PENERAPAN ALGORTIMA *SLIQ* UNTUK PENGKLASIFIKASIAN JURUSAN PADA SMK BAGI SISWA BARU (STUDI KASUS : DATA PENDAFTAR PADA PSB ONLINE JENJANG SMK PROVINSI DKI JAKARTA TAHUN 2008-2010)

SKRIPSI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun oleh :

HENDRAWAN KUNCORO

NIM. 0710960050

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
PROGRAM STUDI TEKNIK INFORMATIKA
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG

2012

LEMBAR PERSETUJUAN

PENERAPAN ALGORTIMA *SLIQ* UNTUK PENGKLASIFIKASIAN JURUSAN SMK DITERIMA BAGI SISWA BARU (STUDI KASUS : DATA PENDAFTAR PADA PSB ONLINE JENJANG SMK PROVINSI DKI JAKARTA TAHUN 2008-2010)

SKRIPSI

Disusun oleh :

HENDRAWAN KUNCORO

NIM. 0710960050

Telah diperiksa dan disetujui oleh :

Pembimbing I,	Pembimbing II,
Lailil Muflikhah S.Kom, M.Sc	**Dian Eka R., Ssi, M.Kom**
NIP. 197411132005012001	**NIP. 197306192002122001**

LEMBAR PENGESAHAN

PENERAPAN ALGORTIMA *SLIQ* UNTUK PENGKLASIFIKASIAN JURUSAN SMK DITERIMA BAGI SISWA BARU (STUDI KASUS : DATA PENDAFTAR PADA PSB ONLINE JENJANG SMK PROVINSI DKI JAKARTA TAHUN 2008-2010)

SKRIPSI

Diajukan untuk memenuhi persyaratan memperolah gelar Sarjana Komputer

Disusun oleh :

HENDRAWAN KUNCORO

NIM. 0710960050

Skripsi ini telah diuji dan dinyatakan lulus tanggal 30 November 2012

Penguji I,	Penguji II,
Dewi Yanti Liliana, S.Kom, M.Kom	**Candra Dewi, S.Kom, M.Sc**
NIP. 198111162005012004	**NIP. 197711142003122001**

Mengetahui
Ketua Program Studi Teknik Informatika

Drs. Marji, M.Si.
NIP. 196708011992031001

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, November 2012

Mahasiswa,

Hendrawan Kuncoro

NIM 0710960050

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT yang telah melimpahkan segala Rahmat, Karunia dan Hidayah-Nya sehingga Penulis dapat menyelesaikan skripsi dengan judul: "**Penerapan Algortima SLIQ untuk Pengklasifikasian Jurusan pada SMK bagi Siswa Baru (Studi Kasus : Data Pendaftar pada PSB Online Jenjang SMK Provinsi DKI Jakarta Tahun 2008-2010)**"

Skripsi ini diajukan sebagai syarat ujian seminar skripsi dalam rangka untuk memperoleh gelar Sarjana Komputer di Fakultas MIPA, Program Studi Ilmu Komputer, Jurusan Matematika, Universitas Brawijaya Malang. Atas terselesaikannya skripsi ini, Penulis mengucapkan terima kasih kepada:

1. Drs. Marji, MT. selaku Ketua Program Studi Teknik Informatika Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya.
2. Lailil Muflikhah, S.Kom., MSc. selaku Dosen Pembimbing Skripsi I.
3. Dian Eka R., S.Si., M.Kom. selaku Dosen Pembimbing Skripsi II.
4. Nurul Hidayat, SPd. MSc. selaku Dosen Penasehat Akademik.
5. Ir. Sutrisno, MT, selaku Ketua Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya.
6. Segenap Bapak dan Ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya.
7. Segenap staf dan karyawan di Program Teknik Informatika Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya yang telah banyak membantu Penulis dalam pelaksanaan penyusunan skripsi ini.
8. Orang tua Penulis dan saudara-saudaraku atas segala dukungan materi dan doa restunya kepada Penulis.
9. Rekan-rekan Program Studi Teknik Informatika yang telah memberikan dukungannya kepada penulis.
11. IceFrog@gmail.com, yang telah memberi kreativitas dan menjalani malam-malam bersama skripsi yang telah disusun ini.

12. Nuzulianti Tsulusia, yang telah memberi dukungan baik dalam bentuk material maupun non material demi terselesaikannya skripsi ini.
13. Semua pihak yang telah membantu terselesaikannya skripsi ini yang tidak dapat kami sebutkan satu per satu.

Penulis menyadari bahwa skripsi ini tentunya tidak terlepas dari berbagai kekurangan dan kesalahan. Oleh karena itu, segala kritik dan saran yang bersifat membangun sangat Penulis harapkan dari berbagai pihak demi penyempurnaan penulisan skripsi ini.

Akhirnya penulis berharap agar skripsi ini dapat memberikan sumbangan dan manfaat bagi semua pihak yang berkepentingan.

Malang, November 2012

Hendrawan Kuncoro

Penulis

ABSTRAK

Hendrawan Kuncoro. 2012. : Penerapan Algortima SLIQ untuk Pengklasifikasian Jurusan pada SMK bagi Siswa Baru (Studi Kasus : Data Pendaftar pada PSB Online Jenjang SMK Provinsi DKI Jakarta Tahun 2008-2010).

Dosen Pembimbing : Lailil Muflikhah, S.Kom., MSc. dan Dian Eka R., S.Si., M.Kom.

PSB atau Penerimaan Siswa Baru adalah penerimaan peserta didik pada TK/RA/BA dan sekolah/madrasah yang dilaksanakan pada tahun awal ajaran baru. Salah satu permasalahan yang terjadi dalam PSB adalah, kebingungan calon siswa SMK dalam memilih jurusan. Pengolahan data yang sesuai adalah klasifikasi dimana mengklasifikasikan jurusan SMK dengan atribut pada data pendaftar PSB tahun-tahun sebelumnya. Sehingga calon siswa dapat menyesuaikan data pendaftarannya dengan jurusan yang akan dipilih. Algoritma klasifikasi yang digunakan dalam penulisan ini adalah algortima *SLIQ* (Supervised Learning In Quest).

SLIQ adalah algoritma *Decision Tree* yang diperkenalkan oleh Manish Mehta pada tahun 1996 dan dikembangkan oleh IBM Almaden Research Center. Manish Mehta pada tahun 1996 juga menyebutkan bahwa *SLIQ* adalah *Decision Tree* pengklasifikasi yang dapat menangani atribut numerikal ataupun kategorikal dengan jumlah data latih dan atribut yang besar sekalipun. Keunggulan inilah yang menjadi alasan utama penulis dalam pemilihan algortima *SLIQ* untuk proses pengklasifikasian. *SLIQ* yang dalam implementasinya melakukan pembentukan *Decision Tree* dangan metode BFS dan pemangkasan tree dengan metode *bottom-up* dengan data latih pendaftar PSB tahun 2008 sampai 2010 menunjukkan hasil yang baik terhadap data uji tahun 2011. Walaupun data uji tiap tahun yang berjumlah antara 3000 sampai 4000 dan data uji sekitar 4000, waktu yang dibutuhkan untuk pembentukan *tree* maupun pengujian tidak lama.

Kata Kunci : *SLIQ*, *Decision Tree*, PSB, SMK**.**

ABSTRACT

Hendrawan Kuncoro. 2012.: Application of SLIQ algorithm for classification of subjects at vocational schools for new students (Case Study: Online Data Apply at PSB Study SMK DKI Jakarta Years 2008-2010).
Advisor : Lailil Muflikhah, S.Kom., MSc. and Eka Dian R., S.Si., M.Kom.

PSB or Admission is admission of students in kindergarten / RA / BA and school / madrasah conducted at the beginning of the new academic year. One of the problems that occur in the PSB is, confusion prospective vocational students in choosing majors. Appropriate data processing is a classification which classifies vocational majors with attributes in PSB data registries at previous years. So that prospective students can customize the data is registered with the department to be selected. Classification algorithm used in this paper are algorithms SLIQ (Supervised Learning In Quest).

SLIQ Decision Tree algorithm is introduced by Manish Mehta in 1996 and developed by IBM Almaden Research Center. Manish Mehta in 1996 was also mentioned that SLIQ Decision Tree classifiers that can handle numerical or categorical attributes with the amount of data and the attributes of a great trainer though. These excellences are the main reason for the selection of authors in SLIQ algorithm for the classification. SLIQ that in the implementation of forming Decision Tree with the BFS method and tree pruning bottom-up method to the data practice registrars PSB 2008 to 2010 showed good results on the test data in 2011. Although the test data each year, amounting to between 3000 to 4000 and test data around 4000, the time required for tree establishment and testing was short.

Keywords: *SLIQ*, *Decision Tree*, PSB, SMK.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

PSB atau Penerimaan Siswa Baru adalah penerimaan peserta didik pada TK/RA/BA dan sekolah/madrasah yang dilaksanakan pada tahun awal ajaran baru. Sistem yang digunakan untuk mendukung agenda kegiatan tahunan ini bisa jadi dibuat secara mandiri oleh sekolah atau dinas pendidikan setempat, namun beberapa dinas pendidikan kota atau provinsi menggunakan sistem penerimaan siswa baru *online* yang disediakan secara gratis oleh Kementrian Pendidikan dan Kebudayaan Republik Indonesia atau menggunakan sistem berbasis *online* lainnya. Salah satu sistem penerimaan siswa baru yang berbasis online adalah SIAP PSB, yang menyediakan aplikasi pendaftaran secara mandiri dan seleksi secara *real-time*.

Adanya sistem Penerimaan Siswa Baru yang menampung data siswa pendaftar, sekolah pilihan dan hasil seleksi yang terus bertambah setiap tahunnya menjadikan PSB sebagai penyedia *data warehouse*. *Data warehouse* ini nantinya bisa dikembangkan menjadi sistem prediksi, rekomendasi atau menjadi data pendukung dalam pembuat keputusan oleh pelaku kegiatan akademik baik siswa, sekolah maupun dinas pendidikan kota atau provinsi. Salah satu permasalahan yang terjadi dalam Penerimaan Siswa Baru adalah, kebingungan calon siswa SMK dalam memilih jurusan. Permasalahan ini sebenarnya bisa disolusikan dengan mengolah data PSB pada tahun-tahun sebelumya sebagai acuan calon siswa SMK dalam memilih jurusan tujuan. Dengan pengolahan yang sesuai, data PSB yang merupakan *data warehouse*, bisa menjadi informasi pendukung yang sangat bermanfaat dalam pemilihan jurusan bagi calon siswa SMK. Pengolahan data yang sesuai adalah klasifikasi dimana mengklasifikasikan jurusan SMK dengan atribut pada data pendaftar PSB tahun-tahun sebelumnya. Sehingga calon siswa dapat menyesuaikan data pendaftarannya dengan jurusan yang akan dipilih.

Klasifikasi adalah salah satu bentuk penggunaan dari data mining. Data mining sendiri merupakan sebuah kegiatan untuk menggali dan mendapatkan

informasi dari data dengan jumlah yang besar (Olson, David, dan Yong Shi, 2007). Beberapa metode klasifikasi yang umum digunakan antara lain *Bayesian Network*, *Adaptive Bayesian* Network, Naïve *Bayes* dan *Decision Tree*. Menurut Pang-Ning Tan dkk pada tahun 2006, metode yang paling sering digunakan dalam klasifikasi adalah *Decision Tree*. Algoritma *ID3, C4.5, C5.0, CART* dan *MARS* adalah beberapa penerapan metode *Decision Tree* untuk permasalahan klasifikasi, termasuk juga algoritma *SLIQ,* yang akan dibahas salam skripsi ini.

SLIQ adalah salah satu algoritma *Decision Tree* yang diperkenalkan oleh Manish Mehta pada tahun 1996 dan dikembangkan oleh IBM Almaden Research Center. *SLIQ* sendiri adalah singkatan dari *Supervised Learning In Quest,* dimana Quest adalah proyek data mining yang dimiliki oleh IBM Almaden Research Center. Algortima klasifikasi ini berusaha untuk mengatasi kelemahan yang dimiliki kebanyakan algoritma klasifikasi lain, yaitu jumlah data latih. Kebanyakan algortima *Decision Tree* dalam klasfikasi memiliki keterbatasan, dimana data latih harus bisa muat kedalam *memory*, kelemahan ini ditangan oleh algoritma *SLIQ* yang dalam penerapannya tidak memperhatikan jumlah data latih atau jumlah atribut dalam setiap data latih tanpa mengurangi akurasi dari *tree* yang dihasilkan. Manish Mehta pada tahun 1996 juga menyebutkan bahwa *SLIQ* adalah *Decision Tree* pengklasifikasi yang dapat menangani atribut numerikal ataupun kategorikal. Keunggulan-keunggulan inilah yang menjadi alasa utama penulis dalam pemilihan algortima *SLIQ* untuk proses pengklasifikasian.

Berdasarkan latar belakang yang dipaparkan maka skripsi ini diberi judul "**Penerapan Algortima SLIQ untuk Pengklasifikasian Jurusan pada SMK bagi Siswa Baru (Studi Kasus : Data Pendaftar pada PSB Online Jenjang SMK Provinsi DKI Jakarta Tahun 2008-2010)**".

1.2 Rumusan Masalah

Rumusan masalah pada skripsi ini adalah sebagai berikut:

1. Bagaimana menerapkan algoritma *SLIQ* pada klasifikasi jurusan SMK diterima bagi Siswa Baru.
2. Mengukur tingkat *precision, recall, f-measure* dan akurasi klasifikasi dengan algortima *SLIQ* dalam penerapannya untuk pengklasifikasian jurusan pada SMK bagi siswa baru.

1.3 Batasan Masalah

Agar pembahasan tidak melebar, maka batasan masalah dalam skripsi ini adalah:

1. Data latih yang digunakan adalah data pendaftar PSB jenjang SMK tahun 2008-2010 tahap I.
2. Atribut yang digunakan dalam data latih adalah nilai UN (Mata pelajaran Matematika, Bahasa Indonesia, Bahasa Inggris, IPA), jenis kelamin, dan lokasi pendaftar.

1.4 Tujuan

Tujuan yang ingin dicapai dari skripsi ini adalah:

1. Melakukan implementasi algoritma *SLIQ* pada klasifikasi jurusan SMK diterima bagi Siswa Baru.
2. Mengetahui tingkat *precision, recall, f-measure* dan akurasi klasifikasi dengan algortima *SLIQ* dalam penerapannya untuk pengklasifikasian jurusan pada SMK bagi siswa baru.

1.5 Manfaat

1. Pembaca dapat memahami implementasi algortima *SLIQ* sebagai pengklasifikasi jurusan SMK diterima.
2. Hasil klasifikasi dapat digunakan siswa dalam menentukan pilhan jurusan, atau digunakan pelaku akademik sebagai bahan pendukung keputusan.

1.6 Sistematika Penulisan

Sistematika penulisan skripsi terdiri dari 5 bab, yaitu:

BAB I PENDAHULUAN

Bab ini menguraikan latar belakang masalah yang akan dibahas, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian dan sistematika penulisan.

BAB II DASAR TEORI

Bab ini menjelaskan teori-teori yang mendasariimplementasi klasifikasi pilihan jurusan SMK diterima dengan algortima *SLIQ*.

BAB III METODE DAN PERANCANGAN

Bab ini menjelaskan metode yang dipakai dan tahapan bagaimana sistem pengklasifikasian dibangun.

BAB IV IMPLEMENTASI DAN PEMBAHASAN

Bab ini membahas implementasi dari sistem pengklasfikasian yang telah dibuat dan uji coba yang dilakukan beserta pembahasannya.

BAB V PENUTUP

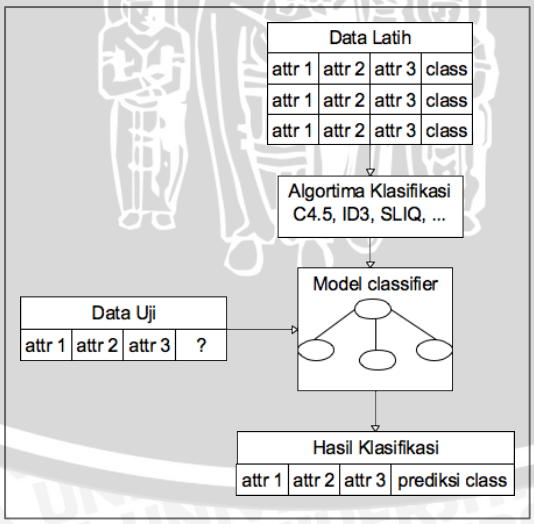
Pada bab ini akan menujukkan kesimpulan dan saran yang dapat digunakan untuk pengembangan berikutnya.

BAB II
DASAR TEORI

2.1 Klasifikasi

Klasifikasi merupakan salah satu bentuk implementasi dari data mining. Klasifikasi memperkirakan nilai atau yang pada implementasinya disebut dengan *class* dari beberapa variabel atau atribut. Sebagai contoh dari nilai atribut cara reproduksi, jumlah kaki dan jenis makanan dapat diperkirakan apakah hewan termasuk kedalam *class* hewan mamalia atau bukan (Han & Kamber 2006).

Dalam klasifikasi secara umum terdapat dua tahapan, yaitu tahapan pembentukan model *classifier* berdasarkan algortima klasifikasi yang digunakan dan data training yang ada kemudian menentukan*class* berdasarkan atribut atau variable menggunakan model *classifier* yang dihasilkan. Tahap kedua adalah evaluasi dari model *classifier* untuk mendapatkan nilai akurasi, jika nilai akurasi sesuai dengan yang diinginkan maka model *classifier* bisa digunakan untuk memprediksi data yang tidak berasal dari data latih (Han & Kamber 2006). Gambar 2.1 menginterpretasikan tahapan klasifikasi secara umum.



Gambar 2.1 Proses Klasfikasi

2.2 Decision Tree

Metode klasifikasi yang umum digunakan adalah dengan menerapkan algoritma *Decision Tree* pada pembentukan model *classifier.*Sebuah *Decision tree* mengklasifikasi data latih dengan mengajukan serangkaian pertanyaan terkait dengan atribut yang ada pada data latih. Setiap pertanyaan ditampung dalam sebuah node, dan setiap node internal pada sebuah node untuk setiap kmungkinan jawaban dari pertanyaan. Karena itu, pertanyaan-pertanyaan tersebut membentuk sebuah hirarki pohon (Kingsfod dan Salzberg, 2008).

Teknik klasfikasi *Decision Tree*menerapkan dua fase : *tree building* (pembentukan pohon) dan *tree pruning*(pemangkasan pohon). Pembentukan pohon diselesaikan dengan metode top-down, pada fase ini *tree* secara rekursif dipartisi sampai data terdapat pada *class* yang sesuai (Hunts dkk, 1966). Pemangkasan pohon dilakukan dengan bottom-up, pada fase ini ditujukan untuk meningkatkan akurasi prediksi dan klasifikasi dari algortima dengan meminimalkan *over-fitting*(*noise* atau detail yang berlebihan pada himpunan data) (Mehta dkk, 1996).

2.2.1 Attribut Selection Measure

*Attribut Selection Measure*bertujuan untuk memilih atribut mana yang paling penting atau paling berguna dari data latih. Atribut ini akan membagi data latih kedalam *class* yang dituju (Han & Kamber 2006). Ukuran yang digunakan untuk menentukan seberapa penting sebuah atribut disebut dengan *information gain.* ID3 menggunakan nilai *information gain* ini untuk memilih kandidat atribut pada setiap langkah dalam pembentukan pohon.

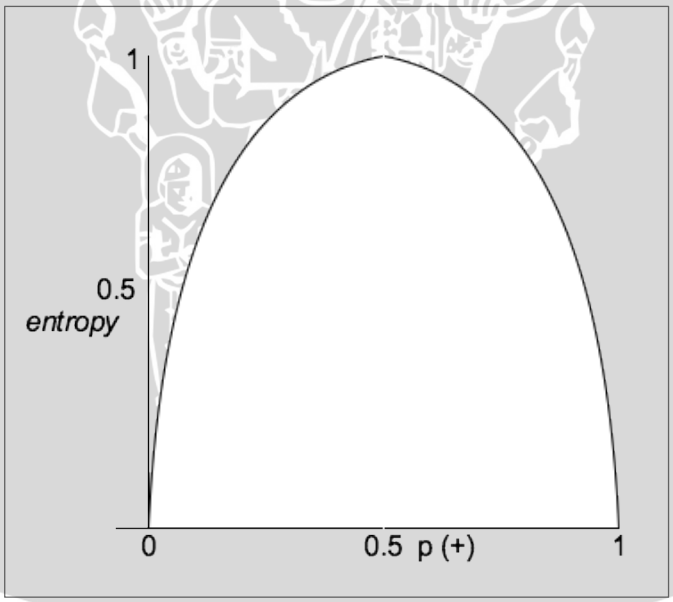
2.2.1.1 Entropy

Untuk menentukan *information gain* dengan tepat, diawali dengan melakukan perhitungan *entropy*yang umum digunakan dalam teori informasi.*Entropy* adalah ukuran kemurnian dari himpunan data latih(Mitchell, 1997). Tanpa mengurangi generalitas, anggap sebuah *decision tree* menghasilkan

dua kategori P (positif) dan N (negatif), dan S adalah himpunan data latih yang mengandung *class* P (+) dan N (-), maka Persamaan 2.1 menunjukkan persamaan *entropy*.

$$Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \quad (2.1)$$

Dimana $p_{\oplus}$ adalah proporsi dari data dengan class positif pada data latih S dan $p_{\ominus}$ adalah proposi data dengan class negatif pada data latih S. Jika semua data latih berada pada *class* yang sama maka nilai *entropy* adalah 0. Sedangkan *entropy* akan bernilai 1 jika data latih terdiri dari data dengan *class* positif dan negatif dengan jumlah yang sama. Jumlah yang tidak sama dari *class* positif dan negatif akan menghasilkan nilai *entropy* antara 0 dan 1. Pernyataan ini digambarkan pada Gambar 2.2.



Gambar 2.2 Kurva Perbandingan Nilai *Entropy* dengan Proporsi *Class* Positif pada Klasifikasi Boolean

Perhitungan entropy dengan persamaan pada Persamaan 2.1 digunakan untuk klasifikasi boolean, atau klasfikasi yang hanya terdiri dari dua kelas target.

Pada klasifikasi dengan *c* kelas target, maka persamaan *entropy* diilustrasikan pada Persamaan 2.2.

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.2}$$

Dimana p_i adalah proporsi dari kelas *i* yang terdapat pada himpunan data latih S.

