KLASIFIKASI BERITA BERBAHASA INGGRIS MENGGUNAKAN ALGORITMA C4.5 BERBASIS ONTOLOGI

SKRIPSI

Diajukan untuk memenuhi persyaratan memperoleh gelar Sarjana Komputer

Disusun oleh :

WINY FIRDASARI

NIM. 105090601111006

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

PROGRAM STUDI INFORMATIKA/ILMU KOMPUTER

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2014

LEMBAR PERSETUJUAN
KLASIFIKASI BERITA BERBAHASA INGGRIS MENGGUNAKAN ALGORITMA C4.5 BERBASIS ONTOLOGI

SKRIPSI

Disusun oleh :

WINY FIRDASARI

NIM. 105090601111006

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I,	Dosen Pembimbing II,
Indriati, ST., M.Kom	**Drs. Achmad Ridok, M.Kom**
NIK. 831013 06 1 2 0035	**NIP. 19680825 199403 1 002**

LEMBAR PENGESAHAN
KLASIFIKASI BERITA BERBAHASA INGGRIS MENGGUNAKAN ALGORITMA C4.5 BERBASIS ONTOLOGI

SKRIPSI

Sebagai salah satu syarat untuk memperoleh
Gelar Sarjana dalam bidang Ilmu Komputer

Disusun oleh:

WINY FIRDASARI
NIM. 105090601111006

Skripsi ini telah diuji dan dinyatakan lulus
pada tanggal 8 Oktober 2014

Dosen Penguji I	Dosen Penguji II
Budi Darma Setiawan, S.Kom., M.Cs	**Edy Santoso, S.Si., M.Kom**
NIK.841015 06 1 1 0090	**NIP.19740414 200312 1 004**

Dosen Penguji III

Barlian Henryranu P, ST., MT
NIK.821024 06 1 1 0254

Mengetahui
Ketua Program Studi Informatika/Ilmu Komputer

Drs. Marji, MT.
NIP.19670801 199203 1 001

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, 27 Agustus 2014

Winy Firdasari

NIM. 105090601111006

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, karena atas segala rahmat dan limpahan hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul **"Klasifikasi Berita Berbahasa Inggris Menggunakan Algoritma C4.5 Berbasis Ontologi"**. Skripsi ini diajukan sebagai syarat untuk memperoleh gelar Sarjana Komputer Strata Satu (S-1) Program Studi Informatika/Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya, Malang.

Penulis menyadari bahwa skripsi ini tidak dapat terselesaikan tanpa bantuan baik moral maupun materiil dari banyak pihak. Untuk itu, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Indriati, ST., M.Kom selaku dosen pembimbing utama yang telah meluangkan waktu untuk memberikan pengarahan dan bimbingan kepada penulis.
2. Drs. Achmad Ridok, M.Kom selaku pembimbing kedua dan selaku dosen penasehat akademik yang telah meluangkan waktu untuk memberikan pengarahan dan bimbingan kepada penulis.
3. Drs. Marji, MT., selaku Ketua Program Studi Informatika/Ilmu Komputer Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya.
4. Ir. Sutrisno, MT, selaku Ketua Program Teknologi Informasi dan 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 dan Ilmu Komputer Universitas Brawijaya.
7. Segenap staf dan karyawan di Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan skripsi ini.

8. Bapak Adi Supriono, Ibu Maria Ulfah, dan Kakak Frastiqa Fahrany, serta seluruh keluarga besar tercinta, terima kasih atas doa restu, kasih sayang, dan perhatian yang tulus serta dukungan yang telah diberikan.
9. Mochamad Jazuly yang telah memberikan semangat dan dukungan selama masa kuliah.
10. Sahabat-sahabat dan rekan seperjuangan penulis selama kuliah, Novia, Afi, Ahlam, Jihad, Fahmi, Dedy, dan Ilham atas segala dukungan dan semua perjuangan selama ini.
11. Rekan-rekan mahasiswa Program Studi Informatika/Ilmu Komputer angkatan 2007, 2008, 2009, dan 2010 yang telah memberikan masukan, dukungan, serta semangat kepada penulis.
12. Semua pihak yang telah terlibat baik secara langsung maupun tidak langsung yang tidak dapat penulis 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, 27 Agustus 2014

Penulis

ABSTRAK

Winy Firdasari. 2014. Klasifikasi Berita Berbahasa Inggris Menggunakan Algoritma C4.5 Berbasis Ontologi

Dosen Pembimbing : Indriati ST., M.Kom dan Drs. Achmad Ridok, M.Kom

Penelitian ini mengimplementasikan algoritma C4.5 berbasis ontologi untuk mengklasifikasikan teks berita berbahasa Inggris. Data yang digunakan pada penelitian adalah *Reuters-21578* 90 kategori. Sistem yang akan dibuat dilengkapi dengan penggunaan *WordNet* sebagai basis ontologinya. Manfaat dari *WordNet* ini adalah untuk menemukan *term* yang saling bersinonim di dalam dokumen latih dan dokumen uji. Algoritma C4.5 merupakan algoritma klasifikasi yang diterapkan pada teknik *decision tree*. Dalam algoritma ini pemilihan atribut yang akan diproses menggunakan *information gain* tertinggi dipilih sebagai *parent* bagi *node* selanjutnya. Dari proses pelatihan pada dokumen latih akan dihasilkan pembentukan *rule tree* yang nantinya akan digunakan untuk pengkategorian dokumen uji. Pengujian untuk sistem menggunakan jumlah dataset yang bervariasi, skenario pertama adalah 40, 80, 120, 160, 200, dan 240 untuk dokumen latih sementara dokumen uji ditentukan tetap jumlahnya yaitu 20. Skenario kedua, perbandingan dokumen latih dan dokumen uji adalah 30% : 70%, 40% : 60%, 50% : 50%, 60% : 40%, 70% : 30% dan 90% : 10% dari total data 200. Sedangkan skenario ketiga adalah menggunakan *K-Fold* dimana kombinasi pertama k = 3 dengan 40 dokumen latih untuk tiap *subset*, kombinasi kedua k = 3 dengan 80 dokumen latih untuk tiap *subset*, dan kombinasi ketiga k = 2 dengan 120 dokumen latih untuk tiap *subset*. Sementara jumlah dokumen uji dibuat tetap yaitu 20 yang bukan termasuk dalam dokumen latih. Hasil uji coba menunjukkan nilai *f-measure* tertinggi sebesar 60.24% pada skenario pertama dengan data latih berjumlah 40 dan data uji berjumlah 20. Nilai *f-measure* yang diperoleh pada saat pengujian menunjukkan hasil yang didapat tidak tergantung pada jumlah dokumen latih, melainkan sangat tergantung pada frekuensi *term* dokumen.

Kata Kunci : C4.5, *decision tree*, klasifikasi teks, ontologi, *WordNet*

ABSTRACT

Winy Firdasari. 2014. Classification of English News Text Using C4.5 Algorithm Based Ontology

Advisor: Indriati ST., M.Kom and Drs. Achmad Ridok, M.Kom

This research implements C4.5 algorithm based ontology algorithm to classify English news text. Reuters-21578 with 90 categories will be used as dataset. The system that will be built is using WordNet as its ontology basis. The function of the WordNet in this case was to find synonymous words within the training document and the testing document. C4.5 Algorithm is a classification algorithm which is implemented in decision tree techniques. In this algorithm the atribute was processed using the highest value of *information gain* was selected as the parents for the next node. Additionally, the training process on the training document resulted in the construction of rule tree which was used for the categorization of the testing document. Testing for system using varying number of datasets, the first scenario is 40, 80, 120, 160, 200, and 240 for training document while the number of testing document which have been tested were constantly 20. The second scenario training document and testing document comparison test was 30% : 70%, 40% : 60%, 50% : 50%, 60% : 40%, 70% : 30% dan 90% : 10% of the data total 200. While the third scenario is using K-Fold where the first combination k = 3 with 40 training documents for each subset, the second combination k = 3 with 80 training documents for each subset, and the third combination k = 2 with 120 training documents for each subset. While the number of testing document which have been tested were constantly 20 were not included in the training documents. The experiment result show that highest f-measure value is 60.24% on 40 for training data and 20 for testing data. Based on those experiment result, it can be concluded that the f-measure value is influenced by the term frequency of documents instead of the number of the training documents.

Keywords : C4.5, decision tree, text classification, ontology, WordNet.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Berita merupakan kumpulan fakta berisi informasi yang dapat menarik perhatian pembaca. Pada zaman sekarang berita sudah menjadi kebutuhan pokok bagi masyarakat luas, karena dengan adanya berita, mereka dapat memperoleh informasi kejadian yang sedang hangat dibicarakan. Selain itu, berita juga dapat menambah wawasan pembaca karena banyak informasi-informasi penting tentang pengetahuan.

Berita dapat disampaikan melalui media cetak seperti surat kabar dan majalah ataupun melalui media elektronik seperti radio, televisi, dan internet. Namun, seiring berkembangnya teknologi informasi pada zaman sekarang, berita lebih banyak didistribusikan melalui internet. Berita-berita yang terdapat di internet sering kali dalam satu topik terdapat informasi yang berbeda-beda, akibatnya pengguna kesulitan mendapatkan informasi yang sesuai. Oleh karena itu, pengklasifikasian berita sangat diperlukan untuk mengatasi hal tersebut [LAN-10]. Untuk mendapatkan informasi dari teks berita membutuhkan waktu yang lama dikarenakan teks merupakan data yang tidak berstruktur tidak seperti data numerik yang dapat diolah dengan mudah menjadi sebuah informasi. Pengorganisasian teks menggunakan *text mining* adalah metode yang tepat untuk memudahkan pengambilan informasi dari teks berita [MUR-13].

Text mining dapat digambarkan sebagai penerapan teknik *data mining* untuk penemuan pengetahuan yang berguna atau menarik dari teks tidak terstruktur [MOO-05]. Adapun tugas khusus dari *text mining* yaitu pengorganisasian dokumen berdasarkan isinya (*text categorization*) [MUR-13].

Beberapa algoritma yang digunakan untuk pengkategorian teks diantaranya adalah *K-Nearest Neighbor* (KNN), *Naïve Bayes Classifier*, dan C4.5. Berdasarkan [KUS-09] menunjukkan bahwa klasifikasi dengan menggunakan *Nearest Neighbor* tidak lebih akurat dari algoritma C4.5 karena banyak kasus yang memiliki kedekatan yang sama dengan klasifikasi yang berbeda.

Sedangkan berdasarkan [SIL-09] menunjukkan bahwa klasifikasi dengan menggunakan algoritma C4.5 lebih baik daripada *Naïve Bayes Classifier* karena algoritma C4.5 memberikan hasil berupa aturan-aturan klasifikasi dalam bentuk *if-then* dan dalam bentuk pohon keputusan (*decision tree*) serta menunjukkan karakteristik data yang terklasifikasi, yaitu berupa frekuensi *term* dalam setiap dokumen.

Beberapa penelitian tentang pengkategorian teks telah banyak dilakukan, diantaranya penelitian [SWA-13], [MUR-13], dan [LAT-12]. Menurut penelitian [LAT-12], pengkategorian teks didasarkan pada morfologi kata. Namun, pada penelitian tersebut terdapat kelemahan pada proses penentuan kemiripan teks, yaitu apabila terdapat teks uji yang memiliki *term* yang berbeda secara morfologi dari *term* pada teks latih padahal kedua *term* tersebut memiliki makna yang sama maka kedua teks tersebut tidak dapat dikatakan mirip. Hal ini memungkinkan teks uji tersebut akan dikelompokkan ke dalam kelompok yang berbeda dari kelompok teks latih tersebut. Untuk mengatasi masalah tersebut, dapat ditambahkan *knowledge background* dengan menggunakan *database WordNet* dimana dapat menghubungkan kata secara konseptual dan semantik serta dapat meningkatkan kinerja dari *text categorization* [HOT-03].

Pada penelitian ini mencoba mengimplementasikan algoritma C4.5 untuk pengklasifikasian dokumen, dimana untuk mendapatkan hasil pengklasifikasian dokumen melalui proses-proses seperti *preprocessing* dan proses klasifikasi untuk menentukan kategorinya. Di dalam proses *preprocessing* terdapat tahap-tahap seperti *case folding, tokenizing, filtering, stemming, dictionary construction ontology extraction,* dan *term weighting*. Sedangkan pada proses klasifikasi terdapat tahap-tahap seperti penentuan atribut untuk inputan melalui perhitungan *information gain*, penentuan *split-point*, dan pembentukan *rule tree.*

Berdasarkan latar belakang yang telah dipaparkan, maka judul yang diambil pada penelitian ini adalah **"Klasifikasi Berita Berbahasa Inggris Menggunakan Algoritma C4.5 Berbasis Ontologi"**.

1.2 Rumusan Masalah

Adapun permasalahan yang dapat dirumuskan dari latar belakang adalah sebagai berikut :

1) Bagaimana implementasi dari algoritma C4.5 dalam mengklasifikasikan berita berbahasa Inggris berbasis ontologi?
2) Bagaimana performansi yang diperoleh dari algoritma C4.5 terhadap pengklasifikasian berita berbahasa Inggris yang diukur melalui *precision*, *recall*, dan *f-measure*?

1.3 Batasan Masalah

Berdasarkan rumusan masalah, penulis perlu memberikan batasan masalah sebagai berikut :

1) Data yang digunakan pada penelitian adalah berita berbahasa Inggris yang dikeluarkan oleh Reuters dan *corpus* yang digunakan adalah Reuters-21578.
2) Teks berita yang akan dipakai pada penelitian ini diambil dari file teks yang telah diubah menjadi format **.txt*.
3) Kategori berita yang digunakan terdiri dari 4 kategori, yaitu *interest, money-fx, trade, crude*.
4) Menggunakan metode *Porter2-Stemmer* untuk proses *stemming*.
5) Menggunakan *WordNet* sebagai basis ontologinya.

1.4 Tujuan

Tujuan dari penelitian ini, antara lain:

1) Mengimplementasikan algoritma C4.5 dalam pengklasifikasian berita berbahasa Inggris berbasis ontologi.
2) Mengukur performansi yang diperoleh dari algoritma C4.5 terhadap pengklasifikasian berita berbahasa Inggris yang diukur melalui *precision*, *recall*, dan *f-measure*.

1.5 Manfaat

Manfaat yang diharapkan dari penelitian ini adalah dapat mereduksi fitur-fitur yang tidak relevan, sehingga dapat meningkatkan performansi dari algoritma.

Selain itu diharapkan dapat lebih efisien dalam pengambilan informasi dari teks berita.

1.6 Sistematika Penulisan

Sistematika penulisan dalam penyusunan skripsi ini adalah sebagai berikut:

1) BAB I PENDAHULUAN

   Bab pendahuluan ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika penulisan dari penelitian ini.

2) BAB II KAJIAN PUSTAKA DAN DASAR TEORI

   Bab kajian pustaka dan dasar teori ini berisi teori, temuan, dan bahan penelitian sebelumnya yang diperoleh dari berbagai referensi yang dijadikan dasar melakukan penelitian. Teori-teori yang akan dibahas pada bab ini adalah teori tentang berita, *text mining*, algoritma C4.5, ontologi, dan teori-teori lain yang berhubungan dengan pengklasifikasian dokumen.

3) BAB III METODE PENELITIAN DAN PERANCANGAN

   Bab metode penelitian dan perancangan ini berisi metode-metode yang dilakukan dalam proses pengklasifikasian beserta perancangan sistemnya, dimana untuk proses perancangannya meliputi perancangan sistem, perancangan algoritma, perhitungan manual, perancangan antarmuka, dan perancangan uji coba.

4) BAB IV IMPLEMENTASI

   Bab implementasi ini berisi hasil implementasi pada sistem, meliputi lingkungan implementasi (spesifikasi perangkat lunak dan perangkat keras), implementasi algoritma C4.5, dan implementasi antarmuka.

5) BAB V ANALISA HASIL DAN PEMBAHASAN

   Bab analisa hasil dan pembahasan ini berisi skenario pengujian, hasil pengujian, serta analisa pengujian dari hasil penerapan algoritma C4.5 untuk pengklasifikasian berita.

6) BAB VI PENUTUP

   Bab penutup ini berisi kesimpulan dari hasil penelitian dan saran untuk pengembangan penelitian kedepannya.

BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

Bab ini berisi tinjauan pustaka dan dasar teori yang berhubungan dengan klasifikasi berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi.

2.1 Tinjauan Pustaka

Tinjauan pustaka pada penelitian ini adalah membahas penelitian [MUR-13] dan [SIL-09]. [MUR-13] melakukan penelitian tentang klasifikasi berita berbahasa Inggris menggunakan *K-Nearest Neighbor* (KNN) berbasis ontologi. Hasil penelitian menunjukkan bahwa beberapa data latih memiliki kemiripan antar kategorinya dilihat dari isinya. Jika nilai "*k*" yang diberikan kecil maka dokumen tersebut tidak dapat diklasifikasikan dengan tepat dikarenakan nilai *scoring* ketika proses KNN yang dimiliki kategori sebenarnya dari dokumen tersebut memiliki nilai lebih kecil daripada kategori lain.

Sedangkan [KUS-09] melakukan penelitian tentang perbandingan metode *Nearest Neighbor* dan algoritma C4.5 untuk menganalisis kemungkinan pengunduran diri calon mahasiswa di STMIK AMIKOM Yogyakarta. Hasil penelitian menunjukkan bahwa klasifikasi dengan menggunakan *Nearest Neighbor* tidak lebih akurat dari algoritma C4.5 karena banyak kasus yang memiliki kedekatan yang sama dengan klasifikasi yang berbeda.

Berdasarkan kedua penelitian tersebut, maka dilakukan penelitian untuk mengetahui apakah algoritma C4.5 tetap memberikan hasil yang baik jika diterapkan pada kasus pengklasifikasian teks berbahasa Inggris berbasis ontologi.

2.2 Klasifikasi Teks

Klasifikasi atau kategorisasi teks adalah proses penempatan suatu teks ke suatu kategori atau kelas sesuai dengan karakteristik dari teks tersebut. Dalam *text mining*, klasifikasi mengacu kepada aktifitas menganalisis atau mempelajari himpunan teks *pre-classified* untuk memperoleh suatu model atau fungsi yang

dapat digunakan untuk mengelompokkan teks lain yang belum diketahui kelasnya ke dalam satu atau lebih kelas *predefined* tersebut [SAL-12].

Klasifikasi teks adalah proses pengelompokan dokumen ke dalam kelas berbeda, dalam tahapannya tiap dokumen *d* menunjuk pada satu kelas tertentu maka dibutuhkan proses untuk menggali informasi dari dokumen tersebut. Sehingga dokumen tersebut harus dapat merepresentasikan dari kelasnya sehingga tiap kata yang muncul dalam dokumen mempunyai nilai [SAL-12].

Klasifikasi teks atau *text categorization* secara garis besar terbagi dalam tiga kelompok yaitu yang pertama adalah klasifikasi berbasis statistik. Metode yang digunakan pada kelompok ini adalah *Naïve Bayesian, K-Nearest Neighbor, Category Center Vector, Support Vector Machine,* dan model *Maximum Entropy.* Kelompok kedua adalah klasifikasi teks berbasiskan keterkaitan dan metode yang digunakan adalah jaringan syaraf tiruan. Kelompok ketiga adalah klasifikasi teks berbasiskan *rule* atau aturan. Metode yang digunakan adalah *decision tree* [XIA-09].

2.3 Berita

Secara sosiologis, berita adalah semua hal yang terjadi di dunia. Dalam gambaran yang sederhana, seperti dilukiskan dengan baik oleh para pakar jurnalistik, berita adalah apa yang ditulis surat kabar, apa yang disiarkan radio, dan apa yang ditayangkan televisi. Berita menampilkan fakta, tetapi tidak setiap fakta merupakan berita. Berita biasanya menyangkut orang-orang, tetapi tidak setiap orang bisa dijadikan berita. Berita merupakan sejumlah peristiwa yang terjadi di dunia, tetapi hanya sebagian kecil saja yang dilaporkan [SUM-05].

2.4 *Data Mining*

Data mining adalah bidang multi-disiplin yang menggambarkan kerja dari statistik, teknologi basis data, kecerdasan buatan, pengenalan pola, mesin pembelajaran, teori informasi, pengambilan pengetahuan, pencarian informasi, komputasi tingkat tinggi, dan visualisasi data [SUM-06]. Sedangkan menurut [HAN-06], *data mining*, sering juga disebut sebagai *knowledge discovery from data (KDD)*, adalah ekstraksi otomatis dari sebuah pola yang merepresentasikan pengetahuan implisit yang disimpan atau didapat dalam *database* yang besar,

gudang data, *web*, tempat penyimpanan informasi yang besar lainnya atau berkas data.

Data mining adalah salah satu bidang yang berkembang sangat cepat pada industri komputer. Salah satu kelebihan *data mining* adalah banyaknya metode dan teknik yang bisa diaplikasikan pada sekumpulan masalah. Keberhasilan dari penggunaan *data mining* bergantung dari kemampuan, kreativitas dari designer data mining itu sendiri. Esensi dari data mining adalah memecahkan seperti memecahakan sebuah *puzzle* [KAN-03].

2.5 *Text Mining*

Text mining adalah salah satu bidang khusus dari *data mining*. *Text mining* merupakan sebuah teknologi baru yang dapat digunakan untuk menambang data yang telah ada dalam sebuah database dengan membuat suatu data berupa teks yang tidak terstruktur menjadi data yang dapat dianalisa [FRA-10].

Text mining juga dapat didefinisikan sebagai suatu proses menggali informasi dimana seorang *user* berinteraksi dengan sekumpulan dokumen menggunakan *tools* analisis yang merupakan komponen-komponen dalam *data mining* yang salah satunya adalah kategorisasi. Tujuan dari *text mining* adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada *text mining* adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur [FEL-07]. Adapun tugas khusus dari *text mining* antara lain yaitu pengkategorisasian teks (*text categorization*) dan pengelompokan teks (*text clustering*) [MUR-13].

Salah satu elemen kunci dari *text mining* adalah kumpulan dokumen yang berbasis teks. Pada prakteknya, *text mining* ditujukan untuk menemukan pola dari sekumpulan dokumen yang jumlahnya sangat besar dan bisa mencapai jumlah ribuan bahkan sampai jutaan. Koleksi dokumen bisa statis, dimana dokumen tidak berubah, dimana dokumen selalu di-*update* sepanjang waktu [TRI-09].

Sistem *text mining* terdiri dari komponen *text preprocessing*, *feature selection*, dan komponen *data mining*. Komponen *text preprocessing* berfungsi untuk mengubah data tekstual yang tidak terstruktur seperti dokumen, ke dalam data terstruktur dan disimpan ke dalam basis data. *Feature selection* akan memilih

kata yang tepat dan berpengaruh pada proses klasifikasi. Komponen terakhir akan menjalankan teknik *data mining* pada *output* dari komponen sebelumnya [LAN-10].

2.6 *Text Preprocessing*

Text Preprocessing merupakan tahapan awal dalam pengolahan *teks*. Struktur informasi yang digali pada *text mining* tentu saja masih berupa informasi sembarang. Oleh karena itu, diperlukan proses pengubahan bentuk menjadi data yang terstruktur sesuai kebutuhannya untuk proses dalam data mining yang biasanya akan menjadi nilai-nilai numerik [LAN-10]. Proses *preprocessing* terdiri dari *case folding, tokenizing, filtering, stemming* dan tahap pembobotan kata yang dilakukan secara sekuensial [MUR-13].

2.6.1 *Case Folding*

Case Folding adalah proses pengubahan karakter huruf besar menjadi huruf kecil. Hanya huruf ‘a‘ sampai ‘z‘ yang diterima. Karakter selain huruf akan dihilangkan dan dianggap sebagai *delimiter* [MAN-09].

2.6.2 *Tokenizing*

Tokenizing adalah proses memotong sebuah urutan karakter menjadi sebuah potongan-potongan, yang disebut *token*, mungkin pada waktu yang sama membuang karakter tertentu, seperti tanda baca [MAN-09].

2.6.3 *Filtering*

Proses *filtering* melakukan penyaringan kata hasil dari *tokenizing*, dimana kata yang tidak relevan dibuang. Proses ini menggunakan pendekatan *stoplist* (membuang kata yang tidak penting) dan *wordlist* (menyimpan kata penting). *Stoplist* adalah sekumpulan kata yang tidak mempunyai relevansi terhadap dokumen namun sering kali muncul dalam sebuah dokumen [MAN-09].

Stopwords adalah daftar kata-kata yang sering digunakan tetapi tidak menggambarkan isi dari dokumen. Contoh *stop words* dalam bahasa Inggris diantaranya adalah *“in”, “and”, “what”, “we”, “they”, “usually”* dan seterusnya. Dalam penelitian ini, daftar *stopwords* diambil dari situs

http://www.ranks.nl/resources/stopwords.html. Di dalam situs tersebut terdapat 671 *stopwords.*

2.6.4 *Stemming*

Stemming merupakan tahap mencari *root* (akar) kata dari tiap kata hasil *filtering*. Pada tahap ini dilakukan proses pengembalian berbagai bentukan kata ke dalam suatu representasi yang sama [LAN-10].

Stemming adalah proses pemotongan berbagai imbuhan atau *affixes* seperti awalan (*prefixes*), sisipan (*infixes*), akhiran (*suffixes*) dan kombinasi awalan dan akhiran (*confixes*). *Stemming* dilakukan setelah proses *filtering.* Tujuan dari *stemming* ini adalah mengurangi bentuk jamak atau bentuk turunan yang berhubungan dengan kata dasar yang umum [MAN-09].

Algoritma yang umum digunakan untuk *stemming* bahasa Inggris adalah *Porter-Stemmer*. Algoritma ini dibuat oleh M.F.Porter pada tahun 1980. Secara garis besar algoritma ini melakukan penghilangan akhiran *morphological* dan *infleksional* yang umum dari bahasa Inggris [POR-80].

Algoritma *Porter-Stemmer* mengalami sedikit perubahan pada September, 2005 menjadi *Porter2-Stemmer*. Hal-hal yang membedakan algoritma *Porter2-Stemmer* dengan versi sebelumnya adalah terdapat penanganan khusus sebagai berikut [POR-05]:

1. Menangani *term-term* pendek yang memiliki akhiran *-e* dan *–y*, seperti *sky, skies, ski, skis* karena terdapat kesalahan penanganan pada bentuk ini, sehingga *term-term* tersebut diperlakukan khusus.
2. Demikian pula terdapat masalah dengan *term* yang berimbuhan *–ing*, terdapat tiga *verb* yang perlu diperlakukan khusus, seperti *die, lie,* dan *tie* menjadi *dying, lying,* dan *tying*.
3. Sedikit berhati-hati dengan bentuk imbuhan *–ing* tertentu, seperti *inning, outing, canning* yang mana tidak bisa di-*stemming* menjadi *in, out, can*.
4. Penghapusan akhiran *–ly* juga memiliki pengecualian, *seperti idly, gently, ugly, early, only, singly*.
5. Penanganan khusus juga dilakukan pada *term* seperti *news* yang bukan bentuk jamak dari *new*, *howe* adalah *family name* sehingga perlu dipisahkan dari *how*,

dan *succeed* bukan merupakan bentuk past participle, sehingga *–ed* tidak boleh dihapus.

6. Beberapa *term non-plural* yang memiliki akhiran *–s* ditambahkan, seperti *atlas, cosmos*, dan lain sebagainya.

Selain penanganan khusus di atas, lima langkah linier tetap sama dengan algoritma *Porter-Stemmer* pada versi sebelumnya. Pada setiap langkah tersebut terdapat beberapa *rule* dan kondisi untuk penghapusan akhiran. Jika aturan *suffix* sesuai dengan *term*, maka kondisi yang ada di dalam rule tersebut akan di cek kondisi mana yang memenuhi, jika sudah memenuhi kondisi maka *rule* akan di-*fire*. Contoh dari kondisi di dalam *rule* yaitu jumlah karakter vokal yang diikuti oleh karakter konsonan pada *stem* harus lebih besar daripada *rule* yang akan dijalankan.

Langkah-langkah algoritma *Porter-Stemmer* adalah sebagai berikut:

1. *Step* 1a : Menghilangkan *plural suffixation*.
2. *Step* 1b : Menghapus *verbal inflection*.
3. *Step* 1b1 : Mengubah *term* yang berimbuhan *–ed* dan *–ing*
4. *Step* 1c : Mengubah akhiran *–y* menjadi *-i*
5. *Step* 2 : mengganti akhiran *–ational* menjadi *–ate, -tional* menjadi *–tion, -enci* menjadi *–ence*, dsb
6. *Step* 3 : pemotongan terhadap *stem* yang berakhiran *–icate, -ative, -alize, -iciti, -ical, -ful*, dan *–ness*
7. *Step* 4 : penghapusan akhiran *–al, -ance, -ence, -er, -ic*, dsb
8. *Step* 5a : pemotongan terhadap huruf *–e*,
9. *Step* 5b : reduksi terhadap huruf konsonan dobel.

Aturan pemilihan *suffix* (akhiran) pada setiap *step* akan dijelaskan pada tabel 2.1 sampai tabel 2.9.

