah *Node* (Fisher, 1987). Setiap *concept* berisikan suatu himpunan obyek dengan propertinya. Pada gambar 2.1 akan ditampilkan contoh *tree* hasil observasi pada algoritma *Cobweb*. *Input*nya terdiri dari 10 buah data yaitu [0 0 0], [0 1 0], [0 1 1], [0 1 1], [1 0 1], [1 0 1], [1 0 1], [1 1 1], [1 0 1], [1 1 0]. Dari gambar tersebut, *cluster* C_1 terdapat 3 (tiga) buah obyek yaitu [1 0 1], [0 1 1], [0 1 0], [0 1 1]. Dengan menggunakan tabel 2.2 berarti *cluster* C_1 berisikan 1 hewan hidup di air, tidak berparu-paru, berdarah panas ([1 0 1]), 2 hewan yang tidak hidup di air, berparu-paru, berdarah panas ([0 1 1]), 1 hewan yang tidak hidup di air, berparu-paru, bukan berdarah panas ([0 1 0]).

Tabel 2.2 Tabel acuan *attribute-value* contoh *clustering*.

Attribute pertama	*Attribute* kedua	*Attribute* ketiga
Angka 1 berarti "hidup di air"	Angka 1 berarti "berparu-paru"	Angka 1 berarti "berdarah panas"
Angka 0 berarti "tidak hidup di air"	Angka 0 berarti "tidak berparu-paru"	Angka 0 berarti "bukan berdarah panas"

Cluster C_2 memiliki 3 buah anak (*child*) yaitu *cluster* C_3, C_4 dan C_5. *Cluster* C_2 memiliki semua obyek yang dimiliki anak-anaknya, yaitu sebanyak 6 obyek. Level 0 terdiri dari C_0 yang merupakan *root*. Level 1 terdiri dari C_1 dan C_2. *Cluster* pada level 1 lebih khusus dibandingkan *cluster* pada level 0. Hal ini dapat dilihat pada anggota *cluster* C_0 yang memiliki semua anggota dari C_1 dan C_2. Untuk penjelasan proses yang terjadi pada setiap pengelompokan obyek dapat dilihat pada subbab selanjutnya mengenai algoritma *Cobweb*.



Gambar 2.1. Konsep pada algoritma *Cobweb* berupa *node*.

2.4 Algoritma *Cobweb*

Algoritma *Cobweb* pertama kali diperkenalkan oleh Douglas Fisher (Fisher, 1987). Algoritma ini menggunakan metode *conceptual clustering* untuk mengorganisasikan data sehingga memaksimalkan kemampuan kesimpulan yang didapat. Algoritma *Cobweb* juga menerapkan *incremental concept* dalam memproses data sehingga setiap informasi (*knowledge*) pada *cluster* dapat diperbaharui terus-menerus. Penggunaan *incremental concept* dikarenakan algoritma *Cobweb* menggunakan strategi *hill-climbing* (Fisher, 1987). Jadi, algoritma *Cobweb* bertujuan untuk memaksimalkan informasi yang ingin didapat melalui prediksi dari informasi setiap anggota dalam sebuah *cluster* dan berlangsung terus-menerus. Tujuan tersebut diharapkan dicapai setiap ada obyek yang ingin dimasukan ke dalam *cluster* hingga obyek terakhir. Oleh karena itu, setiap *cluster* memiliki anggota-anggota yang telah diobservasi terlebih dahulu.

Dalam *clustering*, terkadang ada informasi atau data yang tidak pada *cluster* yang semestinya. Sebagai contoh ikan paus dalam *hierarchy* hewan, dapat digolongkan ke dalam golongan ikan jika dipandang *cluster* ikan beranggotakan hewan-hewan yang mempunyai sirip atau hidup di air. Namun jika dipandang dari cara bernapas dan cara berkembang biak, ikan paus dapat dimasukan ke dalam *cluster* yang lain, bukan *cluster* ikan. Dalam hal inilah informasi dinilai, apakah suatu obyek benar berada pada *cluster* tersebut atau tidak.

Dalam algoritma *Cobweb* ada beberapa hal yang diperlukan untuk mengimplementasi algoritma tersebut. Antara lain:

- Fungsi evaluasi heuristik. Fungsi ini digunakan untuk mengetahui pada *cluster* mana, suatu obyek disimpan.
- Representasi *state* (representasi *concept*) yang berguna untuk menentukan struktur *hierarchy* dan representasi *concept*.
- Operator yang digunakan untuk membangun skema klasifikasi.
- Strategi kontrol yang digunakan untuk mendeskripsikan algoritma *Cobweb*.

2.4.1 Fungsi evaluasi heuristik

Fungsi yang digunakan dalam *Cobweb* yaitu *category utility* dan *partition utility*. Kedua fungsi ini pertama kali diperkenalkan oleh Gluck dan Corter (Gluck, 1985) untuk mengevaluasi kegunaan suatu kategori atau partisi dengan menggunakan strategi perhitungan probabilitas.

Category utility adalah fungsi yang digunakan untuk memberikan nilai dan mengetahui kualitas suatu partisi (Marinchev, 2002). Suatu partisi *cluster* mempunyai nilai *category utility*. Fungsi ini dihitung dari *attribute-value* obyek-obyek yang disimpan dalam partisi *cluster*. Semakin tinggi nilai *category utility*, semakin baik kualitas dari partisi tersebut. Penjelasan mengenai *attribute-value* dapat dilihat pada subbab 2.2. Berikut ini adalah fungsi *category utility*

$$CU(C_k) = P(C_k)\sum_i\sum_j [P(A_i = V_{ij} \mid C_k)^2 - P(A_i = V_{ij})^2]$$

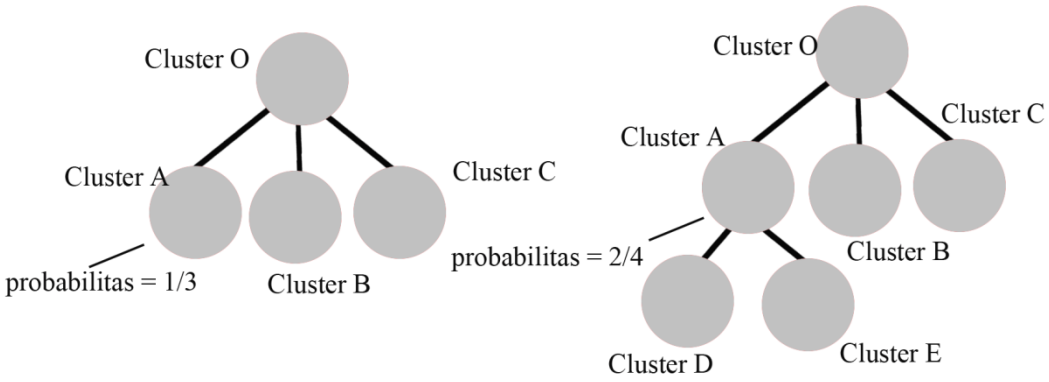
Jumlah yang diperoleh adalah jumlah untuk semua *attribute* A_i dan nilainya V_{ij} dalam sebuah *cluster*. Setiap *cluster* memiliki (Gluck, 1985):

- *Predictability*: probabilitas $P(A_i = V_{ij} \mid C_k)$ menandakan frekuensi kemunculan nilai V_{ij} pada *attribute* A_i dalam *cluster* C_k. $A_i = V_{ij}$ adalah pasangan *attribute-value*.
- *Predictiveness*: probabilitas $P(A_i = V_{ij})$ menandakan frekuensi kemunculan *attribute* A_i dengan nilai V_{ij} pada seluruh *cluster*.
- Probabilitas $P(C_k)$ menyatakan kemungkinan *cluster* C_k terhadap semua *cluster*.
- $CU(C_k)$ menyatakan *category utility cluster* C_k.

Probabilitas sebuah *cluster A*, $P(C_A)$ jika *cluster* di level atasnya memiliki 3 buah *cluster,* dan *cluster A* tersebut serta setiap *node* saudaranya, *cluster B* dan *cluster C,* tidak memiliki anak, maka *cluster* tersebut dan saudaranya memiliki probabilitas $P(C_k)$ sebesar 1/3. Namun jika *cluster A* memiliki 2 *cluster, cluster D*

Implementasi algoritma clustering..., Adolf Pandapotan

dan *cluster E*, dibawahnya yang merupakan level terbawah, sedangkan saudaranya tidak memiliki anak, maka probabilitas *node* tersebut sebesar 1/2. Untuk lebih jelasnya lihat Gambar 2.2.



Gambar 2.2. Probabilitas *node A* yang bukan *node* daun (*leaf*) dengan yang *node* daun (*leaf*).

Fungsi *CU* hanya digunakan pada sebuah partisi dari sebuah *cluster*, sedangkan untuk mengevaluasi beberapa partisi digunakan fungsi *PU*. Fungsi *PU* menghitung nilai rata-rata *CU* dari *cluster*.

Berikut adalah fungsi *PU*

$$PU(\{C_1, C_2, C_3, ..., C_n\}) = \frac{1}{n}\sum_{k}^{n} CU(C_k)$$

Keterangan

1. $PU(\{C_1, C_2, C_3, .., C_n\})$ menyatakan *partition utility* untuk himpunan partisi-partisi *cluster* C_1 hingga *cluster* C_n.
2. $CU(C_k)$ menyatakan *category utility cluster* C_k.

Sebagai contoh pada gambar 1, C_3, C_4, dan C_5 merupakan partisi dari *cluster* C_2 sehingga nilai *PU* dari *cluster* C_2 yaitu

$$PU(C_2) = PU(\{C_3, C_4, C_5\}) = \frac{CU(C_3) + CU(C_4) + CU(C_1)}{3}$$

Berikut contoh fungsi *CU*, misalkan *cluster O* memiliki 2 buah partisi yaitu *cluster* C_A dan *cluster* C_B. Partisi C_A memiliki 2 obyek dengan tiap obyeknya terdiri 3 *attribute* yaitu [1,3,2],[5,3,1]. Probabilitas partisi ini terhadap seluruh *cluster* sebesar 0,5. Obyek-obyek dalam *cluster* O yaitu [1,3,2],[1,1,4],[5,3,1],[2,2,2],[4,1,1]. Dengan menggunakan perhitungan berikut,

- *Attribute* pertama
  - angka 1 = $0,5^2$-$0,4^2$ = 0,09
  - angka 5 = $0,5^2$-$0,2^2$ = 0,21
  - angka 2 = 0^2-$0,2^2$ = -0,04
  - angka 4 = 0^2-$0,2^2$ = -0,04
- *Attribute* kedua
  - angka 3 = 1^2-$0,4^2$ = 0,84
  - angka 1 = 0^2-$0,4^2$ = -0,16
  - angka 2 = 0^2-$0,2^2$ = -0,04
- *Attribute* ketiga
  - angka 1 = $0,5^2$-$0,4^2$ = 0,09
  - angka 2 = $0,5^2$-$0,4^2$ = 0,09
  - angka 4 = 0^2-$0,2^2$ = -0,04

Nilai *CU* partisi C_A sebesar = 0,5 x (0,09+0,21+(-0,04)+(-0,04)+ 0,84+(-0,16)+(-0,04)+0,09+0,09+(-0,04)) = 1.