2.2.1.2 Information Gain

Information gain yang akan digunakan untuk mengukur efektifitas dari atribut dalam mengklasifikasi data, merupakan pengurangan dari *entropy* data yang disebabkan oleh kegiatan mempartisi data latih dengan atribut itu sendiri (Mitchell, 1997). Jika *Gain(S,A)* meruapakan nilai *information gain* dari atribut *A* terhadap himpunan data latih *S*, maka persamaan *information gain* didefiniskan pada Persamaan 2.3.

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|Sv|}{|S|} Entropy(S_v) \tag{2.3}$$

Dimana *Values(A)* adalah himpunan dari nilai atribut A yang mungkin, dan $S_v$adalah sub himpunan dari S dimana atribut A memiliki nilai *v*. Nilai *information gain* dari sebuah atribut menandakan bahwa atribut tersebut semakin penting dan semakin efektif jika digunakan dalam partisi himpunan data latih.

2.2.2 *Tree Building*

Pembentukan pohon dilakukan dengan mempartisi data latih secara rekursif. Data latih dibagi menjadi dua atau lebih partsi menggunakan atribut yang ada

(Mehta dkk, 1996). Gambar 2.3 menunjukkan ilustrasi proses dari pembentukan pohon.

BuatPohon(Data Latih T)
    Partisi(T);
Partisi(Data S)
    if(semua titik pada S berada pada kelas yang sama) then return;
    Evaluasi split untuk setiap atribut A;
    Gunakan split terbaik yang ditemukan pada partisi S menjadi S_1 dan S_2;
    Partisi(S_1);
    Partisi(S_2);

Gambar 2.3 Algortima Tree-Building

Dimana *Evaluate splits* didapat dari proses perhitungan *entropy* dan *information gain.*

2.2.3 Tree Pruning

Fase kedua dari *decision tree* adalah pemangkasan pohon. Pemangkasan pohon memeriksa tree yang dihasilkan pada proses *treebuilding* dengan menggunakan data latih dan kemudian memilih *subtree* dengan tingkat error yang lebih kecil atau dengan batasan tertentu. Terdapat dua pendekatan dalam menghitung tingkat error, dengan menggunakan himpunan data latih, dan dengan menggunakan himpunan data independen yang digunakan khusus untuk mengukur error (Mehta dkk, 1996).

Pendekatan pertama merupakan kategori pengecekan silang. Dimana beberapa contoh dari data latih diambil dan dbentuk *tree*-nya masing-masing untuk kemudian digunakan dalam perhitungan tingkat error dari *subtree* yang dihasilkan pada *tree building* untuk keseluruhan data latih. Pendekatan kedua dilakukan dengan membagi data latih menjadi dua, bagian pertama digunakan

untuk *tree building* dan bagian kedua digunakan untuk *tree pruning* (Mehta dkk, 1996).

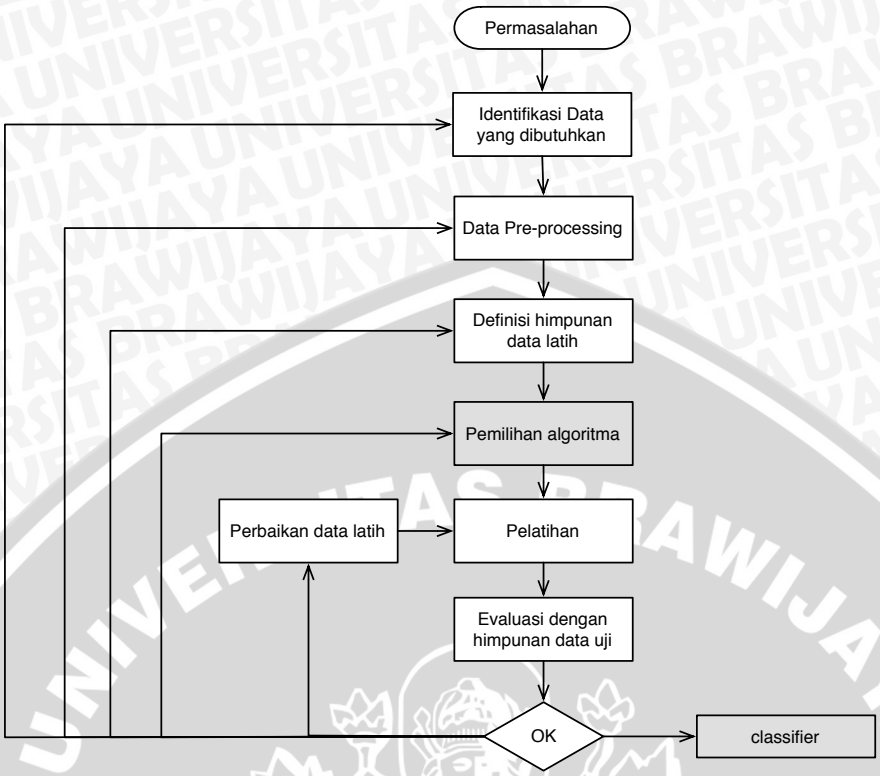
2.3 Algoritma SLIQ

Algoritma *SLIQ* pertama kali diperkenalkan sebagai algortima pembentuk model *classifier* oleh Manish Mehta pada tahung 1996. Dalam penjelasannya di *SLIQ: A Fast Scalable Classifier for Data Mining,* Mehta menyebutkan bahwa *SLIQ* adalah *decision tree* yang dapat menangani atribut numerik maupun kategorikal. Implementasi algoritma *SLIQ* menggunakan teknik *pre-sorting* pada fase *tree building* yang terintegrasi dengan teknik *breadth-first*, dan algortima *tree pruning* dengan prinsip *Minimum Description Length (MDL)*. Kombinasi teknik ini menjadikan algoritma *SLIQ* mampu menangani data dalam ukuran besar baik dalam atribut, kelas, maupun rekord (Mehta dkk, 1996).

SLIQ merupakan kependekan dari *Supervised Learning In Quest*. Seperti namanya SLIQ merupakan algoritma yang belajar secara *supervised*. Sedangkan *Quest* merupakan nama sebuah proyek pada *IBM Almaden Research Center,* dimana algortima ini pertama kali ditemukan (Mehta dkk, 1996)..

2.4 Supervised Machine Learning

Supervised machine learning merupakan pencarian algortima yang berdasarkan contoh yang diberikan dari luar untuk menghasilkan hipotesa umum, yang kemudian dapat menjadi prediksi untuk data berikutnya. Dimana contoh atau data latih yang diberikan dalam *supervised machine learning* memiliki *class* yang diketahui, berkebalikan dengan *unsuperviced machine learning*, dimana contoh yang diberikan tidak memiliki *class* (Kotsiantis, 2007). Gambar 2.4 merupakan ilustrasi proses dari *supervised machine learning*.



Gambar 2.4 Proses Supervised Machine Learning

Berdasarkan gambar 2.4, tahap identifikasi data yang dibutuhkan, pada umumnya masih memiliki *missing values* dan *noise,* sehingga masih dibutuhkan *data preprocessing* (Zhang dkk, 2003).Pada tahap evaluasi dengan himpunan data uji, bisa jadi menghasilkan *classifier* yang tidak sesuai, dimana diatasi dengan perbaikan parameter, perbaikan atau pemilihan ulang algortima, definisi ulang himpunan data latih, pre-processing ulang data latih, atau identifikasi ulang atribut yang dibutuhkan dari data latih.

2.5 Tahapan Algoritma SLIQ

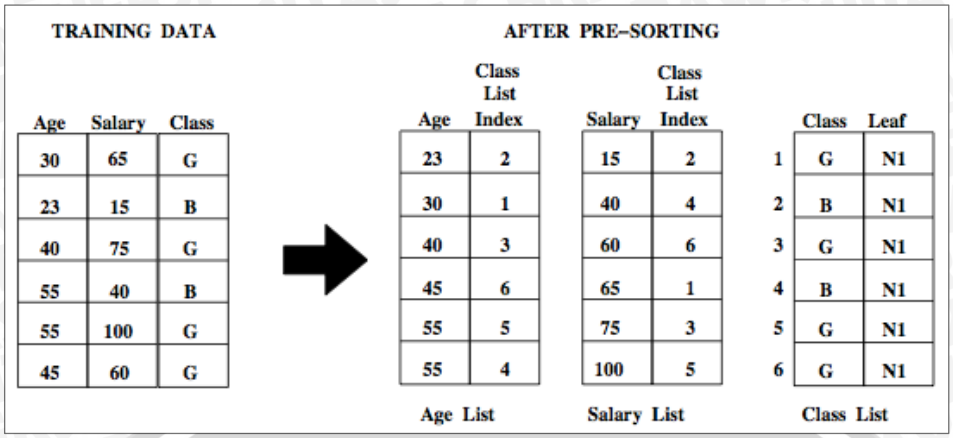
Sebagai salah satu bentuk *decision tree*, SLIQ juga memiliki dua tahap umum, yaitu *tree building* dan *tree pruning*. Dimana yang membedakan SLIQ dengan *decision tree* lainnya adalah teknik *pre-sorting* yang terintegrasi dengan *breadth-first* pada fase pembentukan pohon. Serta fase *tree pruning* dengan menggunakan teknik *Minimum Description Length* (MDL).

2.5.1 *Pre-Sorting* dan *Breadth-First Growth*

Proses *pre-sorting*perlu dilakukan terlebih dahulu terhadap atribut yang bertipe numerik. Menurut Catlett pada tahun 1991, waktu pengurutan adalah faktor dominan untuk menentukan *split* terbaik pada *node* pohon keputusan. Oleh karena itu Mehta pada tahun 1996 mengimplementasikan pengurutan untuk atribut numerik hanya pada awal fase pembentukan pohon dalam algortima SLIQ yang disebut dengan teknik *pre-sorting*, berbeda dengan algoritma *decision tree* lainnya yang melakukan pengurutan pada setiap penentuan *node* (Mehta dkk, 1996).

Proses *pre-sorting* diawali dengan membuat daftar dari setiap atribut yang ada dalam data latih yang disebut daftar atribut dan daftar kelas (*class list*) untuk menampung label kelas dari data atribut tersebut. Setiap entri pada daftar atribut terdapat 2 *field,* yaitu nilai atribut dan index dari daftar kelas. Sedangkan *class list* terdiri dari 2 *field* juga, yaitu label kelas dan referensi ke *leaf* dari *decision tree.* Entri ke-*i* dari daftar kelas akan berkolerasi dengan rekord ke-*i* pada data latih. Setiap *leaf node* pada *decision tree* menunjukkan pemisah pada data latih. Sehingga *class list,* bisa mengidentifikasi termasuk dalam pemisah mana sebuah rekord dari data latih karena *class list* memetakan index dari data latih terhadap *lead node* nya pada *decision tree* (Mehta dkk, 1996).

Pada langkah awal, *leaf* pada semua rekord *class list* diset pada *root* dari *decision tree*. Kemudian setiap atribut numerik diurutkan pada daftar terpisah (daftar atribut) dan di korelasikan dengan index dari *class list.* Sebagai contoh jika sebuah data latih terdiri dari 2 atribut numerik yang diklasifikasikan menjadi kelasnya masing masing, maka akan terdapat 2 daftar atribut yang sudah diurutkan dan 1 daftar kelas. Gambar 2.5 menggambarkan bagaimana proses *pre-sorting* berjalan pada algoritma *SLIQ.*



Gambar 2.5 Pre-Sorting Data Latih dan Hasilnya

2.5.2 Memproses *Node Split*

Dalam memproses *node split*, SLIQ menggunakan teknik *breadth-first*, tidak seperti *classifier decision tree* lainnya yang menggunakan *depth-first*. Akibatnya setiap data dimasukkan, maka *split* pada semua *leaf* akan dievaluasi. Gambar 2.6 menunjukkan algoritma dari evaluasi *split*.

EvaluasiSplit()
for each atribut *A* do
telusuri attribute list dari *A*
for each nilai *v* pada attribute list do
temuakan entry yang berkorespondensi pada class list, dan semenjak yang berkorespondensi dengan class dan node leaf (sebut *l*)
update class histogram pada leaf *l*
if *A* adalah atribut kategorikal then
for each leaf dari tree do
temukan subset dari *A* dengan split terbaik

Gambar 2.6 Algoritma Evaluate Splits

Untuk menghitung *gini splitting-index* dari sebuah atribut dari suatu node, maka dibutuhkan distribusi frekuensi dari nilai kelas yang ada pada data partition yang berkorespondensi dengan node.

$$gini(T) = 1 - \sum p_j^2 \tag{2.4}$$

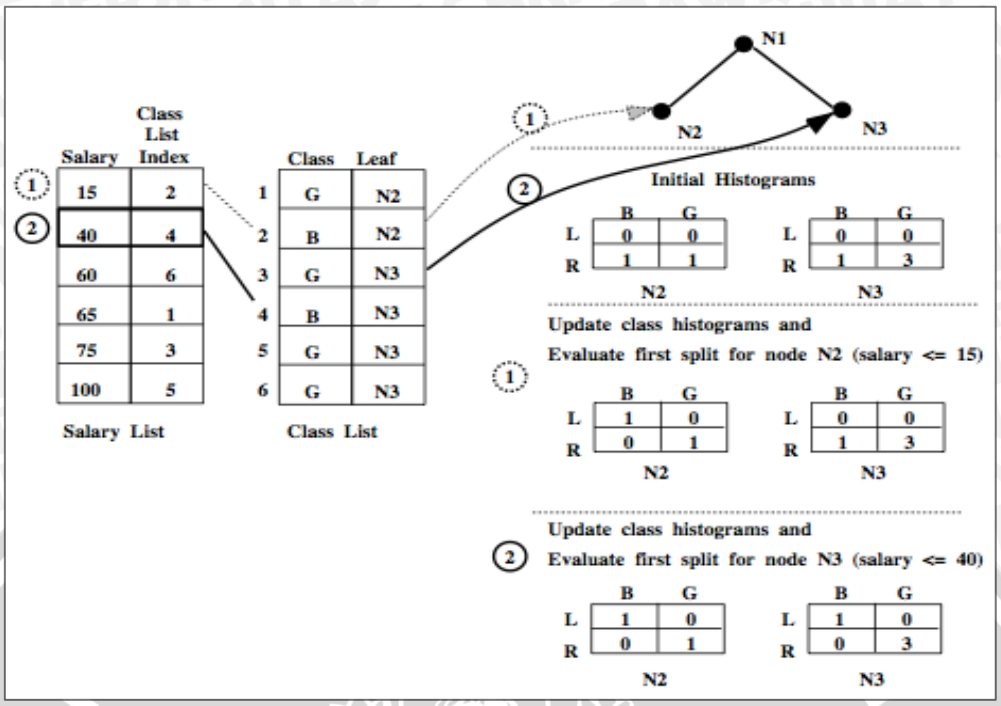
Persamaan 2.4 menunjukkan persamaan *gini*, dimana p_j adalah frekuensi relatif dari kelas *j* pada *T*. Sedangkan untuk persamaan gini split dengan diilustrasikan pada persamaan 2.5 (Nong Ye, 2003).

$$Gini_{split}(T) = \frac{P(T_1)}{P(T)} . Gini(T_1) + \frac{P(T_2)}{P(T)} . Gini(T_2) \tag{2.5}$$

Dimana $Gini(T_1)$ adalah nilai gini indeks untuk split pertama dan $Gini(T_2)$ adalah nilai gini indeks untuk split kedua. $P(T_1)$ dan $P(T_2)$ merupakan frekuensi terjadinya split pertama dan kedua pada data latih, sedangkan $P(T)$ adalah jumlah data latih.

Distribusi disimpan dalam struktur data berbentuk kelas histogram yang terhubung dengan setiap *leaf node*. Untuk atribut numerik, histogram berbentuk daftar yang terdiri dari <*kelas, frekuensi*>. Sedangkan untuk atribut kategorikal, histogram ini berbentuk daftar yang terdiri dari <*nilai atribut, kelas, frekuensi*>.

Daftar atribut diproses secara bersamaan, sesuai dengan algortima Evaluate Split pada Gambar 2.6. Proses *split node* ini dapat diilustrasikan seperti pada Gambar 2.7.



Gambar 2.7 Proses *Split Node*

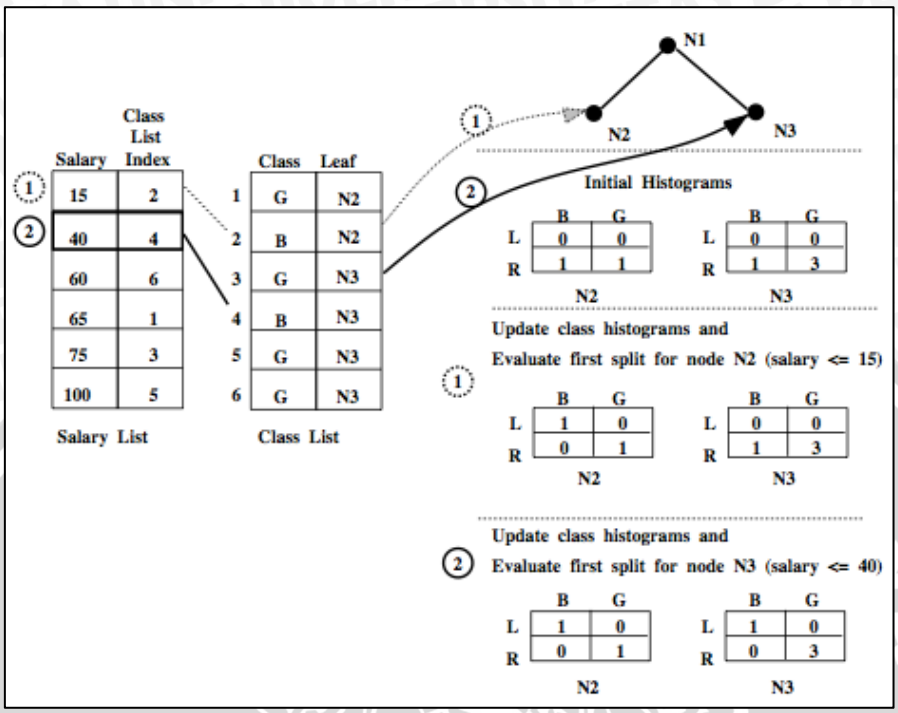
2.5.3 Memperbaharui *Class List*

Langkah selanjutnya adalah membuat child nodes untuk setiap *leaf node* dan memperbaharui daftar kelas. Algoritma dalam memperbaharui daftar kelas bisa dilihat pada Gambar 2.8.

UpdateLabels()
for each atribut *A* yang digunakan pada split do
telusuri attribute list *A*
for each nilai *v* pada the attribute list do
temukan entri yang berkorespondensi pada class list (say *e*)
temukan kelas baru *c* dimana *v* berada dengan mengaplikasi pengujian *e* splitting pada node yang bereferensi dari *e*
update class label untuk e bagi *c*
update node referensi pada e pada child yang berkorespondensi pada kelas *c*

Gambar 2.8 Algortima Update Label Daftar Kelas

Sedangkan proses memperbaharui daftar kelas diilustrasikan pada Gambar 2.9.



Gambar 2.9 Proses Update Daftar Kelas

2.5.4 Subsetting untuk Atribut Kategorikal

Split untuk atribut bertipe kategorikal A adalah bentuk dari A ϵ S', dimana S' c S dan S adalah kemungkinan nilai-nilai dari atribut A. Evaluasi untuk semua subset dari S bisa jadi lama prosesnya, apalagi jika jumlah elemen (cardinality) dari S besar.

SLIQ menggunakan pendekatan hybrid untuk mengatasi hal ini. Jika cardinality dari S lebih kecil dari *threshold* (MAXSETSIZE), maka semua sebset dari S akan dievaluasi. Jika terjadi sebaliknya, maka algoritma greedy digunakan untuk mendapatkan subset yang diinginkan. Algoritma greedy dimulai dengan subset kosong S' dan menambahkan satu element dari S ke dalam S' yang akan memberikan split terbaik. Proses ini diulang sampai tidak ada perbaikan lagi dari split. Pendekatan hybrid ini menemukan subset optimal jika S kecil dan memberikan hasil baik pula pada subset yang lebih besar.

2.5.5 Tree Pruning

Strategi pemangkasan pohon yang digunakan pada *SLIQ* berbasiskan pada prinsip *Minimum Description Length (MDL)*. Prinsip *MDL* mengatakan bahwa model terbaik untuk *encoding* data adalah yang dapat meminimalkan jumlah cost untuk mendeskripsikan data dalam model dan cost untuk mendeskripsikan model itu sendiri. Jika M adalah model yang meng-*encode* data D, maka total cost untuk *encoding* tersebut, *cost (M, D),* didefinisikan pada Persamaan 2.6.

$$\cos t(M,D) = \cos t(D \mid M) + \cos t(M) \tag{2.6}$$

Dimana cost (D | M) adalah cost dalam bit, yang digunakan untuk meng-*encoding* data yang diberikan pada model M dan cost (M) adalah cost untuk meng-*encoding* model M. Dalam konteks *decision tree classifier*, model adalah set dari tree yang diperoleh dengan memangkas (*pruning*) *initial decision tree T*, dan data adalah training set S. Tujuan dari *MDL pruning* adalah untuk mencari subtree dari T yang dapat mendeskripsikan training set S paling baik.

Ada dua komponen dari algoritma pruning, yaitu skema *encoding* yang bertujuan untuk menentukan cost dari encoding data dan model, dan algoritma yang digunakan untuk membandingkan bermacam-macam subtree dari T.

2.5.5.1 Data Encoding

Data encoding, merupakan cost yang diperlukan untuk meng-encode atau mendeskripsikan data yang diberikan oleh model tree. Cost untuk meng-*encode* himpunan data latih S dengan sebuah *decision tree* T didefinisikan sebagai jumlah dari semua klasifikasi yang error. Klasifikasi dari sebuah record disebut error jika klasifikasi yang dihasilkan oleh T tidak sama dengan label kelas aslinya dari record tersebut. Jumlah dari proses salah klasifikasi ini dihitung selama fase pembentukan tree berlangsung. Untuk rumus data encoding pada node yang merupakan leaf digunakan persamaan 2.7.

$$C_{leaf}(n) = \sum_i n_i \log \frac{|F_n|}{|F_n^i|} + \frac{k-1}{2} + \log \frac{|F_n|}{2} + log \frac{\pi^{k/2}}{\Gamma(k/2)} \quad (2.7)$$

Dimana F_n adalah frekuensi rekord yang memiliki kelas n, dan F_n^i merupakan frekuensi rekord yang memiliki kelas selain n. Persamaan 2.8 menunjukkan cost data encoding untuk node yang merupakan split dengan atribut numerik. Sedangkan persamaan 2.9 menunjukkan cost data encoding untuk node split dengan atribut ketagorikal.

$$C_{split}(n) = log\,(v-1) \quad (2.8)$$

$$C_{split}(n) = log\,(2^v - 2) \quad (2.9)$$

Dimana v merupakan jumlah nilai yang berbeda dari atribut tersebut.