Tabel 2.1 ***Stemming Step* 1a**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
NULL	*Sses*	*ss*	*caresses -> caress*
NULL	*Ies*	*i*	*ponies -> poni*

			ties -> tie
NULL	*Ss*	*ss*	*caress -> caress*
NULL	*S*	*NULL*	*cats -> cat*

Tabel 2.2 ***Stemming Step* 1b**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*Eed*	*ee*	*feed -> feed*
			agreed -> agree
*(*v*)*	*Ed*	*NULL*	*plasteres -> plaster*
			bled -> bled
*(*v*)*	*Ing*	*NULL*	*motoring -> motor*
			sing -> sing

Tabel 2.3 ***Stemming Step* 1b1**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
NULL	*At*	*ate*	*conflat(ed) -> conflate*
NULL	*Bl*	*ble*	*troubl(ing) -> trouble*
NULL	*Iz*	*ize*	*siz(ed) -> size*
*(*d and not (*<L> or *<S> or *<Z>))*	*NULL*	*single letter*	*hopp(ing) -> hop*
			tann(ed) -> tan
			fall(ing) -> fall
			hiss(ing) -> hiss
			fizz(ed) -> fizz
*(m=1 and *o)*	*NULL*	*e*	*fail(ing) -> fail*
			fil(ing) -> file

Tabel 2.4 ***Stemming Step* 1c**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
*(*v*)*	*Y*	*i*	*happy -> happi*
			sky -> sky

Tabel 2.5 ***Stemming Step* 2**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*ational*	*Ate*	*relational -> relate*
(m>0)	*tional*	*Tion*	*conditional -> condition*
			rational -> rational
(m>0)	*enci*	*Ence*	*valenci -> valence*
(m>0)	*anci*	*Ance*	*hesitanci -> hesitance*
(m>0)	*izer*	*Ize*	*digitizer -> digitize*
(m>0)	*abli*	*Able*	*conformabli -> conformable*
(m>0)	*alli*	*Al*	*radicalli -> radical*
(m>0)	*ently*	*Ent*	*differently -> different*
(m>0)	*eli*	*E*	*vileli -> vile*
(m>0)	*ousli*	*Ous*	*analogousli -> analogous*
(m>0)	*ization*	*Ize*	*vietnamization -> vietnamize*
(m>0)	*ation*	*Ate*	*predication -> predicate*
(m>0)	*ator*	*Ate*	*operator -> operate*
(m>0)	*alism*	*Al*	*feudalism -> feudal*
(m>0)	*iveness*	*Ive*	*decisiveness -> decisive*
(m>0)	*fullness*	*Ful*	*hopefulness -> hopefull*
(m>0)	*ousness*	*Ous*	*callousness -> callous*
(m>0)	*aliti*	*Al*	*formality -> formal*
(m>0)	*iviti*	*Ive*	*sensitivity -> sensitive*

(m>0)	*bility*	*Ble*	*sensibility -> sensible*

Tabel 2.6 ***Stemming Step* 3**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*Icate*	*ic*	*triplicate -> triplic*
(m>0)	*Ative*	*NULL*	*formative -> form*
(m>0)	*Alize*	*ai*	*formalize -> formal*
(m>0)	*Iciti*	*ic*	*electricity -> electric*
(m>0)	*Ical*	*ic*	*electrical -> electric*
(m>0)	*Ful*	*NULL*	*hopeful -> hope*
(m>0)	*Ness*	*NULL*	*goodness -> good*

Tabel 2.7 ***Stemming Step* 4**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>1)	*al*	*NULL*	*revival -> reviv*
(m>1)	*ance*	*NULL*	*allowance -> allow*
(m>1)	*ence*	*NULL*	*inference -> infer*
(m>1)	*er*	*NULL*	*airliner -> airlin*
(m>1)	*ic*	*NULL*	*gyroscopic -> gyroscop*
(m>1)	*able*	*NULL*	*adjustable -> adjust*
(m>1)	*ible*	*NULL*	*defensible -> defens*
(m>1)	*ant*	*NULL*	*irritant -> irrit*
(m>1)	*ement*	*NULL*	*replacement ->replac*
(m>1)	*ment*	*NULL*	*adjustment -> adjust*
(m>1)	*ent*	*NULL*	*dependent -> depend*
(m>1)	*ion*	*NULL*	*adoption -> adopt*
(m>1)	*ou*	*NULL*	*homologou -> homolog*

(m>1)	*ism*	*NULL*	*communism -> commun*
(m>1)	*ate*	*NULL*	*activate -> activ*
(m>1)	*iti*	*NULL*	*angularity -> angular*
(m>1)	*ous*	*NULL*	*homologous -> homolog*
(m>1)	*ive*	*NULL*	*effective -> effect*
(m>1)	*ize*	*NULL*	*bowdlerize -> bowdler*

Tabel 2.8 ***Stemming Step* 5a**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>1)	*E*	*NULL*	*probate -> probat*
			rate -> rate
*(m=1 and not *o)*	*E*	*NULL*	*cease -> ceas*

Tabel 2.9 ***Stemming Step* 5b**

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
*(m=1 and not *d and *<L>)*	*NULL*	*single letter*	*controll ->control*
			roll -> roll

Tabel 2.10 **Daftar Pengecualian *Term***

Exception
skis -> ski
skies -> sky
dying -> die
lying -> lie
tying -> tie
succeed -> succeed
proceed -> proceed

exceed -> exceed
idly -> idl
gently -> gentl
singly -> singl
news -> news
howe -> howe
atlas -> atlas
cosmos -> cosmos
bias -> bias
andes -> andes

Keterangan:

m : ukuran (*measure*) dari sebuah stem berdasarkan urutan vokal – konsonan

*<X> : *stem* berakhir dengan huruf X

v : *stem* mengandung sebuah vokal

*d : *stem* diakhiri dengan konsonan dobel

*o : *stem* diakhiri dengan konsonan – vokal – konsonan, berurutan, dimana konsonan akhir bukan w, x, atau y

2.6.5 Ontologi

Proses ontologi dilakukan setelah *term* melalui proses *stemming*. Kata "ontologi" telah dikenal pada bidang filsafat sebagai subjek dari keberadaan. Pada konteks kecerdasan buatan, ontologi berarti spesifikasi eksplisit dari konseptualisasi yang saling berbagi. Ontologi digunakan untuk berbagi pengetahuan. Ontologi meningkatan informasi dan pengertian. Ontologi memiliki peran penting pada informasi berbasis komputer yang bersifat heterogen. Ontologi memiliki peran penting pada *web* generasi kedua yang oleh Tim Berners-Lee sebut sebagai *"Semantic Web"*. Mesin pencari atau *search engine* akan menemukan halaman dengan kata yang secara sintaksis berbeda tetapi memiliki kesamaan secara semantik [ELS-05].

Ontologi adalah sebuah deskripsi formal tentang sebuah konsep secara eksplisit dalam sebuah domain, properti dari setiap konsep beserta dengan batasannya. Sebuah konsep di ontologi dapat memiliki objek (*instance*). Secara teknis, ontologi direpresentasikan dalam bentuk *class*, *property*, *facet*, dan *instances*. *Class* menerangkan konsep atau makna dari suatu domain. *Class* adalah kumpulan dari elemen dengan sifat yang sama. Sebuah *class* bisa memiliki sub class yang menerangkan konsep yang lebih spesifik [BAS-10].

2.6.6 *WordNet*

Implementasi ontologi pada klasifikasi teks dapat dibantu dengan menggunakan *WordNet* sebagai basis ontologi. *WordNet* adalah database leksikal untuk bahasa Inggris. Kata benda, kata kerja, kata sifat, dan kata keterangan dikumpulkan menjadi sekumpulan kata yang bersinonim dan dinamakan *synsets*. *WordNet* menyerupai thesaurus, yang mengumpulkan kata berdasarkan maknanya. Tetapi, ada beberapa perbedaan pada *WordNet*. Pertama, *WordNet* menghubungkan tidak hanya antar bentuk kata, *string* dari huruf, tetapi secara spesifik menghubungkan makna dari kata tersebut. Kedua, *WordNet* memberikan nama hubungan semantik antar kata, dimana pengelompokkan kata pada thesaurus tidak mengikuti banyak pola.

WordNet terdiri dari dari hubungan semantik sebagai berikut :

- *Synonymy* adalah relasi dasar pada *WordNet*, karena *WordNet* menggunakan himpunan sinonim yang disebut *synsets* untuk merepresentasikan *senses* kata. *Synonymy* merupakan relasi simetris antar kata.
- *Antonymy* adalah lawan kata dan merupakan hubungan semantik simetris antar kata khususnya pada saat pengelompokkan arti dari kata sifat dan kata keterangan.
- *Hyponymy* (sub-kelas) dan kebalikannya *hypernymy* (super-kelas) adalah hubungan transitif antar *synsets*.
- *Meronymy* (bagian-kelas) dan kebalikannya *holonymy* (keseluruhan-kelas) adalah hubungan semantik yang kompleks pada *WordNet*.
- *Troponymy* adalah untuk kata kerja dimana pada *hyponymy* adalah kata benda.
- *Entailment* adalah hubungan antar kata kerja.

Pada *WordNet*, sebuah kata direpresentasikan string dari karakter ASCII. *WordNet* terdiri dari lebih dari 118.000 kata yang berbeda dan lebih dari 90.000 *senses* [MIL-95].

Pada penelitian ini hubungan semantik yang digunakan adalah *synonymy*. Sekumpulan kata yang saling bersinonim digabungkan dalam sebuah *synset.* Definisi dari relasi *synonymy* sendiri adalah dua buah ekspresi atau kata dikatakan saling bersinonim jika kata tersebut saling menggantikan dalam suatu kalimat maka tidak akan mengubah makna dari kalimat tersebut. Contoh dari *synset* dari kata *person* adalah [*individual*, *someone*, *somebody*, *mortal*, *human*, *drunk person*]. Jika ada suatu kalimat yang memiliki kata *person* di dalamnya kemudian kata *person* diganti dengan *synset* nya maka kalimat tersebut tidak akan berubah makna [MUR-13].

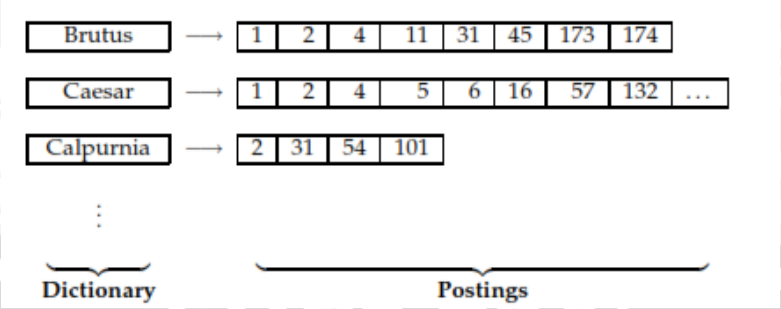
2.6.7 *Dictionary Construction*

Proses *dictionary construction* adalah proses konversi dokumen teks menjadi vektor fitur. *Term* yang dimasukkan ke dalam vektor fitur adalah *term* yang sudah melalui proses *stemming.* Setiap fitur di dalam vektor berkorespondensi dengan kata pada *dictionary* [GUO-04].

Metode yang biasa digunakan untuk *dictionary construction* adalah *Inverted Index* atau disebut juga *Inverted File*. *Inverted index* terdiri dari dua bagian yaitu *term list* dan *posting list. Term-term* yang ada dari dokumen latih disebut *term* list. Istilah kamus atau *dictionary* adalah struktur data yang digunakan sementara sekumpulan *term* disebut *vocabulary*. Untuk setiap *term*, akan terbentuk *list* yang menyimpan pada dokumen mana *term* tersebut muncul. Setiap *item* dari *list* tersebut yang menyimpan jumlah *term* tersebut muncul pada sebuah dokumen disebut *posting. List* dari *posting-posting* tersebut disebut *posting list* atau *inverted list* [MAN-09].

Implementasi dari *Inverted Index* adalah menggunakan struktur data *hashmap. Hashmap* terdiri *key* (kunci) dan *value* (nilai). *Hashmap inverted index* terdiri dari *term-term* yang merupakan *key* serta *posting list* merupakan *value*. *Posting list* berisi frekuensi tiap *term* pada setiap dokumen. Apabila *term* belum ada pada *hashmap* maka *term* akan disimpan dan membuat *posting list* baru.

Apabila *term* sudah ada tetapi nama dokumen belum ada, maka akan dibuat *posting list* baru untuk dokumen tersebut beserta nilai frekuensinya [MUR-13]. Ilustrasi dari *inverted index* dapat dilihat pada gambar 2.1.



Gambar 2.1 Ilustrasi *Inverted Index*

Sumber : [MAN-09]

Dalam proses *dictionary construction* juga dilakukan perhitungan kemunculan *term* tertentu dalam sejumlah dokumen yang disebut dengan *document frequency* [YAN-02].

2.6.8 *Feature Weighting*

Setelah proses *dictionary construction*, maka vektor dokumen tersebut dapat dihitung bobot tiap *term* nya untuk proses klasifikasi. *Feature weighting* (pembobotan) adalah proses memberikan bobot pada tiap-tiap *term.* Metode yang paling banyak digunakan adalah *Term Frequency-Inverse Document Frequency* (TF-IDF). Fungsi dari TF-IDF adalah memberi bobot pada setiap komponen vektor. *Term Frequency* adalah pembobotan *term* berdasarkan perhitungan jumlah *term* yang muncul pada suatu dokumen. Semakin tinggi nilai TF (*Term frequency*) maka semakin penting pula *term* tersebut untuk mendeskripsikan suatu dokumen [SOU-03].

Inverse Document Frequency (IDF) adalah proses mengukur *term* yang jarang muncul pada *corpus*. Penilaiannya menggunakan seluruh dokumen latih yang digunakan. Jika sebuah *term* sering muncul, maka *term* tersebut tidak bisa dianggap sebagai *term* yang mewakili dokumen tersebut. Sebaliknya, jika *term* yang jarang muncul pada *corpus*, maka *term* tersebut dikatakan memiliki hubungan dengan dokumen. TF-IDF mengalikan nilai TF dan IDF. Rumus untuk

menghitung TF-IDF dapat dilihat pada persamaan 2-1 sementara rumus untuk menghitung IDF dapat dilihat pada persamaan 2-2 [SOU-03].

$$wij = tfij \, . \, idfi \quad (2\text{-}1)$$

Dimana,

wij = bobot *term* i pada dokumen j.

$tfij$ = frekuensi *term* i pada dokumen j.

$$idfi = log(\frac{N}{df_i}) \quad (2\text{-}2)$$

Dengan

N = jumlah keseluruhan dokumen.

dfi = banyaknya dokumen yang mempunyai *term* i.

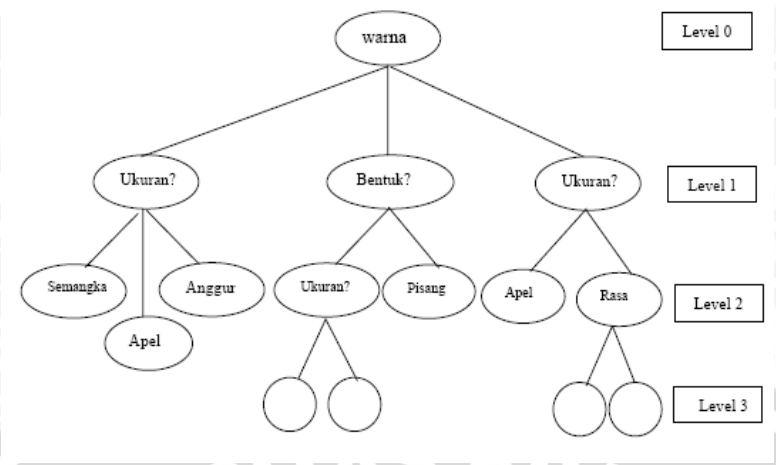
Sehingga persamaan bobot TF-IDF dapat dituliskan seperti pada persamaan 2-3.

$$wij = tfij \, . \, log(\frac{N}{df_i}) \quad (2\text{-}3)$$

2.7 *Decision Tree*

Decision tree adalah sebuah diagram alir yang mirip dengan struktur pohon, dimana setiap *interval node* notasikan atribut yang diuji, setiap cabangnya merepresentasikan hasil dari atribut tersebut, serta *leaf node* merepresentasikan kelas-kelas tertentu atau distribusi dari kelas-kelas [HAN-06].

Langkah ini akan berakhir pada suatu simpul jika pada simpul tersebut sudah ditemukan kelas atau jenis obyeknya. Kalau dalam satu tingkat suatu obyek telah diketahui termasuk dalam kelas tertentu, oleh karena itu berhenti di level tersebut. Jika tidak, maka dilanjutkan dengan pertanyaan di *level* selanjutnya hingga jelas ciri-ciriya dan jenis obyek dapat ditentukan [SAN-07].



Gambar 2.2 Contoh Penggunaan Metode *Decision Tree*

Sumber : [SAN-07]

Walaupun banyak variasi model *decision tree* dengan tingkat kemampuan dan syarat yang berbeda. Secara umum beberapa ciri kasus cocok untuk diterapkan *decision tree* [SAN-07]:

1. Data dapat dinyatakan sebagai pasangan atribut dan nilainya. Misalnya atribut satu data adalah *temp* dan nilainya adalah dingin. Biasanya untuk satu data nilai dari satu atribut tidak terlalu banyak jenisnya.
2. Label/*output* data biasanya bernilai diskrit. *Output* ini bisa bernilai ya atau tidak, sakit atau tidak sakit. Dalam beberapa studi kasus terdapat juga outputnya tidak hanya memiliki dua kelas, tetapi penerapan *decision tree* lebih banyak untuk kasus *binary*.
3. Data mempunyai *missing value*. Misalnya untuk beberapa data, nilai dari suatu atributnya tidak diketahui. Dalam keadaan seperti ini *decision tree* masih mampu memberi solusi yang terbaik.

Algoritma *decision tree* banyak digunakan dalam proses *data mining* karena memiliki beberapa kelebihan, yaitu [HOF-04]:

1. Tidak memerlukan biaya yang mahal saat membangun algoritma ini.
2. Mudah untuk diinterpretasikan.
3. Mudah mengintegrasikan dengan sistem basis data.
4. Memiliki nilai ketelitian yang baik.
5. Dapat menemukan hubungan tak terduga dari suatu data.

6. Dapat menggunakan data pasti/mutlak atau data kontinu.
7. Mengakomodasi data yang hilang.

Secara singkat bahwa *decision tree* merupakan salah satu metode klasifikasi pada *text mining*. Klasifikasi adalah proses menemukan kumpulan pola atau fungsi-fungsi yang mendeskripsikan dan memisahkan kelas data satu dengan lainnya, untuk dapat digunakan memprediksi data yang belum memiliki kelas data tertentu [HAN-06].

2.7.1 Algoritma C4.5

Algoritma C4.5 adalah algoritma yang digunakan untuk menghasilkan pohon keputusan yang dikembangkan oleh Ross Quinlan. Algoritma C4.5 merupakan pengembangan dari algoritma ID3. Pohon keputusan yang dihasilkan oleh C4.5 dapat digunakan untuk klasifikasi. Sejumlah perubahan untuk memperbaiki algoritma ID3 antara lain [XIO-09]:

1. Penanganan data *training* dari atribut dengan *missing value*.
2. Penanganan atribut diskrit dan atribut *continuous*.

Secara umum algoritma C4.5 untuk membangun pohon keputusan adalah sebagai berikut [KAN-03]:

1. Pilih atribut sebagai *root.*
2. Buat cabang untuk masing-masing nilai.
3. Bagi kasus dalam cabang.
4. Ulangi proses untuk masing-masing cabang sampai semua kasus pada cabang memiliki kelas yang sama.

Untuk memilih atribut sebagai *root*, didasarkan pada nilai *information gain* tertinggi dari atribut-atribut yang ada. Formula untuk *information gain* adalah [KAN-03]:

$$I(s_1, s_2, \dots, s_m) = -\sum_{i=1}^{m} p_i \log_2 (p_i) \quad (2\text{-}4)$$

Keterangan :

S = himpunan kasus

s_1 = jumlah sampel

p_i = proporsi kelas

Untuk mendapatkan informasi nilai subset dari atribut A tersebut maka digunakan formula dibawah ini :

$$E(A) = \sum_{j=1}^{y} \frac{s_{1j}+\cdots+s_{mj}}{s} I(s_{1j}, \ldots, s_{mj}) \quad (2\text{-}5)$$

Keterangan :

$\frac{s_{1j}+\cdots+s_{mj}}{s}$ = jumlah subset *j* yang dibagi dengan jumlah sampel S

Untuk mendapatkan nilai *gain* selanjutnya digunakan formula di bawah ini :

$$Gain\ (A) = I\ (s_1, s_2, \ldots, s_m) - E(A) \quad (2\text{-}6)$$

Keterangan :

A = atribut

S = himpunan kasus

s_1 = jumlah sampel

Namun pada kasus klasifikasi teks digunakan perhitungan *information gain* seperti di bawah ini [AGG-12] :

$$I\ (w) = -\sum_{i=1}^{k} P_i \log(P_i) + F(w) \cdot \sum_{i=1}^{k} P_i(w) \log(P_i(w)) + \big(1 - F(w)\big) \cdot \sum_{i=1}^{k} (1 - P_i(w)) \log(1 - P_i(w)) \quad (2\text{-}7)$$

Keterangan :

P_i = probabilitas global dari *class i*

$P_i(w)$ = probabilitas dari *class i* dimana dokumen berisi *term w*

$F(w)$ = fraksi atau jumlah dokumen yang berisi *term* w

2.7.1.1 *Splitting Attribute*

Pada kasus klasifikasi teks ini atribut yang digunakan adalah atribut numeric atau *continuous*, sehingga digunkan metode standart yaitu *binary split.* Tidak seperti atribut nominal, pada atribut numeric atau *continuous* ini setiap atribut memiliki banyak kemungkinan *split-point*. *Split-point* yang dipilih adalah *split-point* terbaik dimana *split-point* ini akan menjadi pembatas nilai-nilai pada atribut *A*. Untuk mencari *split-point* tersebut, nilai-nilai pada atribut harus diurutkan dari yang terkecil sampai yang terbesar, lalu nilai tengah dari pasangan nilai yang berdekatan dianggap sebagai salah satu kemungkinan *split-point*. Oleh karena itu, jika atribut *A* memiliki nilai *v*, maka akan ada *v* - 1 banyaknya

kemungkinan *split-point* yang akan dievaluasi. Sebagai contoh nilai tengah dari nilai a_i dan a_{i+1} adalah :

$$\frac{a_i+a_{i+1}}{2} \quad (2\text{-}8)$$

Jika nilai-nilai dari *A* diurutkan di awal, kemudian menentukan "*best split*" untuk *A* hanya membutuhkan satu kali melewati nilai-nilai itu. Untuk setiap kemungkinan *split-point* untuk *A*, kita mengevaluasi $Info_A(D)$ dengan menggunakan persamaan :

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} x\ Info(D_j) \quad (2\text{-}9)$$

dimana $Info(D_j)$ sesuai dengan persamaan (2-4).

Evaluasi $Info_A(D)$ dimana jumlah partisi adalah dua, yaitu v = 2 (atau j =1,2). Titik atau *point* dengan *information gain* minimum dipilih sebagai *split-point* untuk atribut *A*. Setiap nilai A $\leq$ *split-point* akan terpartisi ke D_1 dan setiap nilai A $>$ *split-point* akan terpartisi ke D_2 [HAN-06].

2.8 Evaluasi

Pada penelitian *text categorization* evaluasi dilakukan ketika proses klasifikasi apakah metode yang digunakan telah mengklasifikasi secara benar. Evaluasi ini biasanya membutuhkan sebuah matrik yang disebut dengan *matrix confusion*. *Matrix confusion* berisi informasi tentang klasifikasi yang aktual dan terprediksi yang dilakukan oleh sistem. Pada penelitian ini metode evaluasi yang digunakan adalah *precision*, *recall*, dan *f-measure* [SEB-02].

Precision adalah jumlah dokumen yang diklasifikasikan dengan benar oleh sistem dibagi dengan jumlah keseluruhan klasifikasi yang dilakukan oleh sistem. *Recall* adalah jumlah dokumen yang terklasifikasi dengan benar oleh sistem dibagi dengan jumlah dokumen yang seharusnya bisa dikenali sistem. *F-measure* merupakan nilai yang mewakili kinerja keseluruhan sistem dan merupakan penggabungan nilai *recall* dan *precision*. Tabel 2.11 menggambarkan *matrix confusion* dan rumus untuk menghitung *precision* dapat dilihat pada persamaan 2-

10, sedangkan rumus untuk menghitung *recall* dan *f-measure* dapat dilihat pada persamaan 2-11 dan 2-12 [DES-09].

Tabel 2.11 Kesesuaian Hasil Kategori

		Hasil klasifikasi dari ahli	
		YES	NO
Hasil klasifikasi dari sistem	YES	*True positive (TP)*	*False Positive (FP)*
	NO	*False Negative (FN)*	*True Negative (TN)*

$$Precision = \frac{TP}{TP+FP} \quad (2\text{-}10)$$

$$Recall = \frac{TP}{TP+FN} \quad (2\text{-}11)$$

$$F - Measure = \frac{2\ x\ recall\ x\ precision}{recall+precision} \quad (2\text{-}12)$$

Dimana :

- ***True positive*** : Contoh positif dikenali dengan benar sebagai positif, misalnya dokumen d merupakan kategori A dan dikenali sebagai kategori A.
- ***False positive*** : Contoh negatif dikenali sebagai positif, misalnya dokumen d bukan kategori A akan tetapi dikenali sebagai kategori A.
- ***True negative*** : Contoh negatif dikenali dengan benar sebagai negatif, misalnya dokumen d bukan kategori A dan dikenali sebagai bukan kategori A.
- ***False negative*** : Contoh positif dikenali sebagai negatif, misalnya dokumen d kategori A tetapi dikenali sebagai bukan kategori A.

Setelah menghitung nilai *precision, recall,* dan *f1-measure* dari setiap kategori, langkah selanjutnya adalah menghitung *Macroveraged F1*. *Macroveraged F1* digunakan untuk mengevaluasi keseluruhan kinerja algoritma pada dataset yang diberikan. *Macroaveraged F1* menghitung nilai *F1* tiap kategori dan kemudian menghitung nilai rata–ratanya [GUO-04]. Definisi dari *macroaveraged* adalah *precision*, *recall* dan *F1-measure* dievaluasi secara lokal pada tiap kategori dan kemudian dievaluasi secara global dengan cara menghitung rata-rata hasil *precision*, *recall* dan *F1-measure* dari kategori–kategori yang

berbeda [SEB-02]. *Macroaveraged precision*, *recall* dan *F1* dapat dirumuskan pada persamaan 2-13 sampai dengan 2-15 :

$$macroaveraged\ precision = \frac{\sum_{i=1}^{m} P(i)}{m} \quad (2\text{-}13)$$

$$macroaveraged\ recall = \frac{\sum_{i=1}^{m} R(i)}{m} \quad (2\text{-}14)$$

$$macroaveraged\ F1 = \frac{\sum_{i=1}^{m} F1(i)}{m} \quad (2\text{-}15)$$

Dimana :

m : Banyaknya kategori/kelas

P(i) : Nilai *precision* untuk kategori ke i

R(i) : Nilai *recall* untuk kategori ke i

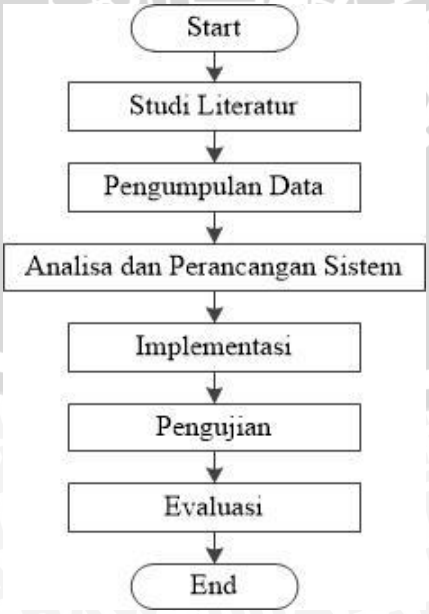
F1(i): Nilai *F1 measure* untuk kategori ke i

BAB III
METODE PENELITIAN DAN PERANCANGAN

Metode penelitian dan perancangan yang digunakan dalam pengklasifikasian berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi adalah sebagai berikut :

1. Melakukan studi literatur mengenai algoritma C4.5 sebagai algoritma klasifikasi untuk menyelesaikan masalah pengklasifikasian berita berbahasa Inggris berbasis ontologi.
2. Mengumpulkan data berita berbahasa Inggris dalam format **.txt* untuk digunakan sebagai data latih dan data uji.
3. Melakukan analisa dan perancangan sistem pengklasifikasian berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi.
4. Mengimplementasikan hasil analisa dan perancangan yang telah dilakukan menjadi sebuah sistem pengklasifikasian teks.
5. Melakukan proses pengujian pada sistem menggunakan data uji.
6. Menganalisa dan mengevaluasi hasil klasifikasi berita yang dilakukan sistem berdasarkan hasil *precision*, *recall* dan *f-measure*.

Alur dari langkah-langkah penelitian yang dilakukan digambarkan pada gambar 3.1.



Gambar 3.1 Alur Penelitian

3.1 Studi Literatur

Studi literatur adalah sebuah tahap untuk mempelajari dan memahami literatur-literatur yang digunakan sebagai acuan penelitian. Literatur-literatur yang digunakan adalah literatur yang terkait dengan pengklasifikasian teks, *text mining*, ontologi, dan algoritma C4.5. Tujuan dari dilakukannya studi literatur ini yaitu untuk menambah pemahaman tentang permasalahan yang diangkat pada penelitian. Literatur yang digunakan diambil dari berbagai sumber, seperti buku, jurnal, *e-book*, internet, serta sumber pustaka lain yang terkait dengan penelitian.

3.2 Data Penelitian

Pada penelitian ini dataset yang digunakan adalah berita berbahasa Inggris yang telah disimpan dalam format **.txt*. Dataset ini diambil dari http://disi.unitn.it/moschitti/corpora.htm. Berita berbahasa Inggris tersebut dikeluarkan oleh Reuters dimana Reuters adalah sebuah kantor berita di Inggris yang merilis *corpus* pada tahun 2000. *Corpus* tersebut berisi berita-berita yang telah dihimpun oleh Reuters untuk kepentingan pengembangan *Natural Language Processing, Information Retrieval*. *Corpus* yang digunakan adalah Reuters-21578. Reuters-21578 memiliki dua jenis yaitu 90 kategori dan 115 kategori, namun yang digunakan pada penelitian ini adalah 90 kategori.

Dataset yang digunakan sebagai data latih pada penelitian ini berjumlah 240 dokumen dimana 60 dokumen untuk masing-masing kelas dengan sub kategori berjumlah empat yaitu *trade, crude, money-fx, interest*. Sedangkan data uji berjumlah 20 dokumen yang bukan termasuk dalam 240 dokumen latih. Berikut ini adalah tabel kategori dan jumlah dokumen yang digunakan pada penelitian.

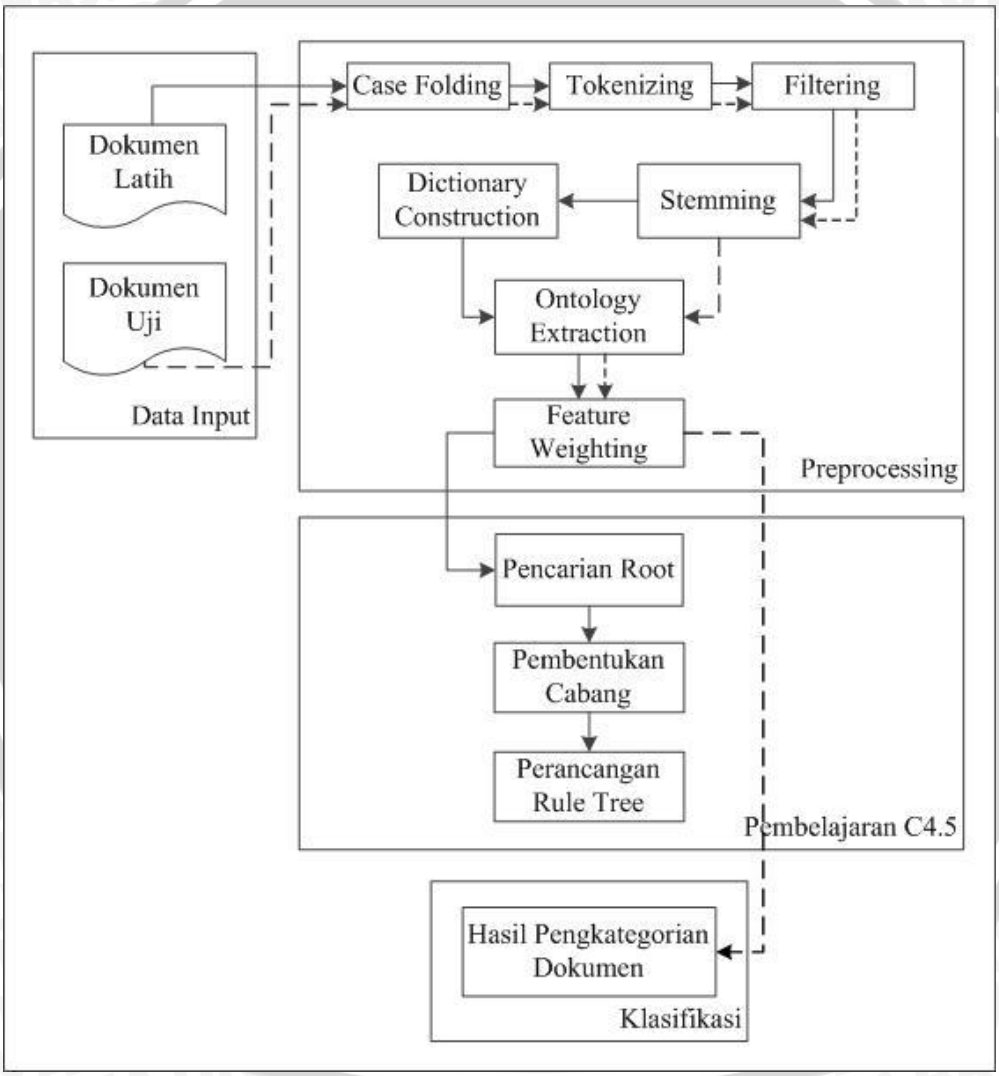
Tabel 3.1 Kategori dan Jumlah Dokumen pada *Corpus*

Kategori	**Jumlah Dokumen**
Trade	60
Crude	60
Money-fx	60
Interest	60
Total	240

3.3 Analisa dan Perancangan Sistem

3.3.1 Deskripsi Umum Sistem

Secara garis besar, sistem pengklasifikasian berita berbahasa Inggris ini memiliki tiga proses, yaitu *preprocessing*, pembelajaran dan klasifikasi yang mengimplementasikan algoritma C4.5. Perancangan pada sistem digambarkan dalam skema perancangan yang ditunjukkan pada gambar 3.2.