2.4.2 Representasi *state* (representasi *concept*)

Sebagai awal dari pembentukan klasifikasi *tree*, dibutuhkan suatu *concept*. Sebuah *concept* merepresentasikan sebuah *state*. Dalam *Cobweb*, suatu *concept* yang umum memiliki semua obyek dari *concept-concept* khusus yang ada dibawahnya. *Concept* juga harus memiliki kepekaan terhadap perubahan dari *attribute-value* yang dimilikinya. Selain itu, sebuah *concept* harus dapat

dievaluasi dengan fungsi *CU* dan *PU*. Oleh karena itu, dalam *Cobweb* sebuah *concept* direpresentasikan dengan *node*.

Untuk menghitung nilai dari probabilitas *attribute-value* sebuah *node*, diperlukan dua buah nilai yaitu jumlah kemunculan suatu *attribute-value* di *node* tersebut dan jumlah kemunculan diseluruh *node*. Sebagai contoh, suatu *concept* untuk *cluster* hewan yang hidup di air termasuk mamalia, dimana air merupakan *attribute-value*, maka *P*(air|mamalia), dihitung dengan

$$\frac{\text{jumlah hewan mamalia hidup di air yang telah diobsevasi}}{\text{jumlah hewan mamalia yang telah diobsevasi}}$$

2.4.3 Operator

Operator pada algoritma *Cobweb* terdapat 4 buah (Fisher, 1987), antara lain

- Memasukan obyek ke dalam *node*.
- Membuat *node* baru (*new node*).
- Mengkombinasikan 2 *node* menjadi 1 *node* (*merge node*).
- Membagi sebuah *node* menjadi beberapa *node* (*split node*).

Berikut akan dijelaskan mengenai operator-operator diatas.

a. Memasukan obyek ke dalam *node*

Obyek dimasukan ke dalam *node y*ang berada dilevel bawahnya. Proses ini dibagi dalam 2 hal yaitu dimasukan sementara dan dimasukan permanen. Obyek dimasukan sementara untuk menghitung nilai *PU*, hal ini digunakan untuk menentukan *node* mana yang terbaik menyimpan obyek tersebut. Sedangkan permanen berarti obyek disimpan secara tetap pada suatu *node*. Untuk lebih jelasnya mengenai pemasukan obyek sementara lihat gambar 2.3.

b. Membuat *node* baru (*new node*)

Dalam operator ini, sebuah *node* baru dibuat untuk menampung obyek yang baru. Dalam algoritma *Cobweb*, memasukan obyek ke dalam *node* baru untuk

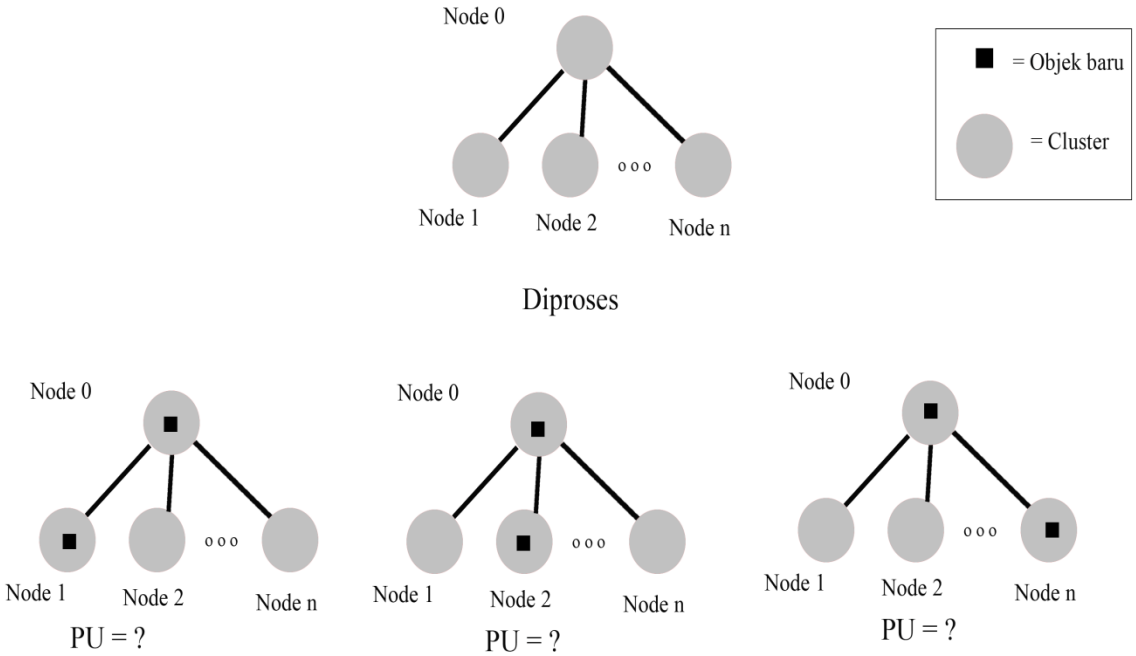
Implementasi algoritma clustering..., Adolf Pandapotan

sementara dilakukan untuk mendapatkan nilai *PU*. Gambar 2.4 adalah ilustrasi operator ini.

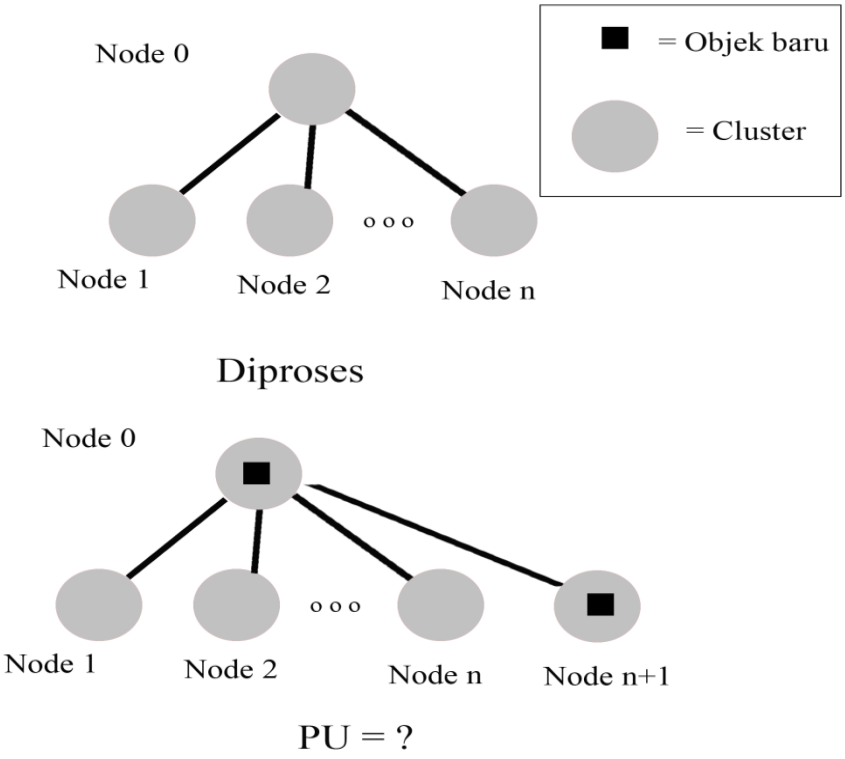
c. Mengkombinasikan 2 *node* menjadi 1 *node* (*merge node*)

Operator 1 dan 2 belum tentu dapat menanggulangi pemilihan obyek awal yang kurang baik, karena pemilihan obyek awal dapat mempengaruhi bentuk dari *hierarchy concept*. Oleh karena itu, alogritma *Cobweb* memiliki operator *merge node* dan *split node*. *Split node* akan dijelaskan selanjutnya.

Merge node menggabungkan 2 buah *node* menjadi 1 *node* dan menjumlah semua *attribute-value* yang dimiliki 2 *node* tersebut. Kedua buah *node y*ang dipilih adalah *node-node* anak yang menyebabkan nilai *PU node* tersebut paling besar ketika suatu obyek baru dimasukan ke dalam *node* anak tersebut. Pemilihan 2 *node* ini menggunakan operator "memasukan obyek ke dalam *node*". Penggabungan 2 *node* anak tersebut menyebabkan nilai *PU node* besar karena nilai *PU* diperoleh dengan mencari rata-rata dari 2 buah *node*. Penjelasan *PU* ada pada subbab 2.3.1. Gambar 2.5 adalah ilustrasi *merge node*.



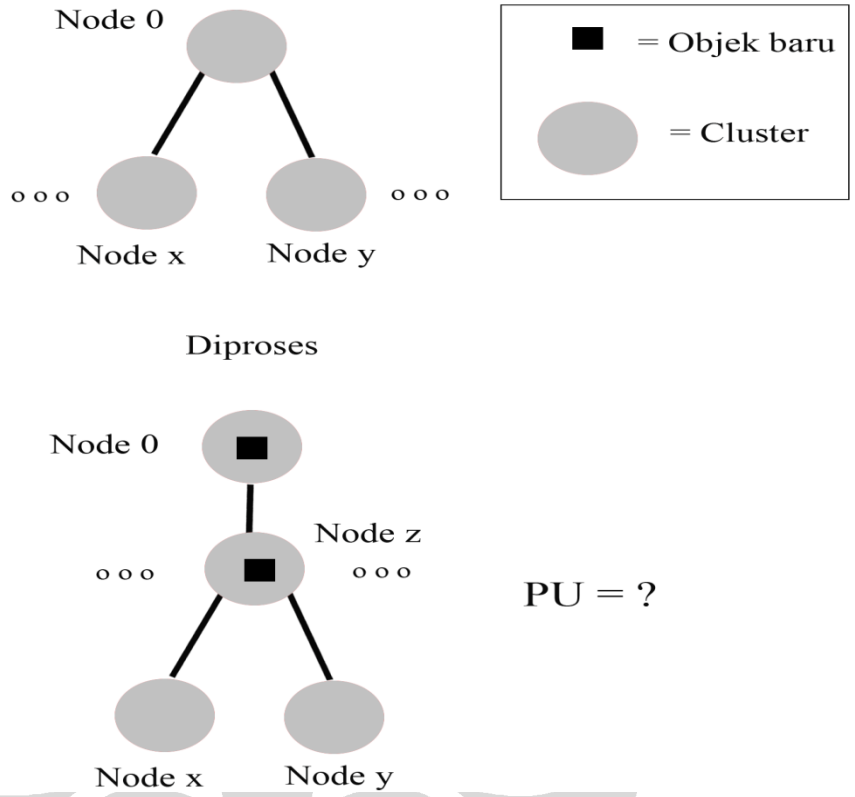
Gambar 2.3 Proses memasukan obyek sementara untuk memilih *node* mana yang terbaik.