2.5.5.2 Model Encoding

Skema *encoding* untuk model harus bisa menyediakan cost untuk mendeskripsikan tree dan cost untuk mendeskripsikan pengujian yang digunakan pada tree dalam setiap node.

- Proses *Encoding Tree*. Diberikan sebuah *decision tree*, node pada decision tree bisa berupa sebuah internal node dengan satu atau dua anak atau leaf node. Jumlah bit yang dibutuhkan untuk meng-encode tree bergantung pada struktur tree yang diizinkan. Ada tiga kemungkinan untuk meng-*encoding* tree:

  1. *Code*$_1$: Node diperbolehkan mempunyai 0 atau 2 anak. Hanya ada dua kemungkinan, jadi dibutuhkan satu bit saja untuk meng-encode setiap node-nya.

2. *Code*$_2$: Node bisa tidak mempunyai anak, anak kiri saja, anak kanan saja, atau anak kiri dan kanan. Karena itu dibutuhkan 2 bit untuk meng-encode 4 kemungkinan nilai dari setiap node-nya.
3. *Code*$_3$: Hanya internal node yang dikaji. Dengan demikian setiap node hanya bisa mempunyai anak kanan saja, anak kiri saja, atau anak kiri dan kanan, dengan jumlah bit yang dibutuhkan untuk meng-encode sebanyak log(3) bits.

- Proses *Encoding Split*. Cost untuk meng-encode split tergantung pada tipe dari atribut yang diuji untuk split:
  1. Atribut Numerik: Jika split memenuhi syarat A ≤ v dimana A adalah atribut numerik dan v adalah bilangan real, maka cost encoding untuk tes ini merupakan overhead dari encoding v, sebut saja P. Walaupun nilai dari P harus di optimalkan berdasarkan tiap tes pada decision tree, tetapi diasumsikan nilai konstan 1 untuk keseluruhan tree.
  2. Atribut Kategorikal: untuk tes dengan syarat A ϵ S, dimana A adalah atribut kategorikal dan S adalah subset dari kemungkinan nilai A, maka cost dihitung dalam 2 langkah. Pertama, hitung tiap nilai tes yang digunakan dalam tree, nA_i, untuk setiap atribut kategorikal Ai. Lalu cost dari tes tersebut dihitung sebagai $ln(nA_i)$.

2.5.5.3 Algoritma Pruning

MDL pruning mengevaluasi panjang kode pada tiap decision tree node untuk menentukan apakah dilakukan konversi node menjadi leaf, memangkas anak kiri atau kanan, atau membiarkan node tetap utuh. Penentuan panjang kode C(*n*), ditentukan sesuai dengan Gambar 2.10.

$C_{leaf}(t)=L(t)+Errors_t$, jika t adalah leaf (1)

$C_{both}(t)=L(t)+L_{test}+C(t1)+C(t2)$, jika t mempunyai anak kiri dan kanan (2)

$C_{left}(t)=L(t)+L_{test}+C(t1)+C'(t2)$, jika t hanya punya t1 sebagai anak (3)

$C_{right}(t) = L(t) + L_{test} + C'(t1) + C(t2)$, jika t hanya punya t2 sebagai anak (4)

Gambar 2.10 Penentuan Panjang kode C(*n*) pada *MDL Pruning*

Untuk kasus pruning parsial dimana $t_1$atau t_2 dipangkas, maka record yang terletak di cabang yang dipangkas tersebut akan di-*encode* menggunakan statistik pada parent node. $C'(t_i)$ menunjukkan cost encoding untuk record anak menggunakan statistik pada parent.

Ada tiga strategi pruning:

1. Full (Penuh): Strategi ini hanya menggunakan opsi (1) dan (2) dari Gambar 2.10. Jika $C_{leaf}(t)$ lebih kecil dari $C_{both}(t)$ untuk node t maka kedua anak akan dipangkas dan node ini akan dikonversikan menjadi leaf. Pendekatan ini akan mengkodekan decision tree hanya menggunakan satu bit saja (metode $Code_1$).
2. Partial (Sebagian): Strategi pemangkasan sebagian ini menggunakan semua opsi pada Gambar 2.10. Setiap node dikonversikan ke dalam opsi dengan panjang kode yang paling pendek. Pendekatan ini menggunakan metode kedua untuk meng-*encoding* tree ($Code_2$) yang membutuhkan 2 bit pada setiap node-nya.
3. Hybrid (Gabungan): Metode hybrid memangkas tree dalam 2 fase. Pertama, menggunakan metode Full (Penuh) untuk mendapatkan tree terkecil dan kemudian menggunakan opsi (2), (3), dan (4) dari Gambar 2.10 untuk memangkas tree lebih jauh.

2.6 Metode Evaluasi

Evalusi percobaan pada *classifier* biasanya lebih diutamakan untuk mengukut kefektifannya daripada efisiensinya, yaitu kemampuannya untuk melakukan pengklasifikasian secara benar (Sebastiani, 2005).

Dalam konteks persoalan klasifikasi, perumusan *precision* dan *recall* pada umumnya menggunakan tabel kesuaian hasil kategori. Dimana untuk klasifikasi dengan *multiclass* tabel hasil kategori, digambarkan pada Tabel 2.1.

Tabel 2.1 Tabel Kesesuaian Hasil Kategori

		Predicted Class		
Known Class		**A**	**B**	**C**
	A	tp_A	e_{AB}	e_{AC}
	B	e_{BA}	tp_B	e_{BC}
	C	e_{CA}	e_{CB}	tp_C

Dimana *tp* menyatakan bahwa kelas prediksi dan kelas pada data uji menunjukkan hasil yang sama, sedangkan *e* menunjukkan sebaliknya, yaitu ketidaksamaan antara kelas pada data uji dengan hasil prediksi.

Sedangkan untuk persamaan *Recall* didefinisikan pada Persamaan 2.10.

$$\text{Re}call = tp / (tp + fn) \tag{2.10}$$

Dimana *tp* adalah jumlah dari hasil prediksi yang true positive atau antara *predicted class* dan *known class* sama. Dan *tp* + *fn* adalah jumlah dari dari data yang diuji. Untuk persamaan *Precision*, didefinisikan pada Persamaan 2.11

$$\text{Pr}ecision = tp / (tp + fp) \tag{2.11}$$

Dimana *tp* adalah hasil prediksi yang true positive, dan *fp* adalah prediksi false positive dari kelas. Perlu diingat bahwa untuk klasifikasi dengan multi kelas, Recall dan Precision dihitung untuk masing masing kelas, kemudian dirata-rata.

Dari *Precision* dan *Recall* dapat dihitung *F-Measure* yang merupakan rata-rata. Persamaan F-Measure dapat dilihat pada Persamaan 2.12.

$$F - Measure = 2.\frac{precision.recall}{precision + recall} \tag{2.12}$$

Evaluasi lain yang dilakukan adalah evaluasi akurasi, dimana persamaan akurasi diilutrasikan pada Persamaan 2.12.

$$Akurasi = \frac{Jumlah\,Prediksi Benar}{Total\,Prediksi} \tag{2.13}$$

2.7 Gambaran Umum Data Penelitian

Pada skripsi ini data penelitian yang digunakan adalah data pendaftar untuk aplikasi SIAP PSB jenjang SMK tahun 2008-2010. Dimana dalam kasus klasifikasi yang akan dilakukan dari data penelitian ini yang menjadi kelas adalah :

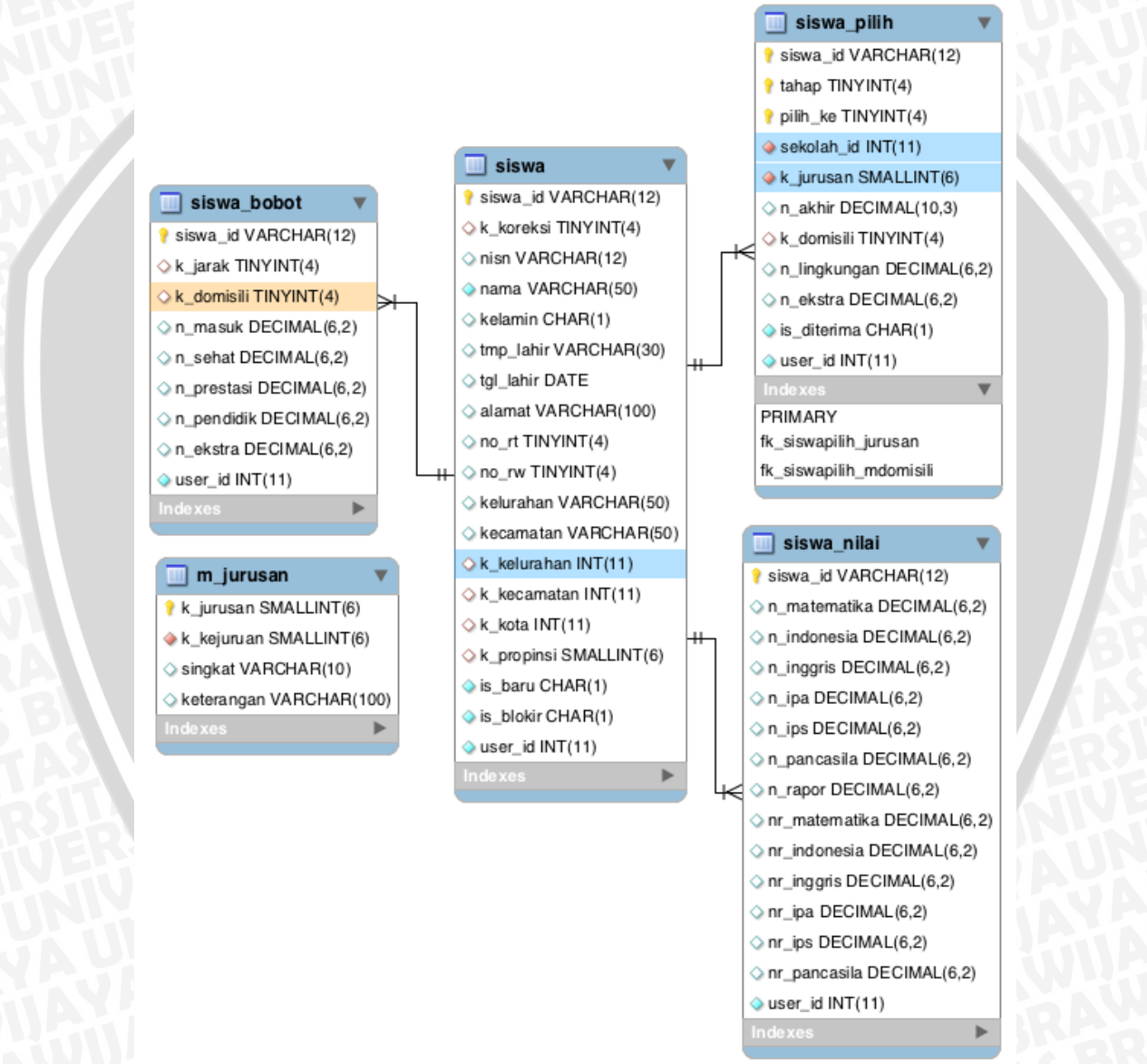
- Jurusan diterima dari pilihan siswa, maksudnya adalah jurusan dimana siswa pendaftar diterima oleh sistem SIAP PSB.

Sedangkan yang menjadi atribut adalah :

- Nilai UN Mata Pelajaran Matematika, dengan rentang antara 0 sampai 10, dan tipe *double*
- Nilai UN Mata Pelajaran Bahasa Indonesia, dengan rentang antara 0 sampai 10, dan tipe *double*
- Nilai UN Mata Pelajaran Bahasa Inggris, dengan rentang antara 0 sampai 10, dan tipe *double*
- Nilai UN Mata Pelajaran IPA, dengan rentang antara 0 sampai 10, dan tipe *double*

- Jenis Kelamin Pendaftar, tipe atribut kategorikal dengan nilai "Laki-laki" dan "Perempuan"
- Lokasi Sekolah Pendaftar, tipe atribut kategorikal dngan nilai "Luar Kota" dan "Dalam Kota"

Untuk jumlah data kurang lebih 4.000 siswa yang diteirma oleh sistem pertahunnya. Adapun struktur dan relasi dari tabel yang digunakan pada penelitian ini terdapat pada basis data PSB terdapat pada gambar 2.11.



Gambar 2.11 Struktur Tabel Basis Data PSB

BAB III

METODE DAN PERANCANGAN SISTEM

3.1 Tahapan pengembangan

Tahapan dalam pengembangan sistem implementasi Algoritma *SLIQ* ini yaitu:

1. Membangun *engine* Decision Tree Builder dengan Algoritma SLIQ.
2. Membangun aplikasi evaluasi *decision tree*.
3. Mengumpulkan data
4. Evaluasi aplikasi yang telah dibangun
5. Membangun Antarmuka
6. Revisi aplikasi

3.2 Kebutuhan Sistem

Dalam membangun sistem rekomendasi tag ini dibutuhkan perangkat keras dan perangkat lunak tertentu.

3.2.1 Perangkat keras

Kebutuhan perangkat keras yang digunakan oleh peneliti dalam melakukan penelitian ini adalah satu buah komputer dengan spesifikasi minimum sebagai berikut :

Penggunaan : Pengembangan
Jenis : Notebook /Desktop
Prosesor : Intel Core 2 Duo CPU dengan kecepatan minimal 2.26GHz
Memory : 5GB
Hardisk : 250 GB
Network : WiFi atau Ethernet

3.2.2 Perangkat lunak

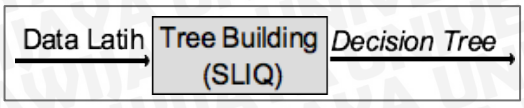
Perangkat lunak yang digunakan dalam pembuatan dan pengujian sistem implementasi algoritma *SLIQ,* adalah Sistem Operasi Mac OS X 10.6.8, dengan aplikasi tambahan JDK versi 1.6.0 untuk pengembangan dan JRE versi SE 6 untuk pengujian sistem.

3.3 Data Penelitian

Data penelitian yang digunakan bersumber dari data pendaftar beserta pilihan jurusannya yang diterima pada aplikasi SIAP PSB tahun 2008 - 2010 untuk jenjang SMK. Sedangkan data yang digunakan data pengujian adalah data pendaftar beserta pilihan jurusannya yang diterima pada aplikasi SIAP PSB tahun 2011. Dalam penelitian ini tabel yang dijadikan referensi adalah tabel yang digunakan adalah *siswa, siswa_pilih, siswa_nilai, siswa_bobot, m_jurusan* seperti diilustrasikan pada gambar 2.11. Dari tabel-tabel tersebut akan didapat atribut pendaftar Nilai UN (Matematika, Bahasa Indonesia, Bahasa Inggris, IPA), Jenis Kelamin, Lokasi Pendaftar (Luar Kota / Dalam Kota), beserta nama jurusan yang diterima oleh sistem.

3.4 Perancangan *Engine Decision Tree Builder* dengan Algoritma *SLIQ*

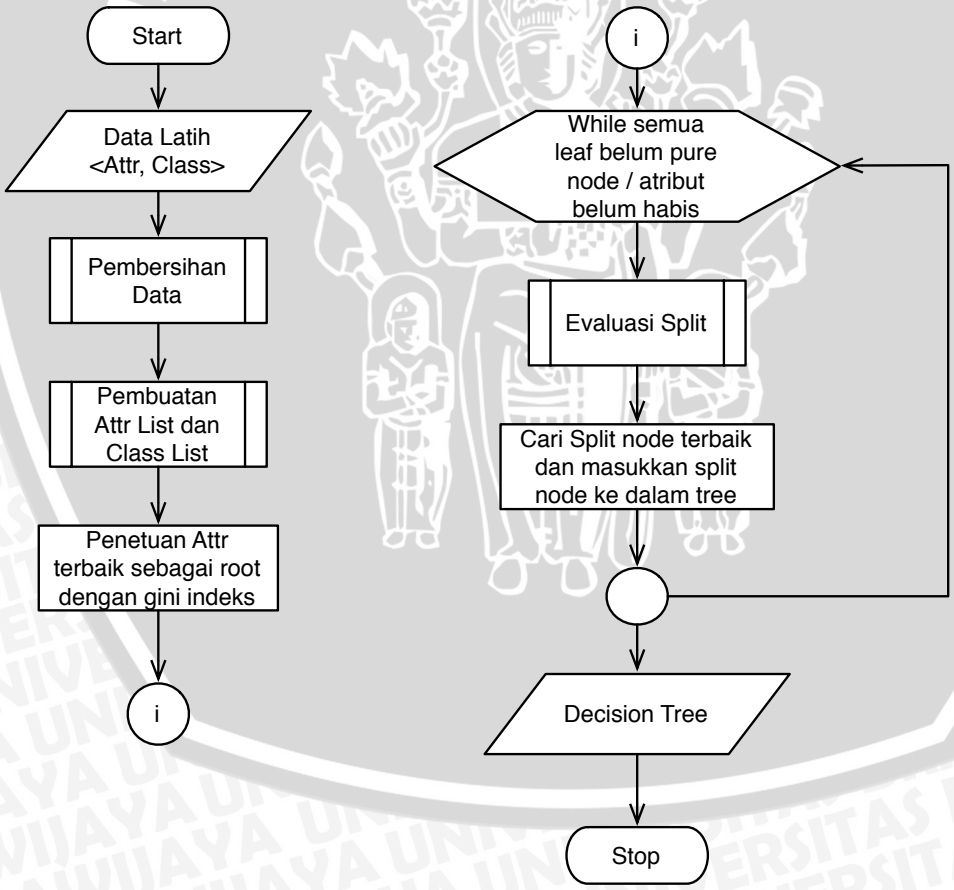
Engine Decision Tree Builder, merupakan layanan yang terdapad dalam sistem, bertujuan untuk menghasilkan *decision tree* yang pada skripsi ini merupakan model *classifier.*Decision tree pada skripsi ini dibangun dengan algoritma SLIQ. Dalam implementasi nya *engine* ini menerima masukan data latih, yaitu data pendaftar aplikasi SIAP PSB tahun 2008-2011 jenjang SMK. Secara umum proses dari *engine decision tree builder* ini diilustrasikan pada gambar 3.1.



Gambar 3.1 Digram Proses Tree Building

3.4.1 Modul *SLIQ Decision Tree Builder*

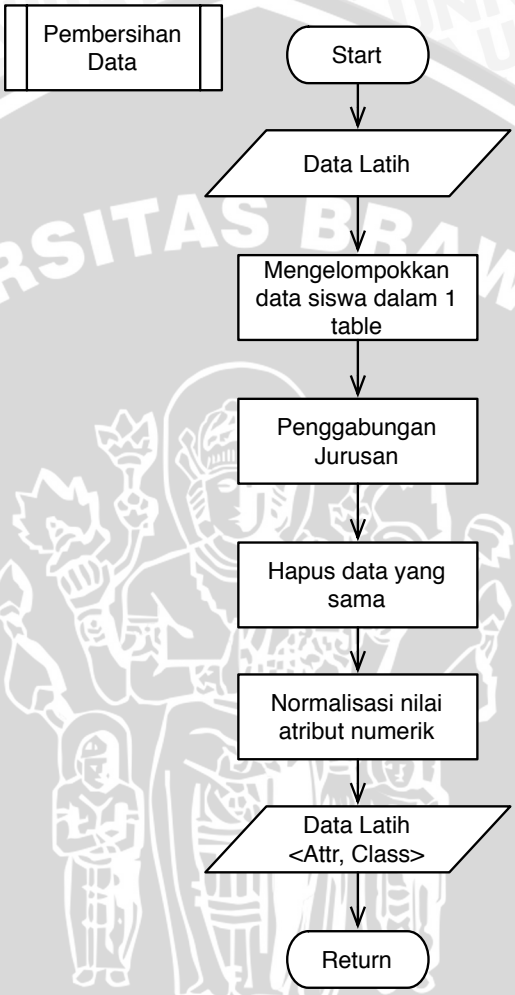
Modul ini berfungsi untuk menghasilkan *decision tree* sebagai model *classifier*. Input dari modul ini adalah data latih siswa pendaftar pada aplikasi SIAP PSB tahun 2008-2010 jenjang SMK. *Decision tree* yang dihasilkan menggunakan algoritma SLIQ ini nantinya akan digunakan pada modul evaluator untuk mengevaluasi data uji siswa pendaftar pada aplikasi SIAP PSB 2011 jenjang SMK. Algoritma dasar dari SLIQ dalam membangun decision tree diilustrasikan pada diagram alir gambar 3.2.



Gambar 3.2 Diagram Alir Algoritma SLIQ

3.4.1.1 Pembersihan Data

Demi menjaga kualitas data latih, yang merupakan masukan untuk *decision tree builder*, beberapa persyaratan dan batasan perlu dipenuhi. Diagram Alir untuk pembersihan data terlihat pada gambar 3.3.



Gambar 3.3 Diagram Alir Pembersihan Data

Beberapa kegiatan yang dilakukan terhadap data siswa pendaftar dan pilihannya antara lain :

1. Mengelompokkan data pendaftar (no ujian, nama siswa) menjadi satu tabel dengan atribut pendaftaran (nilai UN : Matematika, Bahasa Indonesia, Bahasa Inggris, IPA; Jenis kelamin; Lokasi pendaftar) yang awalnya terletak pada tabel yang berbeda. Hal ini dilakukan untuk memudahkan *decision tree builder* dalam melakukukan pembentukan

pohon. Tabel ini merupakan daftar atribut yang menjadi acuan dalam pembentukan pohon.

2. Melakukan penggabungan Jurusan untuk semua SMK pilihan dari 2008-2011. Seperti diterangkan pada bab 3.3, bahwa sekolah pilihan tidak diperhatikan dalam skripsi ini, melainkan jurusan yang menjadi acuan. Penggabungan jurusan merupakan proses dimana semua nama jurusan yang sama dari seluruh SMK peserta SIAP PSB dijadikan satu.
3. Mencari data kembar, data kembar yang dimaksud pada data peneltian yang digunakan pada skripsi ini adalah, jika semua atribut bernilai sama dan kelas yang dihasilkan juga sama pada dua atribut atau lebih, maka hanya akan diambil satu rekord saja. Sebagai contoh jika terdapat data dua siswa pendaftar yang berbeda namun menunjukkan bahwa nilai UN matematika, bahasa Indonesia, bahasa Inggris, IPA, jenis kelamin dan lokasi pendaftar sama persis, dan diterima pada jurusan yang sama (tanpa memperhatikan sekolah pilihan) maka dua data ini dianggap kembar, dan dalam penggunaannya nanti hanya akan digunakan sati dari dua rekord ini.
4. Menetapkan Range nilai atribut numerikal, semenjak atribut numerikal pada data penelitian ini adalah nilai UN matematika, bahasa Indonesia, bahasa Inggris, IPA maka nilai ini akan dinormalisasi dengan rentang 1-10 dan pembulatan maksimal 2 digit dibelakang koma.