Gambar 3.2 Skema Perancangan Sistem

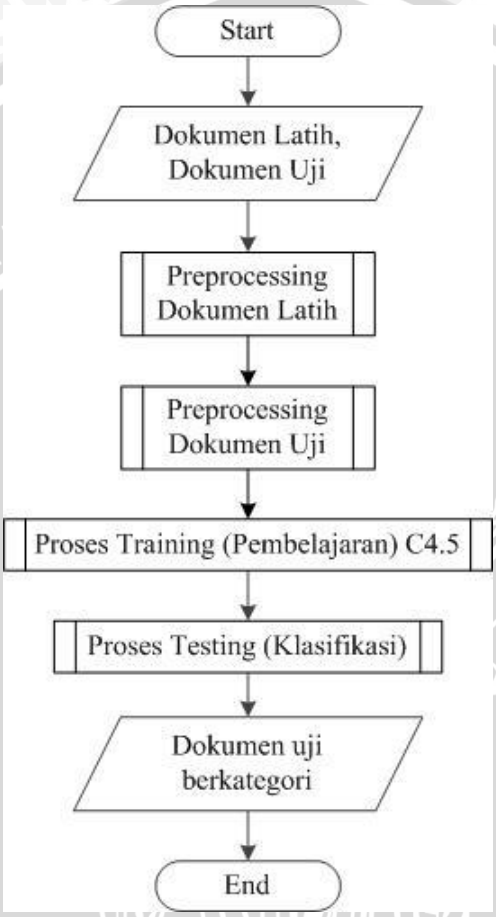
Berikut ini adalah penjelasan mengenai skema perancangan sistem pada gambar 3.2.

1. Data *input* merupakan data masukan yang akan diproses pada tahap *preprocessing*, pembelajaran C4.5, dan klasifikasi. Data *input* terdiri dari dokumen latih dan dokumen uji dimana kedua dokumen tersebut akan mengalami perbedaan proses pada saat *preprocessing* dan pembelajaran C4.5.
2. Tahap selanjutnya setelah terdapat data masukan adalah tahap *preprocessing.* Pada tahap ini data *input* yang berupa dokumen latih diproses melalui *case folding, tokenizing, filtering, stemming, dictionary construction, ontology extraction,* dan *feature weighting*. Sedangkan pada dokumen uji terdapat perbedaan proses karena di antara proses-proses tersebut tidak dilakukan proses *dictionary construction*. Proses *ontology extraction* dilakukan setelah proses *stemming*. *Term-term* dari dokumen uji dicocokkan sinonimnya dengan kata pada database *WordNet* kemudian *term-term* yang saling bersinonim akan digabungkan frekuensi kemunculannya. Proses selanjutnya yaitu dihitung bobotnya pada tahap *feature weighting*.
3. Setelah melalui tahap *preprocessing*, tahap selanjutnya untuk dokumen latih yaitu tahap pembelajaran (*training*) menggunakan algoritma C4.5 yang terdiri dari tahap pencarian *root,* pembentukan cabang, dan perancangan *rule tree.* Pada dokumen uji tidak dilakukan proses ini karena dari hasil perhitungan bobot disertai dengan berpedoman pada *rule* yang telah dibentuk pada proses pembelajaran sudah dapat diketahui kategori dari dokumen uji tersebut.
4. Tahap terakhir adalah tahap klasifikasi. Pada tahap ini menentukan kategori pada dokumen uji dengan berpedoman pada *rule* yang telah terbentuk.

3.3.2 Perancangan Proses

Perancangan proses ini menjelaskan bagaimana proses yang dikerjakan sistem dalam melakukan pengklasifikasian berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi. Dokumen yang digunakan disimpan dalam format **.txt*. Tahap awal dalam pengklasifikasian berita ini adalah tahap *preprocessing* dokumen latih dan dokumen uji. Dari proses *preprocessing* akan didapatkan bobot *term* pada dokumen latih dan dokumen uji. Selanjutnya adalah

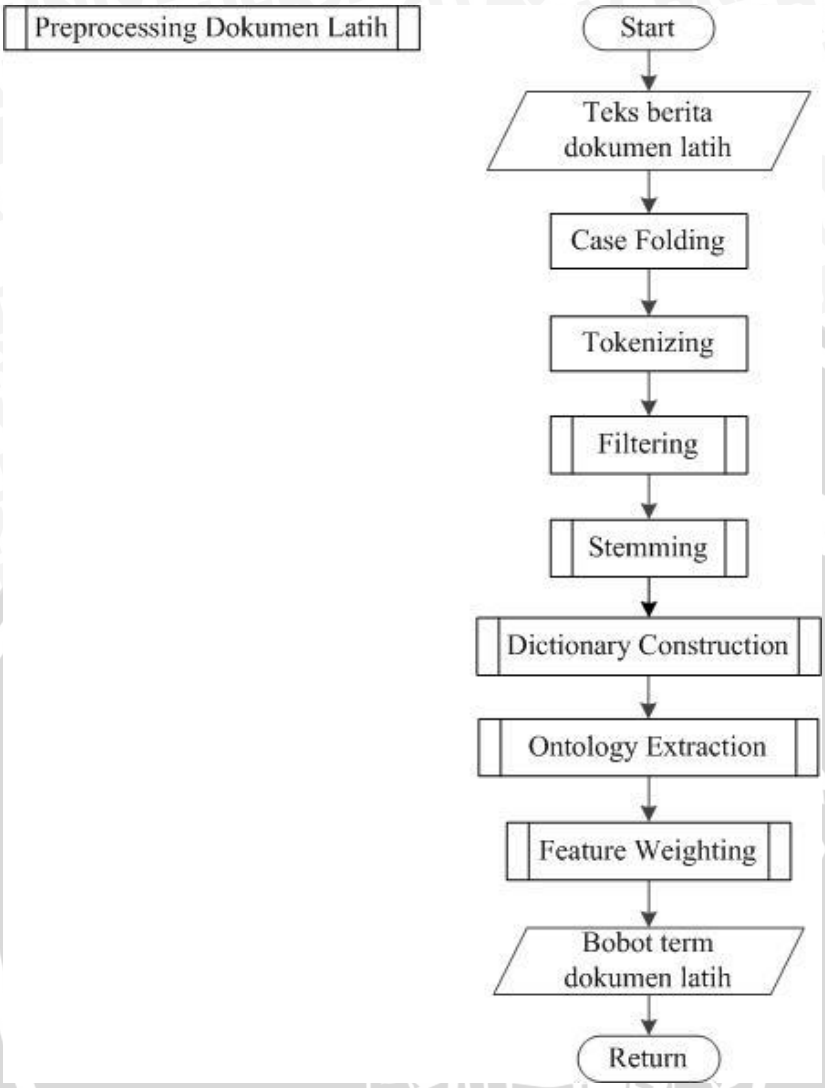
proses pembelajaran C4.5 dimana yang diproses adalah dokumen latih. Dari proses tersebut akan diperoleh pohon keputusan yang disimpan dalam bentuk *rule*. *Rule tree* ini nanti yang akan digunakan sebagai pedoman untuk pengkategorian dokumen uji. Gambar 3.3 menggambarkan *flowchart* sistem secara umum.



Gambar 3.3 *Flowchart* Sistem Secara Umum

3.3.2.1 *Preprocessing* Dokumen Latih

Preprocessing yang dilakukan pada dokumen latih yaitu mengolah dokumen latih menjadi vektor numerik sehingga dapat digunakan untuk proses klasifikasi dokumen. Tahap dari *preprocessing* dokumen latih yaitu *case folding, tokenizing, filtering, stemming, dictionary construction, ontology extraction*, dan *feature weighting*. Tahap *preprocessing* dokumen latih dapat dilihat pada gambar 3.4.



Gambar 3.4 *Flowchart Preprocessing* Dokumen Latih

1. *Case Folding*

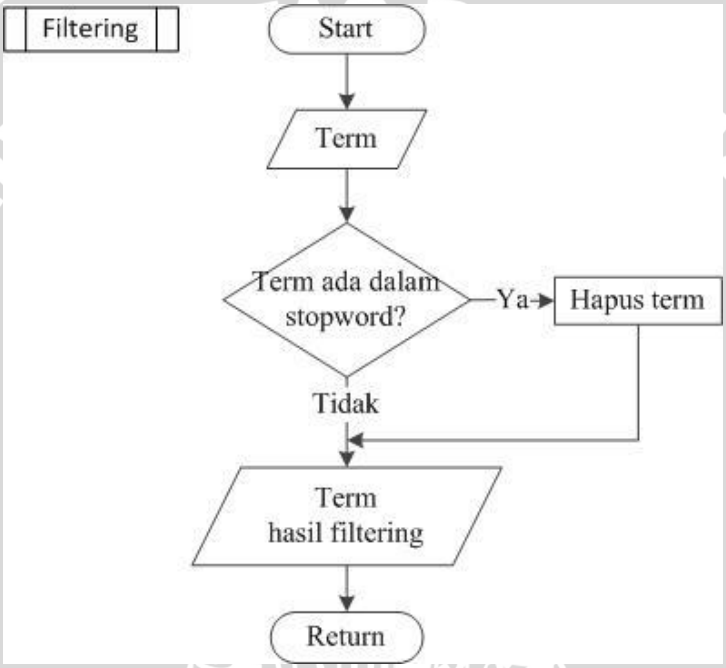
Case folding merupakan proses pengubahan karakter huruf besar yang terkandung di dalam isi berita menjadi karakter huruf kecil. Hanya karakter huruf a-z yang diterima, karakter selain huruf tersebut seperti angka dan tanda baca akan dianggap sebagai spasi atau *delimiter*.

2. *Tokenizing*

Tokenizing merupakan proses setelah dilakukan *case folding*. Pada proses *tokenizing* kalimat berita dipecah menjadi kata tunggal (*token*). Pemecahan kalimat menjadi kata tunggal ini dilakukan dengan menggunakan *split*, yaitu dipisah berdasarkan spasi.

3. *Filtering*

Pada tahap *filtering* setiap kata pada dokumen dicek apakah termasuk kata-kata yang dianggap penting atau tidak. Pengecekan tersebut dilakukan dengan cara membandingkan dengan daftar kata-kata yang dianggap tidak bisa mendeskripsikan isi dokumen yang disimpan pada *stopwords list.* Apabila *term* termasuk ke dalam kata dalam *stopwords list*, maka *term* tersebut akan dibuang. *Flowchart* dari proses *filtering* dapat dilihat pada gambar 3.5.

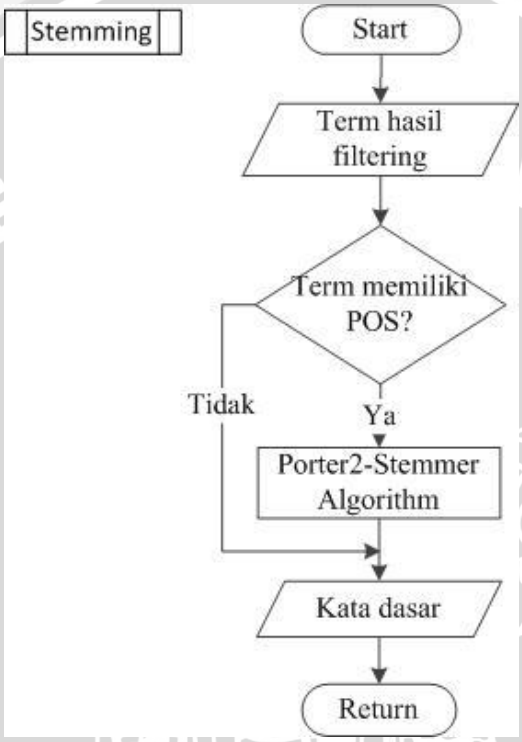


Gambar 3.5 *Flowchart* Proses *Filtering*

4. *Stemming*

Stemming merupakan proses pengubahan kata-kata yang telah melalui proses *filtering* menjadi kata dasar. Metode *stemming* yang digunakan adalah metode *Porter2-Stemmer*. *Porter2-Stemmer* merupakan metode *stemming* untuk bahasa Inggris. *Term-term* yang melalui proses *stemming* ini adalah *term* yang memiliki *Part of Speech* (POS). POS ini diantaranya adalah *noun, adjective, verb,* dan *adverb*. Untuk mengetahui POS dari suatu *term* dapat diketahui dari *WordNet* apakah *term* tersebut memiliki POS atau tidak. Jika *term* tidak memiliki POS, maka *term* tidak akan melalui proses *stemming.* Algoritma ini melakukan reduksi

kata hingga menjadi bentuk kata dasarnya. Algoritma ini memiliki aturan-aturan untuk mereduksi akhiran seperti yang telah digambarkan pada tabel 2.1 sampai dengan tabel 2.9. Algoritma *Porter2-stemmer* terdiri dari 5 langkah pereduksian kata disertai dengan pengecualian seperti yang telah dipaparkan pada bab sebelumnya. *Flowchart* dari proses *stemming* dapat dilihat pada gambar 3.6.

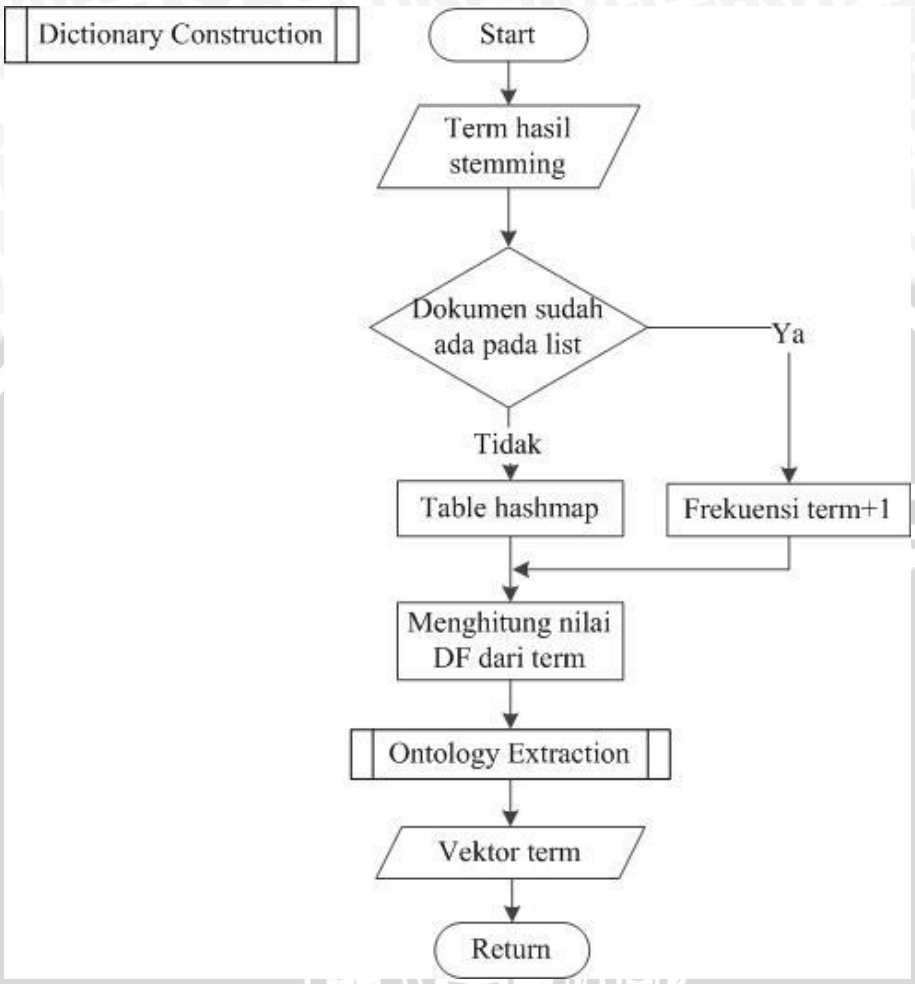


Gambar 3.6 *Flowchart* Proses *Stemming*

5. *Dictionary Construction*

Dictionary construction (pembentukan kamus) merupakan proses penyimpanan *term-term* hasil dari proses *stemming. Term-term* tersebut akan disimpan pada *Inverted Index. Inverted index* menggunakan struktur data *hashmap* yang terdiri *term* yang merupakan *key* (kunci) dan *posting list* yang merupakan *value* (nilai). *Posting list* berisi frekuensi tiap *term* pada setiap dokumen. Apabila *term* belum ada pada *hashmap* maka *term* akan disimpan dan membuat *posting list* baru. Apabila *term* sudah ada tetapi nama dokumen belum ada, maka akan dibuat *posting list* baru untuk dokumen tersebut beserta nilai frekuensinya. *Term-term* yang telah disimpan dalam *inverted index* dihitung

menggunakan metode *Document Frequency* (DF). *Flowchart* dari proses *dictionary construction* dapat dilihat pada gambar 3.7.

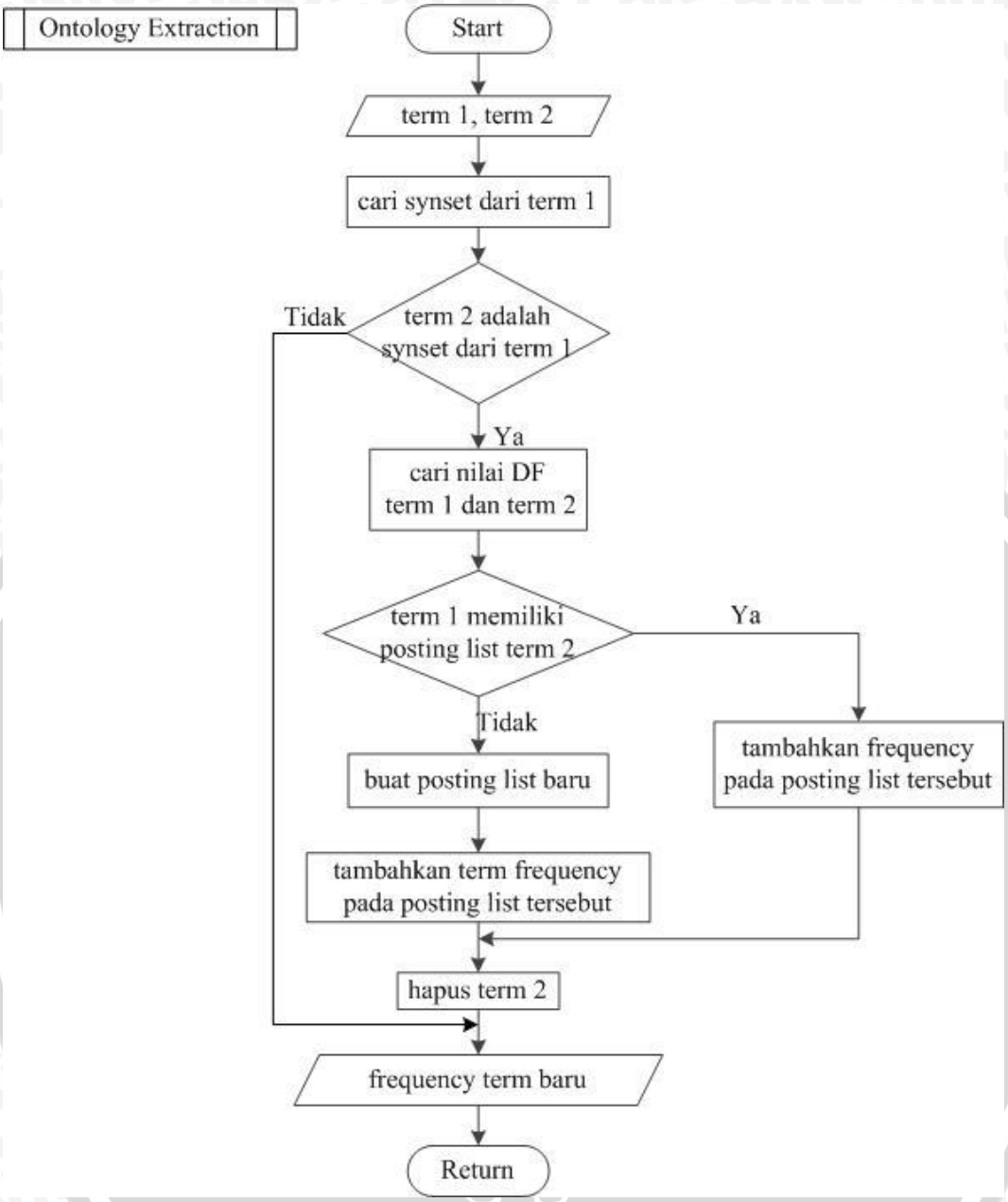


Gambar 3.7 *Flowchart* Proses *Dictionary Construction*

6. *Ontology Extraction*

Ontology extraction merupakan proses untuk mencocokkan sinonim *term-term* pada dokumen dengan kata yang ada pada database *WordNet*. Proses awal adalah dicari *synset* atau sekumpulan kata yang bersinonim dengan term tersebut, apabila termasuk dalam *synset* term tersebut, selanjutnya dicari *Document Frequency* (DF) dari masing-masing *term*, baru kemudian *term* tersebut digabungkan, dan *term* yang sudah digabungkan tersebut akan dihapus dari kamus. Jika DF *term* tersebut mempunyai nilai yang sama, maka *term* yang akan digabungkan adalah salah satu saja. Dalam proses penggabungan *term* terdapat

aturan-aturan, yaitu jika *posting list* dari *term* yang memiliki nilai DF kecil sudah ada pada *posting list term* yang memiliki nilai DF besar, maka frekuensi *term* dari dokumen yang memiliki nilai DF kecil akan ditambahkan ke *posting list* dokumen yang memiliki nilai DF besar. Sedangkan jika *posting list* pada dokumen yang memiliki nilai DF besar belum ada, maka akan dibuat *posting list* baru sesuai dengan *posting list* yang dimiliki oleh *term* yang akan digabungkan. *Flowchart* dari proses *ontology extraction* dapat dilihat pada gambar 3.8.



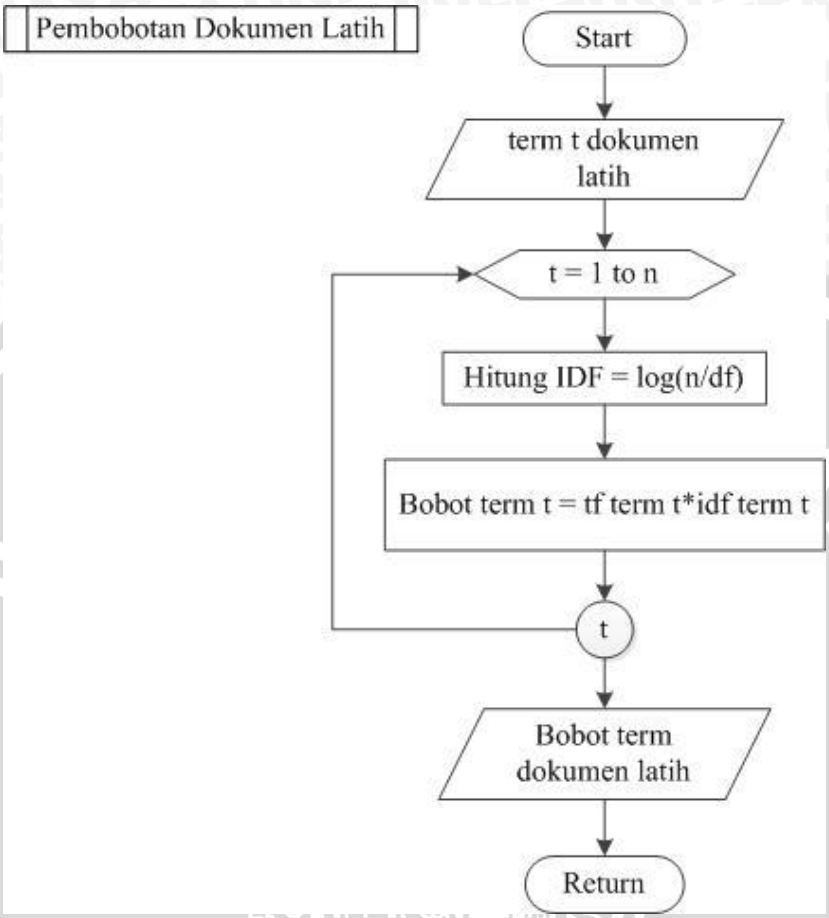
Gambar 3.8 *Flowchart* Proses *Ontology Extraction*

Sumber : [MUR-13]

7. Pembobotan Dokumen Latih

Tahap akhir dari proses *preprocessing* dokumen latih adalah menghitung nilai bobot dari *term* dengan menggunakan metode TF-IDF. TF merupakan jumlah *term* yang muncul pada suatu dokumen. Sedangkan IDF adalah *term* yang jarang muncul pada *corpus.* Untuk mendapatkan nilai TF-IDF dengan cara

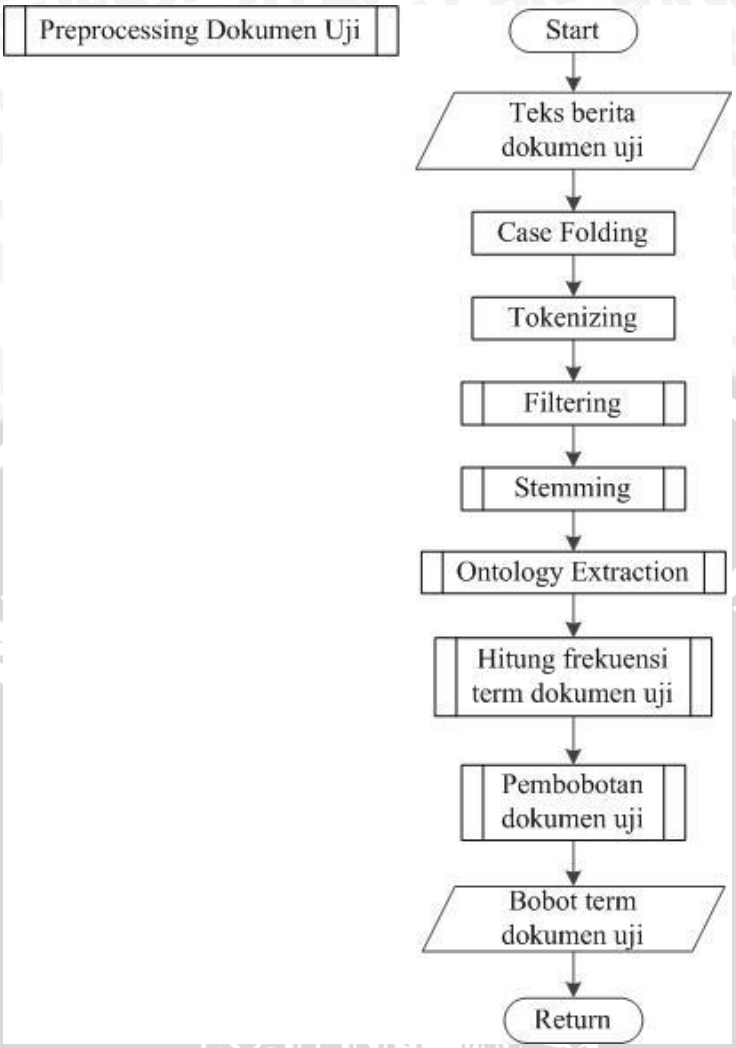
mengalikan nilai TF dan nilai IDF dari *term* tersebut. *Flowchart* dari proses pembobotan dokumen latih dapat dilihat pada gambar 3.9.



Gambar 3.9 *Flowchart* Proses Pembobotan Dokumen Latih

3.3.2.2 *Preprocessing* Dokumen Uji

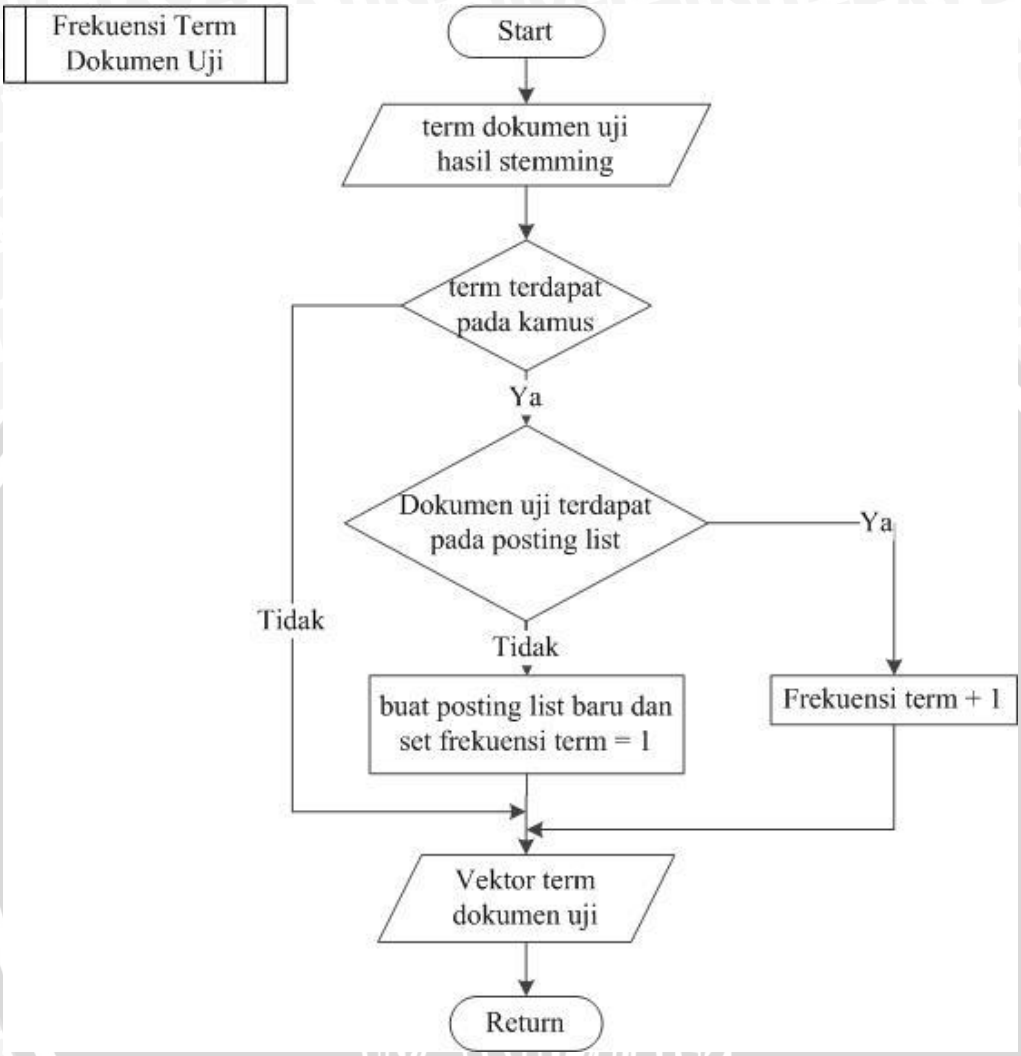
Selain *preprocessing* pada dokumen latih, *preprocessing* juga dilakukan pada dokumen uji namun dengan proses yang berbeda. Tahap preprocessing pada dokumen uji yaitu *case folding, tokenizing, filtering, stemming, ontology extraction*, dan *feature weighting*. *Preprocessing* pada dokumen uji tidak mengalami proses *dictionary construction* karena *term* yang digunakan pada dokumen uji mengacu pada *term* yang ada pada kamus yang telah dibentuk pada saat *preprocessing* dokumen latih. *Flowchart* dari *preprocessing* dokumen uji dapat dilihat pada gambar 3.10.