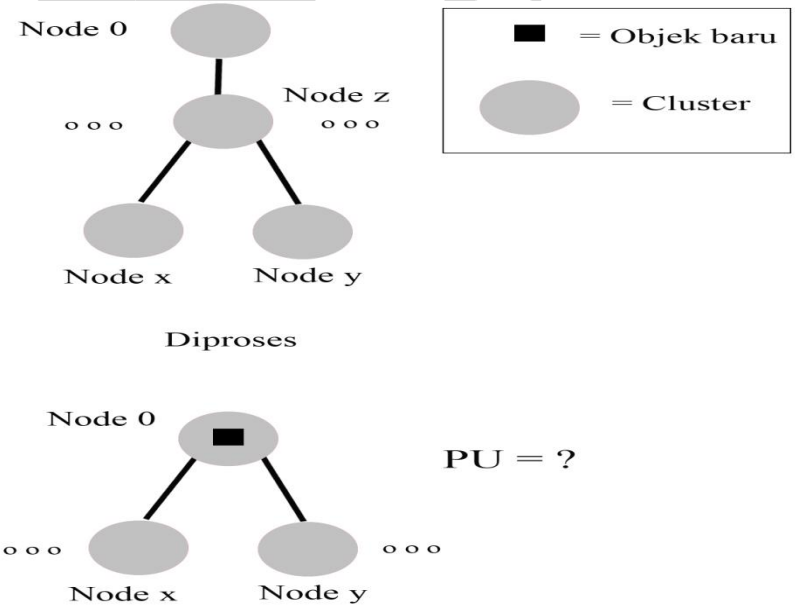
Gambar 2.4 Proses memasukan obyek ke dalam *node* baru yaitu *node n*+1.

d. Membagi sebuah *node* menjadi beberapa *node* (*split node*)

Sama seperti *merge node,* operator *split node* digunakan untuk menanggulangi pemilihan obyek awal. Pada operator *split node, node y*ang dipilih adalah *node* anak yang menyebabkan *node* tersebut memiliki nilai *PU* terbesar. Hasil dari proses *split* tersebut muncul *node-node y*ang sebelumnya anak dari *node* anaknya menjadi *node* anak yang baru. Sehingga jika sebuah *node* memiliki *n* buah anak dan *node* anak, yang menyebabkan *node* tersebut memiliki nilai *PU* terbesar, memiliki *m* buat anak, maka *node* tersebut memiliki (n-1+m) buah anak, Gambar 2.6 adalah ilustrasi dari operasi *split node*.



Gambar 2.5 Proses menggabungkan 2 buah *node* menjadi 1 *node* (*node z*).



Gambar 2.6 Proses *split node* terbaik, *node z*, menjadi beberapa *node*.

2.4.4 Strategi kontrol

Algoritma *Cobweb* memiliki strategi kontrol yang digunakan dalam tahap-tahap pembentuk *hierarchy concept*. Berikut ini akan ditampilkan strategi kontrol algoritma *Cobweb*.

Pseudo code Cobweb:

```
1  FUNCTION COBWEB (Object, Root (of a classification tree))
2  Update count of the root
3  IF Root is a leaf
4  THEN Return the expanded leaf to accommodate the new object
5  ELSE Find that child of Root that beat host Object and perform one of the
   following
6  Consider creating a new class and do so if appropriate
7  Consider node merging and so do if appropriate and call
   COBWEB(Object, Merge Node)
8  Consider node splitting and do so if appropriate and call
   COBWEB(Object, Root)
9  IF none of the above (a, b, or c) were performed
10 THEN call COBWEB(Object, Best child of the Root)
```

Pada inisial awal, obyek dimasukan ke dalam sebuah *node*. *Node* tersebut menjadi *root* dan berada pada level 0. Langkah pertama, FUNCTION COBWEB dipanggil dengan parameter *root* dan obyek baru yang akan di*cluster*. Kemudian pada nomor 3 dicek apakah *root* tersebut merupakan daun (*leaf*) atau tidak, jika *root* merupakan daun (*leaf*) maka daun diekspansi dengan cara memisahkan obyek-obyek yang ada pada parameter *root* ke dalam suatu *node* baru dan memasukan obyek baru, yang merupakan parameter, ke dalam *node* baru yang lain. Namun jika bukan daun, maka dilakukan perhitungan untuk menentukan dimana obyek tersebut disimpan. Pada nomor 5a dilakukan pertimbangan untuk menyimpan obyek tersebut pada *node* baru, jika keputusan ini terbaik, maka obyek disimpan pada *node* baru. Pada nomor 5b dilakukan pertimbangan untuk menyimpan obyek pada *node y*ang merupakan hasil penggabungan 2 *node* terbaik,

jika keputusan ini terbaik, maka obyek disimpan pada *node* gabungan tersebut dan dilakukan iterasi selanjutnya dengan parameter *node* gabungan dan obyek yang sama. Pada nomor 5c dilakukan pertimbangan untuk memecah *node* terbaik sebelum memutuskan kemana obyek tersebut disimpan, jika keputusan ini tepat, maka dilakukan iterasi dengan parameter *root* dan obyek yang sama.

Namun jika tidak ada keputusan terbaik dari nomor 5a hingga 5c, maka obyek disimpan pada *node* terbaik lalu dilakukan iterasi selanjutnya dengan parameter *node* terbaik dan obyek yang sama. Proses ini berlangsung iteratif hingga obyek tersebut berada pada level terbawah dari klasifikasi *tree*. Setiap keputusan ditentukan oleh nilai *PU* terbesar. Berikut akan diilustrasikan contoh proses algoritma *Cobweb*.

Tabel 2.3 Tabel acuan *attribute-value*.

Nama *attribute*	Deskripsi	*Value*
Berparu-paru	Memiliki paru-paru atau tidak	1 (tidak memiliki), 2 (memiliki)
Hidup di air, hidup di darat, hidup di udara	Tempat hidup	1 (air), 2 (darat), 3 (udara)
Jenis makanan	Jenis berdasarkan makanannya	1 (herbivora), 2 (karnivora) atau 3 (omnivora)

Obyek-obyek yang ingin dimasukan yaitu

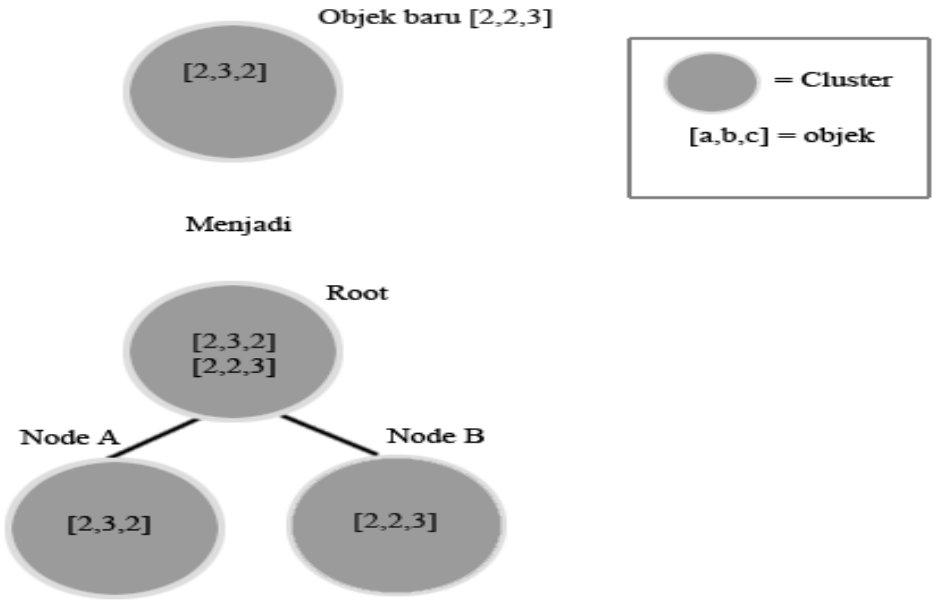
- Burung elang [berparu-paru, hidup di udara, karnivora] berdasarkan tabel 2.3 yaitu [2,3,2].
- Manusia [berparu-paru, hidup di darat, omnivora] berdasarkan tabel 2.3 yaitu [2,2,3].

- Ikan paus [berparu-paru, hidup di air, karnivora] berdasarkan tabel 2.3 yaitu [2,1,2].
- Burung hantu [berparu-paru,hidup di udara, karnivora] berdasarkan tabel 2.3 yaitu [2,3,2].
- Ikan piranha [tidak berparu-paru, hidup di air, karnivora] berdasarkan tabel 2.3 yaitu [1,1,2].

Langkah pertama dalam *Cobweb* yaitu memasukan obyek pertama, burung elang, ke sebuah *node*, *node* ini akan menjadi *root.* Berikutnya akan dijelaskan per tahap untuk 4 obyek lainnya.

a. Memasukan obyek manusia [2,2,3]

FUNCTION COBWEB dipanggil dengan parameter obyek selanjutnya, manusia, dan *root*. *Root* diperiksa apakah sebuah *leaf* (daun) atau tidak, karena *root* merupakan sebuah daun, maka dilakukan proses *expanded leaf* (lihat nomor 3 pada *pseudo code Cobweb*). Hasil dari iterasi ini yaitu *node A* dengan obyek burung elang [2,3,2] dan *node B* dengan obyek manusia [2,2,3]. Lebih jelasnya lihat gambar 2.7.



Gambar 2.7 *Root* mengalami proses *expanded leaf.*

Root yang lama memiliki anggota burung elang [2,3,2]. Sedangkan *root* yang baru memiliki anggota manusia [2,2,3] dan burung elang [2,3,2]. *Root* memiliki semua anggota karena *root* merupakan *cluster* yang paling umum.

b. Memasukan obyek ikan paus [2,1,2]

Pada iterasi pertama, dimasukan obyek ikan paus [2,1,2] pada FUNCTION COBWEB dengan parameter *node root*. Karena *root* bukan daun, maka dilakukan langkah pada *pseudo code* nomor 5. Langkah pertama yaitu menentukan *node* anak yang menyebabkan *root* memiliki nilai *PU* tertinggi. Didapat 2 buah *node* anak yaitu *node A* dan *node B*. *Node A* yaitu *node y*ang memiliki obyek burung elang [2,3,2], sedangkan *node B* memiliki obyek manusia [2,2,3].

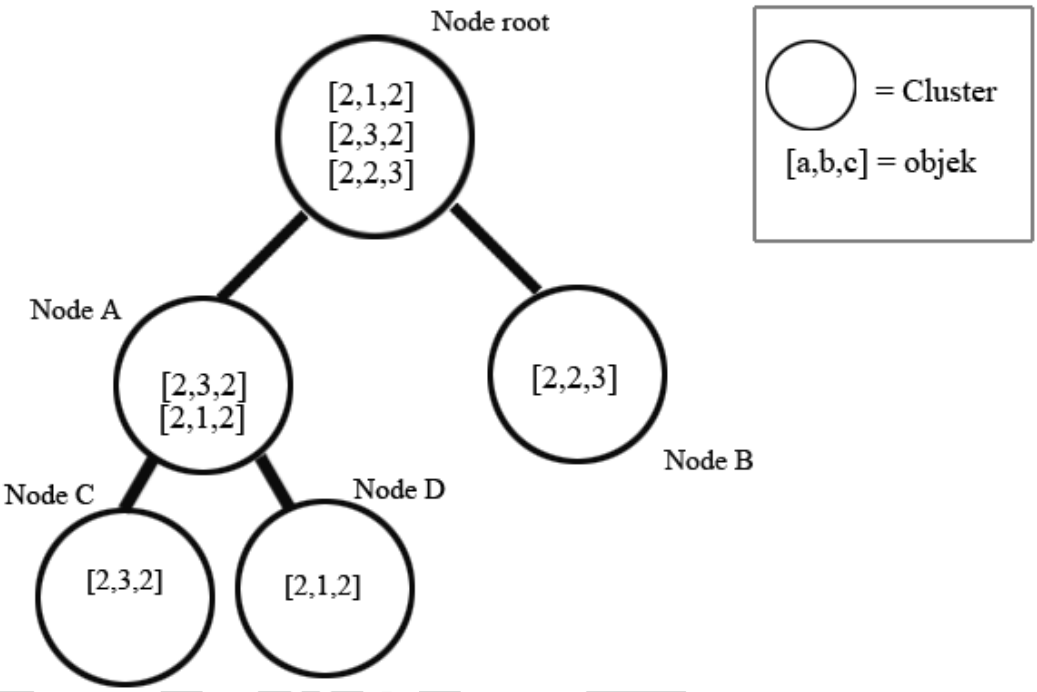
Dengan menggunakan fungsi *CU* pada subbab 2.3.1 didapat nilai *CU node A* dan *node B* jika dimasukan obyek ikan paus berturut-turut sebesar 0,407 dan 0,074. Dan nilai *CU node A* dan *node B* jika tidak dimasukan obyek ikan paus berturut-turut sebesar 0,3703 dan 0,3703. Sehingga nilai *PU node root* jika obyek ikan paus dimasukan ke *node A* lebih besar obyek ikan paus dimasukan ke dalam *node B* yaitu 0,3889. Maka *node A* yang dipilih, lalu dilakukan perhitungan untuk menentukan nilai *PU* terbesar untuk *node root*.