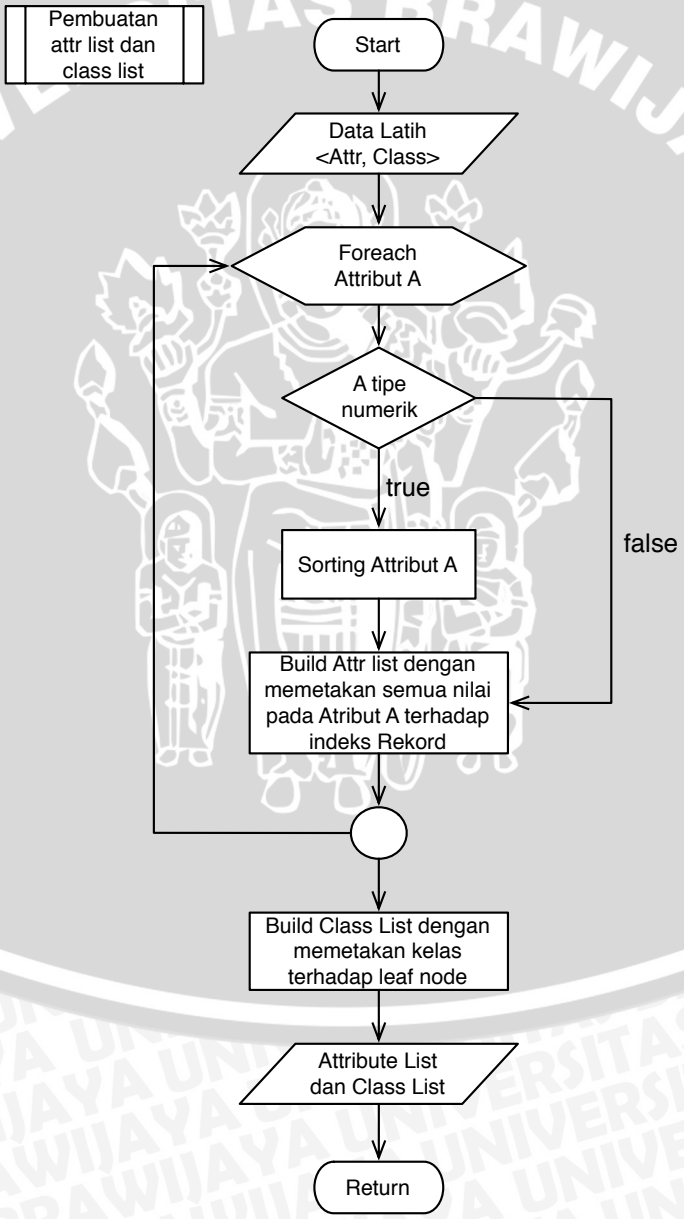
3.4.1.2 Pembuatan Attribute List dan Class List

Pembuatan *Attribute List* dan *Class List* merupakan langkah awal dalam pembuatan *decision tree* dengan algoritma SLIQ. Attribute list merupakan sebuah daftar yang menampung nilai atrribut beserta index dari rekord tersebut berada dalam data latih.

1. Atribut numerik, untuk pembentukan atribut list bagi atribut numerik terlebih dahulu semua nilai dari semua atribut teresebut diurutkan dari nilai terkecil ke besar. Setiap baris dari daftar atribut numerik dikorelasikan dengan index dimana nilai atribut tersebut terletak pada rekord data latih.

2. Atribut kategorikal, untuk pembentukan atribut list pada atribut kategorikal tidak perlu diurutkan seperti atribut numerik. Setiap bari pada atribut kategorikal juga tetap dikorelasikan dengan index dimana nilai atribut tersebut terletak pada rekord data latih.

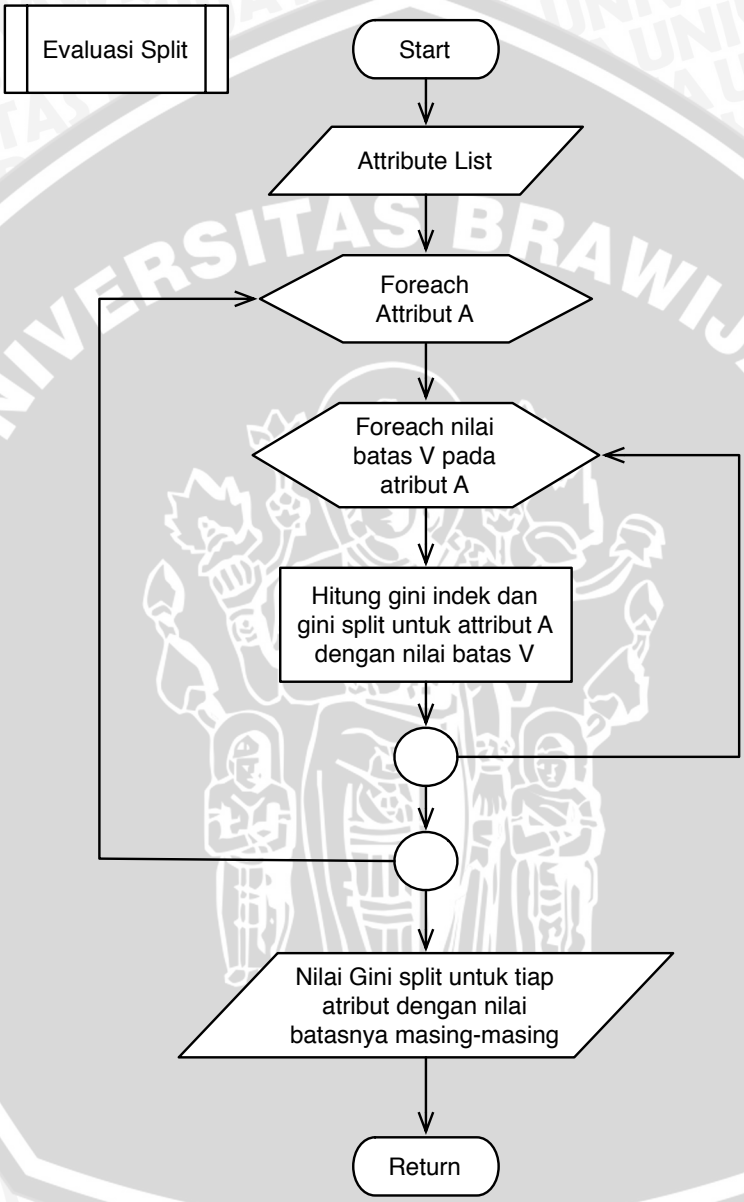
Class List merupakan daftar dimana nama kelas di korelasikan dengan *leaf node,* yang terurut sesuai dengan urutan rekord data latih. Sedangkan inisialisasi *leaf node* pertama adalah root atau N1. Diagram alir untuk pembuatan attribut list dan class list digambarkan pada gambar 3.4.



Gambar 3.4 Diagram Alir Pembuatan Attribute List dan Class List

3.4.1.3 Evaluasi Split

Evaluasi Split merupakan proses dimana sistem mencari split terbaik untuk sebuah atribut. Diagram alir untuk evaluasi split dijelaskan pada Gambar 3.5.



Gambar 3.5 Diagram Alir Evaluasi Split

Pada Gambar 3.5, proses penghitungan gini indeks dilakukan dengan menggunakan Persamaan 2.4 dan 2.5.

3.5 Perancangan Struktur Data

Pada tahap ini akan dijelaskan bagaimana struktur data yang digunakan pada sistem implementasi Algoritma SLIQ ini. Struktur data akan dibagi menjadi dua, yaitu struktur data pada kode yang nantinya akan berbentuk *class* dan disimpan dan diproses dalam memory serta struktur data yang disimpan dalam *harddisk* dalam bentuk tabel pada basisdata.

3.5.1 Struktur Data pada Memory

Struktur data pada memory yang dimaksud adalah dimana sebuah struktur data digunakan dalam kode aplikasi. Baik penyimpanan maupun pengaksesan (I/O) dilakukan dalam memory, hal ini agar memudahkan dan mempercepat aplikasi mengakses data. Bentuk nyata dari struktur data ini pada kode aplikasi adalah sebuah *class*.

3.5.1.1 Strukur Data Attribute List

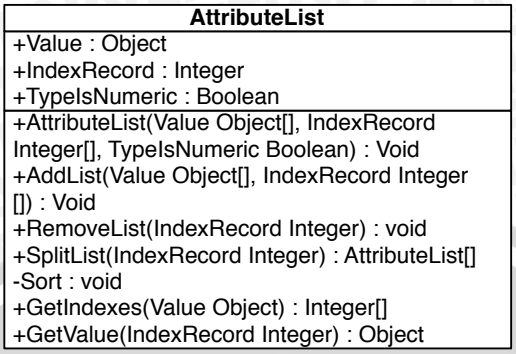
Attribute list dibentuk dalam sebuah *class* yang memiliki properti :

- Value : menyimpan nilai dari atribut.
- IndexRecord : menyimpan index pada rekord dalam data latih dimana value itu ditemukan.
- Type : untuk menyinpan apakan attribute numerikal atau kategorikal

Sedangkan untuk operasi yang dapat dilakukan oleh *class* attrbite list adalah :

- AttributeList : Construct dalam pembuatan attribute list
- AddList : untuk menambahkan item ke dalam list
- RemoveList : untuk menghapus item pada list
- SplitList : untuk membagi list menjadi dua bagian
- Sort : untuk mengurutkan value pada attribute list
- GetIndexes : untuk mendapatkan index record dimana sebuah value ditemukan

- GetValue : untuk mendapatkan value dari attribute list pada index rekord tertentu.



Gambar 3.6 Class Diagram *AttributeList*

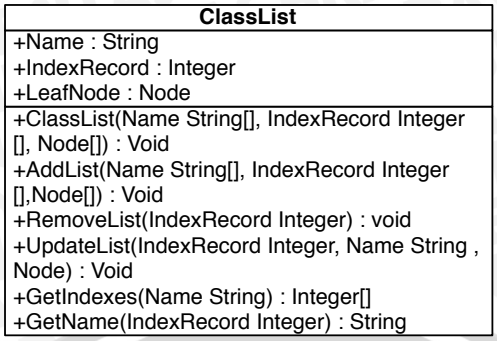
3.5.1.2 Strukur Data Class List

Class list menampung daftar kelas pada rekord dari data latih, *class* ClassList memiliki properti :

- IndexRecord : index dari rekord dimana kelas ditemukan
- Name : merupakan nama kelas
- LeafNode : merupakan leaf node dimana rekord ini berada pada *decision tree*

Method yang terdapat pada *class* ini :

- ClassList : konstruktor untuk membuat *class list*
- GetName : untuk mendapatkan nama kelas dari index rekord
- GetIndexes : untuk mendapatkan index rekord dari daftar kelas dengan nama kelas tertentu
- UpdateList : untuk memperbaharui list
- AddList : untuk menambahkan item pada Class List
- RemoveList : untuk menghapus item pada class list



Gambar 3.7 Class Diagram *ClassList*

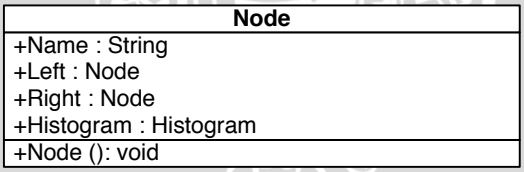
3.5.1.3 Strukur Data Node

Class Node memiliki properti :

- Name : merupakan nama unik dari node
- Left : merupakan cabang sebelah kiri dari node itu sendiri
- Right : merupakan cabang sebelah kanan
- Histogram : merupakan tabel histogram dari setiap node

Operasi yang ada pada node :

- Node : Konstruktor dalam membentuk inisialisasi node

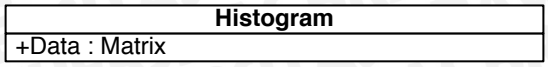


Gambar 3.8 Class Diagram *Node*

3.5.1.4 Strukur Data Histogram

Histogram menyimpan kelas dan korelasi nya dengan frekuensi ditemukan pada data latih setelah dilakukan *split*. Kelas Histogram memiliki properti :

- Data : merupakan data dari tabel histogram



Gambar 3.9 Class Diagram *Histogram*

3.5.1.5 Strukur Data Tree

Kelas Tree menampun pohon keputusan yang dibuat. Propert yang dimiliki kelas ini :

- Root : Node yang merupakan root dari tree

Method yang tersedia adalah :

- Add : untuk menambahkan node pada tree
- ToArrayList : traversing tree menjadi arrayList

Gambar 3.10 Class Diagram *Tree*

3.5.2 Struktur Data pada Tabel Basisdata

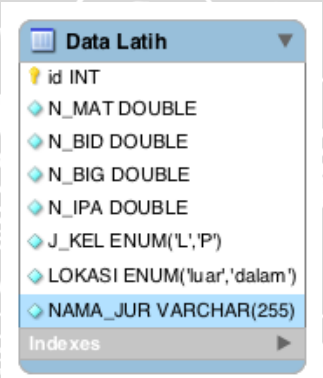
Selanjutnya adalah Struktur data yang disimpan pada *harddisk,* dimana dalam sistem ini diimplementasikan dalam bentuk tabel yang disimpan dalam sebuah basisdata. Untuk penyimpanan dan pengaksesan (*I/O*) dilakukan dengan SQL Queries.

3.5.2.1 Tabel Data Latih

Tabel data latih menyimpan data siswa pendaftar beserta sekolah yang diterima oleh aplikasi SIAP PSB.

Tabel 3.1 Tabel Data Latih

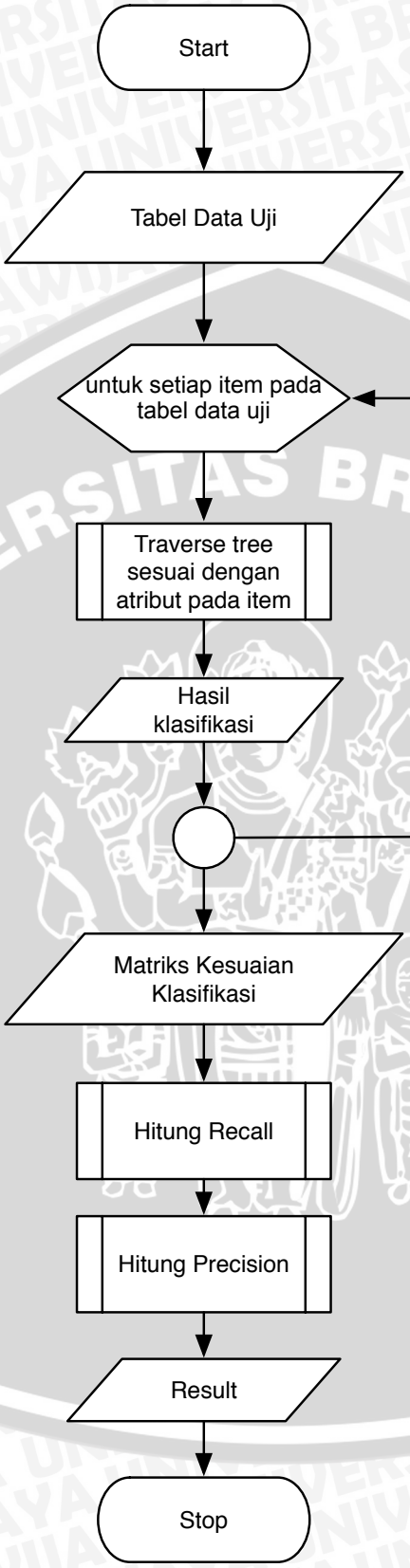
Field	Jenis	Keterangan
id	**int(10)**	
N_MAT	double	
N_BID	double	
N_BIG	double	
N_IPA	double	
J_KEL	enum('L','P')	
LOKASI	enum('luar', 'dalam')	
NAMA_JUR	varchar(255)	



Gambar 3.11 Sruktur Tabel

3.6 Perancangan *Evaluator Decision Tree*

Evaluator bertujuan untuk melakukan pengujian decision tree dalam hal akurasi. Aplikasi ini dibuat dengan tujuan memudahkan penulis dalam menyusun hasil penelitian. Evaluator untuk decision tree ini nantinya akan menerima data uji yang sudah diketahui klasifikasinya kemudian dibandingkan dengan hasil klasifikasi dari decision tree yang dihasilkan melalui algoritma SLIQ. Gambar 3.12 menunjukkan diagram alir dari evaluator Decision Tree.



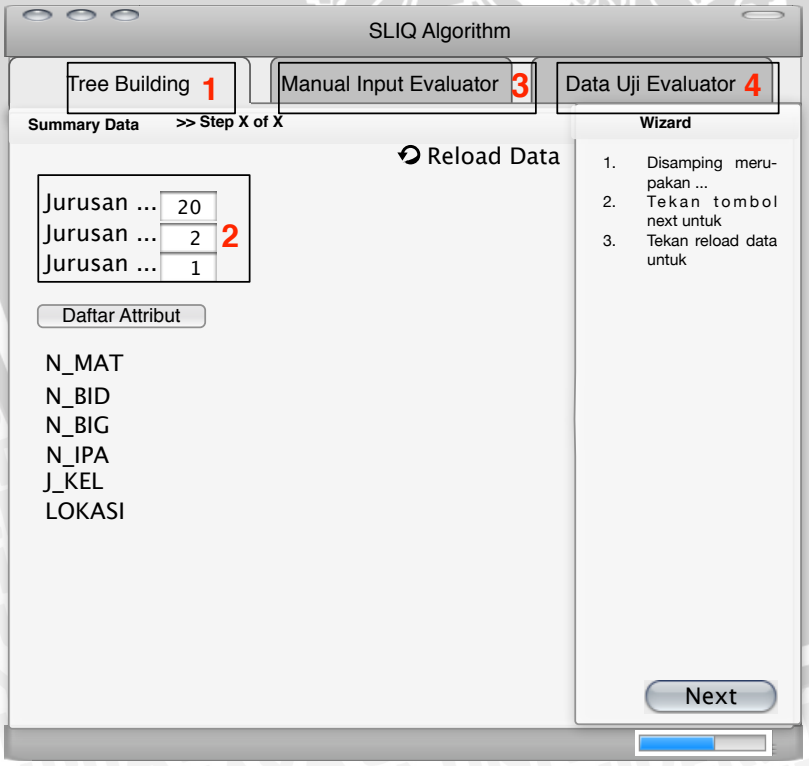
Gambar 3.12 Diagram Alir Evaluator dengan Data Uji

3.7 Perancangan Antarmuka

Sistem dibuat dengan antar muka berbasis aplikasi dekstop, dan diarahkan ke *wizard application*.

3.7.1 Modul Tree Building

Antarmuka untuk modul tree building, diawali dengan penyajian ringakasan data dari tabel yang ada pada konfigurasi. Ringkasan data menampilkan jumlah rekord, jumlah dan nama atribut beserta jenisnya, jumlah dan nama kelas pada klasiifikasi atau range nilai atribut jika numerik. Pada langkah kedua sistem akan melakukan tree building yang kemudian decision tree yang terbentuk akan ditampilkan. Gambar 3.12 mengilustrasikan modul tree building pada langkah pertama.

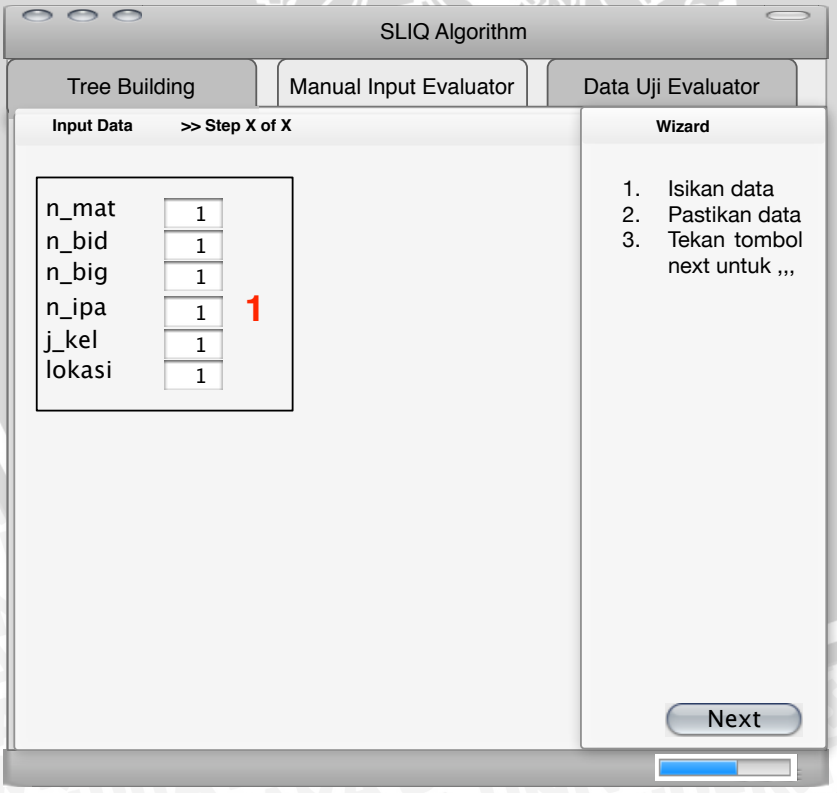


Gambar 3.13 Antarmuka Modul Tree Building

Pada Gambar 3.13, nomor 1 menunjukkan menu untuk modul *tree building*, nomor 2 adalah nama jurusan yang tersedia pada data beserta jumlahnya, yang tidak bisa disunting, sedangkan nomor 3 dan 4 masing-masing menunjukkan menu untuk modul evaluator dengan manual input dan evaluator dengan data uji.

3.7.2 Modul Manual Input Evaluator

Modul manual input evaluator memiliki antar muka awal yang mengijinkan pengguna memasukkan nilai dari atribut dalam klasifikasi. Pada langkah selanjutnya sistem akan menampilkan hasil klasifikasi dari data yang dimasukkan oleh pengguna kedalam sistem berdasarkan *decision tree* yang terbentuk pada modul sebelumnya.



Gambar 3.14 Antarmuka Modul Evalutor

Nomor 1 pada gambar 3.14 merupakan masukan untuk setiap atribut yang akan digunakan untuk menguji *tree*.

3.7.3 Modul Data Uji Evaluator

Modul Data Uji Evaluator, memiliki antar muka yang mirip dengan Modul Tree Building pada Gambar 3.14, dimana sistem menampilkan ringkasan data dari tabel yang digunakan sebagai data uji. Namun yang membedakan adalah pada langkah selanjutnya sistem tidak menampilkan *decision tree*, namun menampilkan matriks kesesuaian klasifikasi dan perhitungan *precision* dan *recall* per-kelas dalam klasifikasi.