Gambar 3.10 *Flowchart Preprocessing* Dokumen Uji

Dari gambar 3.10 dapat dijelaskan bahwa tahap *preprocessing* dokumen uji dimulai dari proses yang sama dengan dokumen latih yaitu *case folding, tokenizing, filtering* dan *stemming.* Setelah proses *stemming* adalah proses *ontology extraction* dimana *synset* dari *term* pada dokumen uji dicocokkan dengan daftar *synset* pada dokumen latih. Jika *term* pada dokumen uji tidak terdapat pada kamus dokumen latih tetapi *term* tersebut ada di dalam daftar *synset* maka *term* tersebut dianggap sama dengan *term* pada dokumen uji yang bersinonim dengan *synset* tersebut. Kemudian *term* yang ada pada dokumen uji dicocokkan dengan kamus pada dokumen latih. *Term* yang tidak ada pada kamus tidak akan dilakukan

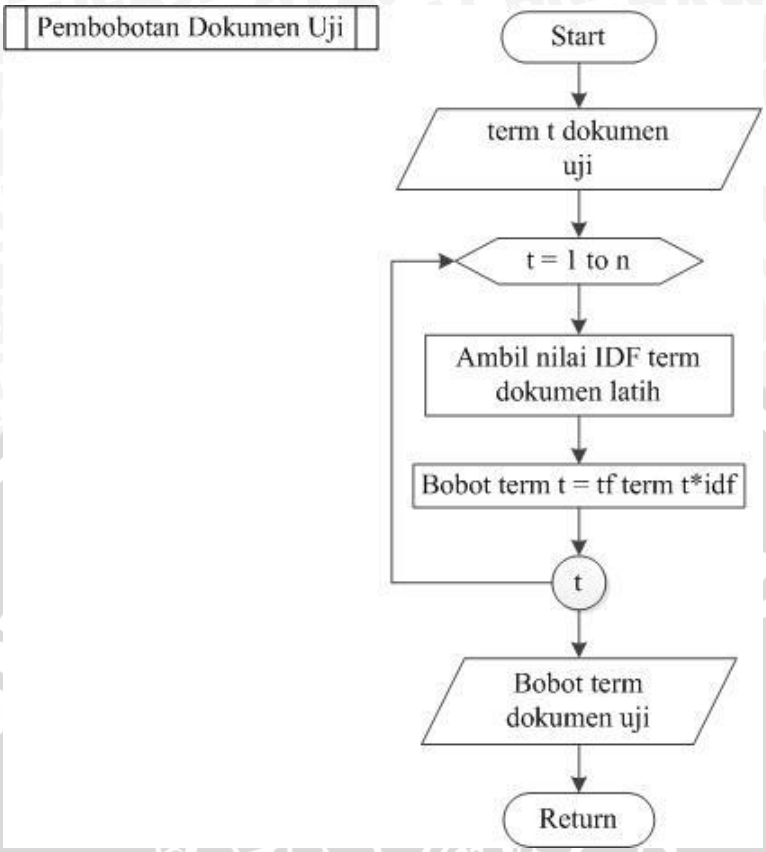
proses lebih lanjut. *Flowchart* dari proses perhitungan frekuensi *term* dokumen uji dapat dilihat pada gambar 3.11.



Gambar 3.11 *Flowchart* Perhitungan Frekuensi *Term* Dokumen Uji

Sumber : [MUR-13]

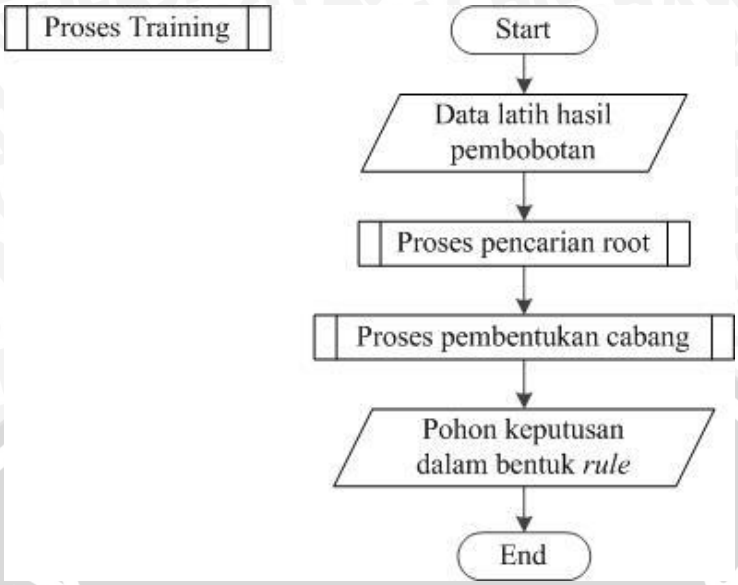
Tahap selanjutnya setelah perhitungan frekuensi *term* adalah tahap *feature weighting* atau perhitungan bobot dokumen uji. Rumus perhitungan bobot ini sama dengan perhitungan bobot pada dokumen latih yaitu mengalikan nilai TF dengan nilai IDF. Nilai TF diambil dari nilai TF dari *term* dokumen uji, sedangkan nilai IDF diambil dari proses perhitungan IDF pada dokumen latih. *Flowchart* dari proses pembobotan dokumen uji dapat dilihat pada gambar 3.12.



Gambar 3.12 *Flowchart* Proses Pembobotan Dokumen Uji

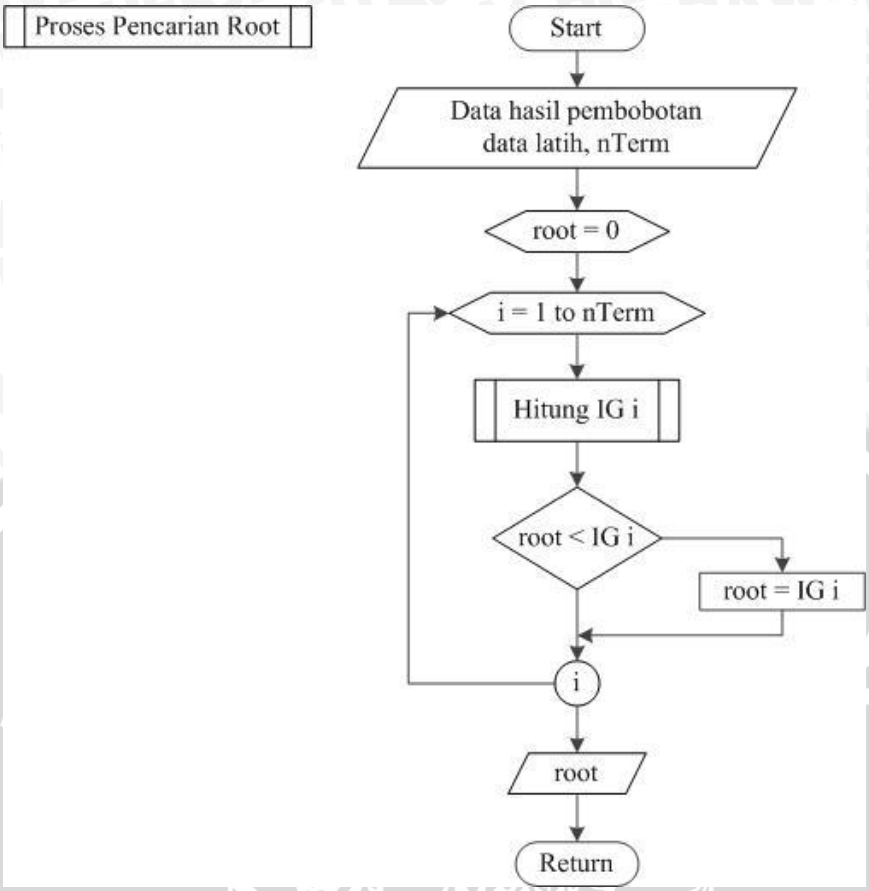
3.3.2.3 Proses *Training* (Pembelajaran) C4.5

Setelah proses pembobotan data latih, proses selanjutnya adalah proses *training* (pembelajaran) menggunakan algoritma C4.5. Data masukan dari proses ini adalah data latih hasil pembobotan. Kemudian dilakukan proses pencarian *root* dan pembentukan cabang. Hasil dari proses *training* berupa pohon keputusan yang disimpan dalam bentuk *rule*. *Flowchart* dari proses *training* (pembelajaran) C4.5 dapat dilihat pada gambar 3.13.



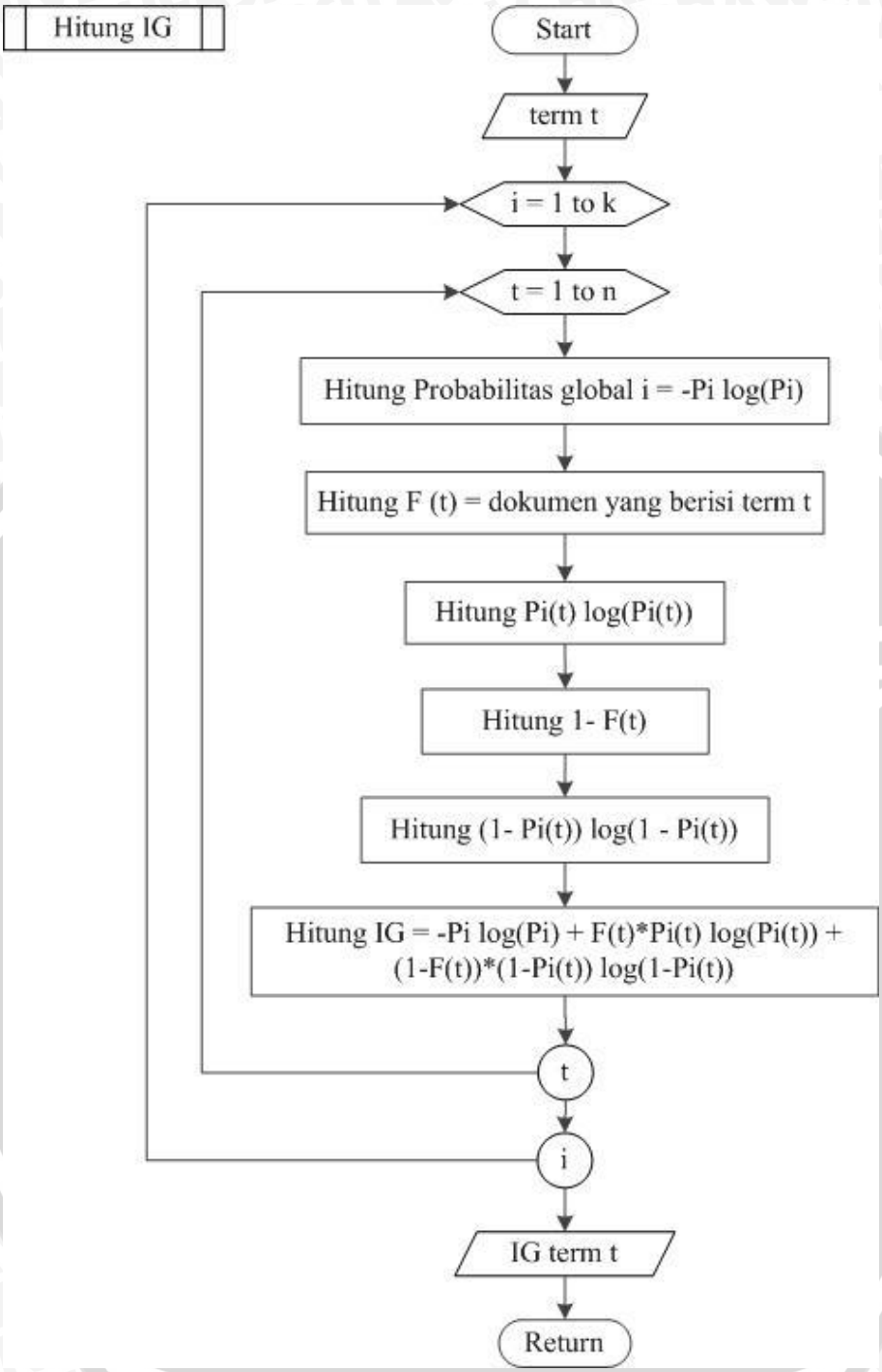
Gambar 3.13 *Flowchart* Proses *Training* (Pembelajaran) C4.5

Proses awal dari *training* (pembelajaran) menggunakan algoritma C4.5 adalah pencarian *root*. Data masukan pada proses ini berasal dari hasil pembobotan. Kemudian dihitung *information gain* (IG) setiap *term* dimana *term* yang memiliki nilai *information gain* (IG) tertinggi ditetapkan sebagai *root*. Hasil dari proses pencarian *root* adalah satu kata sebagai *root*. *Flowchart* dari proses pencarian *root* dapat dilihat pada gambar 3.14.



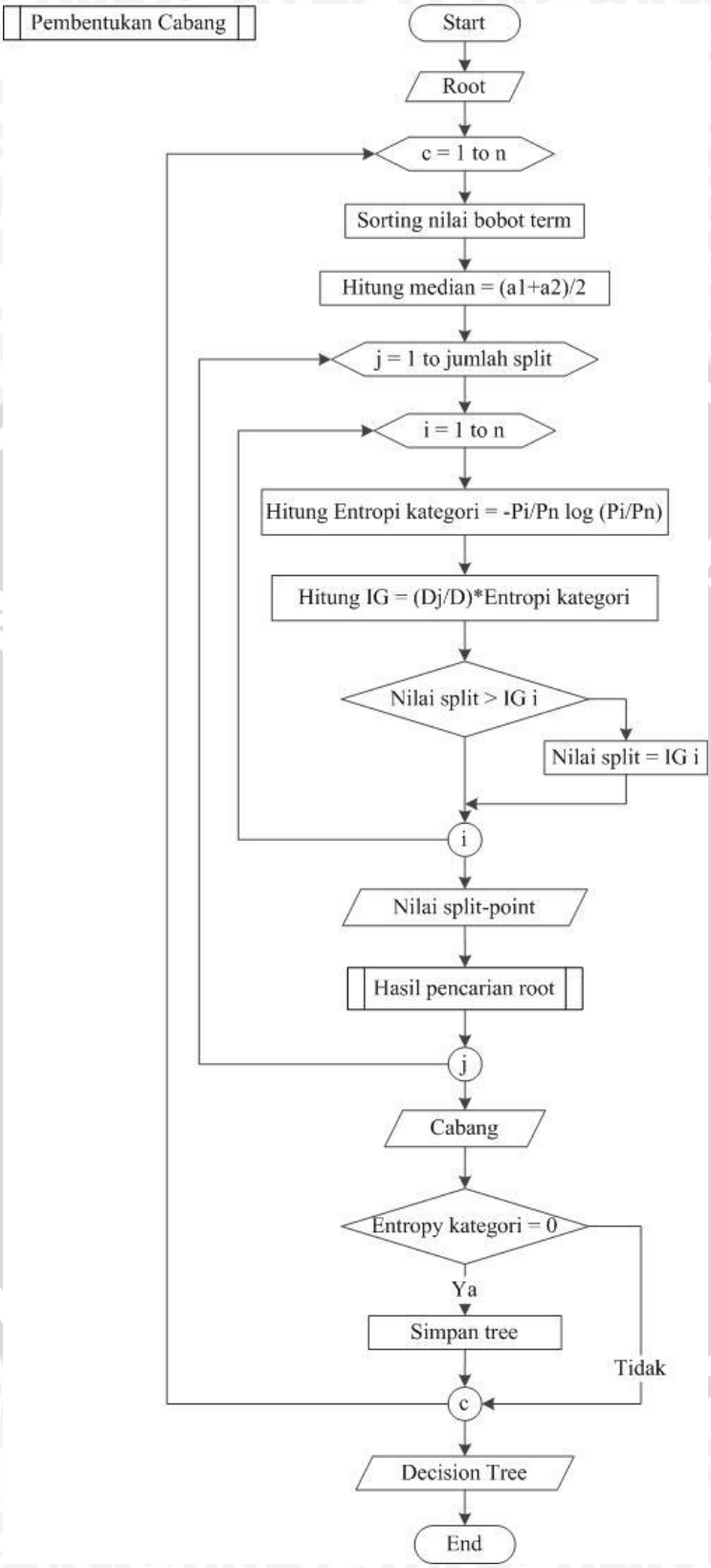
Gambar 3.14 *Flowchart* Proses Pencarian *Root*

Di dalam proses pencarian *root* pada gambar 3.14 terdapat proses perhitungan *information gain* (IG). *Input* dari proses ini adalah *term* t. Kemudian menghitung probabilitas global dari *class* i, menghitung jumlah dokumen yang berisi *term* t, dan menghitung probabilitas dari *class* i dimana dokumen berisi *term* t. Hasil dari proses tersebut diperoleh nilai *information gain* (IG) dari setiap *term*. *Flowchart* dari proses perhitungan *information gain* (IG) dapat dilihat pada gambar 3.15.



Gambar 3.15 *Flowchart* Proses Perhitungan *Information Gain*

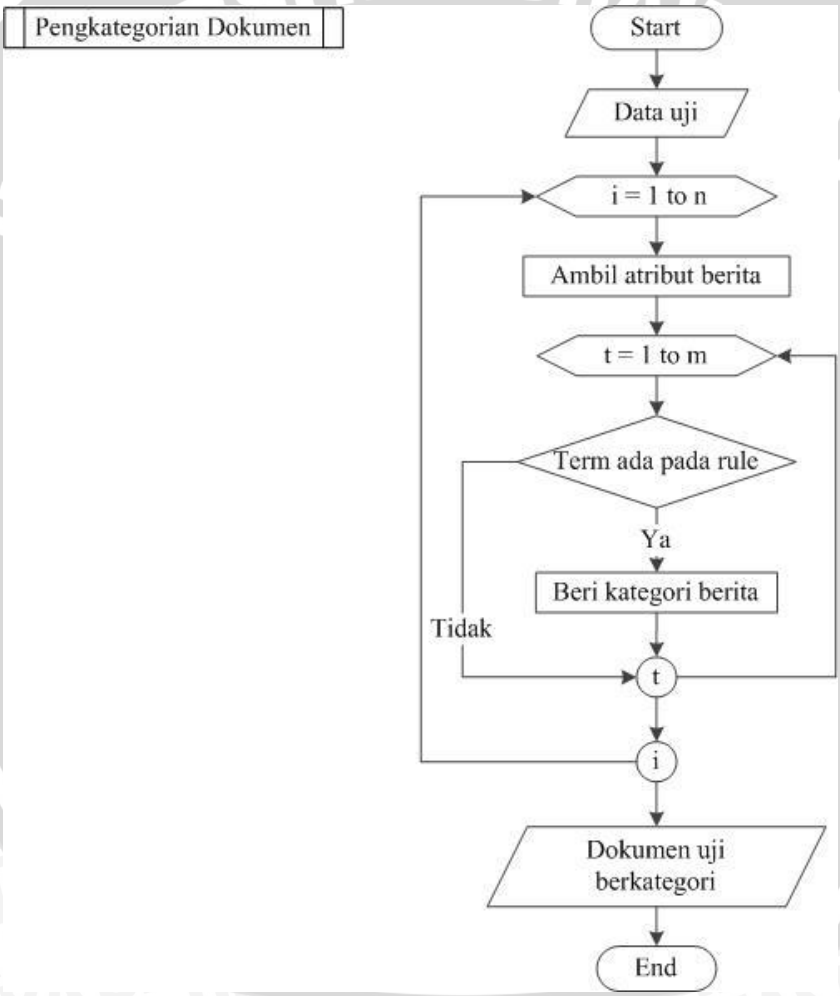
Proses selanjutnya setelah pencarian *root* adalah proses pembentukan cabang. Dari hasil penentuan *root*, selanjutnya dilakukan proses perhitungan untuk menentukan *split-point* atau nilai yang akan digunakan untuk men-*split*. Teknik *splitting* menggunakan *binary split* yaitu ≤ dan >. Data masukan pada proses ini adalah *term* yang telah terpilih menjadi *root*. Kemudian nilai pembobotan pada *term* yang dipilih akan di-*sorting* dan dihitung nilai *median* pada setiap *point*. Selanjutnya dari *splitting* nilai *median* tersebut dihitung nilai *information gain*, dimana *split-point* dengan nilai *information gain* terkecil yang akan dipilih. Setelah diperoleh nilai *split*, kemudian dilakukan proses untuk menentukan cabang dengan cara menghitung *information gain* setiap *term* seperti pada proses pencarian *root*. Proses tersebut akan berjalan rekursif untuk menemukan *node* selanjutnya dan akan berhenti jika dokumen sudah berada pada kategori yang sama atau *entropy* kategori bernilai nol dan pada kondisi tersebut *leaf* akan terbentuk. Hasil dari proses ini adalah pembentukan *decision tree*. *Flowchart* dari proses pembentukan cabang dapat dilihat pada gambar 3.16.



Gambar 3.16 *Flowchart* Proses Pembentukan Cabang

3.3.2.4 Pengkategorian Dokumen Uji

Proses akhir adalah proses pengkategorian dokumen uji. Proses *training* (pembelajaran) yang telah dilakukan sebelumnya berguna untuk membangun *rule* berdasarkan *decision tree* yang terbentuk, sedangkan proses pengkategorian berguna untuk membandingkan nilai data uji berdasarkan *rule* yang telah terbentuk. Hasil dari proses pengkategorian ini berupa dokumen uji yang sudah memiliki kategori. *Flowchart* dari proses pengkategorian dokumen dapat dilihat pada gambar 3.17.



Gambar 3.17 *Flowchart* Proses Pengkategorian Dokumen Uji

Pada *flowchart* di atas, *input* berupa data uji. Pada teks berita uji ke i akan diambil setiap atributnya, yaitu *term* t. Apabila *term* ada pada *rule*, maka beri kategori berita sesuai dengan *rule* yang ada. Proses itu akan berlanjut sampai teks

berita uji ke n, atau sampai semua teks berita uji mendapatkan kategori yang sesuai dengan *rule*.

3.3.3 Perhitungan Manual

Perhitungan manual menjelaskan semua proses mulai dari tahap *preprocessing*, *ontology extraction*, sampai dengan tahap pengklasifikasian berita berbahasa Inggris menggunakan algoritma C4.5. Kategori yang terdapat dalam kasus ini sebanyak 4 kategori, yaitu *trade*, *crude*, *money-fx*, dan *interest*. Dalam perhitungan manual ini diambil masing-masing 2 dokumen dari 4 kategori tersebut, dimana setiap dokumen latih hanya diambil beberapa kalimat inti untuk menyederhanakan proses perhitungan manual. Tabel 3.2 menunjukkan isi dari dokumen latih yang digunakan pada perhitungan manual.

Tabel 3.2 Dokumen latih pada Perhitungan Manual

Indeks Dok	Kategori	Isi Dokumen
D1	*Trade*	In 1985, had a trade surplus of 4 million dollars, trade surplus of 6 billion dollars, and trade surplus of 8 billion dollars. The U.S. had a trade import of 7 billion dollars, trade import of 10 billion dollars, and trade import of 9 billion dollars. We must trade import in 1986, trade import in 1987, and trade import in 1988. He told about trade in 1989, trade in 1990, trade in 1991, trade in 1992, trade in 1993, and trade in 1994.
D2	*Trade*	Trade trade trade trade trade trade trade trade trade trade import import import import import import oil oil oil.
D3	*Crude*	Trade trade trade trade trade trade trade import million million million million million dollar dollar dollar dollar dollar dollar dollar dollar crude oil crude oil oil oil oil oil oil oil price price price price price.
D4	*Crude*	Import oil oil oil oil oil oil oil oil oil oil oil oil price price price price price.
D5	*Money-fx*	Trade trade trade surplus import billion dollar dollar dollar

		dollar dollar oil price.
D6	*Money-fx*	Trade trade trade trade import million dollar dollar dollar dollar dollar dollar dollar dollar price.
D7	*Interest*	Import import import import import price price.
D8	*Interest*	Trade surplus surplus billion billion dollar oil.
D9	Tidak Diketahui	Murphy said it increased its crude oil posted prices by 50 cts a barrel and new posting is 19 dollars a barrel.

Dari dokumen-dokumen di atas yang akan dijadikan contoh perhitungan manual untuk dokumen latih adalah dokumen nomor 1 (D1). Tahap pertama yaitu tahap *preprocessing* dokumen latih yang terdiri dari tahap *case folding, tokenizing, filtering,* dan *stemming*. *Case folding* merupakan proses pengubahan karakter huruf besar menjadi huruf kecil dan menghilangkan karakter selain 'a' sampai 'z'. Proses di bawah ini menunjukkan proses dari *case folding*.

Dokumen sebelum proses *case folding* :

> In 1985, had a trade surplus of 4 million dollars, trade surplus of 6 billion dollars, and trade surplus of 8 billion dollars. The U.S. had a trade import of 7 billion dollars, trade import of 10 billion dollars, and trade import of 9 billion dollars. We must trade import in 1986, trade import in 1987, and trade import in 1988. He told about trade in 1989, trade in 1990, trade in 1991, trade in 1992, trade in 1993, and trade in 1994.

Dokumen setelah proses *case folding* akan menjadi :

> in had a trade surplus of million dollars trade surplus of billion dollars and trade surplus of billion dollars the us had a trade import of billion dollars trade import of billion dollars and trade import of billion dollars we must trade import in trade import in and trade import in he told about trade in trade in trade in trade in trade in and trade in

Tahapan setelah dilakukan *case folding* adalah *tokenizing* dimana kalimat berita diubah menjadi kalimat tunggal (*token*). Tabel 3.4 menunjukkan hasil dari proses *tokenizing*.

Tabel 3.3 Hasil *Tokenizing*

in	dollars	import	of	trade	trade
had	and	of	billion	import	in
a	trade	billion	dollars	in	trade
trade	surplus	dollars	we	he	in
surplus	of	trade	must	told	and
of	billion	import	trade	about	trade
million	dollars	of	import	trade	in
dollars	the	billion	in	in	
trade	us	dollars	trade	trade	
surplus	had	and	import	in	
of	a	trade	in	trade	
billion	trade	import	and	in	

Setelah diperoleh hasil *tokenizing*, tahap selanjutnya adalah menyaring kata-kata penting dan membuang kata-kata yang dianggap tidak penting. Tahap ini disebut dengan tahap *filtering*. Pada tahap ini akan dilakukan pengecekan kata-kata dari hasil *tokenizing* dengan daftar kata pada *stopword*. Apabila terdapat kesamaan kata hasil *tokenizing* dengan kata yang terdapat pada *stopword* maka kata tersebut akan dihapus karena dianggap tidak penting. Tabel 3.4 menunjukkan hasil dari proses *filtering*.

Tabel 3.4 Hasil *Filtering*

trade	million
surplus	billion
import	dollars

Setelah diperoleh kata-kata penting dari proses *filtering*, tahap selanjutnya adalah mengubah kata-kata tersebut menjadi bentuk kata dasarnya. Tahap ini disebut dengan tahap *stemming* dimana algoritma stemming yang digunakan adalah algoritma *Porter2 Stemmer*. Tabel 3.5 menunjukkan hasil dari proses *stemming*.

Tabel 3.5 Hasil *Stemming*

trade → trade	million → million
surplus → surplus	billion → billion
import → import	dollars → dollar

Setelah proses *stemming*, tahap selanjutnya adalah menghitung *term frequency* (TF) dari setiap *term* pada setiap dokumen. Tabel 3.6 menunjukkan hasil dari perhitungan *term frequency* dokumen.

Tabel 3.6 Hasil Perhitungan *Term Frequency*

trade = 15	import = 6
surplus = 3	billion = 5
million = 1	dollar = 6

Setelah dilakukan perhitungan *term frequency* dari semua dokumen latih, tahap selanjutnya adalah menentukan *document frequency*. *Document frequency* adalah jumlah dokumen dimana *term* tersebut muncul. Tabel 3.7 menunjukkan pembentukan *inverted index* serta *document frequency* dari dokumen latih.

Tabel 3.7 *Inverted Index* dan *Document Frequency Term-Term* Dokumen Latih

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**DF**
trade	15	10	7	0	3	4	0	1	6
surplus	3	0	0	0	1	0	0	2	3
import	6	6	1	1	1	1	5	0	7
million	1	0	5	0	0	1	0	0	3
billion	5	0	0	0	1	0	0	2	3
dollar	6	0	8	0	5	8	0	1	5
oil	0	3	8	12	1	0	0	1	5
crude	0	0	2	0	0	0	0	0	2
price	0	0	5	4	1	1	2	0	5

Tahap selanjutnya adalah menggabungkan *term* yang saling bersinonim pada *inverted index* dengan menggunakan database *WordNet.* Tahap ini disebut dengan tahap *ontology extraction*. *Term* yang saling bersinonim akan dihapus salah satunya lalu frekuensi dari kedua *term* tersebut digabungkan. Setelah dilakukan pencocokan untuk setiap *term*, terdapat *term* yang saling bersinonim, yaitu *term crude* dengan *term oil* dan *term million* dengan *term billion*.

Tabel 3.8 *Inverted Index Term* Sebelum *Ontology Extraction*

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**DF**
oil	0	3	8	12	1	0	0	1	5
crude	0	0	2	0	0	0	0	0	1
million	1	0	5	0	0	1	0	0	3
billion	5	0	0	0	1	0	0	2	3

Tabel 3.9 *Inverted Index Term* Sesudah *Ontology Extraction*

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**DF**
oil	0	3	10	12	1	0	0	1	5
billion	6	0	5	0	1	1	0	2	5

Setelah proses *ontology extraction*, maka *inverted index* yang baru telah terbentuk dengan pengurangan *term* pada proses *ontology extraction*. Langkah selanjutnya adalah menentukan *Inverse Document Frequency* (IDF) dari setiap *term* pada *inverted index*. Rumus menghitung IDF dapat dilihat pada persamaan 2-2. Contoh perhitungan IDF salah satu *term* adalah sebagai berikut:

Menghitung *term trade*. Rumus dari IDF adalah sebagai berikut:

$$idf_i = log\left(\frac{N}{df_i}\right)$$

Dengan :

N = jumlah keseluruhan dokumen yaitu 12 dokumen.

df_i = Nilai *document frequency* dari *term* tersebut.