Dari perhitungan didapat bahwa

- Jika memasukan obyek ikan paus ke dalam *node A*, maka nilai *PU node root* sebesar 0,3889.
- Jika dibuat *node* baru, maka nilai *PU* untuk *node root* sebesar 0,37.
- Jika menggabungkan *node A* dan *node B* (*merge node*), maka nilai *PU* untuk *node root* sebesar 0.
- Proses *split* tidak dapat dilakukan karena *node B* tidak memiliki anak.

Dari 4 kemungkinan diatas, maka yang paling baik adalah memasukan obyek ikan paus ke dalam *node A*. *Node root* dimasukan juga obyek ikan paus. Kemudian dilakukan iterasi selanjutnya terhadap *node A*. Karena *node A*

merupakan daun, maka dilakukan *expanded leaf*. Sehingga dibuat *node* baru, misalkan, *node C*, yang berisi obyek burung elang [2,3,2] dan *node* baru lain, misalkan *node D*, berisikan obyek ikan paus [2,1,2]. Gambar 2.8 merupakan gambar hasil pemasukan obyek ikan paus.



Gambar 2.8 Hasil pemasukan obyek ikan paus [2,1,2].

c. Memasukan obyek burung hantu [2,3,2]

Kemudian obyek burung hantu [2,3,2] diproses. Pada iterasi pertama, FUNCTION COBWEB dipanggil dengan parameter *root* dan obyek burung hantu. Lalu dilakukan perhitungan nilai *PU*. Dari perhitungan didapat bahwa

- *Node* terbaik yaitu *node A*, terbaik kedua yaitu *node B*.
- Jika memasukan obyek burung hantu ke dalam *node A*, maka nilai *PU node root* sebesar 0,3333.
- Jika dibuat *node* baru, maka nilai *PU* untuk *node root* sebesar 0,25.

- Jika menggabungkan *node A* dan *node B* (*merge node*), maka nilai *PU* untuk *node root* sebesar 0.
- Proses *split node A* maka nilai *PU node root* sebesar 0,25.

Dari 4 kemungkinan diatas, maka yang paling baik adalah memasukan ke dalam *node A*. Obyek burung hantu juga dimasukan ke dalam *node root*. Iterasi selanjutnya dipanggil FUNCTION COBWEB dengan parameter *node A* dan obyek burung hantu. Dari perhitungan didapat bahwa

- *Node* terbaik yaitu *node C*, terbaik kedua yaitu *node D*.
- Jika memasukan obyek burung hantu ke dalam *node C*, maka nilai *PU node root* sebesar 0,222.
- Jika dibuat *node* baru, maka nilai *PU* untuk *node A* sebesar 0,148.
- Jika menggabungkan *node C* dan *node D* (*merge node*), maka nilai *PU* untuk *node A* sebesar 0.
- Proses *split node C* tidak bisa karena tidak memiliki anak.

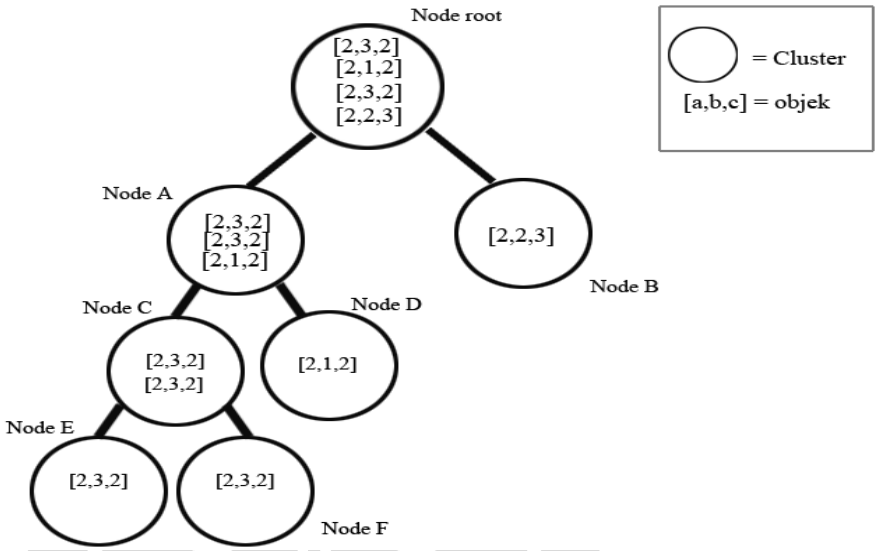
Dari 4 kemungkinan diatas maka obyek burung hantu dimasukan ke dalam *node C* dan *node A*. Selanjutnya karena *node C* adalah *node* daun, maka dilakukan *expanded leaf*. Dibuat *node* baru, misalkan *node E*, berisi obyek burung elang [2,3,2] dan *node* baru lainnya, misalkan *node F*, berisi obyek burung hantu [2,3,2]. Hasil akhir dari pemasukan burung hantu dapat dilihat pada gambar 2.9.

d. Memasukan obyek ikan piranha [1,1,2]

Lalu obyek ikan piranha [1,1,2] akan diproses. Dipanggil FUNCTION COBWEB dengan parameter *root* dan obyek ikan piranha. Karena *root* bukan daun, maka dilakukan langkah pada nomor 5. Didapat bahwa

- *Node* terbaik yaitu *node A*, terbaik kedua yaitu *node B*.
- Jika memasukan obyek ikan piranha ke dalam *node A*, maka nilai *PU node root* sebesar 0,289

- Jika dibuat *node* baru, maka nilai *PU* untuk *node root* sebesar 0,337.



Gambar 2.9 Proses pemasukan obyek burung hantu [2,3,2].

- Jika menggabungkan *node A* dan *node B* (*merge node*) maka nilai *PU root* sebesar 0.
- Proses *split node A* menghasilkan nilai *PU node root* sebesar 0,3413.

Dari 4 kemungkinan tersebut, maka yang paling baik adalah melakukan *split* pada *node A*. *Node A* dibuang dari *tree*. *Node-node* anak *node A* menjadi *node-node* anak *node root*. Iterasi kedua dipanggil FUNCTION COBWEB dengan parameter *node root* dan obyek ikan piranha.

Didapat bahwa

- *Node* terbaik yaitu *node D*, terbaik kedua yaitu *node C*.
- Jika memasukan obyek ikan piranha ke dalam *node D*, maka nilai *PU node root* sebesar 0,359.
- Jika dibuat *node* baru, maka nilai *PU* untuk *node root* sebesar 0,319
- Jika menggabungkan *node C* dan *node D* (*merge node*) maka nilai *PU root* sebesar 0,2899.
- Proses *split node D* tidak bisa dilakukan.

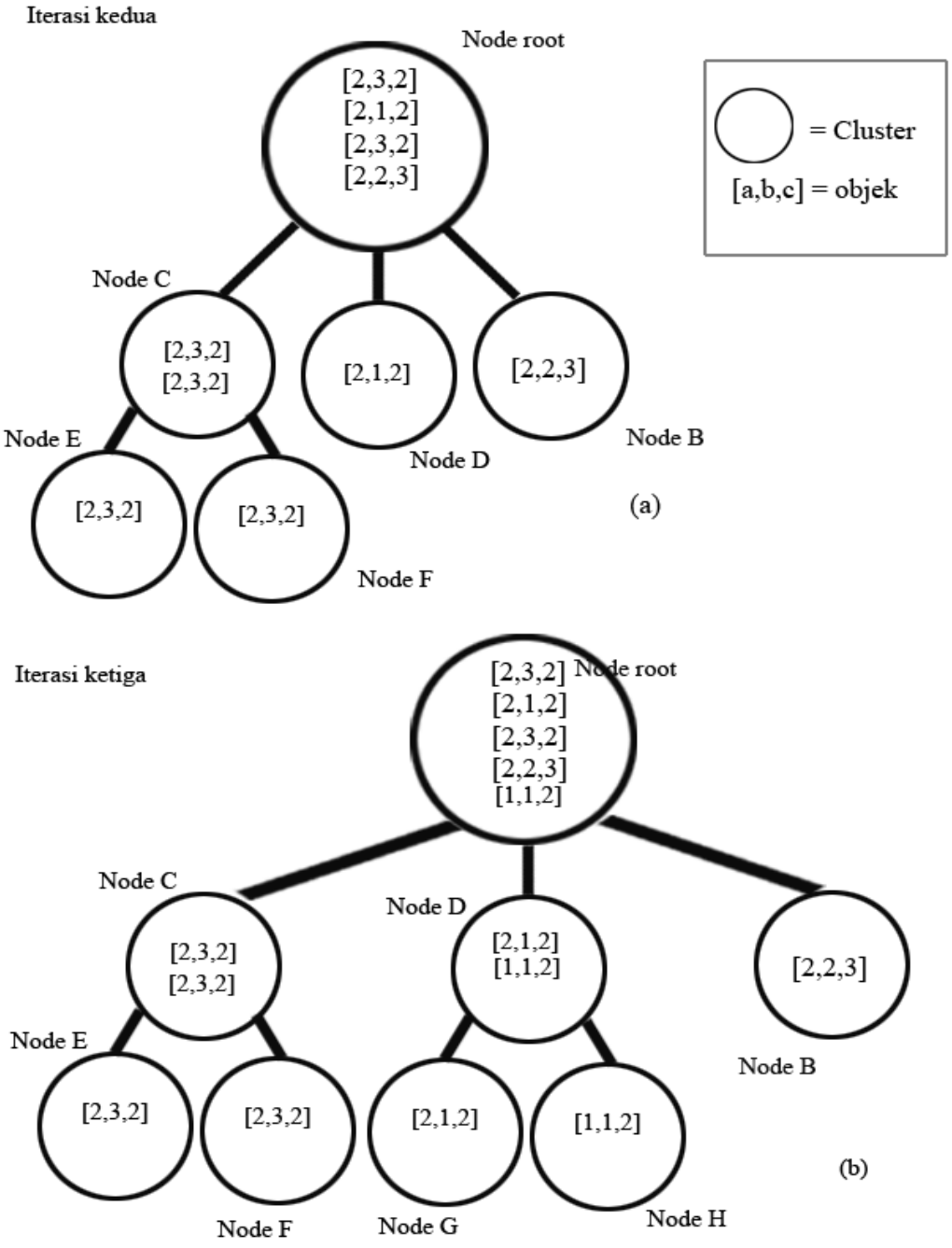
Dari keempat kemungkinan tersebut, yang terbaik yaitu memasukan obyek ikan piranha ke *node D*. Obyek ikan piranha dimasukan ke dalam *node D* dan *node root*. Iterasi selanjutnya dipanggil FUNCTION COBWEB dengan parameter *node D* dan obyek ikan piranha, karena *node D* merupakan *node* daun, makadilakukan *expanded leaf. Node* baru, *node G*, berisikan obyek ikan paus, sedangkan *node* baru lainya, *node H*, berisikan obyek ikan piranha. Hasil akhirnya ada pada gambar 2.10.