3.8 Perancangan Hasil Penelitian

Pengujian yang akan dilakukan bertujuan untuk mengukur tingkat akurasi dari *decision tree* pengklasifikasi yang dibuat. Parameter akurasi yang digunakan yaitu *precission* dan *recall. Precission* mengacu pada seberapa tepat hasil klasifikasi dan *recall* mengacu pada seberapa lengkap dari hasil klasifikasi yang benar dibandingkan keseruhan data uji.

Pengukuran pada Tabel 3.2, menunjukaan *precision* dan *recall* dikorelasikan dengan data latih calon siswa SMK pertahun. Precision dan recall dihitung dari pengujian dengan data pada tahun itu sendiri dan kemudian data pada tahun 2011.

Tabel 3.2 Pengukuran Precision, Recall

No	Data latih	Uji Data latih		Uji Data 2011	
		Precission	*Recall*	*Precission*	*Recall*
1	2008				
2	2009				
3	2010				
4	2008 – 2010				

Tabel 3.3 menunjukkan F-Measure yang didapat dari rata-rata precision dan recall.

Tabel 3.3 Pengukuran F-Measure

No	Data latih	Uji Data latih	Uji Data 2011
		F-Measure	*F-Measure*
1	2008		
2	2009		
3	2010		
4	2008 – 2010		

Pengukuran pada Tabel 3.4 menunjukkan akurasi dari algoritma *SLIQ* dalam pengklasifikasian dibandingkan dengan N data latih.

Tabel 3.4 Pengukuran Akurasi

No	Data latih	Akurasi	
		Uji Data latih	Uji Data 2011
1	2008		
2	2009		
3	2010		
4	2008 – 2010		

Pengujian selanjutnya adalah dengan membandingkan jumlah data latih terhadap hasil pengujian *precision*, *recall*, *F-Measure*, dan akurasi yang ditunjukkan pada Tabel 3.5.

Tabel 3.5 Pengujian Jumlah Data Latih

No	N Data latih	Hasil Uji (500 Data)		
		Precision	Recall	F-Measure
1	500			
2	1000			
3	2000			
4	3000			

3.9 Perhitungan Manual

Penerapan alortima SLIQ akan dijelaskan dalam perhitungan manual dengan data latih pada tabel 3.4.

Tabel 3.6 Contoh Data Latih Klasfikasi Jurusan

n_mat	n_bid	n_big	n_ipa	j_kel	lokasi	jurusan
9.25	6.8	6.6	8.5	P	DALAM	Adm Perkantoran
8.75	6.6	6.6	7.5	P	DALAM	Adm Perkantoran
8.5	6.8	6.6	8.5	L	DALAM	Adm Perkantoran
8.5	6.8	6.2	7.5	P	DALAM	Adm Perkantoran
8.75	6.2	6.6	8.25	P	DALAM	Adm Perkantoran
9	8.6	8.4	8.5	P	LUAR	Adm Perkantoran
9	8.2	8.6	8.5	P	LUAR	Adm Perkantoran
8.75	9	9	8.25	P	LUAR	Adm Perkantoran
9	9.6	8.6	7	P	LUAR	Adm Perkantoran
9	7	8.4	7.25	P	LUAR	Adm Perkantoran
9.75	8.8	9.8	9.25	L	LUAR	Teknik Komp dan Jaringan
9	9.6	8.8	8.5	L	LUAR	Teknik Komp dan Jaringan
9.25	8.2	9	8.75	L	LUAR	Teknik Komp dan

						Jaringan
9	8.4	9.6	8	L	DALAM	Teknik Komp dan Jaringan
8.5	7.6	9	9.75	L	DALAM	Teknik Komp dan Jaringan
8	7.2	6.2	6.5	P	DALAM	Teknik Komp dan Jaringan
10	8.4	6	7.5	L	DALAM	Teknik Komp dan Jaringan
7.75	8.2	8.6	6.75	L	DALAM	Teknik Komp dan Jaringan
8	8	7.6	7.75	P	DALAM	Teknik Komp dan Jaringan
8.5	8.2	6	7	L	DALAM	Teknik Komp dan Jaringan
10	9.2	9	9	L	DALAM	Teknik Komp dan Jaringan
9	7.2	8	6.5	L	DALAM	Teknik Komputer dan Jaringan
8.25	8.4	8.8	6.75	L	LUAR	Teknik Audio Video
8.25	8	6.6	8	P	LUAR	Teknik Audio Video
8	7.2	8.8	7	L	LUAR	Teknik Audio Video
8.25	8	8.4	8.25	L	LUAR	Teknik Audio Video
9	9.4	8.2	9	L	LUAR	Teknik Audio Video
9	9.4	8.6	8.25	L	LUAR	Teknik Audio Video
9.25	7.2	7.6	7.75	L	DALAM	Teknik Audio Video
8	8	6	7.25	L	DALAM	Teknik Audio Video
7.75	7.4	5.6	7	L	DALAM	Teknik Audio Video

Langkah 1. Membuat Attribute List dan Class List

Attribute list terdiri dari 2 *field,* yaitu nilai atribut dan index dari daftar kelas. Dengan contoh data latih tabel 3.6 maka akan terbentuk attribut list seperti pada tabel 3.7, 3.8, 3.9, 3.10, 3.11, 3.12.

Tabel 3.7 Atttribute List N_MAT

n_mat	indeks
7.75	18
7.75	31
8	16
8	19
8	25
8	30
8.25	23
8.25	24
8.25	26
8.5	3
8.5	4
8.5	15
8.5	20
8.75	2
8.75	5
8.75	8
9	6
9	7
9	9
9	10
9	12
9	14
9	22
9	27
9	28
9.25	1
9.25	13
9.25	29
9.75	11

Tabel 3.8 Atttribute List N_BID

n_bid	indeks
6.2	5
6.6	2
6.8	3
6.8	4
6.8	1
7	10
7.2	16
7.2	25
7.2	22
7.2	29
7.4	31
7.6	15
8	19
8	30
8	24
8	26
8.2	18
8.2	20
8.2	7
8.2	13
8.4	23
8.4	14
8.4	17
8.6	6
8.8	11
9	8
9.2	21
9.4	27
9.4	28

10	17
10	21

9.6	9
9.6	12

Tabel 3.9 Atttribute List N_BIG

n_big	indeks
5.6	31
6	30
6	20
6	17
6.2	4
6.2	16
6.6	5
6.6	2
6.6	3
6.6	1
6.6	24
7.6	29
7.6	19
8	22
8.2	27
8.4	10
8.4	26
8.4	6
8.6	18
8.6	7
8.6	28
8.6	9
8.8	25
8.8	23
8.8	12
9	15

Tabel 3.10 Atttribute List N_IPA

n_ipa	indeks
6.5	16
6.5	22
6.75	18
6.75	23
7	31
7	20
7	9
7	25
7.25	30
7.25	10
7.5	17
7.5	4
7.5	2
7.75	29
7.75	19
8	24
8	14
8.25	5
8.25	26
8.25	28
8.25	8
8.5	3
8.5	1
8.5	6
8.5	7
8.5	12

9	13
9	8
9	21
9.6	14
9.8	11

8.75	13
9	27
9	21
9.25	11
9.75	15

Tabel 3.11 Atttribute List J_KEL

j_kel	indeks
P	1
P	2
L	3
P	4
P	5
P	6
P	7
P	8
P	9
P	10
L	11
L	12
L	13
L	14
L	15
P	16
L	17
L	18
P	19
L	20
L	21
L	22
L	23

Tabel 3.12 Atttribute List Lokasi

lokasi	indeks
DALAM	1
DALAM	2
DALAM	3
DALAM	4
DALAM	5
LUAR	6
LUAR	7
LUAR	8
LUAR	9
LUAR	10
LUAR	11
LUAR	12
LUAR	13
DALAM	14
DALAM	15
DALAM	16
DALAM	17
DALAM	18
DALAM	19
DALAM	20
DALAM	21
DALAM	22
LUAR	23

P	24
L	25
L	26
L	27
L	28
L	29
L	30
L	31

LUAR	24
LUAR	25
LUAR	26
LUAR	27
LUAR	28
DALAM	29
DALAM	30
DALAM	31

Sedangkan Tabel 3.13 menunjukkan Class List.

Tabel 3.13 Class List

jurusan	leaf
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Administrasi Perkantoran	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1

Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Komputer dan Jaringan	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1
Teknik Audio Video	N1

Langkah 2. Membuat Histogram untuk setiap atribut.

Histogram adalah tabel yang menyimpan frekuensi dari kemunculan kelas dalam atribut tersebut. Contoh histogram untuk atribut n_mat, dengan nilai batas 7.75, 8, 8.25, 8.5, 8.75, 9, 9.25, 9.75 diilustrasikan pada Tabel 3.14, 3.15, 3.16, 3.17, 3.18, 3.19, 3.20 dan 3.21.

Tabel 3.14 Histogram n_mat nilai batas 7.75

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 7.75)	0	1	1
R (> 7.75)	10	11	8

Tabel 3.15 Histogram n_mat nilai batas 8

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 8)	0	3	3
R (> 8)	10	9	6

Tabel 3.16 Histogram n_mat nilai batas 8.25

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 8.25)	0	3	6
R (> 8.25)	10	9	3

Tabel 3.17 Histogram n_mat nilai batas 8.5

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 8.5)	2	5	6
R (> 8.5)	8	7	3

Tabel 3.18 Histogram n_mat nilai batas 8.75

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 8.75)	5	5	6
R (> 8.75)	5	7	3

Tabel 3.19 Histogram n_mat nilai batas 9

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 9)	9	8	8
R (> 9)	1	4	1

Tabel 3.20 Histogram n_mat nilai batas 9.25

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 9.25)	10	9	9
R (> 9.25)	0	3	0

Tabel 3.21 Histogram n_mat nilai batas 9.75

n_mat	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L (<= 9.75)	10	10	9
R (> 9.75)	0	2	0

Sedangkan untuk histogram dengan atribut kategorikal, dicontohkan dengan atribut jenis kelamin seperti pada tabel 3.22.

Tabel 3.22 Histogram j_kel

j_kel	Administrasi Perkantoran	Teknik Komputer dan Jaringan	Teknik Audio Video
L	1	11	9
P	9	2	1

Langkah 3. Menghitung nilai *gini index* untuk memilih atribut terbaik dan split node.

Untuk setiap atribut dihitung nilai *gini indeks* dengan menggunakan persamaan 2.4 dan 2.5. Contoh perhitungan nilai *gini indeks* pada atribut n_mat dengan nilai batas 7.75 dijelaskan sebagai berikut.

$$N_1 = 1 - \left(\left(\frac{0}{2}\right)^2 + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right) = 0.5$$

$$N_2 = 1 - \left(\left(\frac{10}{29}\right)^2 + \left(\frac{11}{29}\right)^2 + \left(\frac{8}{29}\right)^2\right) = 0.661117717$$

$$Gini(T) = \left(\frac{2}{31}\right).0.5 + \left(\frac{29}{31}\right).0.661117717 = 0.650723026$$

Perhitungan *gini indeks* juga dilakukan untuk nilai batas lainnya yaitu 8, 8.25, 8.5, 8.75, 9, 9.25, 9.75. Begitu pula dengan atribut n_bid, n_big, n_ipa, j_kel dan lokasi. Kemudian dicari atribut dengan nilai *gini* terkecil. Tabel 3.23 menunjukkan hasil perhitungan gini indeks untuk semua atribut.

Tabel 3.23 Hasil Gini Indeks

Atribut	*Gini*
n_mat	0.560117302
n_bid	0.495483871
n_big	0.587956989
n_ipa	0.607444169
j_kel	0.49989899
lokasi	0.620493359

Dari Tabel 3.23 menunjukkan bahwa atribut n_bid memiliki nilai *gini indeks* terkecil, sehingga atribut ini dipilih sebagai split node. Nilai gini dari n_bid terkecil didapat dari nilai batas = 7. Selanjutnya adalah menentukan subset untuk n_bid <=7 dan n_bid >7. Tabel 3.24 dan 3.25 menunjukkan data subset pada atribut n_bid dengan nilai batas = 7.

Tabel 3.24 Tabel data subset n_bid <= 7

n_mat	n_bid	n_big	n_ipa	j_kel	lokasi	jurusan
8.5	6.8	6.2	7.5	P	DALAM	Adm Perkantoran
8.5	6.8	6.6	8.5	L	DALAM	Adm Perkantoran
8.75	6.2	6.6	8.25	P	DALAM	Adm Perkantoran
8.75	6.6	6.6	7.5	P	DALAM	Adm Perkantoran
9.25	6.8	6.6	8.5	P	DALAM	Adm Perkantoran
9	7	8.4	7.25	P	LUAR	Adm Perkantoran
8.5	6.8	6.2	7.5	P	DALAM	Adm Perkantoran

Tabel 3.25 Tabel data subset n_bid > 7

n_mat	n_bid	n_big	n_ipa	j_kel	lokasi	jurusan
9	8.4	9.6	8	L	DALAM	Teknik Komputer dan Jaringan
8.5	7.6	9	9.75	L	DALAM	Teknik Komputer dan Jaringan
8	7.2	6.2	6.5	P	DALAM	Teknik Komputer dan Jaringan
10	8.4	6	7.5	L	DALAM	Teknik Komputer dan Jaringan
7.75	8.2	8.6	6.75	L	DALAM	Teknik Komputer dan Jaringan
8	8	7.6	7.75	P	DALAM	Teknik Komputer dan Jaringan
8.5	8.2	6	7	L	DALAM	Teknik Komputer dan Jaringan
10	9.2	9	9	L	DALAM	Teknik Komputer dan Jaringan

9	7.2	8	6.5	L	DALAM	Teknik Komputer dan Jaringan
9.25	7.2	7.6	7.75	L	DALAM	Teknik Audio Video
8	8	6	7.25	L	DALAM	Teknik Audio Video
7.75	7.4	5.6	7	L	DALAM	Teknik Audio Video
9	8.6	8.4	8.5	P	LUAR	Administrasi Perkantoran
9	8.2	8.6	8.5	P	LUAR	Administrasi Perkantoran
8.75	9	9	8.25	P	LUAR	Administrasi Perkantoran
9	9.6	8.6	7	P	LUAR	Administrasi Perkantoran
9.75	8.8	9.8	9.25	L	LUAR	Teknik Komputer dan Jaringan
9	9.6	8.8	8.5	L	LUAR	Teknik Komputer dan Jaringan
9.25	8.2	9	8.75	L	LUAR	Teknik Komputer dan Jaringan
8.25	8.4	8.8	6.75	L	LUAR	Teknik Audio Video
8.25	8	6.6	8	P	LUAR	Teknik Audio Video
8	7.2	8.8	7	L	LUAR	Teknik Audio Video
8.25	8	8.4	8.25	L	LUAR	Teknik Audio Video
9	9.4	8.2	9	L	LUAR	Teknik Audio Video
9	9.4	8.6	8.25	L	LUAR	Teknik Audio Video

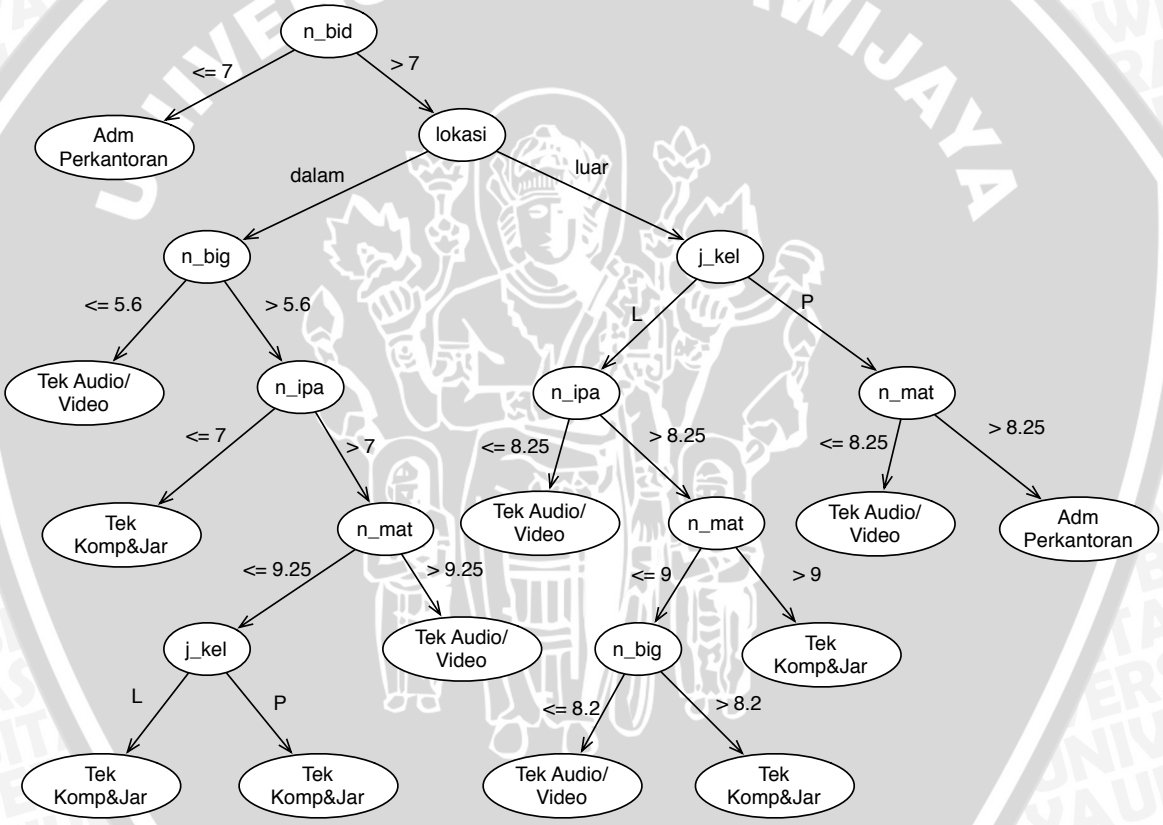
Langkah 4. Pembentukan tree.

Dari Tabel 3.24 menunjukkan bahwa subset telah berada dalam 1 kelas yaitu Administrasi Perkantoran sehingga selanjutnya subtree dengan split node n_bid <=7 memiliki leaf Administrasi Perkantoran.

Tabel 3.25 menunjukkan data subset dengan split node n_bid > 7 memiliki lebih dari 1 kelas, sehingga akan dilakukan langkah 1 sampai langkah 3 kembali untuk menentukan atribut terbaik berikutnya sebagai split node.

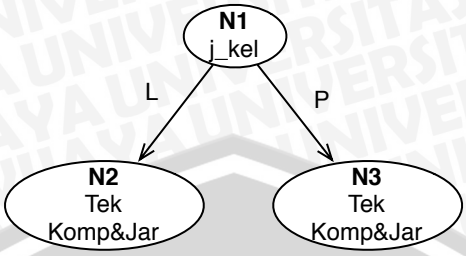
Langkah 1 sampai langkah 3 dilakukan rekursif sampai semua node berada dalam kondisi *pure node*, yaitu dimana semua rekord dalam node tersebut berada dalam 1 kelas. Jika sampai atribut telah habis namun belum menemui pure node maka kelas dengan jumlah terbanyak yang dipilih, jika jumlah kelas sama maka dipilih kelas pertama yang ada pada node tersebut.

Setelah proses pembentukan tree selesai, maka tree yang terbentuk digambarkan pada Gambar 3.15.



Gambar 3.15 Hasil Tree dengan perhitungan manual

Langkah 5. Tree Pruning



Gambar 3.16 Subtree Contoh untuk Tree Pruning

Langkah terakhir adalah tree pruning dengan menggunakan MDL. Karena tree diperbolehkan mempunyai 0 atau 2 anak, maka algoritma MDL pruning yang digunakan adalah strategi *Full* yang dijelaskan pada bab 2.5.5.3 dan menggunakan model encoding $Code_1$ yang dijelaskan pada bab 2.5.5.2. Sehingga cost untuk encode setiap node adalah 1 dengan cost untuk encode node yang merupakan leaf node dengan Persamaan 2.7, Persamaan 2.8 dan 2.9 untuk menghitung node yang merupakan split.

Tree Pruning menggunakan metode bottom-up sehingga cost dihitung dari bagian bawah tree kemudian diteruskan sampai root Contoh perhitungan cost untuk subtree pada Gambar 3.16 adalah sebagai berikut.