Term trade memiliki *document frequency* 6, maka nilai IDF dari *term trade* sesuai rumus adalah sebagai berikut:

$$log\left(\frac{8}{6}\right) = 0.125$$

Nilai IDF dari setiap *term* pada dokumen latihdapat dilihat pada tabel 3.10.

Tabel 3.10 Nilai IDF *Term-Term* pada Dokumen Latih

Term	**DF**	**IDF**
trade	6	0.125
surplus	3	0.426
import	7	0.058
billion	5	0.204
dollar	5	0.204
oil	5	0.204
price	5	0.204

Setelah diperoleh nilai TF dan IDF untuk setiap *term*, langkah selanjutnya adalah menghitung bobot term dengan menggunakan TFIDF, dimana nilai *term frequency* dari setiap *term* pada *Inverted Index* masing-masing dikalikan dengan nilai IDF *term* tersebut. Contoh menghitung nilai dari TFIDF dari *term surplus* adalah sebagai berikut:

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**IDF**
surplus	3	0	0	0	1	0	0	2	0.426

Maka nilai TFIDF dari *term surplus* adalah sebagai berikut:

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**
surplus	3*0.426=1.278	0	0	0	1*0.426=0.426	0	0	2*0.426=0.852

Perhitungan diatas juga dilakukan untuk semua *term* dokumen latih. Hasil perhitungan TFIDF tersebut dapat dilihat pada tabel 3.11.

Tabel 3.11 Hasil Perhitungan TFIDF *Term* Dokumen Latih

Term	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**
trade	1.874	1.249	0.875	0	0.375	0.500	0	0.125
surplus	1.278	0	0	0	0.426	0	0	0.852
import	0.348	0.348	0.058	0.058	0.058	0.058	0.290	0
billion	1.224	0	1.02	0	0.204	0.204	0	0.408
dollar	1.225	0	1.633	0	1.021	1.633	0	0.204
oil	0	0.612	2.041	2.449	0.204	0	0	0.204
price	0	0	1.02	0.816	0.204	0.204	0.408	0
Kategori	*Trade*	*Trade*	*Crude*	*Crude*	*Money-fx*	*Money-fx*	*Interest*	*Interest*

Klasifikasi Dokumen Uji

Preprocessing dokumen latih telah selesai dilakukan lalu langkah selanjutnya adalah mencari frekuensi *term* dari dokumen uji. Sebelumnya setiap dokumen uji juga dilakukan proses *preprocessing* seperti pada dokumen latih tetapi terdapat perbedaan pada salah satu prosesnya. Pada *preprocessing* dokumen uji tidak terdapat proses *dictionary construction* atau pembentukan *inverted index* karena *term* yang diambil adalah *term* yang sudah ada di dalam *inverted index* dokumen latih. Misalkan terdapat dokumen uji sebagai berikut:

Murphy said it increased its crude oil posted prices by 50 cts a barrel and new posting is 19 dollars a barrel.

Awalnya dokumen uji tersebut dilakukan proses *case folding*, *tokenizing*, *filtering*, *stemming*, dan menghitung frekuensi tiap *term* seperti pada dokumen latih. Hasil *preprocessing* sampai dengan proses perhitungan frekuensi *term* pada dokumen uji tersebut dapat dilihat pada tabel 3.12

Tabel 3.12 Hasil *Preprocessing* dan Perhitungan Frekuensi *Term* Dokumen Uji

murphy=1	oil=1	barrel=2
increas =1	post=2	new=1
crude=1	price=1	dollar=1

Setelah mengetahui *term frequency* dari dokumen uji, langkah selanjutnya adalah melakukan proses *ontology extraction*. Proses *ontology extraction* pada dokumen uji sedikit berbeda dengan proses yang dilakukan pada dokumen latih. Setiap *term* pada dokumen uji dicari *synset* nya kemudian dicek apakah *term* tersebut ada pada kamus dokumen latih. Jika *term* tersebut tidak muncul pada kamus dokumen latih maka dicek kembali apakah *term* pada dokumen uji tersebut ada pada daftar *synset* dokumen latih. Jika muncul, maka *term* tersebut dianggap sama dengan *term* pada dokumen latih yang bersinonim dengan *synset*. Contoh *term* ***oil*** dan ***crude.*** *Term oil* terdapat pada kamus dokumen latih maka frekuensi *term* nya adalah 1 sedangkan pada *term crude* tidak terdapat pada kamus dokumen latih tetapi terdapat pada daftar *synset* maka *term crude* pada dokumen uji tersebut dianggap sama dengan *term oil* maka frekuensi term *oil* bertambah satu sementara frekuensi term *crude* adalah 0. Frekuensi term dari dokumen uji setelah proses *ontology extraction* dapat dilihat pada tabel 3.13

Tabel 3.13 *Term Frequency* dari Dokumen Uji Setelah *Ontology Extraction*

murphy =1	post=2	new=1
oil=2	price=1	dollar=1
increas=1	barrel=2	

Langkah berikutnya adalah mencari frekuensi *term* dari dokumen uji. *Term* yang diambil dari dokumen uji adalah *term* yang sudah ada pada *inverted index*. Nilai IDF dari *term-term* tersebut diambil dari nilai IDF *term* yang telah disimpan di *inverted index*. Daftar *term*, nilai TF serta nilai bobot TFIDF dari dokumen uji dapat dilihat pada tabel 3.14

Tabel 3.14 Daftar *term* nilai TF serta nilai bobot TFIDF dari dokumen uji

Dokumen Uji		
Term	**TF**	**TFIDF**
trade		0
surplus		0
import		0
billion		0
dollar	1	0.204
oil	2	0.402
price	1	0.204

Penerapan Algoritma C4.5

Setelah mengetahui hasil pembobotan dokumen latih, proses selanjutnya adalah pembentukan pohon keputusan. Langkah-langkah dalam pembentukan pohon keputusan adalah sebagai berikut:

1. Pembetukan *root* utama

   Proses pembentukan *root* dilakukan dengan cara mencari *information gain* setiap *term* terhadap kategori dengan menggunakan persamaan 2-10. Contoh perhitungan *information gain* salah satu *term* adalah sebagai berikut:

   Menghitung *term dollar*. Rumus dari *information gain* adalah sebagai berikut:

$$I(w) = -\sum_{i=1}^{k} P_i \log(P_i) + F(w) \cdot \sum_{i=1}^{k} P_i(w) \log(P_i(w)) + \left(1 - F(w)\right) \cdot \sum_{i=1}^{k} (1 - P_i(w)) \log(1 - P_i(w))$$

   Dengan :

   P_i = probabilitas global dari *class i*

   $P_i(w)$ = probabilitas dari *class i* dimana dokumen berisi *term w*

   $F(w)$ = fraksi atau jumlah dokumen yang berisi *term* w

Dari tabel 3.11 dapat dilihat bahwa jumlah dokumen sebanyak 8 dengan 2 dokumen berkategori *trade*, 2 dokumen berkategori *crude*, 2 dokumen berkategori *money-fx*, dan 2 dokumen berkategori *interest*. Jumlah dokumen yang berisi *term dollar* hanya sebanyak 5 dokumen dari 8 dokumen dimana muncul pada 1 dokumen *trade*, 1 dokumen *crude*, 2 dokumen *money-fx*, dan 1 dokumen *interest*, maka *information gain* dari *term dollar* sesuai rumus adalah sebagai berikut:

$$\text{I}(dollar) = \left(\left(-\frac{2}{8}\log\left(\frac{2}{8}\right)\right)x\,4\right) +$$

$$5x\left(\left(\frac{1.225}{2}\log\left(\frac{1.225}{2}\right)\right)+\left(\frac{1.633}{2}\log\left(\frac{1.633}{12}\right)\right)+\left(\frac{1.021}{2}\log\left(\frac{1.021}{2}\right)\right)+\left(\frac{1.633}{2}\log\left(\frac{1.633}{2}\right)\right)+\left(\frac{0.204}{2}\log\left(\frac{0.204}{2}\right)\right)\right)+$$

$$(1-5)x\left(\left(\left(1-\frac{1.225}{2}\right)\log\left(1-\frac{1.225}{2}\right)\right)+\left(\left(1-\frac{1.633}{2}\right)\log\left(1-\frac{1.633}{2}\right)\right)+\left(\left(1-\frac{1.021}{2}\right)\log\left(1-\frac{1.021}{2}\right)\right)+\left(\left(1-\frac{1.633}{2}\right)\log\left(1-\frac{1.633}{2}\right)\right)+\left(\left(1-\frac{0.204}{2}\right)\log\left(1-\frac{0.204}{2}\right)\right)\right)$$

$$= 0.474$$

Nilai *information gain* untuk menentukan *root* awal dapat dilihat pada tabel 3.15.

Tabel 3.15 Hasil Perhitungan *Information Gain* untuk Menentukan *Root*

Term	**IG**
trade	-0.592
surplus	0.086
import	-1.878
billion	-0.615
dollar	0.474
oil	0.164
price	-0.855

Dari tabel 3.15 *term* yang memiliki *information gain* tertinggi adalah *term dollar*, maka atribut *dollar* akan menjadi *root* awal. Setelah itu, melakukan proses *split* dengan menggunakan metode *binary splits*. Setiap atribut mempunyai banyak kemungkinan *split-point*. Setiap kemungkinan *split-point* tersebut akan dievaluasi dengan cara mempartisi *learning sample* dengan setiap kemungkinan *split-point* tersebut dengan aturan dimana setiap nilai A $\leq$ *split-point* akan terpartisi ke D_1 dan setiap nilai A $>$ *split-point* akan terpartisi ke D_2. Cara menentukan *split-point* yaitu dengan menghitung nilai tengah *median* dari setiap partisi yang bersebelahan dengan kategori yang berbeda. Langkah-langkah menentukan *split-point* dari *term dollar* adalah sebagai berikut:

- *Sorting* bobot *term dollar* dari nilai terkecil hingga nilai terbesar

Bobot *term dollar* sebelum proses *sorting:*

dollar	1.225	0	1.633	0	1.021	1.633	0	0.204
Kategori	*Trade*	*Trade*	*Crude*	*Crude*	*Money-fx*	*Money-fx*	*Interest*	*Interest*

Hasil *sorting* bobot *term dollar* :

dollar	0	0	0	0.204	1.021	1.225	1.633	1.633
Kategori	*Trade*	*Crude*	*Interest*	*Interest*	*Money-fx*	*Trade*	*Crude*	*Money-fx*

- Menghitung *median* dari setiap *point* dengan menggunakan rumus pada persamaan (2-11) :

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
dollar	0	0	0	0.204	1.021	1.225	1.633	1.633

Hasil perhitungan *median* dari *term dollar* untuk setiap *point*:

$(t_1+ t_2)/2$	$(t_2+ t_3)/2$	$(t_3+ t_4)/2$	$(t_4+ t_5)/2$	$(t_5+ t_6)/2$	$(t_6+ t_7)/2$	$(t_7+ t_8)/2$
0	0	0.102	0.613	1.123	1.429	1.633

- Menghitung *information gain* setiap *split-point* dengan menggunakan persamaan (2-12) dimana *split-point* terbaik adalah yang mempunyai nilai *information gain* terkecil.

Tabel 3.16 Hasil *Split-point* dari Nilai *Median* untuk Atribut *Dollar*

Median	***Split-point***	
0	≤0	>0
0	≤0	>0
0.102	≤0.102	>0.102
0.613	≤0.613	>0.613
1.123	≤1.123	>1.123
1.429	≤1.429	>1.429
1.633	≤1.633	>1.633

Contoh perhitungan *information gain* dari *split-point* 1.429. Dari tabel 3.11 dapat dilihat bahwa pada *term dollar* yang memiliki nilai bobot ≤1.429 terdapat pada D1, D2, D4, D5, D7, D8. Tabel 3.17 menunjukkan daftar dokumen yang memiliki nilai bobot ≤1.429 untuk atribut *dollar*.

Tabel 3.17 Daftar Dokumen dengan Nilai Bobot ≤1.429 untuk Atribut *Dollar*

Term	**D1**	**D2**	**D4**	**D5**	**D7**	**D8**
trade	1.874	1.249	0	0.375	0	0.125
surplus	1.278	0	0	0.426	0	0.852
import	0.348	0.348	0.058	0.058	0.290	0
billion	1.224	0	0	0.204	0	0.408
dollar	1.225	0	0	1.021	0	0.204
oil	0	0.612	2.449	0.204	0	0.204
price	0	0	0.816	0.204	0.408	0
Kategori	*Trade*	*Trade*	*Crude*	*Money-fx*	*Interest*	*Interest*

Dari tabel 3.17 dapat dilihat bahwa jumlah dokumen sebanyak 6 dengan 2 dokumen berkategori *trade*, 1 dokumen berkategori *crude*, 1 dokumen berkategori

money-fx, dan 2 dokumen berkategori *interest*, maka perhitungan *information gain* adalah sebagai berikut:

$Entropy\ (\leq 1.429)$

$$E(2,1,1,2) = \left(\left(-\frac{2}{6}\log\left(\frac{2}{6}\right)\right)x2\right) + \left(\left(-\frac{1}{6}\ log\left(\frac{1}{6}\right)\right)x2\right) = 0.577$$

Selanjutnya untuk *split-point* >1.429, dari tabel 3.11 dapat dilihat bahwa pada *term dollar* yang memiliki nilai bobot >1.429 terdapat pada D3 dan D6. Tabel 3.18 menunjukkan daftar dokumen yang memiliki nilai bobot >1.429 untuk atribut *dollar*.

Tabel 3.18 Daftar Dokumen dengan Nilai Bobot >1.429 untuk Atribut *Dollar*

Term	**D3**	**D6**
trade	0.875	0.500
surplus	0	0
import	0.058	0.058
billion	1.02	0.204
dollar	1.633	1.633
oil	2.041	0
price	1.021	0.204
Kategori	*Crude*	*Money-fx*

Dari tabel 3.18 dapat dilihat bahwa jumlah dokumen sebanyak 2 dengan 1 dokumen berkategori *crude* dan 1 dokumen berkategori *money-fx*, maka perhitungan *information gain* adalah sebagai berikut:

$Entropy\ (> 1.429)$

$$E(1,1) = \left(-\tfrac{1}{2}\log\left(\tfrac{1}{2}\right)\right)x2 = 0.301$$

Sehingga hasil perhitungan *information gain* untuk nilai *split-point* 1.429 adalah sebagai berikut:

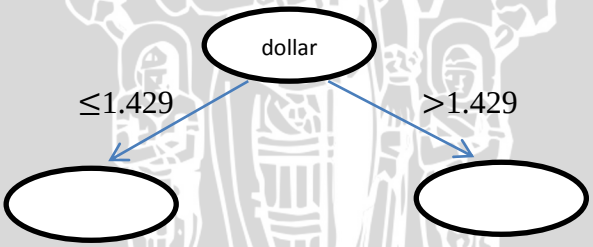
$$I(dollar\ @1.429) = \tfrac{6}{8}.E(dollar \leq 1.429) + \tfrac{2}{8}.E(dollar > 1.429)$$
$$= \tfrac{6}{8}(0.577) + \tfrac{2}{8}(0.301)$$
$$= 0.433 + 0.075$$
$$= 0.508$$

Nilai *information gain* dari setiap *split-point* untuk atribut *dollar* dapat dilihat pada tabel 3.19.

Tabel 3.19 Hasil Perhitungan *Information Gain* untuk setiap *Split-point* pada Atribut *Dollar*

Median	***Split-point***		**IG**
0	≤0	>0	0.600
0	≤0	>0	0.600
0.613	≤0.613	>0.613	0.602
1.123	≤1.123	>1.123	0.600
1.429	≤1.429	>1.429	0.508
1.633	≤1.633	>1.633	0.602

Dari tabel 3.19 dapat dilihat bahwa nilai *information gain* terkecil terdapat pada *split-point* 1.429, maka *split-point* tersebut yang akan digunakan sebagai nilai *split* untuk atribut *dollar*. Gambar 3.18 menunjukkan bahwa atribut *dollar* menjadi *root* awal dengan nilai *binary split* ≤1.429 dan >1.429.



Gambar 3.18 Atribut *Dollar* sebagai *Root* Awal

2. Pembentukan cabang ≤1.429 dari atribut *dollar*

   Proses awal adalah pembentukan cabang ≤1.429 dari atribut *dollar* dengan melakukan perhitungan *information gain* pada dokumen yang mempunyai nilai ≤1.429 pada atribut *dollar*. Hasil pembentukan cabang ≤1.429 dari atribut *dollar* dapat dilihat pada tabel 3.20.

Tabel 3.20 Hasil Pembentukan Cabang ≤1.429 dari Atribut *Dollar*

Term	**D1**	**D2**	**D4**	**D5**	**D7**	**D8**
trade	1.874	1.249	0	0.375	0	0.125
surplus	1.278	0	0	0.426	0	0.852
import	0.348	0.348	0.058	0.058	0.290	0
billion	1.224	0	0	0.204	0	0.408
oil	0	0.612	2.449	0.204	0	0.204
price	0	0	0.816	0.204	0.408	0
Kategori	*Trade*	*Trade*	*Crude*	*Money-fx*	*Interest*	*Interest*

Kemudian dilakukan perhitungan *information gain* dengan menggunakan cara yang sama seperti pada proses pembentukan *root*, sehingga diperoleh hasil perhitungan *information gain* untuk semua *term* yang dapat dilihat pada tabel 3.21.

Tabel 3.21 Hasil Perhitungan *Information Gain* untuk Cabang ≤1.429 dari Atribut *Dollar*

Term	**IG**
trade	0.189
surplus	0.130
import	-1.092
billion	-0.344
oil	3.485
price	0.102

Dari tabel 3.21 *term* yang memiliki *information gain* tertinggi adalah *term oil*, maka atribut *oil* sebagai *node* berikutnya. Setelah itu menentukan *split-point* dengan menggunakan cara yang sama seperti sebelumnya, sehingga diperoleh hasil *split-point* seperti pada tabel 3.22.

Tabel 3.22 Hasil *Split-point* dari Nilai *Median* untuk Atribut *Oil*

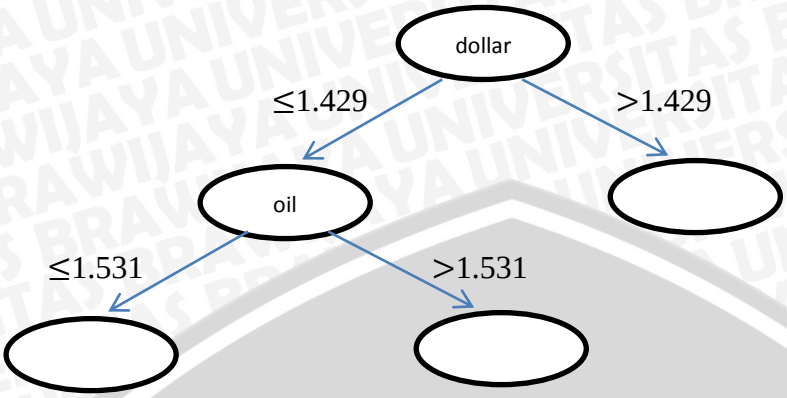
Median	***Split-point***	
0	≤0	>0
0.102	≤0.102	>0.102
0.204	≤0.204	>0.204
0.408	≤0.408	>0. 408
1.531	≤0.612	>0.612

Selanjutnya seperti pada proses sebelumnya yaitu menghitung *information gain* untuk setiap *split-point*. Nilai *information gain* dari setiap *split-point* untuk atribut *oil* dapat dilihat pada tabel 3.23.

Tabel 3.23 Hasil Perhitungan *Information Gain* untuk setiap *Split-point* pada Atribut *Oil*

Median	***Split-point***		**IG**
0	≤0	>0	0.502
0.204	≤0.204	>0.204	0.401
0.408	≤0.408	>0. 408	0.401
1.531	≤1.531	>1.531	0.382

Dari tabel 3.23 dapat dilihat bahwa nilai *information gain* terkecil terdapat pada *split-point* 1.531, maka *split-point* tersebut yang akan digunakan sebagai nilai *split* untuk atribut *oil*. Gambar 3.19 menunjukkan bahwa atribut *oil* sebagai *node* berikutnya dengan nilai binary split ≤1.531 dan >1.531.



Gambar 3.19 *Tree* Hasil Cabang ≤1.429 dari Atribut *Dollar*

3. Pembentukan cabang ≤1.531 dari atribut *oil*

   Proses awal adalah pembentukan cabang ≤1.531 dari atribut *oil* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.20 yang mempunyai nilai ≤1.531 pada atribut *oil*. Hasil pembentukan cabang ≤1.531 dari atribut *oil* dapat dilihat pada tabel 3.24.

Tabel 3.24 Hasil Pembentukan Cabang ≤1.531 dari Atribut *Oil*

Term	**D1**	**D2**	**D5**	**D7**	**D8**
trade	1.874	1.249	0.375	0	0.125
surplus	1.278	0	0.426	0	0.852
import	0.348	0.348	0.058	0.290	0
billion	1.224	0	0.204	0	0.408
price	0	0	0.204	0.408	0
Kategori	*Trade*	*Trade*	*Money-fx*	*Interest*	*Interest*

Kemudian dilakukan perhitungan *information gain* dengan menggunakan cara yang sama seperti pada proses sebelumnya, sehingga diperoleh hasil perhitungan *information gain* untuk semua *term* yang dapat dilihat pada tabel 3.25.

Tabel 3.25 Hasil Perhitungan *Information Gain* untuk Cabang ≤1.531 dari Atribut *Oil*

Term	**IG**
trade	0.069
surplus	0.011
import	-0.713
billion	-0.463
price	0.053

Dari tabel 3.25 *term* yang memiliki *information gain* tertinggi adalah *term trade*, maka atribut *trade* sebagai *node* berikutnya. Setelah itu menentukan *split-point* dengan menggunakan cara yang sama seperti sebelumnya, sehingga diperoleh hasil *split-point* seperti pada tabel 3.26.

Tabel 3.26 Hasil *Split-point* dari Nilai *Median* untuk Atribut *Trade*

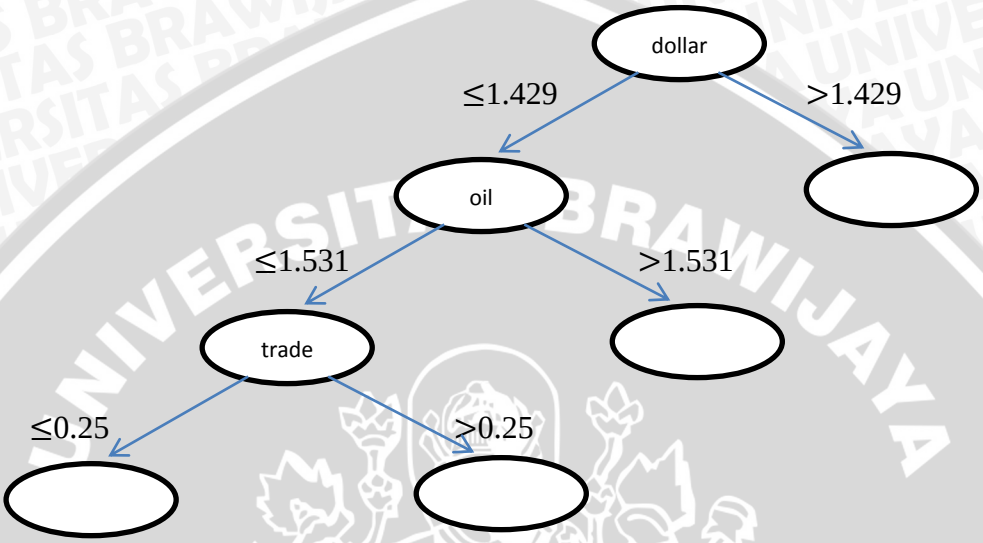
Median	***Split-point***	
0.0625	≤0.0625	>0.0625
0.25	≤0.25	>0.25
0.812	≤0.812	>0.812
1.562	≤1.562	>1.562

Selanjutnya seperti pada proses sebelumnya yaitu menghitung *information gain* untuk setiap *split-point*. Nilai *information gain* dari setiap *split-point* untuk atribut *trade* dapat dilihat pada tabel 3.27.

Tabel 3.27 Hasil Perhitungan *Information Gain* untuk setiap *Split-point* pada Atribut *Trade*

Median	***Split-point***		**IG**
0.25	≤0.25	>0.25	0.166
0.812	≤0.812	>0.812	0.166

Dari tabel 3.27 dapat dilihat bahwa nilai *information gain* bernilai sama pada kedua *split-point*, maka *split-point* dipilih salah satu secara acak yaitu *split-point* 0.25, sehingga *split-point* tersebut yang akan digunakan sebagai nilai *split* untuk atribut *trade*. Gambar 3.20 menunjukkan bahwa atribut *trade* sebagai *node* berikutnya dengan nilai *binary split* ≤0.25 dan >0.25.



Gambar 3.20 *Tree* Hasil Cabang ≤1.531 dari Atribut *Oil*

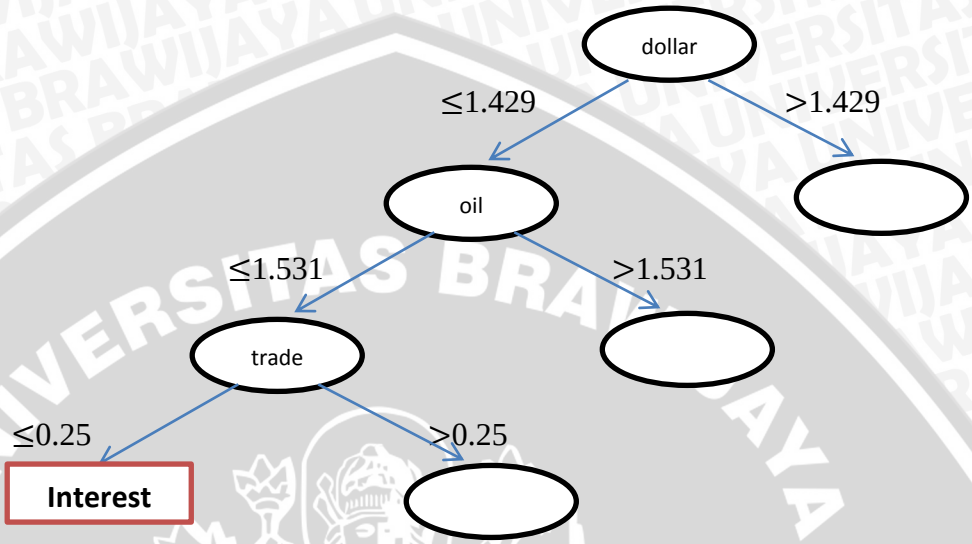
4. Pembentukan cabang ≤0.25 dari atribut *trade*

   Proses awal adalah pembentukan cabang ≤0.25 dari atribut *trade* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.24 yang mempunyai nilai ≤0.25 pada atribut *trade*. Hasil pembentukan cabang ≤0.25 dari atribut *trade* dapat dilihat pada tabel 3.28.

Tabel 3.28 Hasil Pembentukan Cabang ≤0.25 dari Atribut *Trade*

Term	**D7**	**D8**
surplus	0	0.852
import	0.290	0
billion	0	0.408
price	0.408	0
Kategori	*Interest*	*Interest*

Dari tabel 3.28 dapat dilihat bahwa cabang ≤0.25 dari atribut *trade* sudah berada pada kelas atau kategori yang sama, yaitu kategori *interest*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.21 *Tree* Hasil Cabang ≤0.25 dari Atribut *Trade*

5. Pembentukan cabang >0.25 dari atribut *trade*

   Proses awal adalah pembentukan cabang >0.151 dari atribut *trade* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.24 yang mempunyai nilai >0.25 pada atribut *trade*. Hasil pembentukan cabang >0.25 dari atribut *trade* dapat dilihat pada tabel 3.29.

Tabel 3.29 Hasil Pembentukan Cabang >0.25 dari Atribut *Trade*

Term	**D1**	**D2**	**D5**
surplus	1.278	0	0.426
import	0.348	0.348	0.058
billion	1.224	0	0.204
price	0	0	0.204
Kategori	*Trade*	*Trade*	*Money-fx*

Kemudian dilakukan perhitungan *information gain* dengan menggunakan cara yang sama seperti pada proses sebelumnya, sehingga diperoleh hasil perhitungan *information gain* untuk semua *term* yang dapat dilihat pada tabel 3.30.

Tabel 3.30 Hasil Perhitungan *Information Gain* Untuk Cabang >0.25 dari Atribut *Trade*

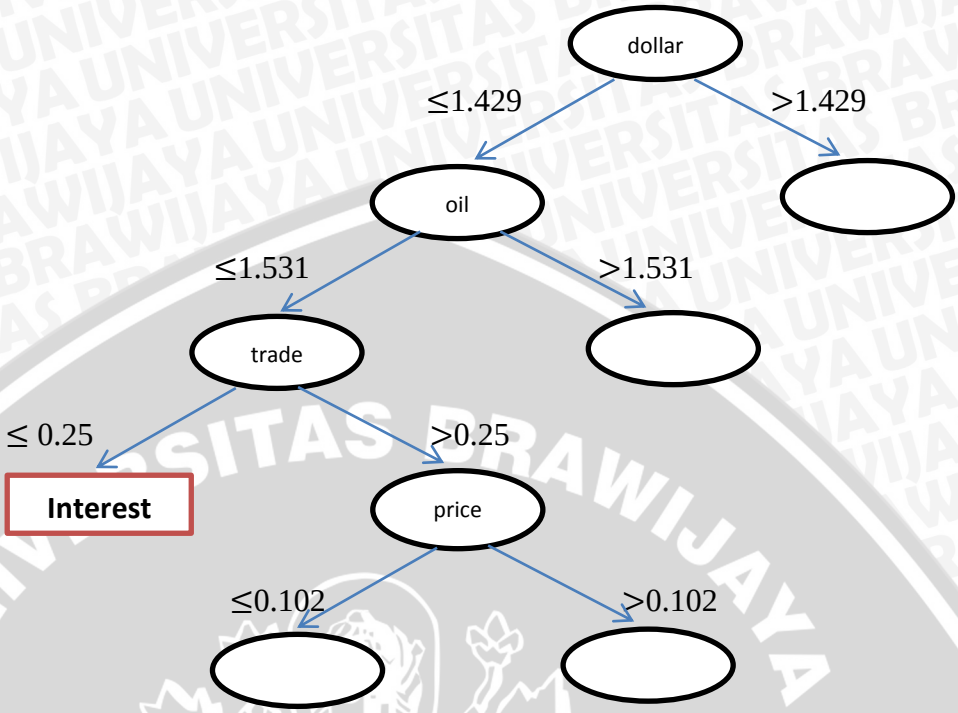
Term	**IG**
surplus	0.010
import	-0.408
billion	-0.028
price	0.136

Dari tabel 3.30 *term* yang memiliki *information gain* tertinggi adalah *term price*, maka atribut *price* sebagai *node* berikutnya. Setelah itu menentukan *split-point* dengan menggunakan cara yang sama seperti sebelumnya, sehingga diperoleh hasil *split-point* seperti pada tabel 3.31.