Hingga hasil akhir dari memasukan kelima obyek tersebut dihasilkan *hirarchy concept* seperti pada gambar 2.10b. *Node-node y*ang dihasilkan yaitu

- *Node root* yang beranggotakan semua obyek.
- *Node B* yang beranggotakan manusia [2,2,3].
- *Node C* yang beranggotakan burung elang [2,3,2] dan burung hantu [2,3,2].
- *Node D* yang beranggotakan ikan paus [2,1,2] dan ikan piranha [1,1,2].
- *Node E* yang beranggotakan burung elang [2,3,2].
- *Node F* yang beranggotakan burung hantu [2,3,2].
- *Node G* yang beranggotakan ikan paus [2,1,2].
- *Node H* yang beranggotakan ikan piranha [1,1,2].

Dari hasil diatas, terlihat bahwa ikan paus dan ikan piranha sama-sama terdapat pada *cluster D*. Kedua obyek ini digolongkan pada golongan mahluk hidup yang hidup di air dan karnivora [_,1,2]. Sedangkan burung elang dan burung hantu tergolong mahluk hidup yang hidup yang memiliki paru-paru, hidup di udara dan karnivora. Manusia dikelompokan pada *cluster* yang berbeda dengan mahluk hidup yang lain.

Dari penjelasan per tahap diatas dapat dilihat bahwa dalam tiap tahapnya dilakukan pengambilan keputusan yang terbaik sehingga mendapatkan hasil akhir yang baik (*hill climbing search*).



Gambar 2.10 Hasil pemasukan obyek ikan piranha [1,1,2].

2.5 Algoritma *Iterate*

Algoritma *Iterate* diperkenalkan oleh Gustam Biswas, Jerry B. Weiberg dan C. Li pada tahun 1995 (Biswas, Weinberg, Li, 1995). Algoritma ini merupakan salah satu algoritma *clustering* dalam *data mining*. Dalam algoritma *Iterate*, penemuan suatu bentuk data menggunakan prediksi. Algoritma *Iterate* berusaha

mencari bentuk pola tersebut yang memungkinkan informasi akan data didapat dari sekumpulan data. Secara umum, algoritma ini hampir mirip dengan *Cobweb*, namun *Cobweb* dipengaruhi oleh urutan dari data, sedangkan *Iterate* tidak dipengaruhi urutan. Perbedaan urutan pada *Cobweb* menyebabkan klasifikasi *tree* yang terbentuk juga berbeda. Algoritma *Cobweb* berfokus pada memaksimalkan prediksi dari sebuah *cluster* sehingga klasifikasi *tree* yang dihasilkan terkadang tidak sesuai dengan yang diharapkan. Sedangkan algoritma *Iterate* menekankan pada menghasilkan kestabilan partisi, memaksimalkan perbedaan partisinya, dan menghasilkan sebuah algoritma yang efisien untuk *data analysis tool* (Biswas, Weinberg, Li, 1995).

Bagian terpenting dalam *Iterate* adalah bagaimana merepresentasikan dan menghasilkan pola-pola dengan membuat grup-grup yang berisikan kumpulan data yang menggambarkan keterkaitan data-data dalam satu grup, namun mempunyai perbedaan yang jelas dengan data-data pada grup yang berbeda.

Data dalam algoritma *Iterate* direpresentasikan sebagai *feature-value pair*, dimana *feature* yang dimaksud merupakan properti dari sebuah obyek. Sebagai contoh, jika obyek adalah sebuah mobil, maka *feature* dari mobil antara lain berat mobil, kecepatan maksimal, dan bentuk mobil. *Feature-value pair* ini sama sepeti *attribute-value pair* dalam algoritma *Cobweb*.

Dalam algoritma *Iterate*, ada 2 ukuran kualitas yang digunakan, yaitu

- *Concept distinctness* atau *inter-class dissimilarity*. Perbedaan antara 2 *concept* diukur dari perbedaan dari deskripsi 2 *class* tersebut. Semakin besar perbedaan antara 2 buah *concept* semakin baik karena pada hasil akhir algorima *Iterate* kualitasnya juga semakin baik.
- *Cohesion* atau *intra-simmilarity.* Ini digunakan untuk mengetahui seberapa baik pengaruh sebuah obyek jika dimasukan ke dalam suatu *class.* Semakin besar *cohesion* dari suatu *class*, maka semakin baik.

Untuk mengevaluasi sebuah partisi dari sebuah *cluster* dalam algoritma *Iterate*, dibutuhkan fungsi *category utility* karena dapat menandakan perubahan

antara *intra-cluster similarity* dan *inter-cluster dissimilarity* (Biswas, Weinberg, Fisher, 1995). Dari fungsi *CU* pada subbab 2.3, peningkatan probabilitas *feature-value* yang dapat diperkirakan pada sebuah *cluster* C_k ($P(A_i=V_{ij} \mid C_k)^2$) yang melebihi jumlah peningkatan probabilitas ($P(A_i = V_{ij})^2$) menunjukan bahwa hubungan antara obyek-obyeknya semakin erat.

Selain itu untuk mengetahui perbedaan antara 2 buah *cluster* (*inter-dissimilarity*) pada *Iterate* digunakan fungsi *variance of distribution match* (Biswas, Weinberg, Fisher, 1995). Makin besar nilai *dissimilarity* antara 2 *cluster*, maka kedua *cluster* tersebut makin berbeda. Berikut nilai *dissimilarity* antara 2 buah *cluster* yaitu *cluster K* dan *cluster L*.

$$\frac{1}{n}\sum_{i}^{n}\sum_{j}[P(A_i = V_{ij} \mid C_k) - P(A_i = V_{ij} \mid C_l)]^2$$

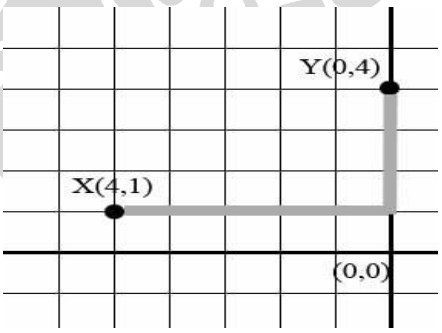
Keterangan

- n = jumlah *attribute*
- $P(A_i = V_{ij} \mid C_k)$ menandakan frekuensi kemunculan nilai V_{ij} pada *attribute* A_i dalam *cluster* C_k. $A_i = V_{ij}$ adalah pasangan *attribute-value*.
- $P(A_i = V_{ij} \mid C_l)$ menandakan frekuensi kemunculan nilai V_{ij} pada *attribute* A_i dalam *cluster* C_l. $A_i = V_{ij}$ adalah pasangan *attribute-value*.

Hasil akhir dari algoritma ini adalah sebuah *hirarchy* dari *concept*, dimana setiap *concept* direpresentasikan oleh sebuah *node*. Setiap *node* memiliki obyek-obyek dimana obyek yang akan dimasukan ke dalam sebuah *node* tergantung pada *bias* ketika obyek tersebut dimasukan ke dalam *node* tersebut. *Bias* adalah semua faktor yang mempengaruhi pembentukan sebuah *cluster* (Biswas, Weinberg, Fisher, 1995). Faktor-faktor ini termasuk batas perkiraan dan algoritma yang digunakan untuk membentuk deskripsi dari suatu *concept*. Oleh karena itu

dibutuhkan suatu *criteria function* untuk membentuk *concept-concept* tersebut. Dan *criteria function* tersebut adalah *category utility* dan *partition utility*.

Dalam algoritma *Cobweb* terdapat 4 buah operator yang digunakan untuk mengetahui kemungkinan dimana sebuah obyek disimpan. Hasil dari algoritma *Cobweb* adalah suatu *hirarchy concept* dalam bentuk klasifikasi *tree*. Sedangkan dalam algoritma *Iterate* terdapat 2 buah operator kontrol yaitu memasukan obyek ke dalam *node* dan membuat *node* baru. Namun selain kedua oprator tersebut, *Iterate* menggunakan fungsi *Anchored Dissimilarity Ordering* (ADO) dan *Iterative Redistribution*. Dua bagian ini digunakan untuk meningkatkan *cohesion* dan *distinction* pada *cluster*. Fungsi ADO digunakan untuk mengurutkan obyek-obyek berdasarkan nilai *similarity* yang dimilikinya terhadap obyek yang lain. Untuk menghitung *similarity* antara kedua obyek tersebut menggunakan Manhattan *distance.* Sebagai contoh jika ada dua buah titik $p_1[x_1,y_1]$ dan p_2 $[x_2,y_2]$, maka nilai Manhattan distancenya sebesar $|x_1 - x_2|+|y_1 - y_2|$(Black, 2006). Jika 2 titik tersebut dianggap sebagai 2 obyek, maka semakin jauh suatu obyek dengan obyek lain, maka semakin besar nilai Manhattan *distance* antara dua obyek tersebut. Contohnya jika ada dua buah obyek $p_1[a_1,b_1,c_1]$ dan $p_2[a_2,b_2,c_2]$, maka nilai Manhattan distancenya sebesar $|a_1 - a_2|+|b_1 - b_2|+|c_1 - c_2|$. Pada gambar 2.11 memperlihatkan Manhattan distance titik x ke titik y sebesar |4-0|+|1-4| = 7.



Gambar 2.11 Manhattan distance antara dua buah titik x dan y adalah 7.

Dalam algoritma *Cobweb* hal tersebut tidak perhatikan. Oleh karena itu perbedaan urutan dari kumpulan obyek yang ingin diobservasi menentukan bentuk dari *hirarchy concept* pada klasifikasi *tree*. Operator pada algoritma

Cobweb menyebabkan obyek pertama memiliki *cluster* yang berbeda dengan obyek yang kedua, walaupun obyek yang kedua memiliki nilai Manhattan *distance* yang kecil (nilai *similarity* besar).

Sedangkan ide *iterative redistribution* adalah obyek dipisahkan dari *cluster* sebenarnya kemudian diobservasi kembali terhadap *cluster-cluster* lain. Untuk penjelasan lebih lengkapnya lihat tahap 3 dari algoritma *Iterate*.

Algoritma *Iterate* terbagi dalam 3 tahap, yaitu

- Pembentukan klasifikasi *tree* dengan menggunakan fungsi *CU* (*classified*).
- Pengumpulan partisi awal yang baik dari klasifikasi *tree* untuk membentuk *cluster-cluster* yang diinginkan (*extract*).
- Proses iterasi terhadap obyek-obyek untuk memaksimalkan pemisahan antara *cluster* (*iterative redistribution*).