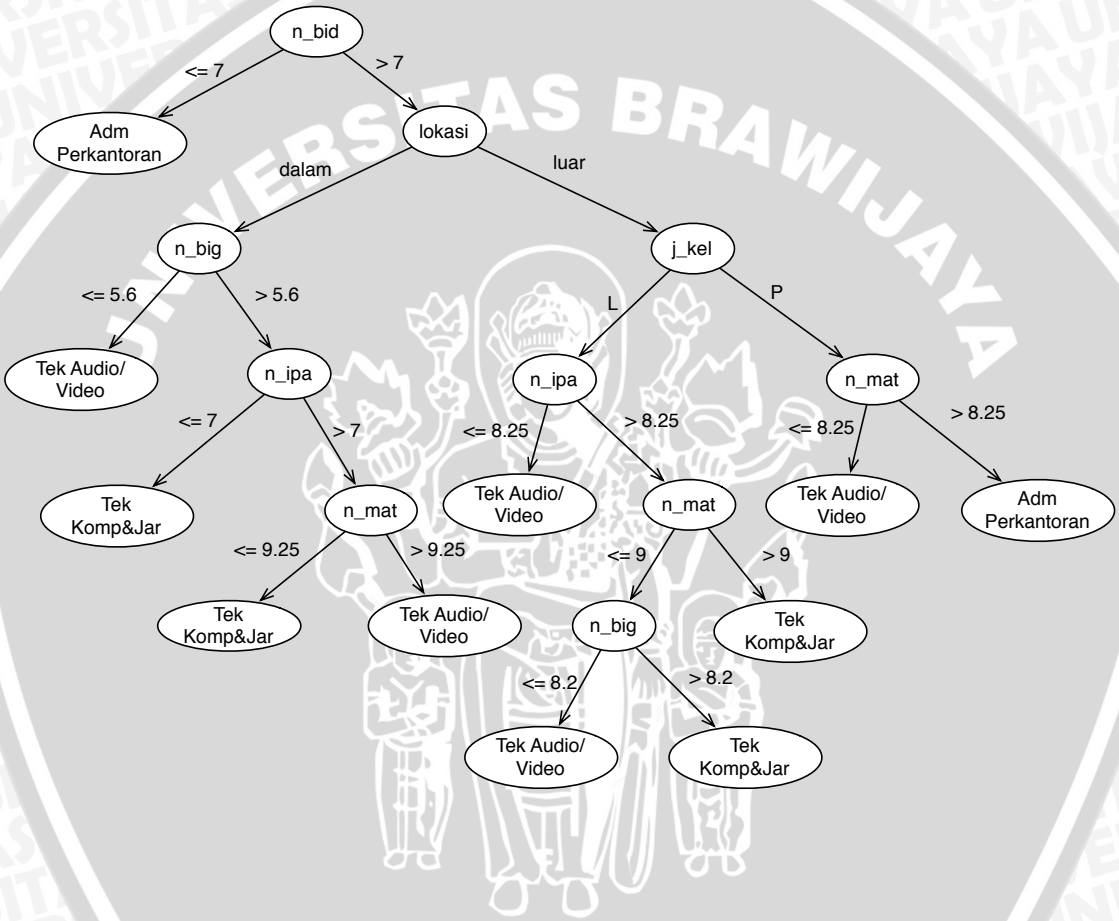
$$C_{leaf}(N_3) = (0 . \log\frac{1}{0} + \frac{2}{2} . \log\frac{1}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)}) + (1 . \log\frac{1}{1} + \frac{2}{2} . \log\frac{1}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)}) + (0 . \log\frac{1}{0} + \frac{2}{2} . \log\frac{1}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)})$$
$$= \mathbf{1.620813}$$

$$C_{leaf}(N_2) = (0 . \log\frac{2}{0} + \frac{2}{2} . \log\frac{2}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)}) + (2 . \log\frac{2}{2} + \frac{2}{2} . \log\frac{2}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)}) + (2 . \log\frac{2}{2} + \frac{2}{2} . \log\frac{2}{2} + \log\frac{\pi^{3/2}}{\Gamma(3/2)})$$
$$= \mathbf{2.523903}$$

Untuk N_1 karena merupakan node split dengan atribut kategorikal jenis kelamin yang memiliki nilai L dan P maka

$$C_{split}(N_1) = \log(2^2 - 1) = \mathbf{0.477121}$$

Karena $C_{leaf}(N) + 1 \leq C_{split}(N_1) + 1 + C_{leaf}(N_3) + C_{leaf}(N_2)$, maka N_1 di konversi menjadi leaf dengan kelas Teknik Komputer dan Jaringan. Langkah Tree Pruning ini dilakukan secara rekursif dengan metode bottom-up sampai node *root*.



Gambar 3.17 Hasil Tree Pruning

Langkah 6. Ekstraksi Decision Tree menjadi Decision Rule.

Dari Gambar 3.17 maka decision rule yang terbentuk adalah :

1. **IF** n_bid <= 7 **THEN** jurusan = Administrasi Perkantoran
2. **IF** n_bid > 7 **AND** lokasi = dalam **AND** n_big <= 5.6 **THEN** jurusan = Teknik Audio Video
3. **IF** n_bid > 7 AND lokasi = dalam **AND** n_big > 5.6 **AND** n_ipa <= 7 **THEN** jurusan = Teknik Komputer dan Jaringan
4. **IF** n_bid > 7 **AND** lokasi = dalam and n_big > 5.6 **AND** n_ipa > 7 **AND** n_mat <= 9.25 **THEN** jurusan = Teknik Komputer dan Jaringan
5. **IF** n_bid > 7 **AND** lokasi = dalam and n_big > 5.6 **AND** n_ipa > 7 **AND** n_mat > 9.25 **THEN** jurusan = Teknik Audio Video
6. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = L **AND** n_ipa <= 8.25 **THEN** jurusan = Teknik Audio Video
7. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = L **AND** n_ipa > 8.25 **AND** n_mat <= 9 **AND** n_big <= 8.2 **THEN** jurusan = Teknik Audio Video
8. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = L **AND** n_ipa > 8.25 **AND** n_mat <= 9 **AND** n_big > 8.2 **THEN** jurusan = Teknik Komputer dan Jaringan
9. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = L **AND** n_ipa > 8.25 **AND** n_mat > 9 **THEN** jurusan = Teknik Komputer dan Jaringan
10. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = P **AND** n_mat <= 8.25 **THEN** jurusan = Teknik Audio Video
11. **IF** n_bid > 7 **AND** lokasi = luar **AND** j_kel = P **AND** n_mat > 8.25 **THEN** jurusan = Administasi Perkantoran

3.9.1 Perhitungan Manual *Precission*, *Recall* dan Akurasi

Untuk perhitungan manual *precission* dan *recall* ditunjukkan pada Tabel 3.26.

Tabel 3.26 Pengukuran Precision, Recall

No	Kelas	Precission	Recall
1	Adm Perkantoran	10 / 14 = **0.71**	10 / 10 = 1
2	Teknik Komputer dan Jaringan	10 / 14 = **0.71**	10 / 12 = **0.83**
3	Teknik Audio Video	8 / 11 = 0.72	8 / 9 = **0.89**
	Rata-Rata	**0.713**	**0.91**

Sedangkan untuk pengkuran akurasi dengan perhitungan manual ditunjukkan pada tabel 3.27.

Tabel 3.27 Pengukuran Akurasi

No	N Data Latih	Jumlah Benar	Akurasi
1	31	28	28 / 31 = **0.903**

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi pada skripsi ini terbagi menjadi dua bagian, yaitu lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan untuk pengembangan aplikasi implementasi dan pengujian pada skripsi ini antara lain:

Jenis : Notebook /Laptop
Prosesor : Intel Core 2 Duo @ 2,26 GHz
Memory : 8 GB
Hardisk : 250 GB
Network : WiFi

Perangkat pengembangan yang digunakan yaitu komputer jenis notebook/laptop dengan kemampuan komputasi standar yang mana cukup untuk menjalankan beberapa aplikasi pengembangan seperti server basis data, aplikasi editor java dan *MySQL* client secara bersamaan.

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam perangkat keras untuk melakukan implementasi dan pengujian dalam skripsi ini antara lain:

- Sistem Operasi Mac OS X versi 10.8.2
- MAMP (*MySQL* server)
- Editor Java (Eclipse Indigo)
- Sequel Pro (aplikasi administrasi basis data)
- JDK versi 1.6.0_35 (Java 6 update 35)

- Subersion client (tools untuk versioning aplikasi)

4.2 Persiapan Data

Data yang digunakan dalam skripsi ini terdiri dari data pelatihan, dan data uji. Sesuai dengan judul skripsi ini, data merupakan data calon siswa SMK yang diterima dari sistem SIAP PSB Online untuk provinsi DKI Jakarta. Untuk data pelatihan adalah data calon siswa SMK pada tahun 2008 yang terdiri dari 4.113 siswa, data calon siswa SMK pada tahun 2009 yang terdiri dari 4.130 siswa dan data calon siswa SMK pada tahun 2010 yang terdiri dari 3.767 siswa. Untuk data pengujian akhir digunakan data calon siswa SMK pada tahun 2011 yang terdiri dari 3.884 siswa.

Metode penyimpanan data dilakukan dengan mengumpulkan data calon siswa SMK pertahun kedalam masing-masing sebuah tabel dalam sebuah basis data. Masing-masing tabel memetakan siswa yang disimbolkan dengan *id* dengan atribut klasifikasinya yaitu nilai matematika, nilai Bahasa Indonesia, nilai Bahasa Inggris, nilai IPA, jenis kelamin, lokasi pendaftar, dan jurusan siswa diterima.

4.3 Implementasi Program

Implementasi program terbagi menjadi dua modul utama yaitu modul *Tree Builder* dan Modul *Evaluator*.

4.3.1 Implementasi Modul *Tree Builder*

Implementasi Modul *Tree Builder* terdiri dari beberapa tahap, yaitu :

1. Tahap Pembentukan *Attribute List* dan *Class List*
2. Tahap Pengurutan *Attribute List* tipe numerik
3. Tahap Pembuatan *Histogram*
4. Tahap Pembuatan *SLIQ Tree*
5. Tahap *Pruning SLIQ Tree*

Berikut akan dijelaskan rincian dari tahapan-tahapan yang dilakukan tersebut.

1. Tahap Pembentukan *Attribute List* dan *Class List*

Tahap pembentukan attribute list dan class list dijalankan setelah aplikasi mengumpulkan data record latih yang diambil dari basis data. Pembentukan Attribute List menggunakan *thread-ing* yang dijalankan oleh class *AttributeListCreatorThread* pada gambar 4.1.

```
36      public AttributeList call()
37              throws Exception {
38          Log.getLogger().info(
39                  "AttributeListCreatorThread untuk "
40                          + this.attrName
41                          + " started");
42          AttributeList attributeList = new AttributeList(
43                  this.attrName);
44          for (Record record : this.classRecords) {
45              for (Attribute attribute : record
46                      .getAttributes()) {
47                  if (attribute
48                          .getName()
49                          .compareToIgnoreCase(
50                                  this.attrName) == 0) {
51                      try {
52                          attributeList
53                                  .addList(
54                                          -1,
55                                          attribute,
56                                          record.getIndexRecord(),
57                                          false);
58                      } catch (Exception e) {
59                          Log.getLogger()
60                                  .error(e.getMessage());
61                          break;
62                      }
63                  }
64              }
65          }
66          Log.getLogger().info(
67                  "AttributeListCreatorThread untuk "
68                          + this.attrName
69                          + " finished");
70          return attributeList;
71      }
```

Gambar 4.1 Kode Sumber Class *AttributeListCreatorThread*

Pada Gambar 4.1 baris 44 menunjukkan bahwa attribute list dibentuk dengan melakukan perulangan pada setiap record dari data latih. Kemudian pada baris 45, untuk setiap record tersebut dilakukan perulangan pada masing-masing atribut, sampai attribut yang sedang dibentuk attribut list nya ditemukan, sehingga pada baris 52, program akan memasukkan atribute yang sesuai pada attribute list. Pada class *AttributeListCreatorThread* terdiri dari sebuah method yang bernama *Call()* method yang berfungsi untuk membentuk sebuah attribute list, dan akan mengembalikan obyek bertipe data AttributeList jika berhasil, atau melempar *Exception* ketika mengalami kegagalan atau *error*.

Sedangkan pembentukan class list menggunakan *thread-ing* yang dijalankan oleh class *ClassListCreatorThread* pada gambar 4.2. Dimana class list didapat dari record yang berasal dari basis data, kemudian diinisialisasi dengan node root yang pada implementasi merupakan node berisi data *null*.

```
34  public ClassList call()
35          throws Exception {
36      Log.getLogger()
37              .info("ClassListCreatorThread started");
38      ClassList classList = new ClassList();
39      for (Record record : this.classRecords) {
40          classList
41                  .addList(
42                          record.getClassName(),
43                          record.getIndexRecord(),
44                          new TreeNode<Node>(
45                                  new Node(
46                                          0,
47                                          null,
48                                          null)),
49                          false);
50      }
51      Log.getLogger()
52              .info("ClassListCreatorThread finished");
53      return classList;
54  }
```

Gambar 4.2 Kode Sumber Class *ClassListCreatorThread*

Pada Gambar 4.2 baris 39, class list dibentuk dengan melakukan perulangan terhadap record yang ada pada data latih, kemudian pada baris 40, anggota class list akan dimasukkan dengan informasi berisi nama kelas, indeks rekord, dan node yang berisi *null*. Method yang terdapat pada ClassListCreatorThread adalah method *Call()* yang digunakan untuk inisialisasi ClassList, dan mengembalikan tipe data ClassList.

2. Tahap Pengurutan *Attribute List* tipe numerik

Tahap pengurutan attribute list bertipe data numerik dilakukan oleh *thread-ing* yang dijalankan oleh class *AttributeListQuickSorterThread* dengan menggunakan algoritma *quick sorting*. Kode sumber class *AttributeListQuickSorterThread* ditunjukkan pada gambar 4.3.

```
64  for (AttributeListItem value : this.toSort
65          .getItems()) {
66      if (Double.valueOf(value.getAttribute()
67              .getValue().toString()) <= Double
68              .valueOf(pivot.getAttribute()
69                      .getValue().toString()))
70          less.add(value);
71      else
72          greater.add(value);
73  }
74
75  AttributeList result = new AttributeList(
76          this.toSort.getAttributeName());
77  result.setIsNumerical(this.toSort.isNumerical());
78  result.setIsCategorical(this.toSort.isCategorical());
79
80  ExecutorService quickSortExecutor = Executors
81          .newCachedThreadPool();
82  Log.getLogger().info(
83          "Thread -1 Sorting Attribute "
84                  + this.toSort.getAttributeName()
85                  + " starting");
86  Future<AttributeList> lessResult = quickSortExecutor
87          .submit(new QuickSorting(less));
88
89  Log.getLogger().info(
90          "Thread -2 Sorting Attribute "
91                  + this.toSort.getAttributeName()
92                  + " starting");
93  Future<AttributeList> greaterResult = quickSortExecutor
94          .submit(new QuickSorting(greater));
95
96  result.addAll(lessResult.get().getItems());
97  Log.getLogger().info(
98          "Thread Sorting Attribute "
99                  + this.toSort.getAttributeName()
100                 + " finished 50%");
101 result.add(pivot);
```

Gambar 4.3 Kode Sumber Class *AttributeListQuickSorterThread*

Pada Gambar 4.3 baris 64, dilakukan perulangan untuk setiap item dalam AttributeList, untuk dibagi menjadi dua, yaitu yang nilainya kurang dari *pivot* ke dalam AttributeList baru dengan instance *less* dan yang lebih besar pada instance *greater*. Dimana inisialisasi nilai pivot, diambil dari item yang terletak di tengah-tengah AttributeList. Kemudian pada baris 86 dan baris 93, dilakukan pemanggilan threading secara rekursif untuk AttributeList yang masuk kedalam

kategori *less* dan *greater*. Pada baris 96, dan 101 Attribute List baru terbentuk dengan menggabungkan item yang berada pada kategori *less*, *pivot dan greater*. Method yang terdapat pada class AttributeListQuickSorterThread adalah *Call()*, method untuk mengurutkan AttributeList dengan algoritma QuickSorting. Mengembalikan obyek AttributeList yang telah terurut. Melemparkan *Exception* ketika terjadi *error*.

3. Tahap Pembuatan *Histogram*

Tahap pembuatan histogram dilakukan setelah attribut list terbentuk, dengan cara mencari nilai unik dari masing masing attribute list, dan kemudian dihitung frekunsi kemunculannya pada record beserta pemetaannya pada kelas dalam kasus klasifikasi. Proses pembentukan histogram dijalankan oleh *thread-ing HistogramCreatorThread*. Gambar 4.4 menunjukkan kode sumber *HistogramCreatorThread.* Perlu diketahui class *HistogramCreatorThread* juga digunakan dalam melakukan proses pembaharuan histogram, pada proses splitting node pembentukan tree.

```
49  if (this.attributeList.isNumerical()) {
50      this.histogram = new Histogram(
51              this.attributeList
52                      .getAttributeName(),
53              this.splitter);
54      this.histogram
55              .setAttribute(this.attributeList
56                      .getItems().get(0)
57                      .getAttribute());
58
59      // Split!!
60      HashMap<Boolean, HashMap<String, Integer>> count =
61              this.attributeList
62              .splitOn(this.splitter,
63                      this.classList);
64      for (boolean countKey : count
65              .keySet()) {
66          this.histogram.setData(countKey,
67                  count.get(countKey));
68      }
69  } else {
70      this.histogram = new Histogram(
71              this.attributeList
72                      .getAttributeName(),
73              this.attributeList
74                      .getDistinctCategories());
75      this.histogram
76              .setAttribute(this.attributeList
77                      .getItems().get(0)
78                      .getAttribute());
79
80      // Split!!
81      for (String category : this.attributeList
82              .getDistinctCategories()) {
83          HashMap<String, Integer> count = this.attributeList
84                  .splitOn(category,
85                          this.classList);
86          this.histogram.setData(category,
87                  (count));
88      }
89  }
90
91  this.histogram.calcGiniIndeks();
```

Gambar 4.4 Kode Sumber Class *HistogramCreatorThread*

Gambar 4.4 pada baris 49 menunjukkan perbedaaan perlakukan terhadap pembuatan histogram dengan tipe data numerik dan kategorikal. Baris 60 pada

kode melakukan split pada attribute list bertipe data numerik, sedangkan untuk attribute list dengan tipe data kategorikal, split dilakukan pada baris 81. Setelah histgoram terbentuk akan dilakukan perhitngan gini indeks yang dieksekusi pada baris 91. Method yang terdapat pada class HistogramCreatorThread adalah *Call ()* yang digunakan untuk membentuk histogram baru dan mengskusi method untuk menghitung nilai Gini Indeks. Mengembalikan obyek histogram, dan melemparkan *Exception* jika terjadi *error.*

4. Tahap Pembuatan *SLIQ Tree*

Tahap pembuatan Tree dengan algoritma SLIQ, yang implementasinya dilakukan dengan motode BFS (Breadth First Search), dimana pembentukan tree dilakukan secara paralel ke samping, tanpa harus menunggu sampai sebuah node mencapai *leaf*-nya. Secara aplikasi proses ini, diimplementasikan dengan paralel dan rekursif, sampai ditemukan *pure node* pada masing-masing *thread* atau atribut yang ditelusuri pada masing-masing *thread* sudah habis. Cuplikan kode sumber untuk class *TreeBuilderThread* bisa dilihat pada Gambar 4.5.

```
170 if (histogramSelected
171         .isNumerical()) {
172     // update node (Root)
173 TreeNode<Node> left = new TreeNode<Node>(
174         new Node(
175                 this.tree
176                         .getNumberOfNodes(),
177                 histogramSelected,
178                 true));
179 this.root.addChild(left);
180
181 TreeNode<Node> right = new TreeNode<Node>(
182         new Node(
183                 this.tree
184                         .getNumberOfNodes(),
185                 histogramSelected,
186                 false));
187 this.root.addChild(right);
188
189 // update class list
190 ExecutorService attributeListIndexRecordSplitExecutor =
191     Executors
192         .newCachedThreadPool();
193 Future<ArrayList<Integer>> indexRecordLeftCreator =
194 attributeListIndexRecordSplitExecutor
195         .submit(new AttributeListIndexRecordSplitThread(
196                 attributeLists,
197                 histogramSelected
198                         .getAttrName(),
199                 histogramSelected
200                         .getSplitterNumerical(),
201                 true));
202
203 Future<ArrayList<Integer>> indexRecordRightCreator =
204 attributeListIndexRecordSplitExecutor
205         .submit(new AttributeListIndexRecordSplitThread(
206                 attributeLists,
207                 histogramSelected
208                         .getAttrName(),
209                 histogramSelected
210                         .getSplitterNumerical(),
211                 false));
```

Gambar 4.5 Kode Sumber Class *TreeBuilderThread*

Pada cuplikan kode sumber Gambar 4.5, baris 170 memulai pembentukan tree jika tipe data pada histogram merupakan numerik. Pada baris 172 sampai

187, menunjukkan proses update node, yaitu menambahkan child left dan right, sedangkan baris 190 sampai 211 menunjukkan proses update class list. Method yang terdapat pada class TreeBuilderThread :

1. Call() : Membentuk tree, dengan parameter Attribute List dan Class List. Mengembalikan tree jika sukses, dan melempar *Exception* jika *error.*
2. AttributeListIndexRecordSplitThread.Call() : Melakukan splitting terhadap Attribute List, mengembalikan obyek AttributeList jika sukses.
3. AttributeListCopierThread.Call() : Menyalin Attribute List dengan index record tertentu. Mengembalikan obyek AttributeList jika sukses.

5. Tahap *Pruning SLIQ Tree*

Tahap pemangkasan tree dilakukan setelah tree terbentuk, pemangkasan dilakukan dari leaf atau node terakhir dari tree, dengan menggunakan metode *MDL Pruning*. Pemangkasan ini dilakukan sampai node root dari tree (bottom-up), atau sudah tidak ada lagi node yang bisa dipangkas. Cuplikan kode sumber dari *MdlTreePrunerThread* yang melakukan proses pruning bisa dilihat pada gambar 4.6.

```
// prune!!
int i = 0;
for (TreeNode<Node> node : parentleaves) {
    if (isPrune.get(i)) {
        ArrayList<Integer> idRecords =
                new ArrayList<Integer>();
        idRecords.addAll(node
                .getChildAt(0)
                .getChildAt(0).getData()
                .getRecordId());
        idRecords.addAll(node
                .getChildAt(1)
                .getChildAt(0).getData()
                .getRecordId());

        String classified = this.classList
                .findMostClasifiedAt(idRecords);

        node.getChildAt(0)
                .removeChildren();
        node.getChildAt(1)
                .removeChildren();
        node.removeChildren();

        // create new node for replace
        TreeNode<Node> newNode = new TreeNode<Node>(
                new Node(
                        this.tree
                                .getNumberOfNodes(),
                        classified));

        // save index record
        newNode.getData().setRecordId(
                idRecords);

        node.addChild(newNode);
        // update class list
        this.classList
                .updateAtIndexRecords(
                        idRecords, newNode);
    }
    i++;
}
```

Gambar 4.6 Kode Sumber Class *MdlTreePrunerThread*

Pada Gambar 4.6 baris 85, dilakukan perulangan untuk setiap node yang bertipe parent dari leaf, jika setelah dilakukan pengecekan perlu dilakukan pruning, maka pada baris 101 sampai 105 menunjukkan dilakukannya pemangkasan pada dengan menghilangkan children, kemudian pada baris 108 dibentuk node baru sebagai pengganti. Method yang terdapat pada class *MdlTreePrunerThread* adalah *Call()* yang melakukan pengecekan apakah node perlu dipangkas, dan melakukan pemangkasan jika diperlukan. Mengembailkan obyek tree atau *Exception* jika *error.*

4.3.2 Implementasi Modul *Evaluator*

Evaluator pada implementasi skripsi ini adalah dengan mecari nilai dari akurasi, *precision* dan *recall*. Untuk mendapatkan nilai evaluator ini dilakukan beberapa tahap, yaitu :

1. Tahap Perhitungan Akurasi

Akurasi dihitung dengan membandingkan *decision rule* yang didapat dari tree dengan data pengujian, semakin banyak data uji yang berada dalam *decision rule* maka semakin tinggi nilai akurasi. Pada aplikasi, penghitungan akurasi dilakukan oleh *thread-ing* yang dijalankan oleh class *AccurationCalculatorThread.* Cuplikan kode sumber dari class *AccurationCalculatorThread* bisa dilihat pada Gambar 4.7.

```
336 while (!checkNode.equals(this.tree
337         .getRoot()) && ruleAttrMacth) {
338     ruleAttrMacth = false;
339     String attributeName = checkNode
340             .getData().getHistogram()
341             .getAttrName();
342     for (Attribute attribute : attributes) {
343         if (checkNode.getData()
344                 .getHistogram()
345                 .isNumerical()) {
346             if (attribute
347                     .getName()
348                     .compareToIgnoreCase(
349                             attributeName) == 0) {
350                 if (checkNode
351                         .getData()
352                         .isLessOrEqual()) {
353                     if (Double
354                             .valueOf(attribute
355                                     .getValue()
356                                     .toString()) <= checkNode
357                             .getData()
358                             .getHistogram()
359                             .getSplitterNumerical()) {
360                         ruleAttrMacth = true;
361                         break;
362                     }
363                 } else {
364                     if (Double
365                             .valueOf(attribute
366                                     .getValue()
367                                     .toString()) > checkNode
368                             .getData()
369                             .getHistogram()
370                             .getSplitterNumerical()) {
371                         ruleAttrMacth = true;
372                         break;
373                     }
374                 }
375             }
376         } else {
```

Gambar 4.7 Kode Sumber Class *AccurationCalculatorThread*

Pada Gambar 4.7 baris 336, kode program melakukan traversing tree untuk mencari rule yang sesuai dengan record yang sedang diperiksa. Method yang terdapat pada class *AccurationCalculatorThread* adalah *Call()* untuk menghitung jumlah data yang benar dan sesuai dengan tree yang dihasilkan, mengembalikan nilai akurasi dalam tipe data *double*.