Tabel 3.31 Hasil *Split-point* dari Nilai *Median* untuk Atribut *Price*

Median	***Split-point***	
0.102	≤0.102	>0.102

Dari tabel 3.31 dapat dilihat bahwa hanya terdapat satu *split-point* yaitu 0.102, sehingga *split-point* tersebut dapat digunakan tanpa harus menghitung nilai *information gain* terlebih dahulu. Gambar 3.22 menunjukkan bahwa atribut *price* sebagai *node* berikutnya dengan nilai *binary split* ≤0.102 dan >0.102.



Gambar 3.22 *Tree* Hasil Cabang >0.25 dari Atribut *Trade*

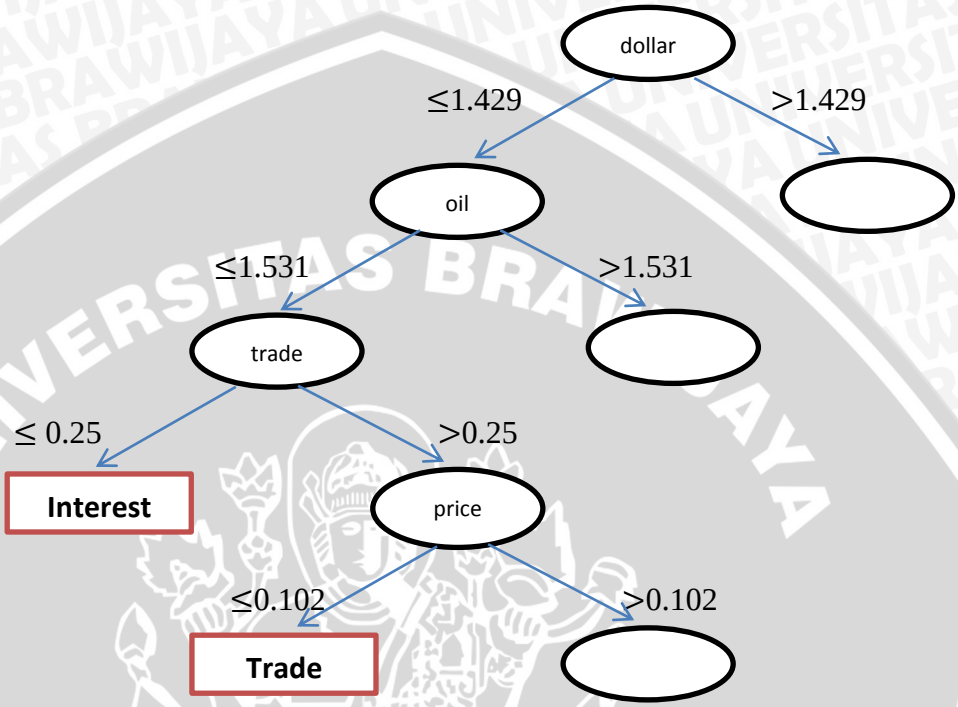
6. Pembentukan cabang ≤0.102 dari atribut *price*

   Proses awal adalah pembentukan cabang ≤0.102 dari atribut *price* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.29 yang mempunyai nilai ≤0.102 pada atribut *price*. Hasil pembentukan cabang ≤0.102 dari atribut *price* dapat dilihat pada tabel 3.32.

Tabel 3.32 Hasil Pembentukan Cabang ≤0.102 dari Atribut *Price*

Term	**D1**	**D2**
surplus	1.278	0
import	0.348	0.348
billion	1.224	0
Kategori	*Trade*	*Trade*

Dari tabel 3.32 dapat dilihat bahwa cabang ≤0.102 dari atribut *price* sudah berada pada kelas atau kategori yang sama, yaitu kategori *trade*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.23 *Tree* Hasil Cabang ≤0.102 dari Atribut *Price*

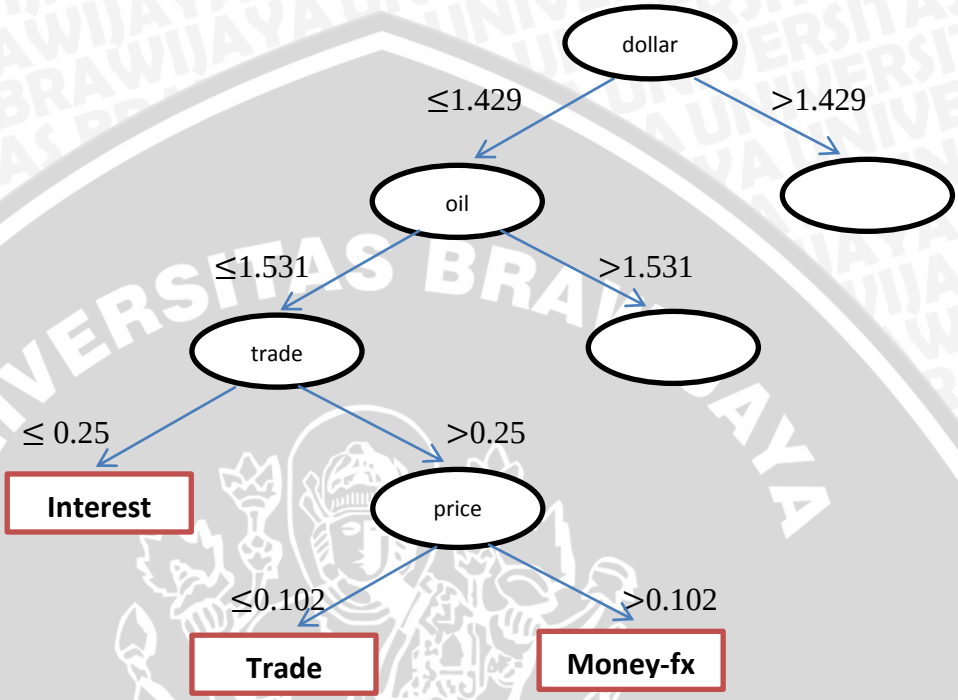
7. Pembentukan cabang >0.102 dari atribut *price*

   Proses awal adalah pembentukan cabang >0.102 dari atribut *price* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.29 yang mempunyai nilai >0.102 pada atribut *price*. Hasil pembentukan cabang >0.102 dari atribut *price* dapat dilihat pada tabel 3.33.

Tabel 3.33 Hasil Pembentukan Cabang >0.102 dari Atribut *Price*

Term	**D5**
surplus	0.426
import	0.058
billion	0.204
Kategori	*Money-fx*

Dari tabel 3.33 dapat dilihat bahwa cabang >0.102 dari atribut *price* sudah berada pada kelas atau kategori yang sama, yaitu kategori *money-fx*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.24 *Tree* Hasil Cabang >0.102 dari Atribut *Price*

8. Pembentukan cabang >1.531 dari atribut *oil*

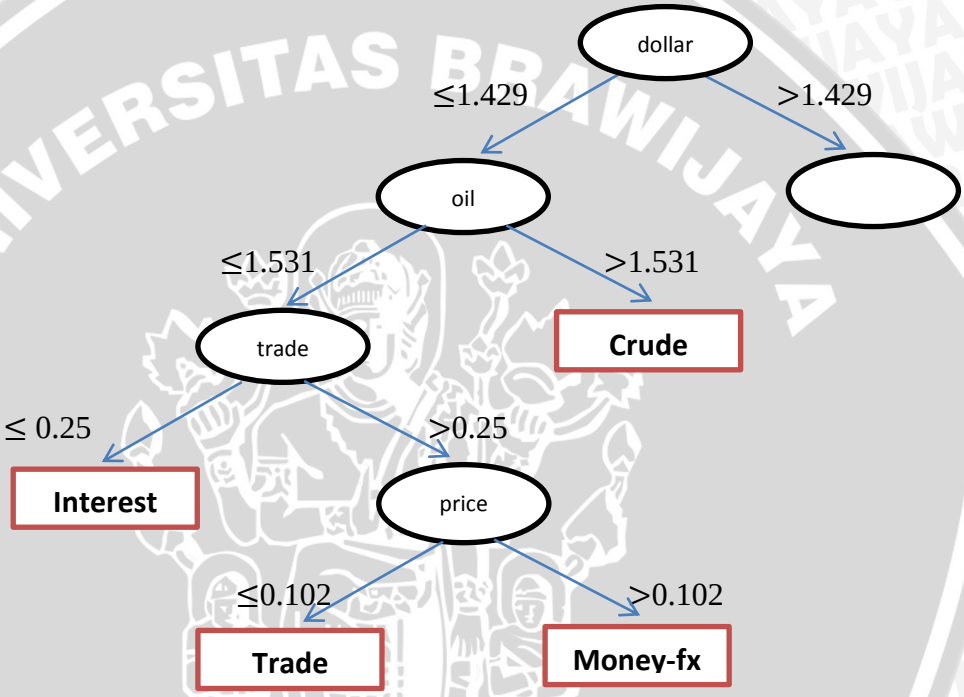
   Proses awal adalah pembentukan cabang >1.531 dari atribut *oil* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.20 yang mempunyai nilai >1.531 pada atribut *oil*. Hasil pembentukan cabang >1.531 dari atribut *oil* dapat dilihat pada tabel 3.34.

Tabel 3.34 Hasil Pembentukan Cabang >1.531 dari Atribut *Oil*

Term	**D4**
trade	0
surplus	0
import	0.058
billion	0

price	0.816
Kategori	*Crude*

Dari tabel 3.34 dapat dilihat bahwa cabang >1.531 dari atribut *oil* sudah berada pada kelas atau kategori yang sama, yaitu kategori *crude*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.25 *Tree* Hasil Cabang >1.531 dari Atribut *Oil*

9. Pembentukan cabang >1.429 dari atribut *dollar*

   Proses awal adalah pembentukan cabang >1.429 dari atribut *dollar* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.11 yang mempunyai nilai >1.429 pada atribut *dollar*. Hasil pembentukan cabang >1.429 dari atribut *dollar* dapat dilihat pada tabel 3.35.

Tabel 3.35 Hasil Pembentukan Cabang >1.429 dari Atribut *Dollar*

Term	**D3**	**D6**
trade	0.875	0.500
surplus	0	0
import	0.058	0.058
billion	1.02	0.204
oil	2.041	0
price	1.021	0.204
Kategori	*Crude*	*Money-fx*

Kemudian dilakukan perhitungan *information gain* dengan menggunakan cara yang sama seperti pada proses sebelumnya, sehingga diperoleh hasil perhitungan *information gain* untuk semua *term* yang dapat dilihat pada tabel 3.36.

Tabel 3.36 Hasil Perhitungan *Information Gain* untuk Cabang >1.429 dari Atribut *Dollar*

Term	**IG**
trade	0.162
surplus	0.301
import	0.063
billion	0.116
oil	0.934
price	0.116

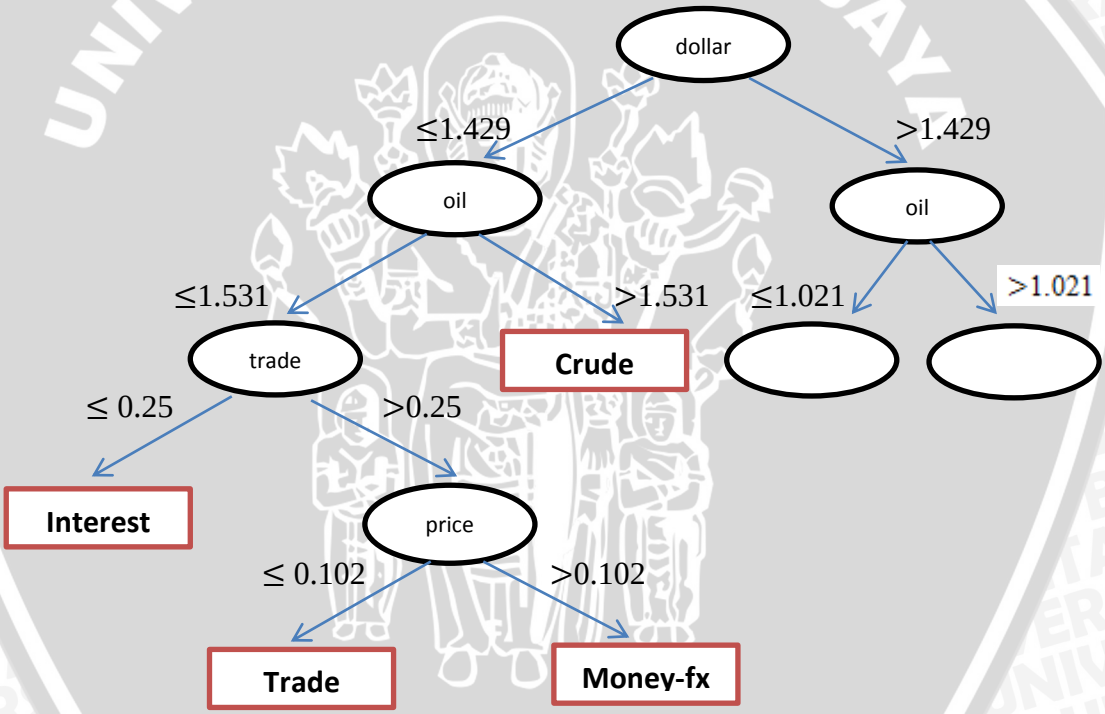
Dari tabel 3.36 *term* yang memiliki *information gain* tertinggi adalah *term oil*, maka atribut *oil* sebagai *node* berikutnya. Setelah itu menentukan *split-point*

dengan menggunakan cara yang sama seperti sebelumnya, sehingga diperoleh hasil *split-point* seperti pada tabel 3.37.

Tabel 3.37 Hasil *Split-point* dari Nilai *Median* untuk Atribut *Oil*

Median	***Split-point***	
1.021	≤1.021	>1.021

Dari tabel 3.37 dapat dilihat bahwa hanya terdapat satu *split-point* yaitu 1.021, sehingga *split-point* tersebut dapat digunakan tanpa harus menghitung nilai *information gain* terlebih dahulu. Gambar 3.26 menunjukkan bahwa atribut *oil* sebagai *node* berikutnya dengan nilai *binary split* ≤1.021 dan >1.021.



Gambar 3.26 *Tree* Hasil Cabang >1.429 dari Atribut *Dollar*

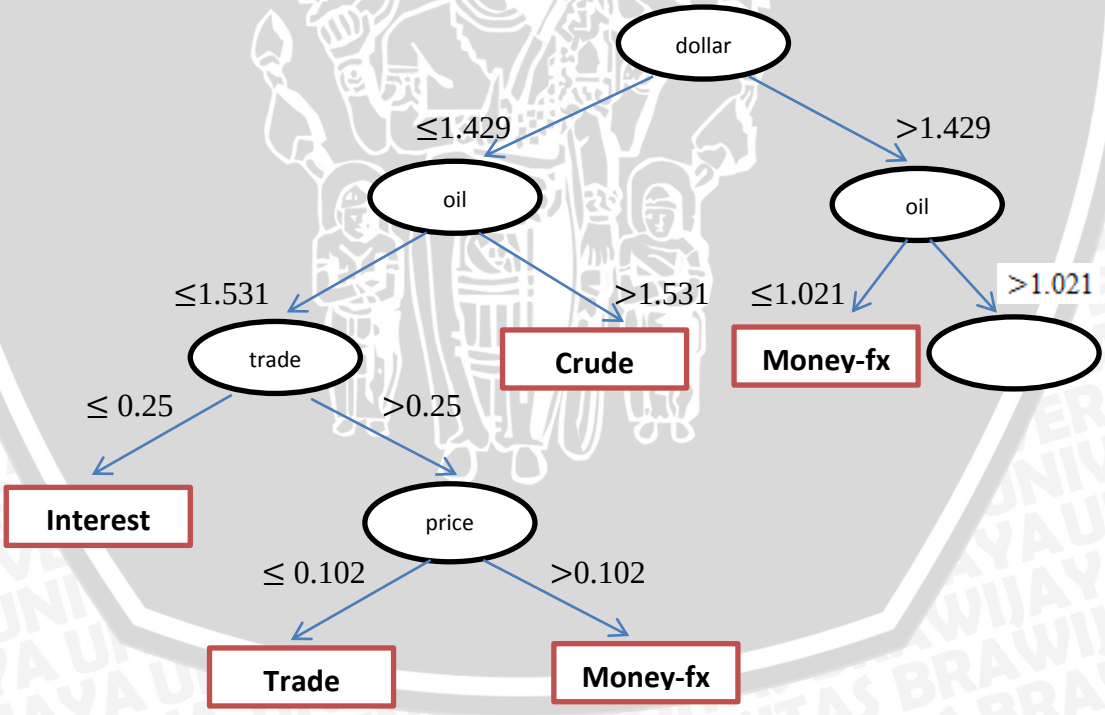
10. Pembentukan cabang ≤1.021 dari atribut *oil*

Proses awal adalah pembentukan cabang ≤1.021 dari atribut *oil* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.35 yang mempunyai nilai ≤1.021 pada atribut *oil*. Hasil pembentukan cabang ≤1.021 dari atribut *oil* dapat dilihat pada tabel 3.38.

Tabel 3.38 Hasil Pembentukan Cabang ≤1.021 dari Atribut *Oil*

Term	**D6**
trade	0.500
surplus	0
import	0.058
billion	0.204
price	0.204
Kategori	*Money-fx*

Dari tabel 3.38 dapat dilihat bahwa cabang ≤1.021 dari atribut *oil* sudah berada pada kelas atau kategori yang sama, yaitu kategori *money-fx*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.27 *Tree* Hasil Cabang ≤1.021 dari Atribut *Oil*

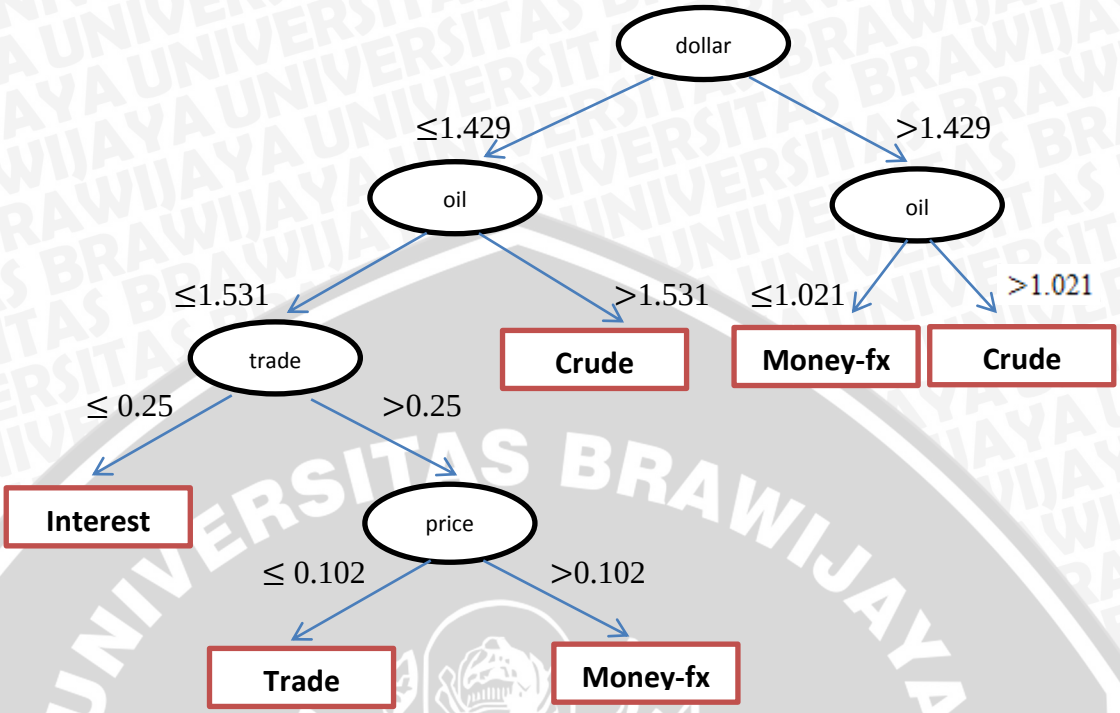
11. Pembentukan cabang >1.021 dari atribut *oil*

Proses awal adalah pembentukan cabang >1.021 dari atribut *oil* dengan melakukan perhitungan *information gain* pada dokumen yang terdapat pada tabel 3.35 yang mempunyai nilai >1.021 pada atribut *oil*. Hasil pembentukan cabang >1.021 dari atribut *oil* dapat dilihat pada tabel 3.39.

Tabel 3.39 Hasil Pembentukan Cabang >1.021 dari Atribut *Oil*

Term	**D3**
trade	0.875
surplus	0
import	0.058
billion	1.02
price	1.021
Kategori	*Crude*

Dari tabel 3.39 dapat dilihat bahwa cabang >1.021 dari atribut *oil* sudah berada pada kelas atau kategori yang sama, yaitu kategori *crude*. Sehingga terbentuk *tree* sebagai berikut.



Gambar 3.28 *Tree* Hasil Cabang >1.021 dari Atribut *Oil*

Perancangan *Rule Tree*

Setelah *tree* terbentuk, selanjutnya dilakukan perubahan menjadi *rule*. Berikut ini merupakan pembentukan *rule tree*:

1. **IF** atribut *dollar* = ≤1.429 **AND** atribut *oil* = ≤1.531 **AND** atribut *trade* = ≤0.25 **THEN** kategori ***interest.***
2. **IF** atribut *dollar* = ≤1.429 **AND** atribut *oil* = ≤1.531 **AND** atribut *trade* = >0.25 **AND** atribut *price* = ≤0.102 **THEN** kategori ***trade.***
3. **IF** atribut *dollar* = ≤1.429 **AND** atribut *oil* = ≤1.531 **AND** atribut *trade* = >0.25 **AND** atribut *price* = >0.102 **THEN** kategori ***money-fx.***
4. **IF** atribut *dollar* = ≤1.429 **AND** atribut *oil* = ≤1.531 **THEN** kategori ***crude.***
5. **IF** atribut *dollar* = >1.429 **AND** atribut *oil* = ≤1.021 **THEN** kategori ***money-fx.***
6. **IF** atribut *dollar* = >1.429 **AND** atribut *oil* = >1.021 **THEN** kategori ***crude.***

Perancangan Proses Klasifikasi

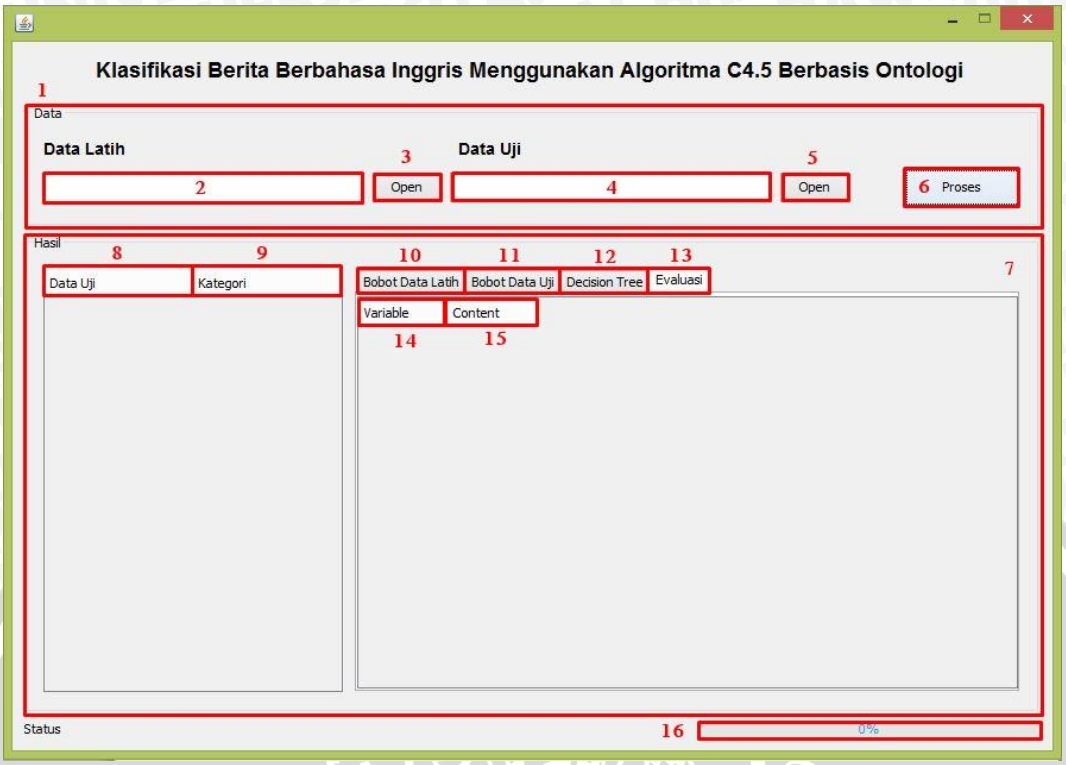
Proses klasifikasi ini dilakukan sesuai dengan hasil *rule* yang dibentuk pada proses pelatihan. *Input* pada proses ini adalah dokumen uji yang telah melalui proses pembobotan.

Dari tabel 3.14 dapat dilihat bahwa dokumen uji memenuhi *rule* ke-1, yaitu **IF** atribut *dollar* = ≤1.429 **AND** atribut *oil* = ≤1.531 **AND** atribut *trade* = ≤0.25 **THEN** kategori ***interest***, jadi dokumen uji tersebut termasuk dalam kategori berita tentang ***interest.***

Setelah melakukan beberapa pengujian, langkah selanjutnya adalah menghitung *precision*, *recall*, dan *f-measure* berdasarkan persamaan (2-10) sampai dengan (2-12).

3.4 Rancangan Antarmuka

Rancangan antarmuka dari sistem klasifikasi berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi secara umum terdiri dari bagian *input*, *output*, tombol proses. Input dari sistem berupa dokumen latih dan dokumen uji. Output dari sistem adalah kategori dokumen uji. Gambar 3.29 menunjukkan rancangan antarmuka sistem.



Gambar 3.29 Rancangan Antarmuka Sistem

Penjelasan dari setiap bagian rancangan antarmuka sistem yang telah ditandai oleh label angka berwarna merah adalah sebagai berikut:

1. Panel *input* data beserta tombol proses.
2. *Field* untuk menampilkan *folder* atau direktori dari dokumen latih.
3. *Button* untuk membuka dan memilih *folder* atau direktori dari dokumen latih.
4. *Field* untuk menampilkan *folder* atau direktori dari dokumen uji.
5. *Button* untuk membuka dan memilih *folder* atau direktori dari dokumen uji.
6. *Button* untuk mengeksekusi proses *preprocessing* dan klasifikasi pada dokumen latih dan dokumen uji.
7. Panel *output* atau hasil dari *preprocessing* dan klasifikasi.
8. *Field* untuk menampilkan data uji yang telah dipilih sebelumnya.
9. *Field* untuk menampilkan kategori dari dokumen uji yang telah diproses menggunakan algoritma C4.5.
10. *Tab* bobot data latih menampilkan hasil proses perhitungan bobot tiap *term* pada dokumen latih menggunakan TFIDF.

11. *Tab* bobot data uji menampilkan hasil proses perhitungan bobot tiap *term* pada dokumen uji menggunakan TFIDF.
12. *Tab decision tree* menampilkan hasil pembentukan pohon keputusan menggunakan algoritma C4.5.
13. *Tab* evaluasi berisi hasil evaluasi sistem yaitu nilai *recall*, *precision*, dan *f-measure*.
14. Tabel *variable* pada *tab* evaluasi berisi variabel untuk jumlah data latih, jumlah data uji, *precision*, *recall*, dan *f-measure*.
15. Tabel *content* pada *tab* evaluasi berisi nilai untuk jumlah data latih, jumlah data uji, *precision*, *recall*, dan *f-measure*.
16. *Progress bar* digunakan untuk mengetahui berapa lama proses berlangsung.

3.5 Rancangan Uji Coba

Setelah sistem selesai diimplementasikan maka langkah selanjutnya adalah melakukan serangkaian uji coba untuk mengetahui kinerja dari sistem ini. Pengujian pertama adalah jumlah dokumen uji dibuat tetap jumlahnya dalam setiap percobaan. Pengujian kedua adalah jumlah dokumen uji dibuat bervariasi dalam setiap percobaan. Pengujian ketiga adalah menggunakan *K-Fold* dimana dataset dibagi menjadi sejumlah K buah partisi secara acak, kemudian dilakukan sejumlah K kali eksperimen menggunakan data partisi ke-K sebagai data uji dan sisa partisi lainnya sebagai data latih. Pengukuran evaluasi sistem yang digunakan adalah *precision*, *recall* dan *f-measure*.

Tabel 3.40 merupakan rancangan pengujian untuk jumlah dokumen uji yang dibuat tetap jumlahnya dengan perbandingan data latih dan data uji adalah 2:1, 4:1, 6:1, 8:1, 10:1, dan 12:1. Tabel 3.41 merupakan rancangan pengujian untuk jumlah dokumen uji yang dibuat bervariasi jumlahnya dengan perbandingan data latih dan data uji adalah 30% : 70%, 40% : 60%, 50% : 50%, 60% : 40%, 70% : 30% dan 90% : 10%. Tabel 3.42 merupakan rancangan pengujian untuk *K-Fold* dengan menggunakan tiga kombinasi, yaitu kombinasi pertama dan kedua menggunakan k = 3, dan kombinasi ketiga menggunakan k = 2. Pada kombinasi pertama data latih dibagi menjadi 3 *subset* (S_1, S_2, S_3) dimana pada tiap *subset* terdapat 40 dokumen latih yang berbeda. Pada kombinasi kedua data latih dibagi

menjadi 3 *subset* (S_1, S_2, S_3) dimana pada tiap *subset* terdapat 80 dokumen latih yang berbeda. Pada kombinasi ketiga data latih dibagi menjadi 2 *subset* (S_1, S_2) dimana pada tiap *subset* terdapat 120 dokumen latih yang berbeda. Ketiga kombinasi tersebut menggunakan data uji yang sama yaitu berjumlah 20 dokumen uji yang bukan termasuk dalam dokumen latih.

Tabel 3.40 Rancangan Pengujian Dokumen Uji dengan Jumlah Tetap

Uji Coba	Perbandingan jumlah data		Jumlah data sebenarnya		*Precision (%)*	*Recall (%)*	*F-Measure (%)*
	Data latih	Data uji	Data latih	Data uji			
1	2	1	40	20			
2	4	1	80	20			
3	6	1	120	20			
4	8	1	160	20			
5	10	1	200	20			
6	12	1	240	20			

Tabel 3.41 Rancangan Pengujian Dokumen Uji dengan Jumlah Bervariasi

Uji Coba	Perbandingan jumlah data		Jumlah data sebenarnya		*Precision (%)*	*Recall (%)*	*F-Measure (%)*
	Data latih	Data uji	Data latih	Data uji			
1	30%	70%	60	140			
2	40%	60%	80	120			
3	50%	50%	100	100			
4	60%	40%	120	80			
5	70%	30%	140	60			
6	90%	10%	180	20			

Tabel 3.42 Rancangan Pengujian menggunakan *K-Fold*

Kombinasi ke-	Uji Coba	Jumlah data		*Precision (%)*	*Recall (%)*	*F-Measure (%)*
		Data latih	Data uji			
1	1	$S_1 = 40$	$S_4 = 20$			
	2	$S_2 = 40$	$S_4 = 20$			
	3	$S_3 = 40$	$S_4 = 20$			
2	1	$S_1 = 80$	$S_4 = 20$			
	2	$S_2 = 80$	$S_4 = 20$			
	3	$S_3 = 80$	$S_4 = 20$			
3	1	$S_1 = 120$	$S_3 = 20$			
	2	$S_2 = 120$	$S_3 = 20$			

BAB IV
IMPLEMENTASI

Bab ini berisi lingkungan implementasi, implementasi program, dan implementasi antarmuka.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang sesuai dibutuhkan agar proses-proses dapat diimplementasikan dan berjalan tepat. Sistem dibangun pada lingkungan implementasi perangkat keras dan perangkat lunak. Lingkungan implementasi perangkat keras dan perangkat lunak yang digunakan adalah sebagai berikut :

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pembuatan sistem memiliki beberapa spesifikasi sebagai berikut:

1. Processor Intel® Core™ i3 CPU M 330 @2.13GHZ
2. Memory 2 GB
3. Harddisk 320 GB
4. Monitor 14”

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan untuk implementasi sistem ini adalah sebagai berikut:

1. Sistem Operasi Windows 8 Enterprise 32-bit
2. Java Development Kit (JDK) 1.7 (build 1.7.0_45)
3. Netbeans IDE 8.0
4. WordNet 3.1
5. Text Editor Notepad
6. Microsoft Office Visio 2007

4.2 Implementasi Program

4.2.1 Kelas dan *Method*

Pada sub bab ini membahas mengenai kelas-kelas utama yang terdapat pada sistem beserta *method* pada setiap kelas beserta kegunaannya. Kelas-kelas

utama tersebut terdiri dari kelas `TextProcessing.java`, `WordNetLib.java`, `Weighting.java`, dan `C45Processing.java`

1. Kelas `TextProcessing.java`

   Kelas ini digunakan untuk membantu semua operasi pada teks. *Method-method* dari kelas `TextProcessing.java` dapat dilihat pada tabel 4.1.