2.5.1 Pembentukan klasifikasi *tree* (*classified*)

Algoritma pembentukan klasifikasi *tree* sebagai berikut

Initialize: Set $\boldsymbol{L} = N_1$,

O_1 = set of data objects to be *cluster*ed

Loop till ***L*** empty

Get first element from ***L*** , say N_k

If O_k contains more than one object

Sort objects according to anchored dissmililarity ordering (ADO) scheme

Loop for every object in O_k

If first objects in O_k

Create new *node* N_{k+c} as child of N_k

Place object in *node* N_{k+c}

Else:

Place object in all child *node* one by one and compute *partition* scores (*PU*) for each one

Place object as new child *node* under N_k and compute *partition* score

assign object to *node* for which the *partition* score is highest, and update $A_i = V_{ij}$ count for *node*

End Loop

Place new children in ***L***

End If

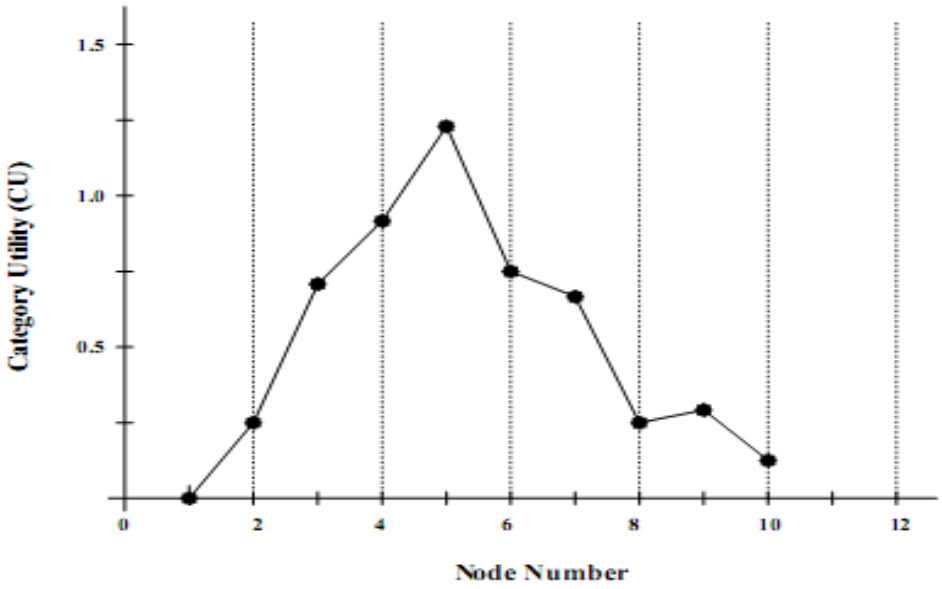
End Loop

Keterangan:

- N_i: *Node i* dalam klasifikasi *tree.*
- O_i: Kumpulan obyek dalam *node* N_i.
- N_1: Root dari klasifikasi *tree.*

O_1: Kumpulan obyek pada *root*, merupakan kumpulan obyek yang ingin di*cluster*.

Tree terbentuk pada tahap ini secara *breadth-first*, level per level mulai dari level atas (level 0) hingga level terbawa. Hasil dari tahap pertama ini yaitu klasifikasi *tree* yang berbentuk *hirarchy concept*.



Gambar 2.12 Grafik nilai *CU* sepanjang penelusuran *node* di tiap levelnya.

2.5.2 Pengumpulan partisi awal yang "baik" dari klasifikasi *tree* untuk membentuk *cluster-cluster* yang diinginkan (*extract*)

Langkah kedua ini digunakan untuk mencari *node-node* pada *tree* yang mempunyai nilai *CU* lebih rendah dari pada *node* orang tuanya. *Node-node* tersebut kemudian dikumpulkan agar dapat dilakukan perbaikan terhadap nilai *CU node-node* tersebut. Hasil dari langkah pertama yaitu klasifikasi *tree*, memiliki grafik nilai *CU* naik hingga pada level tertentu kemudian pada level selanjutnya, nilai *CU* menurun.

Pada gambar 2.12 ditampilkan grafik nilai *CU* sepanjang penelusuran *node* tiap levelnya. Setiap *node y*ang "baik" membentuk suatu grup dan dikumpulkan dalam *initial partition.*

Hasil akhir dari tahap ini yaitu kumpulan grup dimana *node* level teratasnya memiliki nilai *CU* rendah dan akan diperbaiki pada tahap ketiga.

Berikut ini algoritma untuk langkah kedua

Initialize: set *node N = root*

Set list = {*N*}

Loop till *list* empty

Get *children* (*N*)

If $CU_N > CU_C$, $\forall$ *c* ∈ *children* (*N*)

N defines a group in the initial partition (i)

Else

$\forall$ *c* such that *c* ∈ *children* (*N*) and $CU_N \leq CU_C$

add *c* to *list* (ii)

$\forall$ *c* such that *c* ∈ *children* (*N*) and $CU_N > CU_C$

Make *c* a group in initial partition (iii)

End If

Remove *N* from *list*

Set *N* = first elemen of *list*

End Loop

Pada algoritma diatas, dilakukan pengecekan nilai *CU* mulai dari anak-anak dari *root*. Di (i), setiap *node y*ang memiliki *node* anak dengan nilai *CU* lebih kecil dari *node* tersebut,maka *node* tersebut membentuk grup dan disimpan dalam *initial partition* sebagai sub-*tree*. Pada (ii), jika suatu *node* ternyata memiliki nilai *CU* lebih rendah daripada nilai *CU node* orang tuanya, maka *node* itu membentuk grup dan disimpan sebagai *node* akan diproses selanjutnya. Pada (iii), tiap *node*, yang akan diperbaiki nilai *CU*nya, disimpan sebagai sub-*tree* dalam *initial*

partition sehingga *node-node* anaknya tidak dicek karena nilai *CU node* anaknya berdasarkan gambar 2.12 memiliki nilai lebih rendah dari nilai *CU node* tersebut. *Node-node* pada sub-*tree* inilah yang merupakan anggota dari grup yang dibentuk oleh *node* orang tuanya.

2.5.3 Iterative redistribution

Seperti yang sudah dijelaskan sebelumnya, langkah ini dilakukan pada *node-node* dari langkah kedua yang memiliki nilai *CU* lebih rendah dari *node* orang tuanya dan *node y*ang nilai *CU*nya lebih besar dari semua nilai *CU node-node* anaknya. Iterasi ini dilakukan dengan *iteration redistribution*, namun tidak dilakukan pada *node*, melainkan pada setiap obyek di *node* tersebut. Proses iterasi tiap obyek menggunakan fungsi *category match measure* (*CM*). Berikut fungsi dari *CM*

$$CM_{dk} = P(C_k) \sum_{i,j \in \{A_i\}_d} (P(A_i = V_{ij} \mid C_k)^2 - P(A_i = V_{ij})^2)$$

Keterangan:

- CM_{dk} = *Category Match Measure*, nilai *CU* ketika sebuah obyek *d* disimpan dalam *node* C_k.
- $\{A_i\}_d$ = *Attribute i* pada *node* C_k ketika obyek d disimpan.
- $P(A_i = V_{ij} \mid C_k)$ menandakan frekuensi kemunculan nilai V_{ij} pada *attribute* A_i dalam *cluster* C_k. $A_i = V_{ij}$ adalah pasangan *attribute-value*.
- $P(A_i = V_{ij})$ menandakan frekuensi kemunculan *attribute* A_i dengan nilai V_{ij} pada seluruh *cluster*.

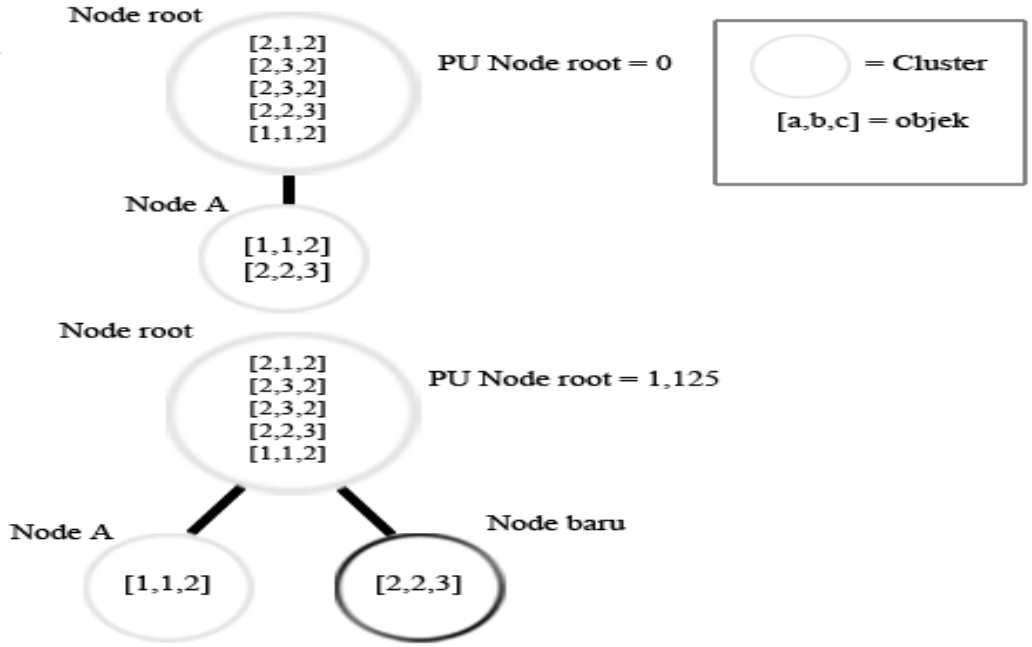
Untuk setiap obyek *d* pada *node* disuatu grup yang ada di *initial partition*, dimasukan ke dalam *node* di grup lain lalu dihitung nilai *CU*nya. *Node* di grup lain yang memiliki nilai *CU* terbesar akan menyimpan obyek *d* tersebut. C_k adalah setiap *node y*ang membentuk grup di *initial partition*. *Iterative redistribution* ini dilakukan hingga semua obyek pada grup-grup *initial partition* selesai dilakukan.

Berikut adalah contoh algoritma *Iterate* dengan menggunakan obyek-obyek ikan paus [2,1,2], burung elang [2,3,2], manusia [2,2,3], burung hantu [2,3,2], ikan piranha [1,1,2].

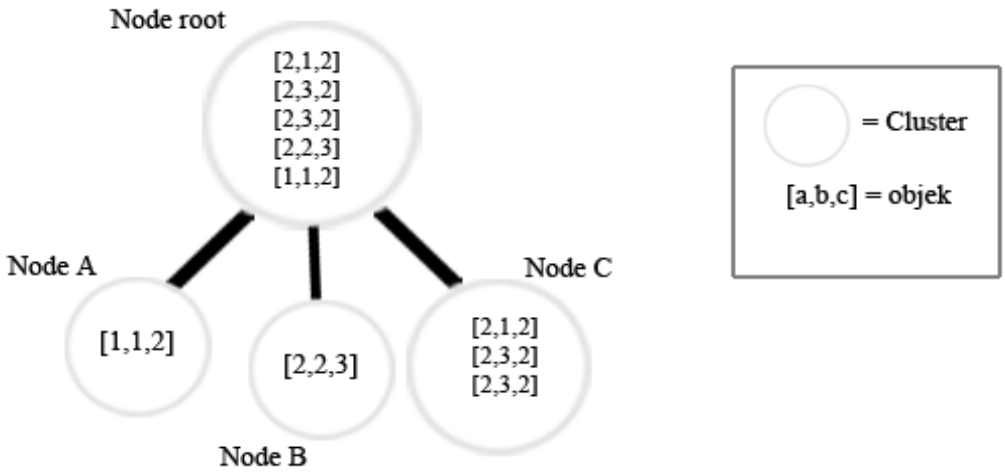
Pada iterasi pertama, *node root* dibuat dengan jumlah *attribute* 3. Semua obyek dimasukan ke dalam *node root*. Obyek-obyek tersebut hanya disimpan saja dan tidak memperbaharui *attribute-attribute node root*.