2. Tahap Perhitungan *True Positive* dan *False Negative*

True Positive didapat dari membandingkan *decision rule* yang didapat dari tree dengan data pengujian, dimana sebuah data pengujian dikatakan *true positive,* jika data pengujian dapat diklasifikasi oleh *decision rule.* Sedangkan data uji dikatakan *False Negative,* jika jurusan ada pada *decision rule*, namun dari segi atribut tidak ditemukan pada *decision rule*. Implementasi dari perhitungan ini dilakukan dengan *thread-ing* yang dijalankan oleh class *TruePositiveFalseNegativeCalculatorThread*. Gambar 4.8 menunjukkan cuplikan kode sumber *TruePositiveFalseNegativeCalculatorThread.*

```
51  for (Record record : this.records) {
52      if (record.getClassName().compareToIgnoreCase(
53              this.className) == 0) {
54          // check apakah sama dengan rule yang ada
55          //(jika sama, maka true
56          // positive)
57          boolean isSameRule = true;
58          for (TreeNode<Node> node : this.classMapper
59                  .get(this.className)) {
60              TreeNode<Node> checkNode = node;
61              isSameRule = true;
62              while (!checkNode.getParent().equals(
63                      this.tree.getRoot())) {
64                  checkNode = checkNode.getParent();
65                  Attribute currentAttribute = new Attribute();
66                  for (Attribute attribute : record
67                          .getAttributes()) {
68                      if (attribute
69                              .getName()
70                              .compareToIgnoreCase(
71                                  checkNode
72                                      .getData()
73                                      .getHistogram()
74                                      .getAttrName()) == 0) {
75
76                          currentAttribute = attribute;
77                          break;
78                      }
79                  }
```

Gambar 4.8 Kode Sumber Class *TruePositiveFalseNegativeCalculatorThread*

3. Tahap Perhitungan *False Positive*

False Postive merupakan data uji yang dari segi atribut dan nilainya terdapat pada decision rule namun hasil kelasmya tidak sesuai. Perhitungan *False Positive* dilakuakn oleh *thread-ing* yang dijalankan oleh *FalsePositiveCalculatorThread*. Cuplikan kode sumber *FalsePositiveCalculatorThread* dapat dilihat pada Gambar 4.9.

```
448 for (Record record : this.records) {
449     if (this.classMapper.containsKey(record
450             .getClassName())) {
451 
452         boolean isSameRule = true;
453         boolean haveSameRule = false;
454         TreeNode<Node> selectedNode = new TreeNode<Node>();
455 
456         for (TreeNode<Node> node : this.classifiedNode) {
457             TreeNode<Node> checkNode = node;
458             selectedNode = node;
459             isSameRule = true;
460             while (!checkNode.getParent().equals(
461                     this.tree.getRoot())) {
462                 checkNode = checkNode.getParent();
463                 Attribute currentAttribute = new Attribute();
464                 for (Attribute attribute : record
465                         .getAttributes()) {
466                     if (attribute
467                             .getName()
468                             .compareToIgnoreCase(
469                                     checkNode
470                                             .getData()
471                                             .getHistogram()
472                                             .getAttrName()) == 0) {
473                         currentAttribute = attribute;
474                         break;
475                     }
476                 }
```

Gambar 4.9 Kode Sumber Class *FalsePositiveCalculatorThread*

4. Tahap Perhitungan *Recall, Precision* dan *F-Measure*

Recall didapat dengan menghitung rata-rata dari *True Positive* dibagi *True Positive* yang telah dijumlahkan dengan *False Negative.* Sedangkan F-Measure merupakan *harmonic-mean* dari precision dan recall. Proses ini dilakukan oleh fungsi *calcPrecisionRecallFMeasure*. Cuplikan kode sumber dari fungsi *calcPrecisionRecallFMeasure* dapat dilihat pada Gambar 4.10.

```
193 double avgPrecision = 0.0;
194 for (String className : classMapperUji) {
195
196     double precision = 0.0;
197     if ((double) (truePositive.get(className) + falsePositive
198             .get(className)) > 0)
199         precision = (double) truePositive
200                 .get(className)
201                 / (double) (truePositive
202                         .get(className) + falsePositive
203                         .get(className));
204
205     avgPrecision += precision;
206 }
207
208 returnData.put("avg_precision",
209         (avgPrecision / (double) classMapperUji
210                 .size()));
211
212 double fMeasure = 2 * ((returnData
213         .get("avg_precision") * returnData
214         .get("avg_recall")) / (returnData
215         .get("avg_precision") + returnData
216         .get("avg_recall")));
217
218 returnData.put("f_measure", fMeasure);
```

Gambar 4.10 Kode Sumber Fungsi *calcPrecisionRecallFMeasure*

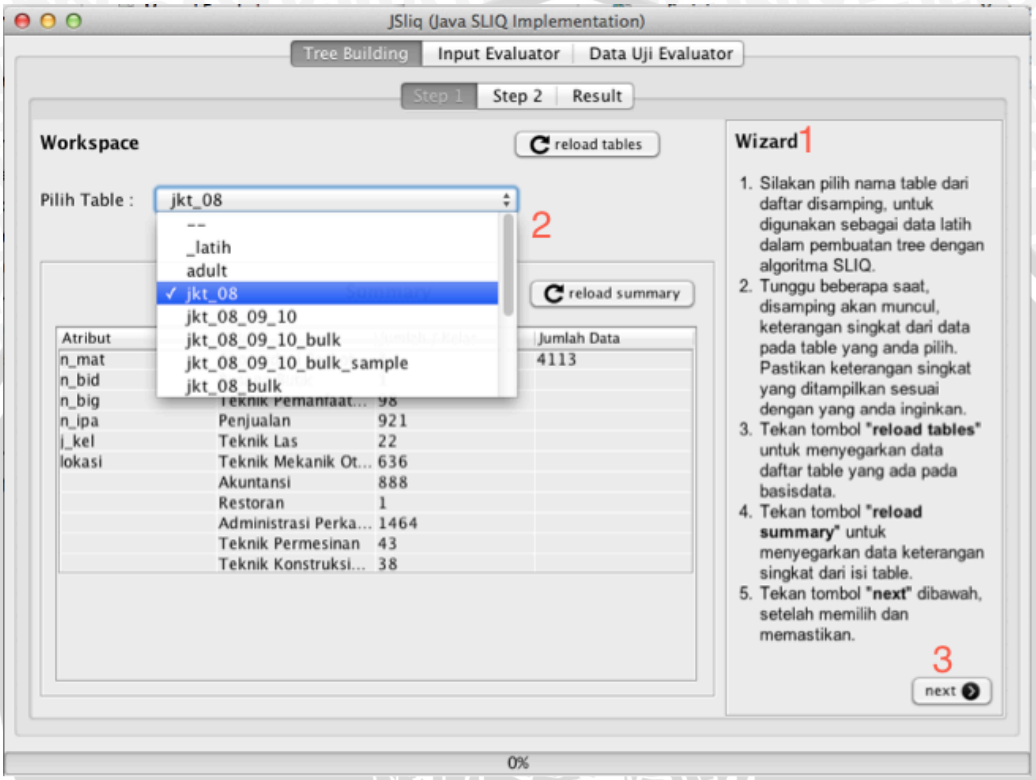
4.4 Implementasi Antar Muka

Antar Muka dari implementasi aplikasi terbagi menjadi bagian Antar Muka Pemilihan Data Latih, Antar Muka Hasil Pembentukan Tree, Antar Muka Evaluator untuk Data Uji dan Antar Muka Evaluator untuk Data Input.

4.4.1 Antar Muka Pemilihan Data Latih

Antar muka pemilihan data latih menampilkan daftar tabel yang ada pada basis data, sebagai pilihan data latih untuk pembentukan *tree* dengan algoritma

SLIQ dalam bentuk *combobox*. Setelah salah satu tabel dipilih, maka di bagian bawah akan ditampilkan ringkasan data dari table tersebut, antara lain atribut yang ada, jumlah data dan kelas beserta jumlahnya. Untuk lebih jelasnya tampilan dari antar muka pemilihan data bisa dilihat pada Gambar 4.11.

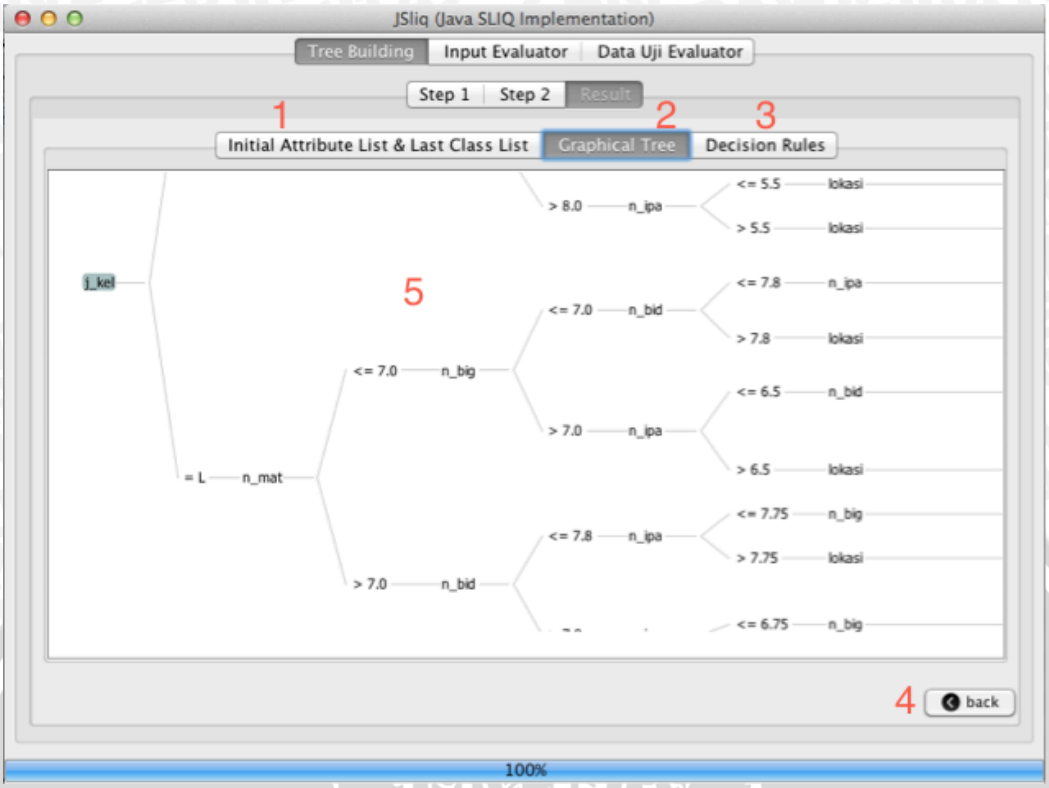


Gambar 4.11 Tampilan Antar Muka Pemilihan Data Latih

Nomor 1 adalah petunjukan penggunaan modul, nomor 2 adalah combo box untuk memilih tabel yang akan dijadikan data latih dan nomor 3 adalah tombol next untuk menuju ke langkah berikutnya setelah memilih data latih.

4.4.2 Antar Muka Hasil Pembentukan Tree

Pada antar muka ini, aplikasi menampilkan tiga jenis informasi, yaitu attribute list dan class list yang terinisialisasi, grafik *tree* terbentuk dan *decision rule* yang dihasilkan. Tampilan antar muka bisa dilihat pada Gambar 4.12.

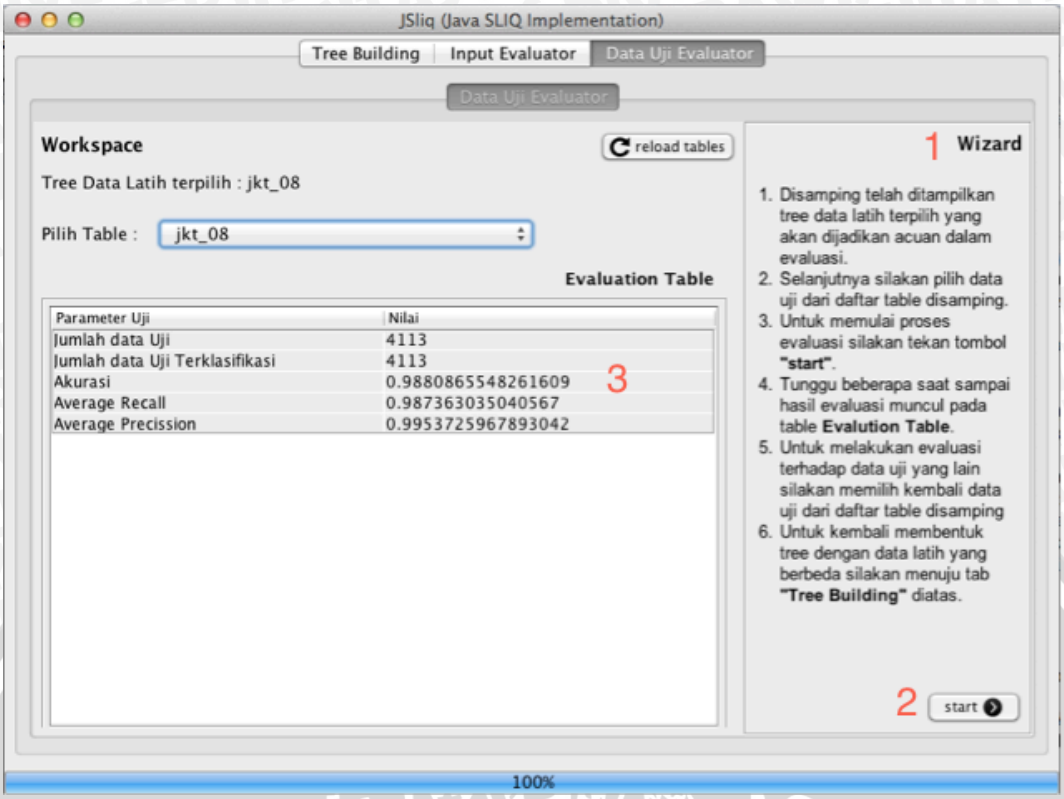


Gambar 4.12 Tampilan Antar Muka Hasil Pembentukan Tree

Pada Gambar 4.12, nomor 1 adalah tab untuk menampilkan inisialisasi AttributeList dan ClassList. Nomor 2 adalah tab untuk menampilkan hasil tree dalam bentuk grafis, dan nomor 3 untuk menampilkan decision rule. Sedangkan nomor 4 adalah tombol untuk kembali ke langkah sebelumnya, dan nomor 5 merupakan contoh tampilan tree dalam bentuk grafis.

4.4.3 Antar Muka Evaluator Data Uji

Antar muka ini digunakan untuk melakukan pengujian terhadap tree terbentuk dengan data uji yang juga diambil dari tabel-tabel yang ada pada basis data. Sama seperti antar muka pemilihan data, antar muka evaluator ini juga menampilkan pilihan data uji yang diambil dari daftar tabel pada basis data dalam bentuk *combobox*. Namun setelah data uji dipilih, dan tombol "*start*" ditekan, maka antar muka akan menampilkan hasil pengujian, yang mengandung informasi akurasi, precision dan recall. Untuk lebih jelasnya antar muka evaluator data uji bisa dilihat pada Gambar 4.13.

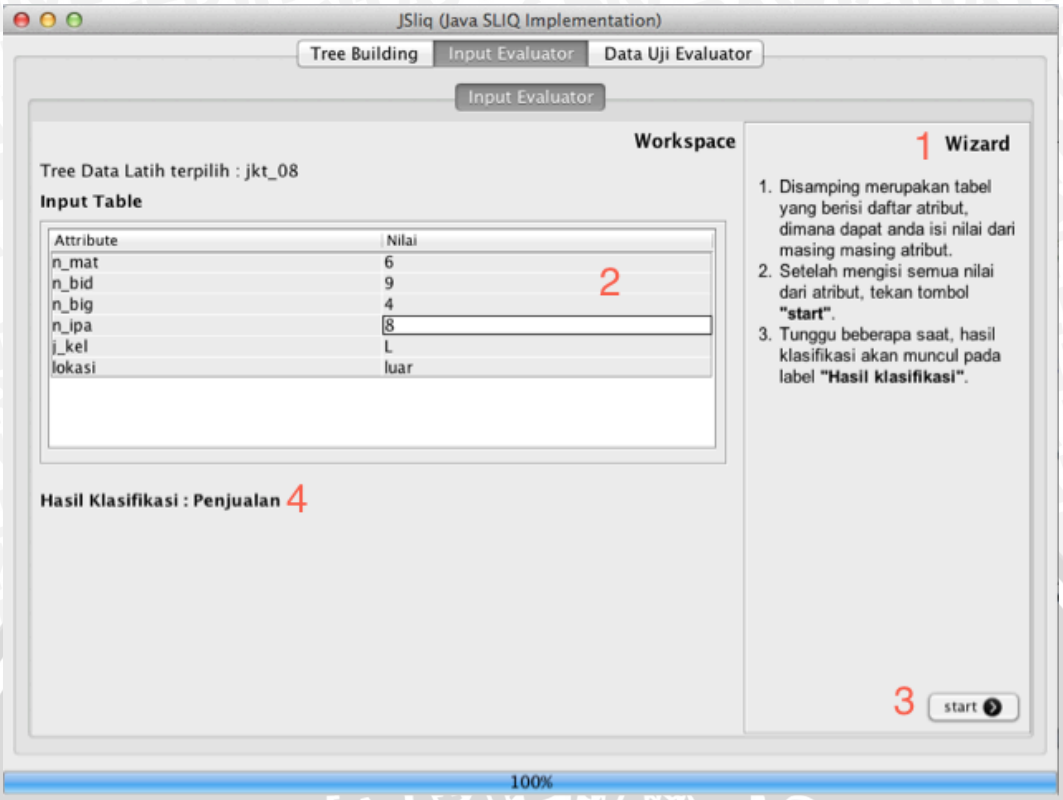


Gambar 4.13 Tampilan Antar Muka Evaluator Data Uji

Nomor 1 pada Gambar 4.13 adalah petunjuk penggunaan modul, nomor 2 adalah tombol untuk memulai perhitungan evaluasi dan nomor 3 adalah contoh dari hasil perhitungan evaluasi.

4.4.4 Antar Muka Evaluator Data Input

Antar muka evalutor data input, menampilkan daftar atribut yang ada pada data latih, untuk kemdian dapat diubah nilainya. Setelah tombol "*start*" ditekan aplikasi akan melakukan pencocokan dengan *decision rule* yang ada untuk menemukan kelas hasil. Tampilan Antar Muka Evalutor Data Input dapat dilihat pada gambar 4.14.



Gambar 4.14 Tampilan Antar Muka Evaluator Data Input

Pada Gambar 4.14 nomor 1 adalah petunjuk penggunaan modul, nomor 2 adalah tabel untuk input data, nomor 3 adalah tombol untuk memulai pengklasifikasian dan nomor 4 adalah hasil klasifikasi.

4.5 Implementasi Uji Coba dan Analisa Hasil

Uji coba dilakukan pada data yang telah dipersiapkan, seperti pada subbab 4.3. Data calon siswa SMK tahun 2008, 2009 dan 2010 masing masing dijadikan data latih, kemudian diuji akurasi, precision dan recall nya terhadap data latih itu sendiri, kemudian juga diuji pada data pada tahun 2011. Selanjutnya data 2008 sampai 2010 digabungkan dan diuji terhadap data gabungan tersebut, kemudian diuji dengan data pada tahun 2011. Rincian bagaimana implementasi pelatihan dan pengujian dilakukan pada skripsi ini adalah sebagai berikut :

4.5.1 Pengujian *Precision*, *Recall* dan *F-Measure*

Pengujian yang dilakukan pertama adalah mendapatkan nilai *precision* dan *recall* dari *decision tree* yang dihasilkan. Ringkasan data tiap tahun dari calon siswa SMK provinsi DKI Jakarta dapat dilihat pada Tabel 4.1.

Tabel 4.1 Ringkasan Data pertahun Calon Siswa SMK Provinsi DKI Jakarta

No	Data Tahun	Jumlah Data
1	2008	4113
2	2009	4130
3	2010	4107
4	2011	3884

Dari data tiap tahun tersebut pada Tabel 4.2 dapat dilihat hasil pengujian *precision* dan *recall* dari *decision tree* yang terbentuk.