Tabel 4.1 *Method-method* pada Kelas `TextProcessing.java`

Nama *Method*	Keterangan
`Tokenizing()`	*Method* untuk mengubah teks menjadi karakter huruf kecil (proses *case folding*) dan memisah teks ke dalam kata-kata berdasarkan spasi.
`Filtering()`	*Method* untuk menghapus *term* yang tidak diperlukan menurut daftar *stopwords.*
`Stemming()`	*Method* untuk mengubah *term* menjadi kata dasarnya.
`HitungFrekDataLatih()`	*Method* untuk menghitung frekuensi *term* yang terdapat pada data latih.
`HitungFrekDataUji()`	*Method* untuk menghitung frekuensi *term* yang terdapat pada data uji.

2. Kelas `WordNetLib.java`

   Kelas ini digunakan untuk melakukan proses ontologi. *Method-method* dari kelas `WordNetLib.java` dapat dilihat pada tabel 4.2.

Tabel 4.2 *Method-method* pada Kelas `WordNetLib.java`

Nama *Method*	Keterangan
`OpenDict()`	*Method* untuk inisialisasi kamus wordnet.
`isSynonim(String word1, String word2)`	*Method* untuk menentukan apakah kedua kata bersinonim.
`HitungDF(int [] frek)`	*Method* untuk menghitung dokumen frekuensi.
`CombineFrek(int [] frek1, int [] frek2)`	*Method* untuk menggabung 2 term yang bersinonim (menjumlahkan

	frekuensinya).
`OntologyProses(HashMap<String, int[]> FREKUENSI)`	*Method* untuk proses ontologi yaitu membandingkan *term*.

3. Kelas `Weighting.java`

   Kelas ini digunakan untuk menghitung bobot setiap dokumen. *Method-method* dari kelas `Weighting.java` dapat dilihat pada tabel 4.3.

Tabel 4.3 *Method-method* pada Kelas `Weighting.java`

Nama *Method*	Keterangan
`TFIDF()`	*Method* untuk menghitung bobot data latih menggunakan metode TFIDF.
`TFIDFUJI()`	*Method* untuk menghitung bobot data uji.

4. Kelas `C45Processing.java`

   Kelas ini digunakan untuk proses klasifikasi menggunakan algoritma C4.5. *Method-method* dari kelas `C45Processing.java` dapat dilihat pada tabel 4.4.

Tabel 4.4 *Method-method* pada Kelas `C45Processing.java`

Nama *Method*	Keterangan
`CreateTree()`	*Method* untuk membentuk *tree*.
`CreateLeafNode (PreNode NodeProses, Stack UrutanNode)`	*Method* untuk membentuk *node* dan *leaf tree*.
`ReadTree()`	*Method* untuk membaca hasil pembentukan *tree*.

4.2.2 Tahapan Pemrosesan

Proses pertama pada sistem klasifikasi berita berbahasa inggris menggunakan algoritma C4.5 berbasis ontologi adalah proses *preprocessing*. Pada kelas `TextProcessing.java` ini berisi *method-method* untuk *preprocessing* dokumen latih dan dokumen uji, yaitu terdiri dari proses *case folding, tokenizing, filtering, stemming,* dan perhitungan *term frequency,* serta pembentukan kamus (*dictionary construction*) khusus untuk dokumen latih. Proses awal dimulai dari

proses *case folding*. Setelah proses *case folding*, langkah selanjutnya adalah *tokenizing* dengan menggunakan fungsi *split* dari *Java*. Fungsi *split* ini digunakan untuk memisah kata berdasarkan spasi. *Source code* dari proses *case folding* dan *tokenizing* dapat dilihat pada *source code* 4.1. Proses selanjutnya adalah proses *filtering*, yaitu membuang *term* yang terdapat pada daftar *stopword*. Daftar *stopword* ini diambil dari kelas *PublicResource*. *Source code* dari proses *filtering* dapat dilihat pada *source code* 4.2. Setelah melalui *filtering*, *term-term* yang bukan termasuk dalam *stopword* akan melalui proses *stemming* dimana aturan dari *stemming* diambil dari kelas `englishStemmer.java`. *Source code* dari proses *stemming* dapat dilihat pada *source code* 4.3.

```
public ArrayList<String> Tokenizing(){
// menghapus seluruh karakter kecuali a-zA-Z dan [spasi]
        Text = Text.replaceAll("[^a-zA-Z ]", "").toLowerCase();
// menghapus spasi double (lebih dari satu spasi)
        Text = Text.replaceAll("\\s+", " ");
        Text = Text.trim();
// memisah kata berdasarkan spasi
        Terms = new ArrayList<>(Arrays.asList(Text.split(" ")));
        return Terms;
    }
```

Source Code* 4.1 Proses *Case Folding* dan *Tokenizing

```
public ArrayList<String> Filtering(){
        for (int i=0;i<Terms.size();i++) {
            if(PublicResource.STOPWORDS.contains(Terms.get(i))){
                Terms.remove(i);
                i--;
            }
        }
        return Terms;
    }
```

Source Code* 4.2 Proses *Filtering

```
public ArrayList<String> Stemming(){
        englishStemmer eStemmer = new englishStemmer();
        for(int i=0;i<Terms.size();i++){
            eStemmer.setCurrent(Terms.get(i));
            eStemmer.stem();
            Terms.set(i, eStemmer.getCurrent());
        }
        if(Mode == 0){
```

```
            HitungFrekDataLatih();
        }else{
            HitungFrekDataUji();
        }
        return Terms;
    }
```

Source Code* 4.3 Proses *Stemming

Kemudian *term* hasil *stemming* akan disimpan pada *Inverted Index*. Proses ini disebut pembentukan kamus (*dictionary construction*). *Inverted index* menggunakan struktur data *hashmap* yang terdiri dari *term* dan *posting list* yang merupakan nilai dari frekuensi tiap *term* pada setiap dokumen. Apabila *term* belum ada pada *hashmap* maka *term* akan disimpan dan membuat *posting list* baru. Apabila *term* sudah ada tetapi nama dokumen belum ada, maka akan dibuat *posting list* baru untuk dokumen tersebut beserta nilai frekuensinya. *Source code* dari proses *dictionary construction* dokumen latih dapat dilihat pada *source code* 4.4.

```
private void HitungFrekDataLatih(){
        for(String Term : Terms){
         if(PublicResource.FREKUENSI_DATALATIH.containsKey(Term)){
           // Jika term sudah terdapat di hashmap
PublicResource.FREKUENSI_DATALATIH.get(Term)[DokumenKe] += 1;
            }else{
           // Jika term belum terdapat di hashmap
           int [] frek = new int[PublicResource.JUMLAH_DATALATIH];
           frek[DokumenKe] = 1;
           PublicResource.FREKUENSI_DATALATIH.put(Term, frek);
            }
        }
    }
```

Source Code* 4.4 Proses *Dictionary Construction

Selanjutnya *term-term* yang telah disimpan dalam *inverted index* dihitung menggunakan metode *Document Frequency* (DF). *Source code* untuk menghitung *Document Frequency* dapat dilihat pada *source code* 4.5.

```
String [] Keys =
PublicResource.FREKUENSI_DATALATIH.keySet().toArray(new
String[0]);
        for(String key : Keys){
            int [] frekValue =
PublicResource.FREKUENSI_DATALATIH.get(key);
            int DF = 0;
```

```
            for(int frek : frekValue){
                if(frek != 0){
                    DF++;
                }
            }
```

***Source Code* 4.5 Proses Menghitung *Document Frequency* (DF)**

Setelah proses perhitungan *Document Frequency* (DF), selanjutnya adalah proses *ontology extraction* yaitu menggabungkan *term* yang saling bersinonim menggunakan database *WordNet*. Proses awal adalah dicari *synset* atau sekumpulan kata yang bersinonim dengan *term* tersebut. Apabila termasuk dalam *synset*, selanjutnya dicari *Document Frequency* (DF) dari masing-masing *term*, baru kemudian *term* tersebut digabungkan. *Term* yang memiliki DF lebih kecil akan digabungkan dengan *term* yang memiliki DF lebih besar. *Source code* dari proses pengecekan sinonim *term* dapat dilihat pada *source code* 4.6. *Source code* dari proses *ontology extraction* dapat dilihat pada *source code* 4.7.

```
public boolean isSynonim(String word1, String word2){
        boolean isSynonim = false;
        POS [] posType = {POS.NOUN, POS.VERB, POS.ADVERB,
POS.ADJECTIVE};
        IIndexWord idxWord;
        for (POS posType1 : posType) {
            idxWord = dict.getIndexWord(word1, posType1);
            if(idxWord != null){
                List<IWordID> wordID = idxWord.getWordIDs();
                IWord word;
                ISynset synset;
                for(IWordID iwordID : wordID){
                    word = dict.getWord(iwordID);
                    synset = word.getSynset();
                    for(IWord w : synset.getWords()){
                        if(word2.equalsIgnoreCase(w.getLemma())){
                            isSynonim = true;
                            return isSynonim;
                        }
                    }
                }
            }
        }
        return isSynonim;
    }
```

***Source Code* 4.6 Proses Pengecekan Sinonim**

```
private void OntologyProses(HashMap<String, int[]> FREKUENSI){
        ArrayList<String> Keys = new
ArrayList<>(FREKUENSI.keySet());

        int first = 0;
        int banding = 1;
        while(first < (Keys.size()-1)){
            mainFrame.setStatusKonten("Membandingkan
"+Keys.get(first)+" dengan "+Keys.get(banding));
            if(isSynonim(Keys.get(first), Keys.get(banding))){
                if(HitungDF(FREKUENSI.get(Keys.get(first))) >
HitungDF(FREKUENSI.get(Keys.get(banding)))){
                    CombineFrek(FREKUENSI.get(Keys.get(first)),
                            FREKUENSI.get(Keys.get(banding)));
                    FREKUENSI.remove(Keys.get(banding));
                    Keys.remove(banding);
                }else{
                    CombineFrek(FREKUENSI.get(Keys.get(banding)),
                            FREKUENSI.get(Keys.get(first)));
                    FREKUENSI.remove(Keys.get(first));
                    Keys.remove(first);
                    banding = first + 1;
                }
            }else{
                banding++;
            }

            if(banding == Keys.size()){
                first++;
                banding = first + 1;
            }
//              System.out.println(first+" == "+(Keys.size()-1)+"
||| "+banding+" ==>>"+100 * first/(Keys.size()-1));
            setProgress(100 * first/(Keys.size()-1));
        }
    }
```

Source Code* 4.7 Proses *Ontology Extraction

Selanjutnya adalah proses pembobotan setiap *term* dari dokumen latih dengan menggunakan metode TFIDF. Sebelumnya dilakukan perhitungan IDF terlebih dahulu yang ditunjukkan *source code* 4.8. Kemudian menghitung TFIDF dari setiap *term* dengan cara mengalikan *term frequency* setiap *term* pada setiap *posting list* dengan nilai IDF (*inverse document frequency*) dari *term* tersebut. *Source code* dari proses pembobotan dokumen latih dapat dilihat pada *source code* 4.9.

```
FileOperation op = new FileOperation();
        // Hitung IDF
        String [] Keys =
PublicResource.FREKUENSI_DATALATIH.keySet().toArray(new
String[0]);
        for(String key : Keys){
            int [] frekValue =
PublicResource.FREKUENSI_DATALATIH.get(key);
            int DF = 0;
            for(int frek : frekValue){
                if(frek != 0){
                    DF++;
                }
            }
            double IDF = Math.log10((double) frekValue.length /
DF);
            PublicResource.DATA_IDF.put(key, IDF);
        }
        op.SaveIDF();
```

***Source Code* 4.8 Proses Perhitungan IDF**

```
for(String key : Keys){
            int [] frekValue =
PublicResource.FREKUENSI_DATALATIH.get(key);
            double [] WeightValue = new double[frekValue.length];
            for(int i=0;i<WeightValue.length;i++){
                WeightValue[i] = (double) frekValue[i] *
PublicResource.DATA_IDF.get(key);
            }

            PublicResource.WEIGHT_DATALATIH.put(key, WeightValue);
        }
        op.SaveTFIDF();
```

***Source Code* 4.9 Proses Pembobotan Dokumen Latih**

Tahap-tahap untuk *preprocessing* dokumen uji adalah *case folding*, *tokenizing*, *filtering*, *stemming*, *ontology extraction*, dan perhitungan *term frequency* untuk pembobotan dokumen uji. Proses *case folding*, *tokenizing*, *filtering*, dan *stemming*, dilakukan sama seperti pada *preprocessing* dokumen latih. Selanjutnya dilakukan proses *ontology extraction* dan perhitungan *term frequency* untuk pembobotan pada dokumen uji. Pada dokumen uji tidak terdapat proses *dictionary construction* tetapi bobot dari dokumen uji ditentukan berdasarkan kemuculan *term* tersebut pada *inverted index* dokumen latih. *Source code* dari proses *ontology extraction* dan perhitungan *term frequency* dari

dokumen uji dapat dilihat pada *source code* 4.10. Setelah dihitung *term frequency* dari dokumen uji maka langkah selanjutnya adalah menghitung TFIDF dari dokumen uji. IDF yang digunakan pada penghitungan bobot dokumen uji adalah IDF pada dokumen latih. Proses penghitungan TFIDF dokumen uji dapat dilihat pada *source code* 4.11.

```
private void HitungFrekDataUji(){
        String [] keysetUji =
PublicResource.FREKUENSI_DATAUJI.keySet().toArray(new String[0]);
        WordNetLib wnLib = new WordNetLib(null, Mode);
        for(String Term : Terms){
if(PublicResource.FREKUENSI_DATAUJI.containsKey(Term)){
                // Jika term sudah terdapat di hashmap
PublicResource.FREKUENSI_DATAUJI.get(Term)[DokumenKe] += 1;
            }else{
                for(String key : keysetUji){
                    if(wnLib.isSynonim(Term, key)){
PublicResource.FREKUENSI_DATAUJI.get(key)[DokumenKe] += 1;
                        break;
                    }
                }
            }
        }
}
```

***Source Code* 4.10 Proses *Ontology Extraction* dan Perhitungan *Term Frequency* Dokumen Uji**

```
public static void TFIDFUJI(){
        String [] Keys =
PublicResource.FREKUENSI_DATAUJI.keySet().toArray(new String[0]);
        for(String key : Keys){
            int [] frekValue =
PublicResource.FREKUENSI_DATAUJI.get(key);
            double [] WeightValue = new double[frekValue.length];
            for(int i=0;i<WeightValue.length;i++){
                WeightValue[i] = (double) frekValue[i] *
PublicResource.DATA_IDF.get(key);
            }

            PublicResource.WEIGHT_DATAUJI.put(key, WeightValue);
        }
    }
```

***Source Code* 4.11 Proses Pembobotan Dokumen Uji**

Tahapan setelah pembobotan *term* yaitu pembentukan *root* utama. Proses ini diawali dengan perhitungan *information gain* pada kelas

`C45Processing.java`. *Source code* dari proses pembentukan *root* melalui perhitungan *information gain* dapat dilihat pada *source code* 4.12.

```
public void CreateTree(){
        // Hitung IG tiap term
        String [] keysKategori =
PublicResource.DATA_JUMDOK_KAT.keySet().toArray(new String[0]);
        HashMap<String, Double>  nilaiIG = new HashMap<>();
        String [] keysTerm =
PublicResource.WEIGHT_DATALATIH.keySet().toArray(new String[0]);
        for(String term : keysTerm){
            // Hitung Probabilitas global
            double ProbGlobal = 0.0;
            for(String kat : keysKategori){
                Integer jumdok =
PublicResource.DATA_JUMDOK_KAT.get(kat);
                if(jumdok > 0){
                    double Pi = (double)
jumdok/PublicResource.JUMLAH_DATALATIH;
                    double pg = Pi * Math.log10(Pi);
                    ProbGlobal += pg;
                }
            }
            //Hitung F(w)
            double F_w = 0.0;
            double [] tfidf_w =
PublicResource.WEIGHT_DATALATIH.get(term);
            for(double w : tfidf_w){
                if(w > 0){
                    F_w += 1;
                }
            }
            //Hitung Probabilitas Class i
            double ProbClass = 0.0;
            for(int i=0;i<tfidf_w.length;i++){
                String kategori =
PublicResource.DOCUMENTS.get(i).getKategori();
                Integer JumDok =
PublicResource.DATA_JUMDOK_KAT.get(kategori);
                if(JumDok > 0 && tfidf_w[i] > 0){
                    double Pi_w = tfidf_w[i]/JumDok;
                    double pc = Pi_w * Math.log10(Pi_w);
                    if(Double.isNaN(pc)){
                        pc = 0;
                    }
                    ProbClass += pc;
                }
            }
            double F_w_1 = 1 - F_w;
            double ProbClass_1 = 0.0;
```

```
            for(int i=0;i<tfidf_w.length;i++){
                String kategori =
PublicResource.DOCUMENTS.get(i).getKategori();
                Integer JumDok =
PublicResource.DATA_JUMDOK_KAT.get(kategori);
                if(JumDok > 0 && tfidf_w[i] > 0){
                    double Pi_w = tfidf_w[i]/JumDok;
                    double pc = (1 - Pi_w) * Math.log10(1 - Pi_w);
                    if(Double.isNaN(pc)){
                        pc = 0;
                    }
                    ProbClass_1 += pc;
                }
            }
            double IG_w = (-1 * ProbGlobal) + (F_w * ProbClass) +
(F_w_1 * ProbClass_1);
            nilaiIG.put(term, IG_w);
        }
        // get maksimum IG
        double IGmaks = -99.0;
        String Termmaks = "";
        for(String term : keysTerm){
            if(nilaiIG.get(term) > IGmaks){
                IGmaks = nilaiIG.get(term);
                Termmaks = term;
            }
        }
```

Source Code* 4.12 Proses Pembentukan *Root* melalui Perhitungan *Information Gain

Pada proses perhitungan *information gain* tersebut akan diperoleh *term* dengan nilai *information gain* terbesar dimana akan menjadi *root*. Selanjutnya untuk membentuk cabang, dilakukan proses perhitungan untuk menentukan *split-point* atau nilai yang akan digunakan untuk men-*split*. Teknik *splitting* menggunakan *binary split* yaitu ≤ dan >. Dari nilai pembobotan *term* yang dipilih akan di-*sorting* kemudian dihitung nilai *median* pada setiap *point*, selanjutnya dari *splitting* nilai *median* tersebut dihitung nilai *information gain*, dimana *split-point* dengan nilai *information gain* terkecil yang akan dipilih. *Source code* dari proses penentuan *split-point* dapat dilihat pada *source code* 4.13.

```
  // Split term
       double [] TFIDFTerpilih =
PublicResource.WEIGHT_DATALATIH.get(Termmaks);
       ArrayList<TFIDF> TFIDFSort = new ArrayList<>();
```

```
        for(int i=0;i<TFIDFTerpilih.length;i++){
            TFIDFSort.add(new TFIDF(TFIDFTerpilih[i],
PublicResource.DOCUMENTS.get(i).getKategori()));
        }
        for(int i=0;i<TFIDFSort.size();i++){
                for(int j=0;j<TFIDFSort.size()-1;j++){
                    if(TFIDFSort.get(j).getNilaiTFIDF() >
        TFIDFSort.get(j+1).getNilaiTFIDF()){
                        TFIDF temp = TFIDFSort.get(j);
                        TFIDFSort.set(j, TFIDFSort.get(j+1));
                        TFIDFSort.set(j+1, temp);
                    }
                }
         }
        // Hitung median
              ArrayList<Double> HasilMedian = new
        ArrayList<>();
              for(int i=0;i<TFIDFSort.size()-1;i++){

        if(!TFIDFSort.get(i).getKategori().equals(TFIDFSort.get(i
        +1).getKategori())){
                    double median = 0.0;
                    if(TFIDFSort.get(i+1).getNilaiTFIDF() !=
        0){
                        median =
        (TFIDFSort.get(i).getNilaiTFIDF() +
        TFIDFSort.get(i+1).getNilaiTFIDF()) / 2.0;
                    }
                    HasilMedian.add(median);
                }
              }
        // Hitung IG tiap split point
              double minIGPoint = 99;
              double MedianTerpilih = 0.0;
        for(double med : HasilMedian){
                ArrayList<Dokumen> DokKurangDari = new
        ArrayList<>();
                ArrayList<Dokumen> DokLebihDari = new
        ArrayList<>();
                for(int i=0;i<TFIDFTerpilih.length;i++){
                    if(TFIDFTerpilih[i] <= med){

        DokKurangDari.add(PublicResource.DOCUMENTS.get(i));
                    }else{

        DokLebihDari.add(PublicResource.DOCUMENTS.get(i));
                    }
                }
                // Hitung Entropy tiap bagian
                HashMap<String, Integer> MapKD =
```

```
HitungDokKategori(DokKurangDari);
            HashMap<String, Integer> MapLD =
HitungDokKategori(DokLebihDari);
            double entropyKD = 0.0;
            double entropyLD = 0.0;

            String [] KeyKD = MapKD.keySet().toArray(new
String[0]);
            String [] KeyLD = MapLD.keySet().toArray(new
String[0]);
for(String key : KeyKD){
  entropyKD += (-1 * ((double)MapKD.get(key) /
  DokKurangDari.size())) *
  Math.log10((double)MapKD.get(key) /
  DokKurangDari.size());
}
for(String key : KeyLD){
  entropyLD += (-1 * ((double)MapLD.get(key) /
  DokLebihDari.size())) *
  Math.log10((double)MapLD.get(key) /
  DokLebihDari.size());
 }
double nilaiig = (((double)DokKurangDari.size() /
TFIDFTerpilih.length) * entropyKD) +
(((double)DokLebihDari.size() / TFIDFTerpilih.length) *
entropyLD);
// menentukan nilai ig minimum
   if(nilaiig < minIGPoint){
      minIGPoint = nilaiig;
      MedianTerpilih = med;
   }
}
```

Source Code* 4.13 Proses Penentuan *Split-point

Selanjutnya setelah menemukan *split-point*, mencari dokumen-dokumen yang terdapat pada masing-masing cabang (cabang ≤ dan cabang >) dengan melihat nilai pembobotan TFIDF. Kemudian menghitung *information gain* tiap *term* dan menentukan *split-point* seperti pada proses sebelumnya. *Source code* dari proses pencarian dokumen untuk pembentukan *node* selanjutnya dapat dilihat pada *source code* 4.14.

```
// Menyiapkan data untuk pembentukan node berikutnya
ArrayList<Dokumen> DokKurangDari = new ArrayList<>();
ArrayList<Dokumen> DokLebihDari = new ArrayList<>();
ArrayList<Integer> IndKurangDari = new ArrayList<>();
ArrayList<Integer> IndLebihDari = new ArrayList<>();
for(int i=0;i<TFIDFTerpilih.length;i++){
```

```
   if(TFIDFTerpilih[i] <= MedianTerpilih){
      DokKurangDari.add(PublicResource.DOCUMENTS.get(i));
      IndKurangDari.add(i);
   }else{
      DokLebihDari.add(PublicResource.DOCUMENTS.get(i));
      IndLebihDari.add(i);
   }
}
HashMap<String, double[]> WeightKurangDari = new HashMap<>();
HashMap<String, double[]> WeightLebihDari = new HashMap<>();
for(String term : keysTerm){
  if(!term.equals(Termmaks)){
    double [] valueKD = new double[IndKurangDari.size()];
    for(int i=0;i<IndKurangDari.size();i++){
        valueKD[i] =
PublicResource.WEIGHT_DATALATIH.get(term)[IndKurangDari.get(i)];
    }
    double [] valueLD = new double[IndLebihDari.size()];
    for(int i=0;i<IndLebihDari.size();i++){
        valueLD[i] =
PublicResource.WEIGHT_DATALATIH.get(term)[IndLebihDari.get(i)];
    }
    WeightKurangDari.put(term, valueKD);
    WeightLebihDari.put(term, valueLD);
  }
}
HashMap<String, Integer> JumDokKD =
HitungDokKategori(DokKurangDari);
HashMap<String, Integer> JumDokLD =
HitungDokKategori(DokLebihDari);
```

***Source Code* 4.14 Proses Pencarian Dokumen Untuk *Node* Selanjutnya**

Kemudian setelah ditemukan kandidat dokumen untuk *node* selanjutnya, dilakukan proses perhitungan seperti pada proses sebelumnya. Proses tersebut akan berjalan rekursif untuk menemukan *node* selanjutnya dan akan berhenti jika dokumen sudah berada pada kategori yang sama dan pada kondisi tersebut *leaf* akan terbentuk. *Source code* dari proses pembentukan *leaf* dapat dilihat pada *source code* 4.15.

```
private void CreateLeafNode(PreNode NodeProses, Stack UrutanNode){
  System.out.println("Create Node");
  // Cek bahwa Dokumen sudah dalam satu kategori
  boolean isSatuKat = true;
  for(int i=1;i<NodeProses.getDocuments().size();i++){
    if(!NodeProses.getDocuments().get(0).getKategori().equals(Node
    Proses.getDocuments().get(i).getKategori())){
       isSatuKat = false;
       break;
    }
  }
```

Source Code* 4.15 Proses Pembentukan *Leaf

Setelah proses pembentukan *tree* selesai, selanjutnya adalah proses pengkategorian dokumen. Proses penelusuran dokumen uji pada *tree* adalah berdasarkan nilai pembobotan *term.* Penelusuran dimulai dari *root* kemudian ke *node* selanjutnya hingga menemukan kategori pada *leaf*. *Source code* proses pengkategorian dokumen dapat dilihat pada *source code* 4.16.

```
public String SearchKategoriTree(int dokumenKe){
   String KategoriFinal = "";
   NodeV2 nodeChecked = PublicResource.NODE_ROOT_V2;
   while(KategoriFinal.equals("")){
     String Term = nodeChecked.getTerm();
     if(!Term.equals("")){
       double SplitPoint = nodeChecked.getSplitPoint();
       ArrayList<NodeV2> child = nodeChecked.getChild();
       if(PublicResource.WEIGHT_DATAUJI.get(Term)[dokumenKe] <=
       SplitPoint){
           nodeChecked = child.get(0);
       }else{
          nodeChecked = child.get(1);
       }
       if(nodeChecked == null){
          return "Unknown";
       }
     }else{
        KategoriFinal = nodeChecked.getKategori();
     }
   }
   return KategoriFinal;
}
```

***Source Code* 4.16 Proses Pengkategorian Dokumen**

Proses akhir pada implementasi ini yaitu menguji kinerja sistem dalam melakukan pengklasifikasian data dengan menggunakan metode *precision, recall,* dan *f-measure* sesuai pada perancangan evaluasi pada bab sebelumnya. *Precision* adalah jumlah dokumen yang diklasifikasikan dengan benar oleh sistem dibagi dengan jumlah keseluruhan klasifikasi yang dilakukan oleh sistem. *Recall* adalah jumlah dokumen yang terklasifikasi dengan benar oleh sistem dibagi dengan jumlah dokumen yang seharusnya bisa dikenali sistem. *F-measure* merupakan nilai yang mewakili kinerja keseluruhan sistem dan merupakan penggabungan nilai *recall* dan *precision.* Proses perhitungan *precision, recall,* dan *f-measure* dapat dilihat pada *source code* 4.17.

```
private double [] HitungAkurasi(String kategori){
        int BenarDikenali = 0;
        int Dikenali = 0;
        int Sebenarnya = 0
        for(Dokumen dok : PublicResource.DOCUMENTS_UJI){
            if(dok.getKategori().equals(kategori) ||
dok.getKategoriDikenali().equals(kategori)){
                // Benar dikenali sebagai "crude"

if(dok.getKategori().equals(dok.getKategoriDikenali())){
                    BenarDikenali += 1;
                }

                // Dikenali sebagai "crude"
                if(dok.getKategoriDikenali().equals(kategori)){
                    Dikenali += 1;
                }

                // Sebenarnya "crude"
                if(dok.getKategori().equals(kategori)){
                    Sebenarnya += 1;
                }
            }
        }
        double Precision = 0.0;
        if(Dikenali > 0){
            Precision = (double)BenarDikenali/Dikenali;
        }
        double Recall = 0.0;
        if(Sebenarnya > 0){
            Recall = (double)BenarDikenali/Sebenarnya;
        }
        double FMeasure = 0.0;
        if((Precision+Recall) > 0){
```

```
            FMeasure = (2*Precision*Recall) / (Precision+Recall);
        }
        double [] Final = new double[3];
        Final [0] = Precision;
        Final [1] = Recall;
        Final [2] = FMeasure;
        return Final;
    }

    public void HitungFinalAkurasi(){
        double Precision = 0;
        double Recall = 0;
        double FMeasure = 0;
        String [] kategoris = new String[4];
        kategoris[0] = "crude";
        kategoris[1] = "interest";
        kategoris[2] = "moneyfx";
        kategoris[3] = "trade";
        int jumlahKategori = 0;
        for(String kat : kategoris){
            int jumlah = 0;
            for(Dokumen dok : PublicResource.DOCUMENTS_UJI){
                if(dok.getKategori().equals(kat)){
                    jumlah++;
                }
            }
            if(jumlah > 0){
                jumlahKategori++;
            }
        }
        for(String kat : kategoris){
            double [] hasil = HitungAkurasi(kat);
            Precision += hasil[0];
            Recall += hasil[1];
            FMeasure += hasil[2];
        }
        Precision = Precision / jumlahKategori;
        Recall = Recall / jumlahKategori;
        FMeasure = FMeasure / jumlahKategori;
```

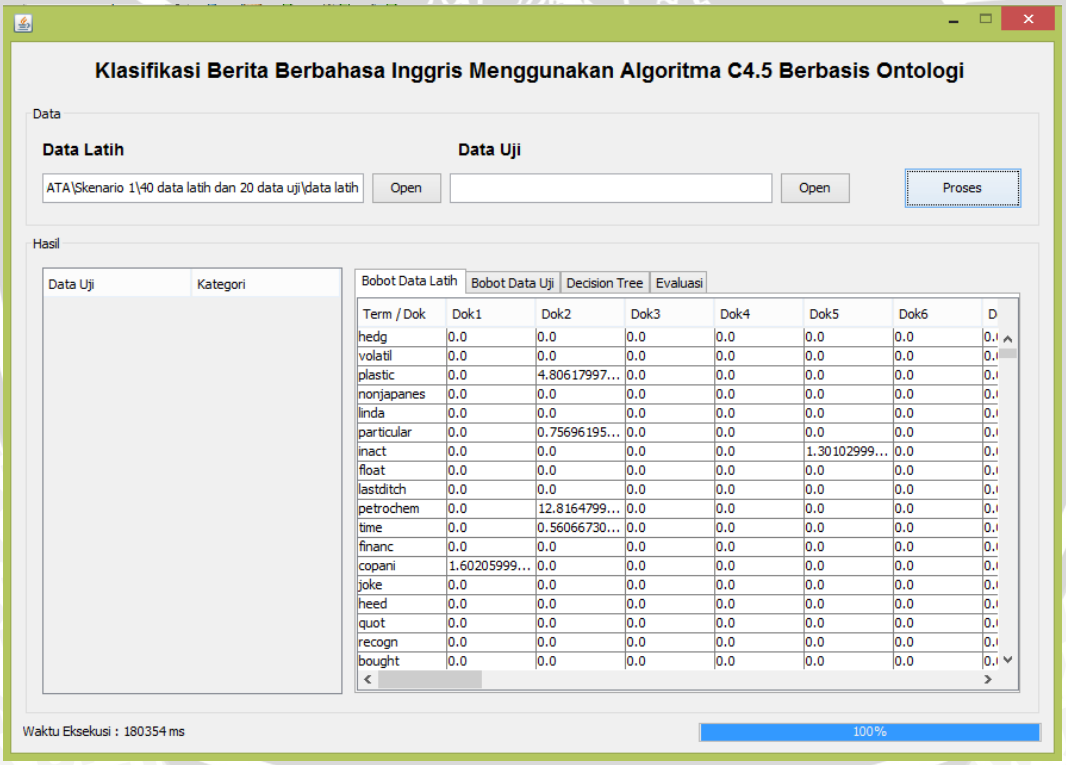
Source Code* 4.17 Proses Perhitungan *Precision, Recall,* dan *F-measure

4.3 Implementasi Antarmuka

Implementasi antarmuka diterapkan berdasarkan pada bab sebelumnya. Antarmuka sistem klasifikasi berita berbahasa inggris menggunakan algoritma C4.5 berbasis ontologi ini terdiri dari beberapa tombol dan *tab*. Tombol utama pada aplikasi ini adalah tombol proses. Tombol lainnya adalah tombol untuk

membuka direktori dokumen latih dan dokumen uji. Sedangkan *tab-tab* pada aplikasi ini terdiri dari pada *tab* bobot data latih, bobot data uji, *decision tree*, dan evaluasi.