- ***Node root***
  - Obyek-obyek pada *root* diurutkan menggunakan ADO. Urutan dari kelima obyek mulai dari obyek yang terdekat dengan yang lain, yaitu ikan paus [2,1,2], burung elang [2,3,2], burung hantu [2,3,2], manusia [2,2,3] dan ikan piranha [1,1,2].
  - Obyek ikan piranha [1,1,2] diproses pertama karena obyek yang paling jauh. Obyek ini dimasukan ke dalam *node y*ang baru, misalkan *node A*, lalu perbaharui *attribute-attribute*nya. Probabilitas *node A* sebesar 1.
  - Obyek selanjutnya, manusia [2,2,3], diproses. Nilai *PU root* jika dimasukan ke *node A* sebesar 0, sedangkan jika ke dalam *node* baru sebesar 0,75. Dibuat *node* baru, misalkan *node B*, dan dimasukan obyek manusia [2,2,3], diperbaharui *attribute-attribute* dan probabilitasnya menjadi 0,5. Probabilitas *node A* juga 0,5. Untuk lebih jelas lihat gambar 2.13.
  - Obyek burung hantu [2,3,2] diproses, jika dimasukan ke dalam *node A*, nilai *PU root*nya sebesar 0,444, begitu juga jika *node B*, *PU root*nya sebesar 0,444. Sedangkan jika ke dalam *node* baru, sebesar 0,518. Maka dibuat *node* baru, misalkan *node C*, berisikan obyek burung hantu, probabilitasnya 0,33, begitu juga *node A* dan *node B*.
  - Selanjutnya obyek burung elang [2,3,2], jika ke dalam *node A*, nilai *PU root*nya sebesar 0,256, jika *node B*, sebesar 0,257, jika di *node C* sebesar 0,4583, jika di *node y*ang baru, sebesar 0,3437. Yang terbesar jika ke

dalam *node C*. Kemudian obyek burung elang [2,3,2] dimasukan ke *node C* dan *attribute-attribute*nya diperbaharui.



Gambar 2.13 *PU root* jika obyek manusia [2,2,3] dimasukan pada *node A* dan *node* baru.



Gambar 2.14 Hasil akhir iterasi *node root.*

- Kemudian diproses obyek ikan paus [2,1,2], jika dimasukan ke dalam *node A*, nilai *PU root*-nya sebesar 0,299, jika *node B*, sebesar 0,226, jika di *node C* sebesar 0,359. Sedangkan jika ke dalam *node* baru, sebesar

0,3199. Obyek ikan paus [2,1,2] dimasukan ke dalam *node C* dan *attribute-attribute*nya diperbaharui. *Node*A memiliki obyek ikan piranha. *Node C* memiliki 3 obyek yaitu ikan paus [2,1,2], burung elang [2,3,2], burung elang [2,3,2], dan ikan paus [2,1,2]. *Node B* hanya memiliki manusia [2,2,3]. Semua *node* baru, *node A, node B, node C,* dimasukan ke dalam kumpulan *node-node y*ang akan diiterasi selanjutnya. Disimpan pada *set L* dan *node root* dibuang dari *set L*, kemudian Iterasi selanjutnya dilakukan pada *node A*. Hasil dari iterasi pada *node root* dapat dilihat pada gambar 2.14.

- ***Node A***

  Karena *node A* memiliki satu obyek maka tidak dilanjutkan. *Node*A dibuang dari *set L* dan iterasi selanjutnya dilakukan pada *node B.*

- ***Node B***

  Karena *node B* memiliki satu obyek maka tidak dilanjutkan. *Node B* dibuang dari *set L* dan iterasi selanjutnya dilakukan pada *node C*.

- ***Node C***

  - Iterasi pada *node D* dimulai dengan mengurutkan obyek dengan ADO. Obyek-obyek terurut menjadi burung elang [2,3,2], burung hantu [2,3,2] dan ikan paus [2,1,2]. Obyek pertama yang dimasukan ikan paus [2,1,2]. Karena obyek pertama, ikan paus [2,1,2], maka disimpan pada *node* baru, misal *node D*. *Attribute-attribute*nya diperbaharui dan probabilitasnya sebesar 1 karena *node C* hanya punya 1 *node* anak.

  - Kemudian obyek burung hantu [2,3,2] diproses, jika dimasukan ke dalam *node D*, nilai *PU node C* sebesar 0, jika ke dalam *node* baru, PUnya sebesar 0,25. Obyek burung hantu [2,3,2] dimasukan ke *node* baru, misalkan *node E*, perbaharui *attribute-attribute*nya dan probabilitasnya 0,5. Probabilitas *node D* juga menjadi 0,5.

- Obyek burung elang [2,3,2] selanjutnya diproses. Jika ke dalam *node D*, *PU node C* sebesar -0,0138, jika dimasukan pada *node E*, *PU node C* sebesar 0,222, jika dimasukan ke *node* baru *PU*nya sebesar 0,148. Obyek burung elang [2,3,2] dimasukan ke *node E*, perbaharui *attribute-attribute*nya. Semua *node* baru, *node D*, *node E* dan F dimasukan ke *set L*. *Node C* dibuang dari *set L* dan iterasi selanjutnya pada *node D*.

- ***Node D***

  Karena *node D* memiliki satu obyek maka tidak dilanjutkan. *Node D* dibuang dari *set L* dan iterasi selanjutnya dilakukan pada *node E*.

- ***Node* E**

  - Iterasi pada *node E* dimulai dengan mengurutkan obyek dengan ADO. Obyek-obyek terurut menjadi burung elang [2,3,2] dan burung hantu [2,3,2]. Obyek pertama yang dimasukan burung hantu [2,3,2]. Karena obyek pertama, maka disimpan pada *node* baru, misal *node F*. *Attribute-attribute*nya diperbaharui dan probabilitasnya sebesar 1 karena *node E* hanya punya satu *node* anak.

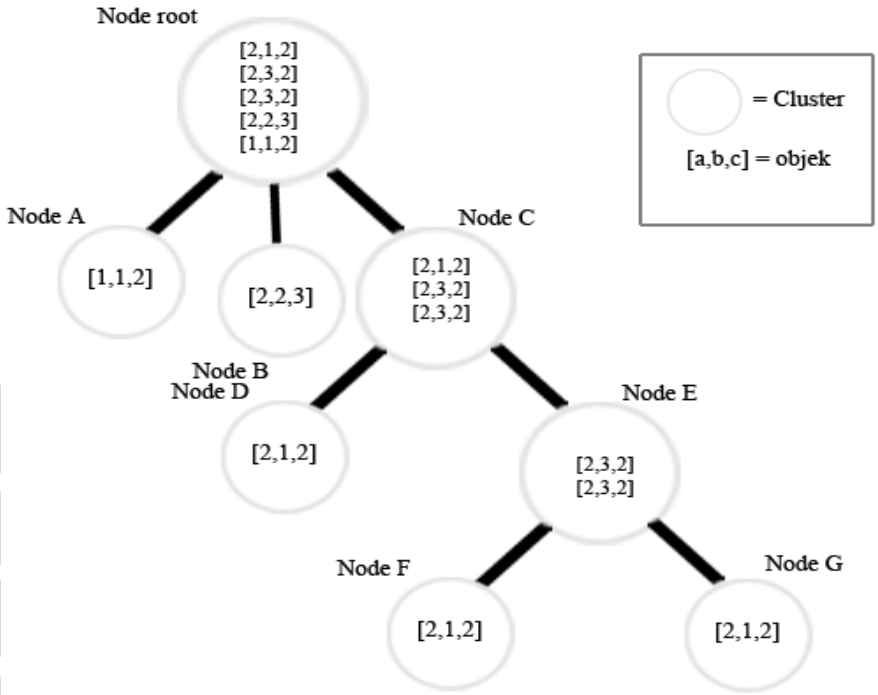
  - Obyek burung elang [2,3,2] diproses, jika dimasukan ke dalam *node F*, nilai *PU node E* sebesar 0, jika ke dalam *node* baru, PUnya sebesar 0. Karena sama maka dipilih untuk membuat *node* baru, misal *node G*, obyek tersebut dimasukan dan *attribute-attribute*nya diperbaharui. Semua *node* baru, *node F* dan G, dimasukan ke *set L* dan *node E* dibuang. Iterasi selanjutnya dilakukan pada *node F*.

- ***Node F***

  Karena *node F* memiliki satu obyek maka tidak dilanjutkan. *Node F* dibuang dari *set L* dan iterasi selanjutnya dilakukan pada *node G*.

- ***Node G***

  Karena *node G* memiliki satu obyek maka tidak dilanjutkan. *Node G* dibuang dari *set L* dan iterasi selanjutnya dilakukan pada *node H*. Hasil akhir dari iterasi diatas dapat dilihat pada gambar 2.15.



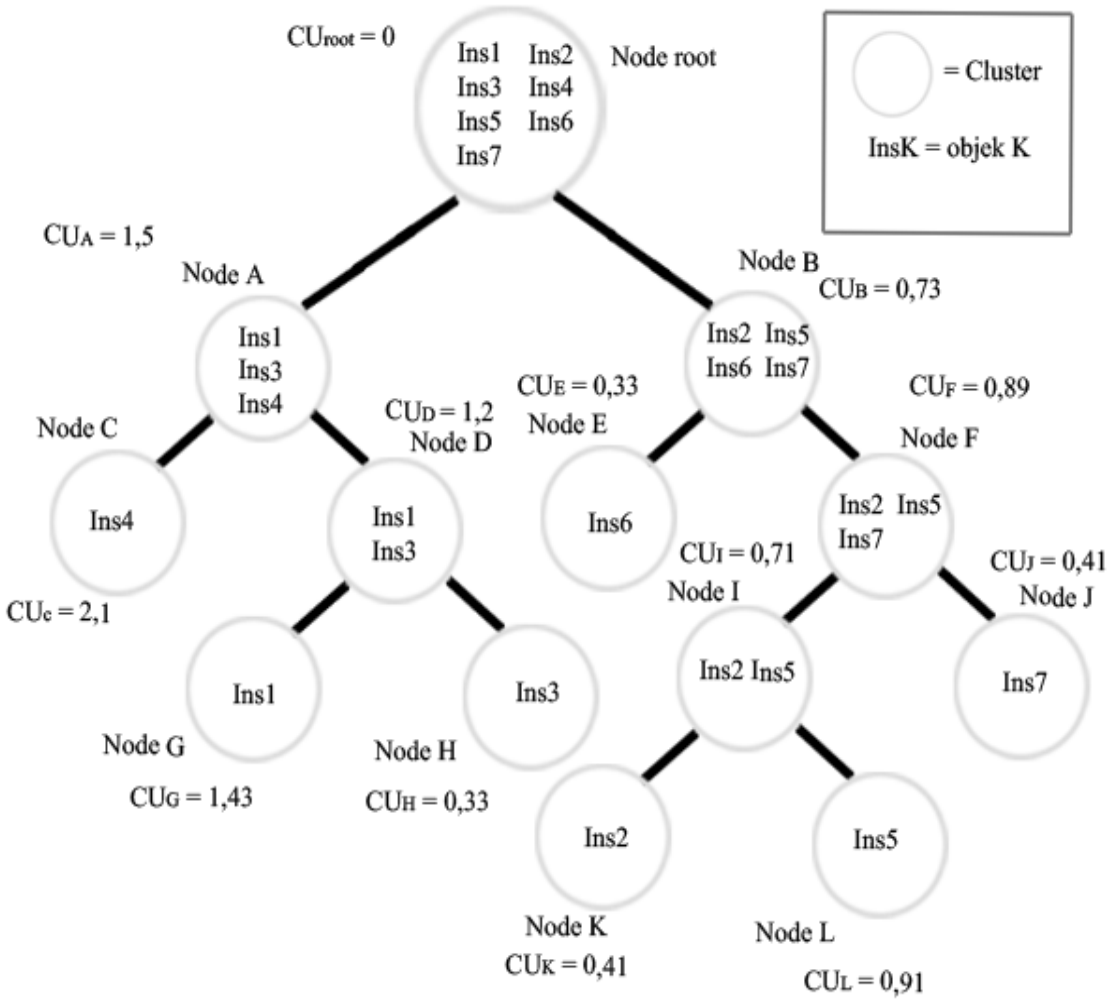
Gambar 2.15 Hasil akhir langkah pertama.