Tabel 4.2 Hasil Pengujian *Precision* dan *Recall*

No	Data latih	Uji Data latih		Uji Data 2011	
		Precission	*Recall*	*Precission*	*Recall*
1	2008	0.99349	0.98736	0.33356	0.3914
2	2009	0.89933	0.74237	0.75279	0.5673
3	2010	0.98022	0.82521	0.75193	0.6610
4	2008 – 2010	0.98298	0.57256	0.85142	0.7252

Tabel 4.2 menujukkan bagaimana *decision tree* mampu menangani keseluruhan data uji yang digambarkan dengan nilai *recall* dan bagaimana *decision tree* mempu memprediksi kelas yang sesuai berdasarkan rule pada data uji. Dilihat dari hasil pengujian, menunjukkan bahwa pengujian *decision tree* terhadap data latihnya sendiri menghasilkan nilai *precision* yang tinggi, yaitu lebih dari 0.8 atau lebih dari 80%. Hasil *precision* tertinggi diperoleh dari data latih tahun 2008. Sedangkan nilai *precision* terkecil ditunjukkan oleh data tahun 2009, hal ini menunjukkan bahwa decision tree mempu membuat rule dari atribut

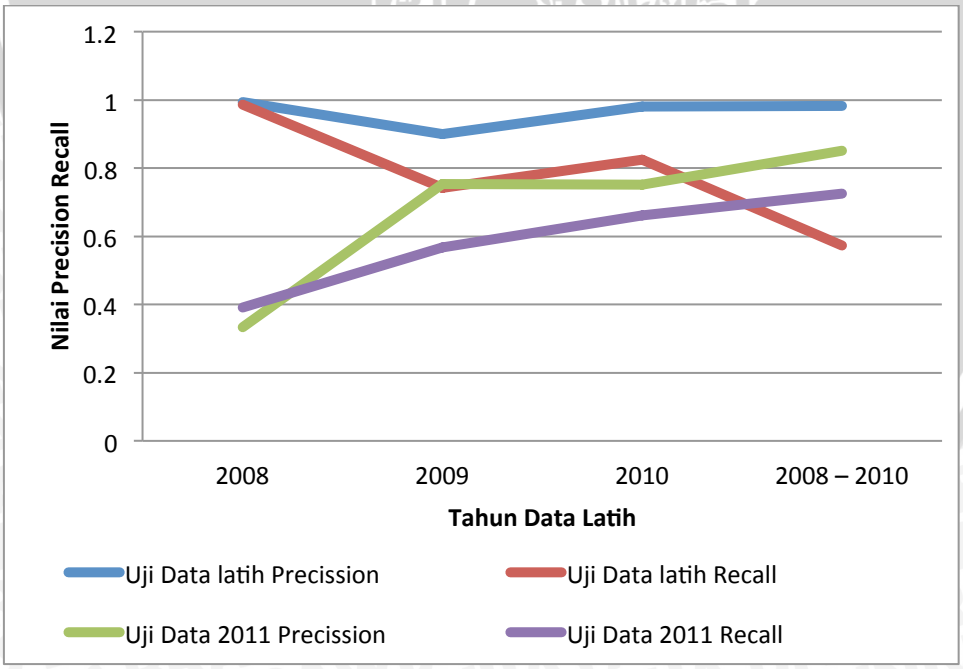
yang ada pada data, namun hasil klasifikasi pada sejumlah data yaitu 0.1 atau 10% tidak sesuai dengan rule yang ada. Tabel 4.3 menunjukkan F-Measure yang didapat dari masing-masing *precision* dan *recall*.

Tabel 4.3 Hasil Pengujian *F-Measure*

No	Data latih	Uji Data latih	Uji Data 2011
		F-Measure	*F-Measure*
1	2008	0.99042	0.36111
2	2009	0.81334	0.64880
3	2010	0.89606	0.69884
4	2008 – 2010	0.72363	0.78329

4.5.1.1 Analisa Hasil Pengujian *Precision, Recall* dan *F-Measure*

Dari hasil pengujian *precision, recall* yang ditunjukkan pada Tabel 4.2 , dapat dibuat grafik hubungan data latih dengan hasil pengujian *precision recall*, seperti digambarkan pada Gambar 4.15.



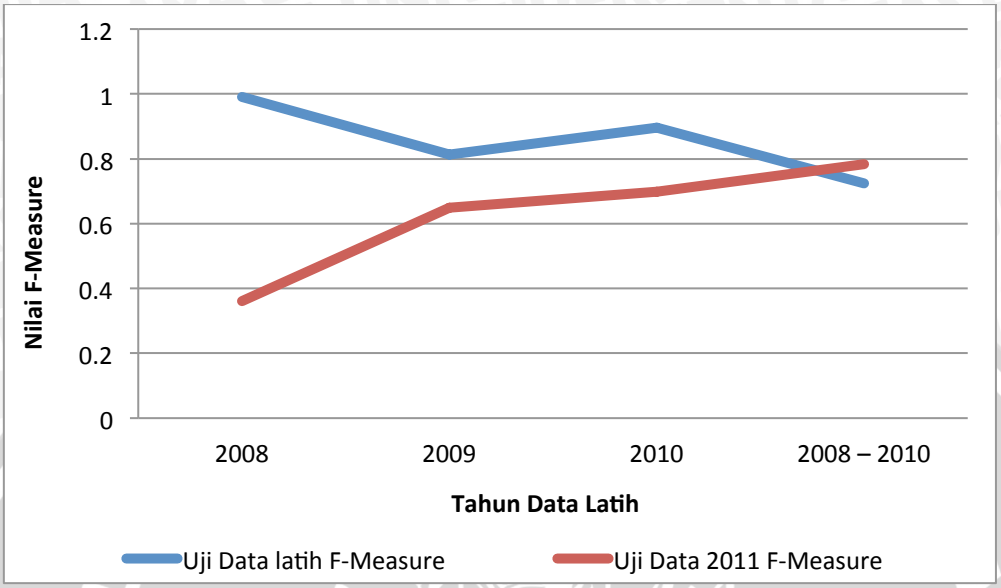
Gambar 4.15 Grafik *Precision*, *Recall*

Nilai *recall* untuk data tahun 2008, 2009 dan 2010 menunjukkan angka beragam dan paling kecil 0.7 yaitu pada tahun 2009. Hal ini menunjukkan decision tree mampu meng-klasifikasi keseluruhan data pada data latihnya masing-masing. Nilai *recall* terkecil terjadi pada data gabungan tahun 2008-2010 yang diuji terhadap data latihnya sendiri yaitu 0.57 atau 57%. Nilai ini menunjukkan 43% data tidak dapat ditangani oleh *decision tree*, yaitu sebanyak 43% data memiliki jurusan yang didefinisikan pada *decision tree*, akan tetapi attribut pada data tersebut tidak terdapat pada rule yang dihasilkan *decision tree.* Hal ini terjadi karena terlalu beragamnya data gabungan, baik dari atributnya (n_mat, n_bid, n_big, n_ipa, j_kel, lokasi) maupun jurusan yang ada.

Sedangkan untuk pengujian dengan data calon siswa SMK pada tahun 2011 menunjukkan nilai *precision* dan *recall* yang tidak terlalu tinggi, kecuali untuk data gabungan 2008-2010 yang menunjukkan nilai *precision* dan *recall* lebih dari 0.7. Hasil ini menyatakan bahwa data gabungan 2008-2010 mampu mempresentasikan data yang ada pada tahun 2011. Sedangkan untuk nilai terkecil didapat oleh data pada tahun 2008 yaitu kurang lebih 0.3 atau 30% saja. Ini menunjukkan bahwa kurang lebih 70% data uji pada tahun 2011 tidak dapat dimodelkan oleh *decision tree* yang dihasilkan dengan baik. Hal bisa terjadi karena adanya perubahan aturan dalam penerimaan siswa SMK, baik dari sisi sistem SIAP PSB Online ataupun dari sisi sekolah, atau kecenderungan pemilihan jurusan yang dilakukan oleh siswa.

Dari grafik yang ditunjukkan Gambar 4.15, jika diperhatikan pengukuran nilai *recall* dan *precision* dengan data uji 2011 menunjukkan trend positif (seri yang berwarna hijau dan ungu), dimana setiap tahunnya data, nilai *precision* dan *recall* semakin tinggi, trend positif ini juga ditunjukkan oleh nilai *f-measure* pada data uji tahun 2011. Hasil ini menunjukkan bahwa setiap bertambah tahun, *decision tree* yang dibentuk oleh data latih, semakin mampu dalam melakukan klasifikasi terhadap data uji tahun 2011. Hasil ini disebabkan oleh semakin miripnya pola pemilihan jurusan beserta aturan dalam penerimaan siswa baru dari tahun ke tahun terhadap data uji tahun 2011. Sehingga pada hasil pengujian dengan data latih gabungan 2008-2010, memiliki nilai paling tinggi dalam

precision, recall dan *f-measure,* karena data gabungan ini paling mampu mengklasifikasikan data uji tahun 2011.



Gambar 4.16 Grafik *F-Measure*

Seperti halnya pada Gambar 4.15, Gambar 4.16 merupakan grafik hasil pengujian F-Measure, dimaan f-mesure merupakan rata-rata dari *precision* dan *recall*. Representasi nilai *f-measure* juga menunjukkan nilai yang cenderung mengalami penurunan pada pengujian dengan data latih sendiri, namun justru menunjukkan trend positif pada uji data 2011, dimana pada Gambar 4.16 ditunjukkan dengan seri berwarna merah. Hasil ini menujukkan bagaimana data latih gabugan tahun 2008-2010 dapat mengklasifikasikan data uji 2011 paling baik, hal ini dikarenakan terjadinya penyusutan jumlah jurusan dari tahun 2008 ke tahun 2011, sehingga meningkatkan precision dari *decision tree*.

4.5.2 Pengujian Akurasi

Pengujian selanjutnya adalah pengujian akurasi, akurasi menunjukkan perbandingan jumlah data uji yang dapat diklasifikasikan oleh *decision tree* dengan tepat dibandingkan dengan jumlah keseluruhan data uji tersebut. Pada

Tabel 4.4 dapat dilihat bagaimana hasil akurasi dari *decision tree* terhadap data ujinya.

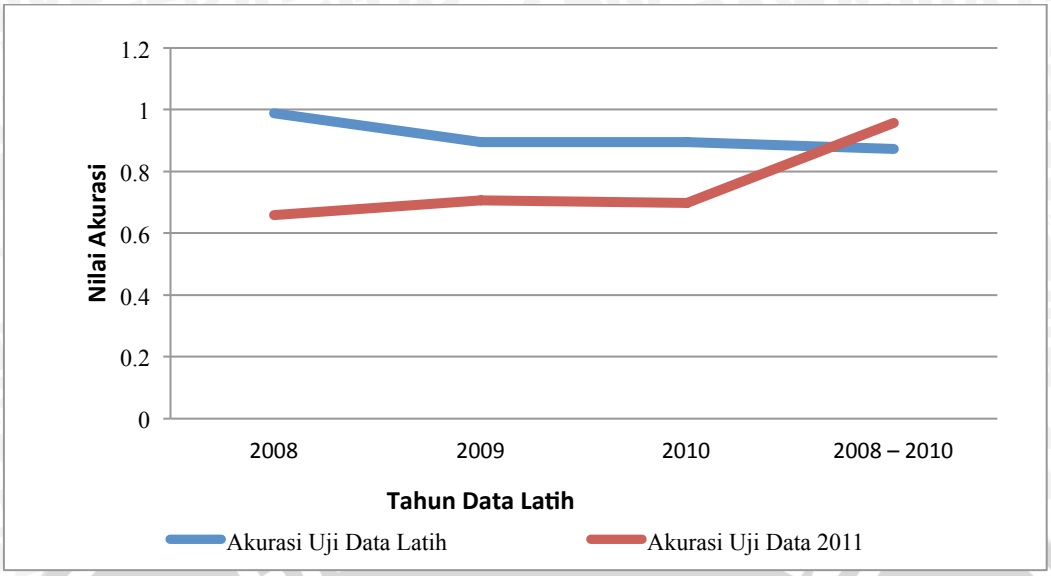
Tabel 4.4 Hasil Pengujian Akurasi

No	Data latih	Akurasi	
		Uji Data Latih	*Uji Data 2011*
1	2008	0.98808	0.65881
2	2009	0.89472	0.70567
3	2010	0.89457	0.69791
4	2008 – 2010	0.87295	0.95829

Dari hasil pengujian akurasi yang ditunjukkan pada Tabel 4.3, dapat dilihat nilai Akurasi tertingi diperoleh oleh *decision tree* dengan data latih data calon siswa SMK tahun 2008 dan data uji adalahd ata latihnya sendiri. Sedangkan untuk data uji tahun 2011, nilai akurasi tertinggi ada pada *decision tree* dengan data latih gabungan data calon siswa SMK dari tahun 2008 sampai tahun 2011.

4.5.2.1 Analisa Hasil Pengujian Akurasi

Berdasarkan hasil pengujian akurasi yang ada pada Tabel 4.4, dapat dilihat bagaimana pengaruh data latih terhadap akurasi *decision tree*. Gambar 4.17 menunjukkan grafik akurasi dari hasil uji coba.



Gambar 4.17 Grafik *Akurasi*

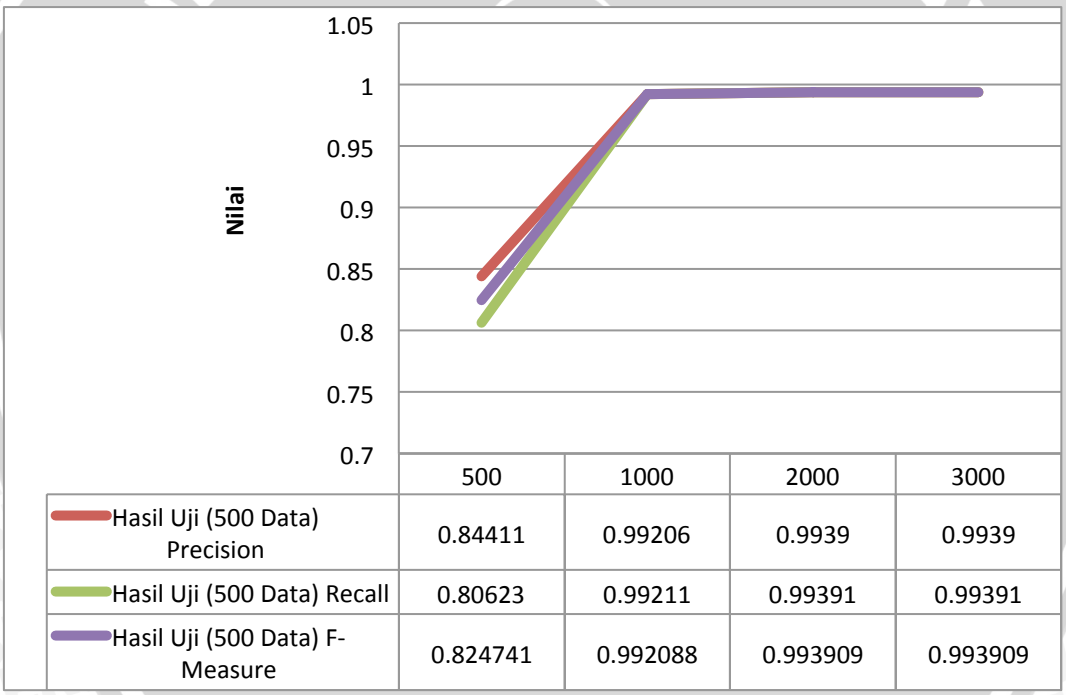
Berdasarkan hasil pengujian akurasi dapat dilihat untuk nilai akurasi pada pengujian dengan data uji sama dengan data latih, menunjukkan nilai yang tinggi, dimana minimal 0.87, atau sebanyak 87% data uji dapat diklasifikasi oleh *decision tree* dengan tepat. Namun untuk nilai akurasi pada pengujian terhadap data uji adalah data tahun 2011, untuk *decision tree* yang dibentuk dari data latih, tahun 2008, 2009, dan 2010 menunjukkan angka kurang lebih 65-70% data uji. Ini disebabkan untuk masing-masing data pertahun belum cukup untuk mengklasifikasikan data pada tahun 2011. Sedangkan untuk *decision tree* yang dibentuk dari data latih yang merupakan data gabungan calon siswa SMK pada tahun 2008 – 2010, menunjukkan nilai akurasi kurang lebih 0.96 atau 96% data uji pada tahun 2011, bisa diklasifikasikan dengan tepat. Hasil in terjadi karena data latih pada tahun 2008 – 2011 yang digabungkan, sehingga menjadi lebih umum, ternyata dapat menyelesaikan sebagian besar data uji tahun 2011. Nilai akurasi yang tidak dapat mencapai 100% dikarenakan ada beberapa data yang merupakan pencilan.

Walaupun pengukuran akurasi terhadap *decision tree* dengan data uji yang sama dengan data latih menunjukkan trend negatif, namun sebaliknya, pengujian terhadap data uji 2011, menunjukkan trend positif (seri yang berwarna merah). Nilai akurasi yang cenderung naik, setiap tahunnya menyatakan bahwa semakin

tahun dari tahun 2008 - 2010, hasil *decision tree* dapat semakin baik mengklasifikasikan data uji tahun 2011.

4.5.3 Pengujian Jumlah Data Latih

Pengujian data latih berfungsi untuk mengetahui bagaimana pengaruh jumlah data latih terhadap hasil pengujian. Pada percobaan ini digunakan data latih adalah data pada tahun 2008, dimana 500 data uji yang diambil adalah data pada tahun 2008 juga namun bukan merupakan bagian dari data latih. Gambar 4.18 menunjukkan hasil pengujian pada tahun 2008.



	500	1000	2000	3000
Hasil Uji (500 Data) Precision	0.84411	0.99206	0.9939	0.9939
Hasil Uji (500 Data) Recall	0.80623	0.99211	0.99391	0.99391
Hasil Uji (500 Data) F-Measure	0.824741	0.992088	0.993909	0.993909

Gambar 4.18 Grafik Pengujian Jumlah Data Latih Tahun 2008

4.5.3.1 Analisa Hasil Pengujian Jumlah Data Latih

Dari Gambar 4.17, menunjukkan bahwa nilai *precision* dan *recall* terkecil diperoleh oleh data latih dengan jumlah 1000, sedangkan untuk nilai *precision, recall* dan *f-measure* pada data latih dengan jumlah 2000 dan 3000 memiliki nilai

yang sama. Peningkatan data latih dari 1000 menjadi 2000 juga meningkatkan tingkat *precision* dan *recall* dari 500 data uji yang digunakan. Hal ini menyimpulkan bahwa peningkatan data latih dapat meningkatkan pula tingkat akurasi dari *decision tree* yang dibentuk. Sedangkan untuk data latih dengan jumlah 2000 dan 3000 menunjukkan nilai yang landai, atau tidak mengalami peningkatan maupun penurunan, karena jumlah data 2000 sudah cukup untuk mengklasifikasi sejumlah 500 data uji, sedangkan variasi yang diperoleh data latih pada data latih 3000 tidak mengalamin perubahan yang signifikan dibanding data latih jumlah 2000, sehingga *decision tree* yang dihasilkan tidak berubah.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil uji dan analisis yang telah dilakukan dapat diambil beberapa kesimpulan sebagai berikut:

1. Telah diimplementasi algoritima *SLIQ* untuk melakukan klasifikasi siswa SMK yang diterima dalam jurusan, dalam bentuk aplikasi *desktop* berbasis Java. Adapun implementasi dilakukan dengan menerapkan pemanfaatan fitur *multithreading* yang dimiliki oleh Java. Langkah-langkah utama yang diterapkan dalam aplikasi adalah :
    1. Pembentukan AttributeList dan ClassList.
    2. Pengurutan AttributeList tipe numerik dengan algortima *QuickSorting*.
    3. Pembentukan *tree* dengan metode BFS (Breadth First Search).
    4. Pemangkasan *tree* dengan metode *bottom-up*.
2. Telah diketahui tingkat akurasi dalam pengklasifikasian jurusan siswa SMK baru menggunakan algoritma *SLIQ,* dengan bantuan aplikasi yang telah dibangun. Tingkat akurasi untuk *decision tree* yang dibentuk algoritma SLIQ dengan data uji sama dengan data latih, paling kecil 0.87 dan paling besar 0.98. Sedangkan untuk nilai akurasi untuk data uji data tahun 2011, memiliki nilai paling kecil 0.65 dan nilai akurasi paling besar 0.95 yang data latihnya merupakan data gabungan dari tahun 2008-2010. Nilai *precision* terendah untuk data uji tahun 2011 diperoleh *tree* dengan data latih tahun 2008 sebesar 0.33, sedangkan yang tertinggi diperoleh *tree* dengan data latih gabungan tahun 2008-2010 sebesar 0.85. Nilai *recall* terendah untuk data uji tahun 2011 diperoleh *tree* dengan data latih tahun 2008 sebesar 0.39, sedangkan yang tertinggi diperoleh *tree* dengan data latih gabungan tahun 2008-2010 sebesar 0.75. Sedangkan Nilai *f-measure* data latih 2008 memiliki nilai terkecil jika diujikan dengan tahun 2011, yaitu sebesar 0.36, sedangkan nilai *f-measure* tertinggi diperoleh data latih gabungan tahun 2008-2010, sebesar 0.78.

5.2 Saran

Beberapa saran yang dapat disampaikan untuk penelitian selanjutnya yaitu :

1. Untuk menambah kecepatan dalam membentuk *decision tree*, proses *build tree* dan *pruning tree* dapat dilakukan terintegrasi atau dilakukan secara bersamaan.
2. Untuk meningkatkan akurasi, *precision* dan *recall*, dapat ditambahkan atribut pada data latih, seperti asal sekolah, nilai UAS dan atribut-atribut lainnya yang dapat menjadi bahan pendukung dalam klasifikasi jurusan siswa baru SMK.

DAFTAR PUSTAKA

[CAL-91] Calett, J.1991. *Megainduction: Machine Learning on Very Large Databases*. University of Sydney.

[COM-07] Compumine. 2007. *Evaluating a Classification Model* . http://www.compumine.com/web/public/newsletter/20071/precision-recall. Diakses pada tanggal 29 Mei 2012

[HAN - 06] Han Jiawei, Micheline Kamber. 2006. *Data Mining : Concept and Techniques, 2nd ed*. Morgan Kaufmann Publishers

[HUN-66] Hunt, E,B., Marin. dan Stone, P.J. 1966. *Experiments in Induction*. New York. Academic Press

[KIN-08] Kingsford Carl, Steven L Salzberg. 2008. *What are Decision Trees?*. Nature Publishing Group

[KOT-07] Kotsiantis, S.B. 2007. *Supervised Machine Learning: A Review of Classification Techniques.* Department of Computer Science and Technology

[MAN-96] Manish Mehta, Rakesh Agrawal dan Jorma Rissanen. 1996.*SLIQ: A Fast Scalable Classifier for Data Mining*. IBM Almaden Research Center

[MIT-97] Mitchell, Tom M. 1997. *Machine Learning. Singapore*. McGraw-Hill

[OLS-07] Olson, David, dan Yong Shi. 2007. *Intoduction to Business Data Mining*. New York. McGraw-Hill

[PAN-06] Pang-Ning Tan, Michael Steinbach dan Vipin Kumar. 2006. *Introduction to Data Mining*. Addison-Wesley

[RAS-00] Rastogi, Rajeev dan Kyuseok Shim. 2000. *PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning*. Kluwer Academic Publishers

[YE-03] Ye, Nong. 2003. *The Handbook of Data Mining*. London. Lawrence Erlbaum Associates

[ZHAN-03] Zhang, S., Zhang, C., Yang, Q. 2003. *Data Preparation for Data Mining. Applied Artificial Intelligence, Volume 17*. Taylor & Francis