Untuk proses langkah kedua dan ketiga, contoh tidak diberikan karena memerlukan data yang besar. Namun diberikan ilustrasinya. Gambar 2.16 adalah gambar acuan untuk ilustrasi tersebut.

Langkah dua dimulai dengan menelusuri *node* mulai dari *node root* dahulu ke *node-node* daun secara *breath first*. Berikut penjelasannya.

- *Node N* adalah *node root. Node root* dimasukan ke dalam *set list*. Lalu dilihat apakah semua *CU* anak-anak *root* lebih kecil dari CU_{root}. Karena tidak lebih kecil, maka dicek satu per satu anak *root*. *CU node A* (CU_A) lebih besar dari CU_{root} maka dimasukan ke dalam *set list*. *CU node B* (CU_B) lebih besar dari CU_{root} maka dimasukan ke dalam *set list*. Kemudian

node root dibuang dari *set list. N* selanjutnya adalah *node A. Set list* berisi *node A* dan *node B.*



Gambar 2.16 Ilustrasi langkah kedua dan ketiga *Iterate*

- *CU* semua anak *node A* tidak lebih kecil dari CU_A. Sehingga dicek satu per satu. CU_C lebih besar dari CU_A maka dimasukan ke dalam *set list*. CU_D lebih kecil dari CU_A maka *node D* membuat grup untuk diperbaiki, misalkan grup *D*. Grup *D* berisikan *node G* dan *node H*. Kemudian *node A* dibuang dari *set list*. *N* selanjutnya adalah *node B*. *Set list* berisikan *node B* dan *node C*.

- *CU* semua anak *node B* tidak lebih kecil dari CU_B sehingga dicek satu per satu. CU_E lebih kecil dari CU_B namun *node E* adalah *node* daun(*leaf*) maka tidak membuat grup. CU_F lebih besar dari CU_B maka *node F* dimasukan ke dalam *set list*. Kemudian *node B* dibuang dari *set list*. N selanjutnya adalah *node C*. *Set list* berisikan *node C* dan *node F*.

- *Node C* merupakan *node* daun sehingga dibuang dari *set list*. N selanjutnya adalah *node F*. *Set list* berisikan *node F*.

- *CU* semua anak *node F* tidak lebih kecil dari CU_F sehingga dicek satu per satu. CU_I lebih kecil dari CU_F maka *node I* membuat grup, misalkan grup *I*. Grup *I* berisikan *node K* dan *node L*. CU_J lebih kecil dari CU_F namun *node J* adalah *node* daun(*leaf*) maka tidak membuat grup. Kemudian *node F* dibuang dari *set list*. Langkah kedua selesai.

Langkah ketiga mengambil 2 grup yaitu grup *D* dan *I* sebagai *input*. Pertama-tama, setiap obyek dari *node D* (*node* paling umum di grup *D*) dimasukan sementara pada *node I* (*node* paling umum di grup *I*). Misalkan obyek Ins1, CU_D jika obyek Ins1 berada di *node D* sebesar 0,41. Sedangkan jika pada *node I* menyebabkan CU_I lebih besar dari CU_D maka Ins1 disimpan pada *node I*. Jika orang tua *node A* bukan *root*, maka Ins1 dibuang dari *node-node y*ang berada pada jalur (*path*) dari *root* hingga *node A*. Kemudian *node G*, yang hanya mempunyai obyek Ins1, dibuang dari grup *D* dan dilakukan penentuan penyimpanan pada grup *I*. Penentuannya ada 2 pilihan yaitu disimpan pada *node* terbaik atau pada *node* baru. Misalkan $CU_{\text{k+Ins1}}$ (*CU node L* jika menyimpan Ins1) lebih besar jika menyimpan Ins1 dari pada $CU_{\text{L+Ins1}}$ (*CU node L* jika menyimpan Ins1) dan $CU_{node\ \text{baru}}$ (*CU* jika Ins1 disimpan pada *node* baru), maka Ins1 disimpan pada *node K*. Kemudian karena *node K* memiliki 2 obyek, maka obyek lama, Ins2, disimpan pada *node* baru, misal *node M*, dan obyek Ins1 juga disimpan pada *node* baru, misal *node P*. Sehingga *node K* memiliki 2 anak. Begitu seterusnya hingga obyek terakhir pada *node D*. Kemudian hal yang sama dilakukan pada grup *I*.

2.6 WEKA

WEKA (Waikato *Environment for Knowledge Analysis*) salah *data mining* tools yang *open source*. WEKA dibuat oleh Universitas Waikato, New Zealand. Tools ini dibuat dengan menggunakan bahasa pemrograman Java (Hall, n.d.).

Dalam WEKA terdapat beberapa bagian dari *data mining*, antara lain *data preprocessing*, *clustering*, *Classification*, *Regression*, *Visualization*, dan *Feature Selection*.

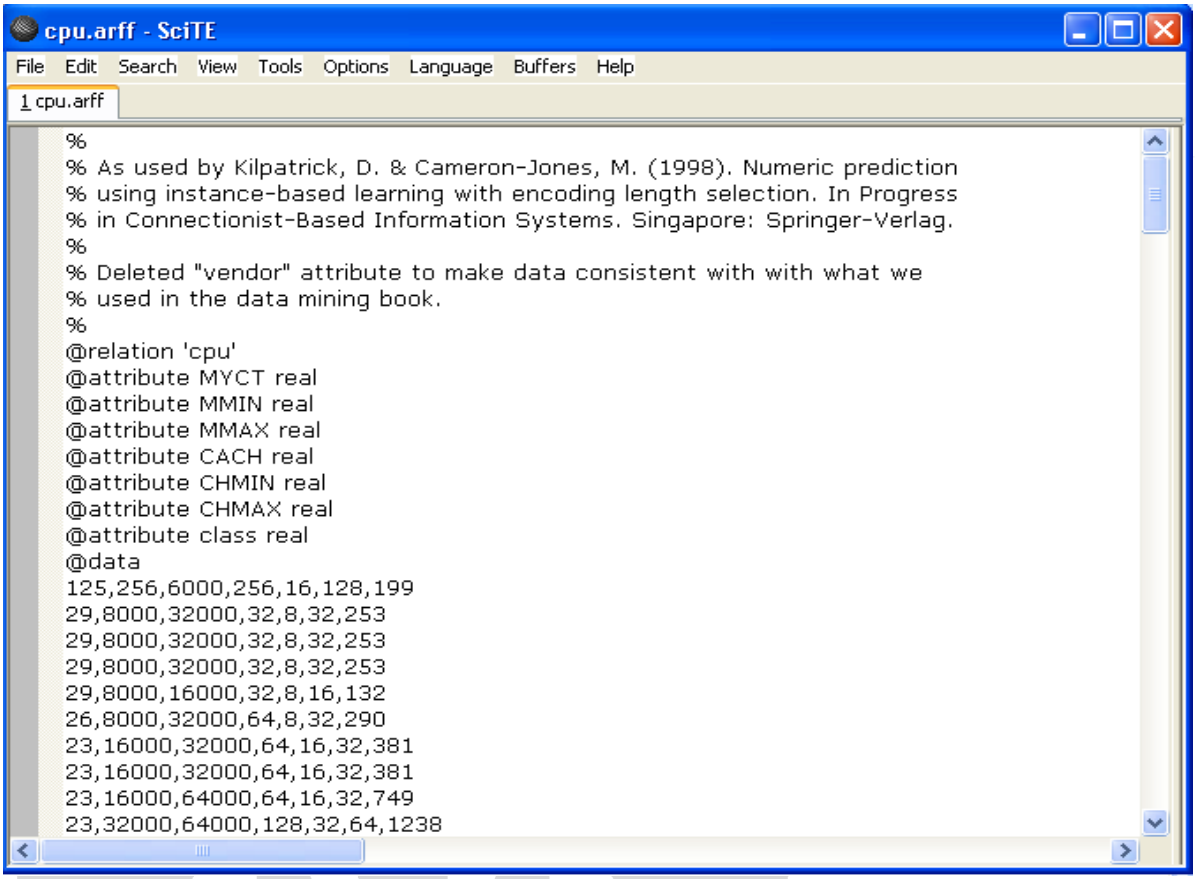
clustering, *Classification* dan *Regression* merupakan teknik-teknik dalam *data mining*. Dalam WEKA, *Classification* dan *Regression* menjadi satu bagian yaitu *classify*. Selain itu, WEKA juga menyediakan teknik *Association*. Contoh algoritma-algoritma yang terdapat dalam WEKA antara lain BayesNet, ZeroR, NBTree (*classify*), *Cobweb*, EM, FartheFirst (*clustering*), Appriori, PredictiveAppriori, Tertius (*association*).

Input untuk aplikasi ini dapat berupa tabel *database*, *file* CSV, dan lain-lain. *Input* ini sebelum diproses terlebih dahulu diubah menggunakan *data preprocessing*. WEKA juga menyediakan format *file* tersendiri yaitu *arff*. Format *file* ini merupakan buatan WEKA. *File* format *arrf* terdiri dari 3 bagian yaitu *relation*, *attribute*, dan data. Untuk lebih jelasnya lihat gambar 2.17

Berikut penjelasan untuk format *arff* (Waikato, 2006) pada gambar 2.12

- *Header*, terdiri dari:
    a. Relation dengan simbol *@relation* yang menandakan relasi antar data.
    b. *Attribute* dengan simbol *@attribute*. Format untuk *attribute* yaitu *@attribute* <nama *attribute*> <tipe data>. Tipe data antara lain numerik, <*nominal-spesification*>, *string*, dan *date* (format tanggal).
- Data, yang ditandai dengan simbol @data dan merupakan kumpulan data-data yang akan diproses.

Implementasi algoritma clustering..., Adolf Pandapotan



Gambar 2.17 Contoh *file* format *arff*.