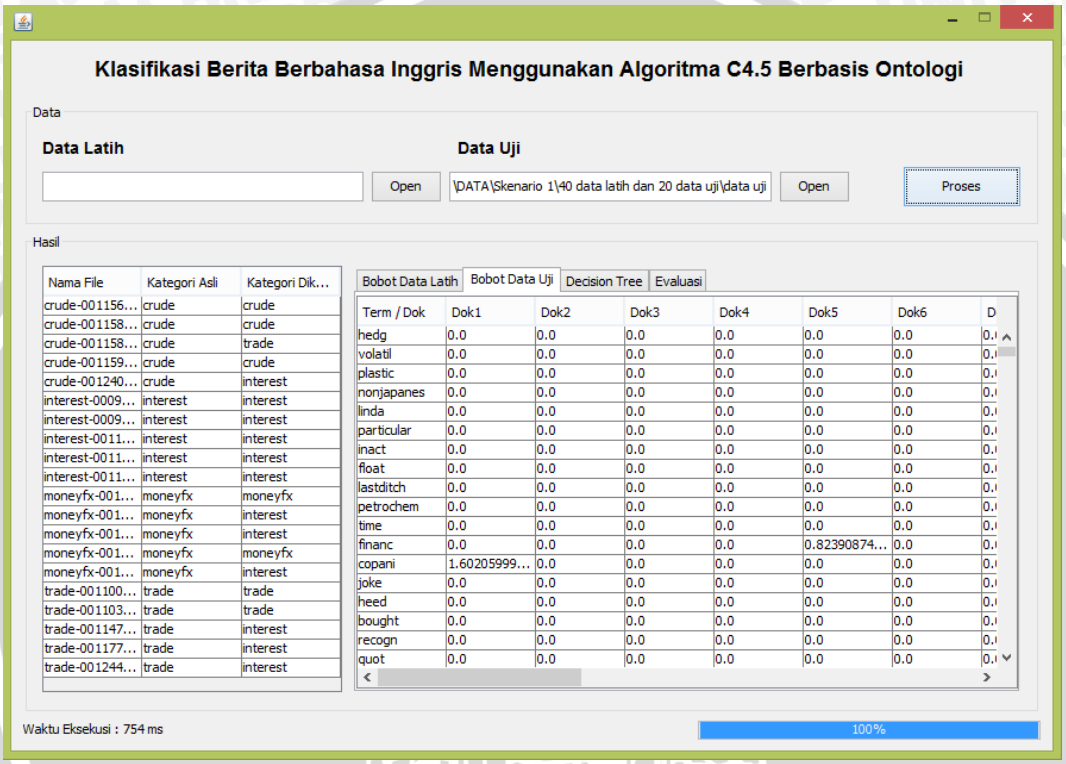
Pada langkah awal adalah mengklik tombol *open* untuk membuka direktori dokumen latih, dimana hanya dapat memilih direktori tidak dapat memilih satu *file* dokumen. Selanjutnya mengklik tombol proses, dimana ini merupakan tahap *preprocessing* sekaligus proses *training* (pembelajaran) pada data latih menggunakan algoritma C4.5. Hasil dari proses ini akan diperoleh nilai bobot data latih yang ditampilkan pada *tab* bobot data latih dan pembentukan *rule tree*. Gambar 4.1 menunjukkan antarmuka setelah proses *preprocessing* dan *training* (pembelajaran) data latih selesai dijalankan.



Gambar 4.1 Tampilan Antarmuka Ketika *Preprocessing* dan *Training* (Pembelajaran) Data Latih Selesai Dijalankan

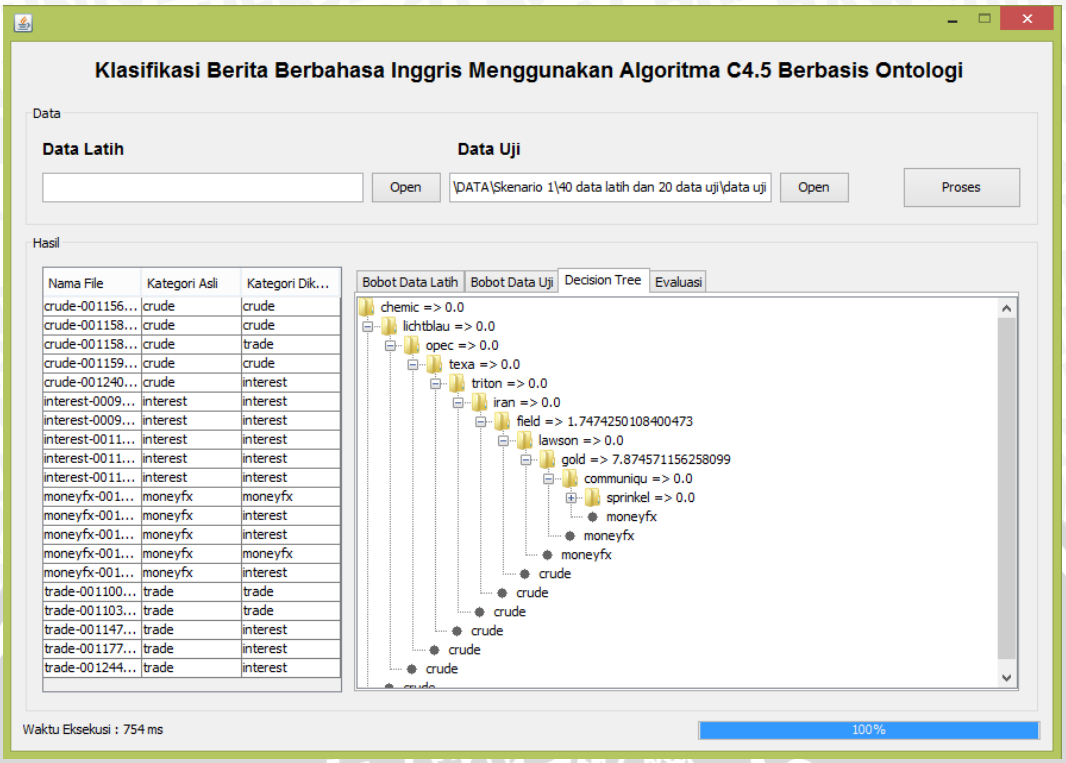
Setelah *preprocessing* dan *training* (pembelajaran) C4.5 pada data latih selesai dijalankan, selanjutnya adalah proses *preprocessing* dan pengkategorian data uji. Proses ini dilakukan dengan cara mengklik tombol *open* untuk membuka

direktori dokumen uji, dimana hanya dapat memilih direktori tidak dapat memilih satu *file* dokumen. Selanjutnya mengklik tombol proses. Hasil dari proses ini akan diperoleh nilai bobot data uji, pengkategorian dokumen uji dalam bentuk *decision tree*, dan hasil evaluasi sistem. Tampilan antarmuka *tab* bobot data uji dapat dilihat pada gambar 4.2.



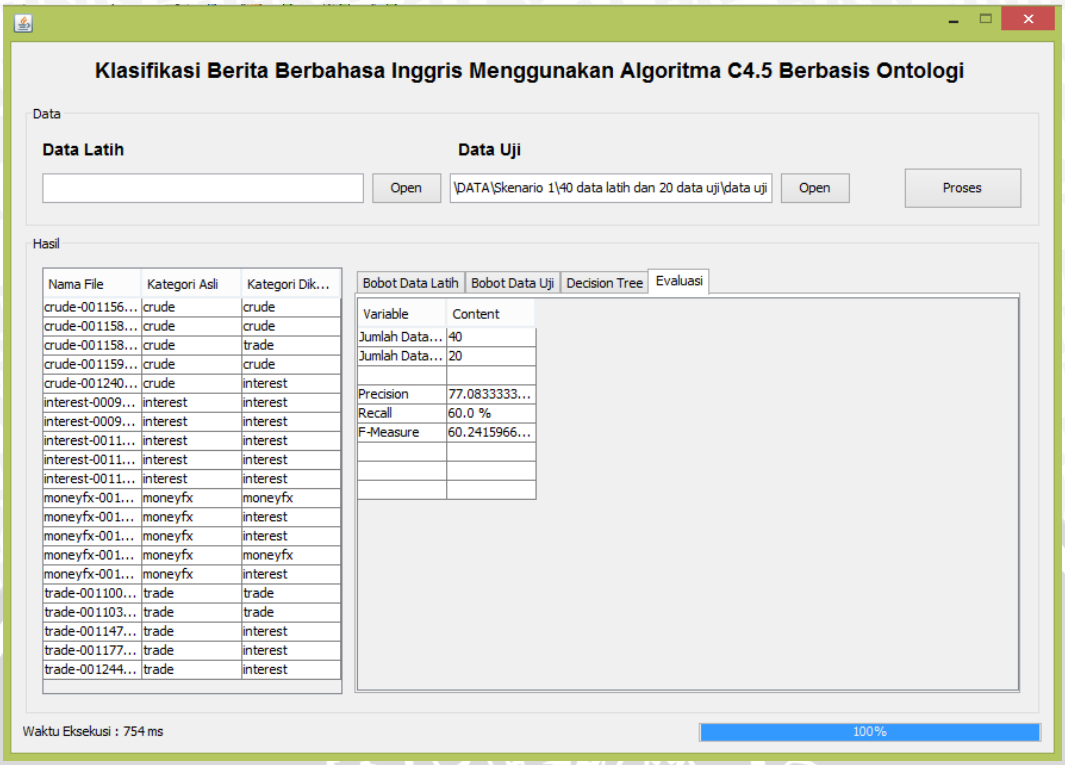
Gambar 4.2 Tampilan Antarmuka *Tab* Bobot Data Uji

Pengkategorian dokumen uji dalam bentuk *decision tree* diperoleh dari pembentukan *rule tree* yang dihasilkan pada proses *training* (pembelajaran) data latih. Tampilan antarmuka *tab decision tree* dapat dilihat pada gambar 4.3.



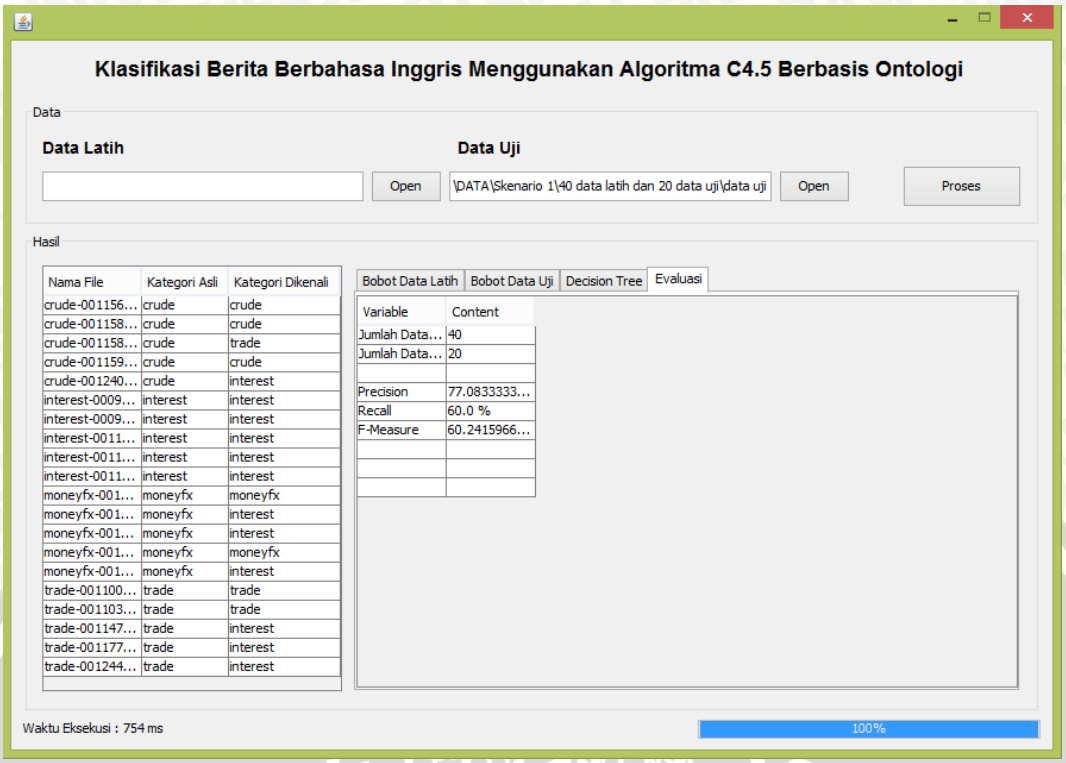
Gambar 4.3 Tampilan Antarmuka *Tab Decision Tree*

Proses terakhir yang dihasilkan yaitu evaluasi sistem yang akan ditampilkan pada *tab* evaluasi. Tab ini terdiri dari dua kolom, yaitu kolom parameter (*variable*) dan nilai (*content*). Kolom parameter (*variable*) terdiri dari jumlah dokumen latih dan jumlah dokumen uji yang diinputkan serta *precision*, *recall*, dan *f-measure* dari setiap kategori. Sedangkan kolom nilai (*content*) berupa angka dari parameter (*variable*) jumlah dokumen latih, jumlah dokumen uji, *precision*, *recall*, dan *f-measure*. Tampilan antarmuka dari *tab* evaluasi dapat dilihat pada gambar 4.4.



Gambar 4.4 Tampilan Antarmuka *Tab* Evaluasi

Kategori dari setiap dokumen uji hasil perhitungan sistem akan ditampilkan pada *field* yang berada di sebelah kiri, dimana terdapat kolom nama *file*, kategori asal, dan kategori dikenali. Tampilan antarmuka dari hasil pengkategorian dapat dilihat pada gambar 4.5.



Gambar 4.5 Tampilan Antarmuka Hasil Pengkategorian

BAB V
ANALISA HASIL DAN PEMBAHASAN

Bab ini berisi skenario pengujian beserta analisa hasil dari pengujian yang telah dilakukan. Bab ini terdiri dari sub-bab skenario pengujian, hasil pengujian, dan analisa hasil.

5.1 Skenario Pengujian

Pengujian pada sistem ini terbagi menjadi tiga pengujian. Pengujian pertama adalah jumlah dokumen uji dibuat tetap jumlahnya dalam setiap percobaan. Pengujian kedua adalah jumlah dokumen uji dibuat bervariasi dalam setiap percobaan. Pengujian ketiga adalah menggunakan *K-Fold* dimana dataset dibagi menjadi sejumlah K buah partisi secara acak, kemudian dilakukan sejumlah K kali eksperimen menggunakan data partisi ke-K sebagai data uji dan sisa partisi lainnya sebagai data latih. Pada setiap pengujian dilakukan penghitungan *precision, recall,* dan *f-measure.*

5.1.1 Skenario Dokumen Uji dengan Jumlah Tetap

Pengujian ini dilakukan sebanyak 6 kali dengan perbandingan data latih dan data uji adalah 2:1, 4:1, 6:1, 8:1, 10:1, dan 12:1. Jumlah data uji dibuat tetap jumlahnya dalam setiap percobaan dikarenakan untuk menguji apakah semakin banyak jumlah data latih yang digunakan akan mempengaruhi hasil evaluasi sistem. Jumlah data latih dan data uji yang digunakan pada skenario uji coba ini dapat dilihat pada tabel 5.1.

Tabel 5.1 Skenario Dokumen Uji dengan Jumlah Tetap

No	Perbandingan jumlah data		Jumlah data sebenarnya		Total Data
	Data latih	Data uji	Data latih	Data uji	
1	2	1	40	20	60
2	4	1	80	20	100
3	6	1	120	20	140
4	8	1	160	20	180

5	10	1	200	20	220
6	12	1	240	20	240

5.1.2 Skenario Dokumen Uji dengan Jumlah Bervariasi

Pengujian ini dilakukan sebanyak 6 kali dengan perbandingan prosentase data latih dan data uji adalah 30% : 70%, 40% : 60%, 50% : 50%, 60% : 40%, 70% : 30% dan 90% : 10%. Jumlah data uji dibuat bervariasi jumlahnya dalam setiap percobaan dikarenakan untuk menguji apakah mempengaruhi hasil evaluasi sistem. Jumlah data latih dan data uji yang digunakan pada skenario uji coba ini dapat dilihat pada tabel 5.2.

Tabel 5.2 Skenario Dokumen Uji dengan Jumlah Bervariasi

No	Perbandingan jumlah data		Jumlah data sebenarnya		Total Data
	Data latih	Data uji	Data latih	Data uji	
1	30%	70%	60	140	200
2	40%	60%	80	120	200
3	50%	50%	100	100	200
4	60%	40%	120	80	200
5	70%	30%	140	60	200
6	90%	10%	180	20	200

5.1.3 Skenario Menggunakan *K-Fold*

Pengujian untuk *K-Fold* dengan menggunakan tiga kombinasi, yaitu kombinasi pertama dan kedua menggunakan k = 3, dan kombinasi ketiga menggunakan k = 2. Pada kombinasi pertama data latih dibagi menjadi 3 *subset* (S_1, S_2, S_3) dimana pada tiap *subset* terdapat 40 dokumen latih yang berbeda. Pada kombinasi kedua data latih dibagi menjadi 3 *subset* (S_1, S_2, S_3) dimana pada tiap *subset* terdapat 80 dokumen latih yang berbeda. Pada kombinasi ketiga data latih dibagi menjadi 2 *subset* (S_1, S_2) dimana pada tiap *subset* terdapat 120 dokumen latih yang berbeda. Ketiga kombinasi tersebut menggunakan data uji

yang sama yaitu berjumlah 20 dokumen uji yang bukan termasuk dalam dokumen latih. Jumlah data latih dan data uji yang digunakan pada skenario uji coba ini dapat dilihat pada tabel 5.3.

Tabel 5.3 Skenario Menggunakan *K-Fold*

Kombinasi ke-	Uji Coba	Jumlah data		Total Data
		Data latih	Data uji	
1	1	S_1 = 40	S_4 = 20	60
	2	S_2 = 40	S_4 = 20	60
	3	S_3 = 40	S_4 = 20	60
2	1	S_1 = 80	S_4 = 20	100
	2	S_2 = 80	S_4 = 20	100
	3	S_3 = 80	S_4 = 20	100
3	1	S_1 = 120	S_3 = 20	140
	2	S_2 = 120	S_3 = 20	140

5.2 Hasil Pengujian

Pada setiap perbandingan data hanya dilakukan satu kali pengujian, karena hasil dari pengujian tidak menghasilkan bilangan random, artinya hasil dari setiap pengujian selalu menghasilkan nilai yang sama.

5.2.1 Hasil Pengujian Dokumen Uji dengan Jumlah Tetap

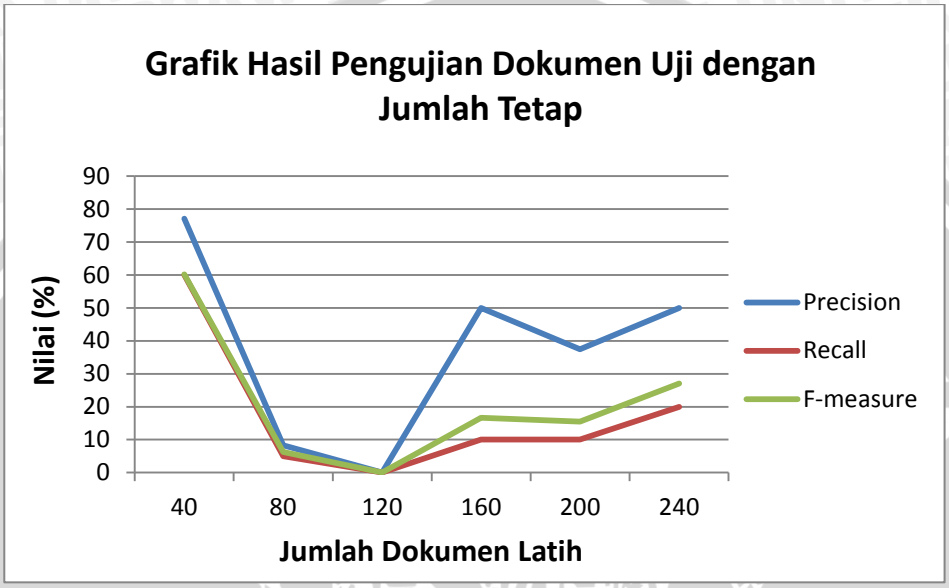
Pengujian ini dilakukan sebanyak 6 kali dengan jumlah data uji dibuat tetap. Hasil pengujian dokumen uji dengan jumlah tetap dapat dilihat pada tabel 5.4.

Tabel 5.4 Hasil Pengujian Dokumen Uji dengan Jumlah Tetap

Jumlah Data Latih	Jumlah Data Uji	Total Data	*Precision* (%)	*Recall* (%)	*F-measure* (%)	Waktu Eksekusi (ms)
40	20	60	77.08	60	60.24	280605
80	20	100	8.33	5	6.25	614184
120	20	140	0	0	0	1239529
160	20	180	50	10	16.67	2294092

200	20	220	37.5	10	15.48	2997165
240	20	260	50	20	27.08	3075109

Representasi grafik pada tabel 5.4 hasil pengujian dokumen uji dengan jumlah tetap dapat dilihat pada Gambar 5.1.



Gambar 5.1 Grafik Hasil Pengujian Dokumen Uji dengan Jumlah Tetap

Dari tabel 5.4 dan grafik pada gambar 5.1 dapat diketahui bahwa nilai *f-measure* tertinggi untuk pengujian dokumen uji dengan jumlah tetap adalah 60.24% ketika data latih berjumlah 40 dan data uji berjumlah 20.

5.2.2 Hasil Pengujian Dokumen Uji dengan Jumlah Bervariasi

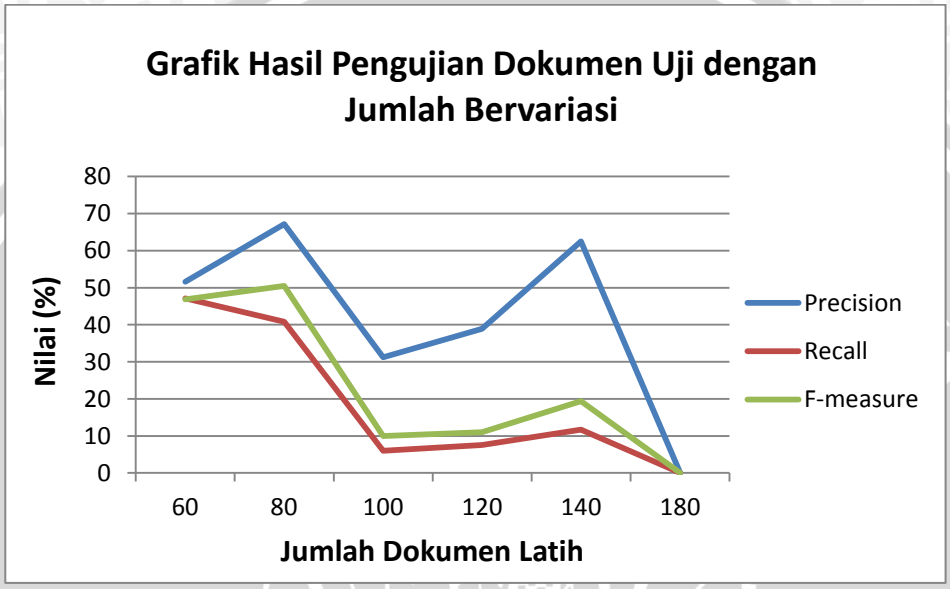
Pengujian ini dilakukan sebanyak 6 kali dengan jumlah data uji dibuat bervariasi. Hasil pengujian dokumen uji dengan jumlah bervariasi dapat dilihat pada tabel 5.5.

Tabel 5.5 Hasil Pengujian Dokumen Uji dengan Jumlah Bervariasi

Jumlah Data Latih	Jumlah Data Uji	Total Data	*Precision* (%)	*Recall* (%)	*F-measure* (%)	Waktu Eksekusi (ms)
60	140	200	51.62	47.14	46.89	445022
80	120	200	67.17	40.83	50.48	846952
100	100	200	31.25	6	9.96	1103002

120	80	200	38.89	7.5	11	878605
140	60	200	62.5	11.67	19.35	1354550
180	20	200	0	0	0	1686731

Representasi grafik pada tabel 5.5 hasil pengujian dokumen uji dengan jumlah bervariasi dapat dilihat pada Gambar 5.2.



Gambar 5.2 Grafik Hasil Pengujian Dokumen Uji dengan Jumlah Bervariasi

Dari tabel 5.5 dan grafik pada gambar 5.2 dapat diketahui bahwa nilai *f-measure* tertinggi untuk pengujian dokumen uji dengan jumlah bervariasi adalah 50.48% ketika data latih berjumlah 80 dan data uji berjumlah 120.

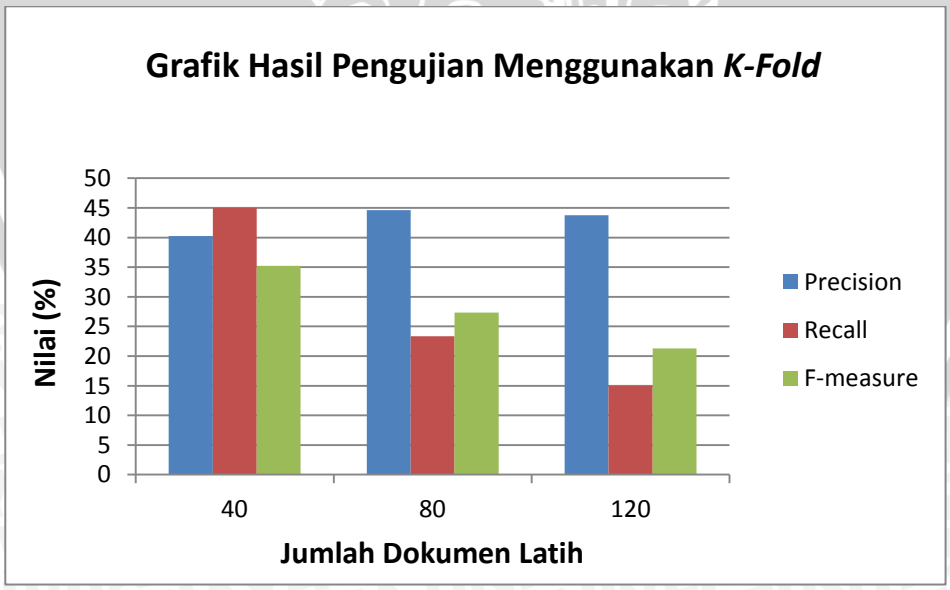
5.2.3 Hasil Pengujian Menggunakan *K-Fold*

Pengujian ini dengan menggunakan tiga kombinasi, yaitu kombinasi pertama dan kedua menggunakan k = 3, dan kombinasi ketiga menggunakan k = 2. Hasil pengujian menggunakan *K-Fold* dapat dilihat pada tabel 5.6.

Tabel 5.6 Hasil Pengujian Menggunakan *K-Fold*

Kombinasi ke-	Uji Coba	Jumlah data		*Precision* (%)	*Recall* (%)	*F-Measure* (%)	Waktu Eksekusi (ms)
		Data latih	Data uji				
1	1	S_1 = 40	S_4 = 20	33.33	50	37.5	602028
	2	S_2 = 40	S_4 = 20	54.61	55	48.17	906213
	3	S_3 = 40	S_4 = 20	32.81	30	20	558712
	Rata-rata :			40.25	45	35.22	
2	1	S_1 = 80	S_4 = 20	50.57	35	34.37	1427629
	2	S_2 = 80	S_4 = 20	33.33	15	20.53	1771452
	3	S_3 = 80	S_4 = 20	50	20	27.08	1799317
	Rata-rata			44.63	23.33	27.33	
3	1	S_1 = 120	S_3 = 20	37.5	10	15.48	2176891
	2	S_2 = 120	S_3 = 20	50	20	27.08	2281562
	Rata-rata :			43.75	15	21.28	

Representasi grafik pada tabel 5.6 hasil pengujian menggunakan *K-Fold* dapat dilihat pada Gambar 5.3.



Gambar 5.3 Grafik Hasil Pengujian Menggunakan *K-Fold*

Dari tabel 5.6 dan grafik pada gambar 5.3 dapat diketahui bahwa nilai rata-rata *f-measure* tertinggi adalah 35.22% terjadi ketika kombinasi ke-1 dengan jumlah dokumen latih 40 dan dokumen uji 20. Dari tiga kali uji coba tersebut, nilai *f-measure* tertinggi terjadi ketika uji coba ke-2 yaitu sebesar 48.17%.

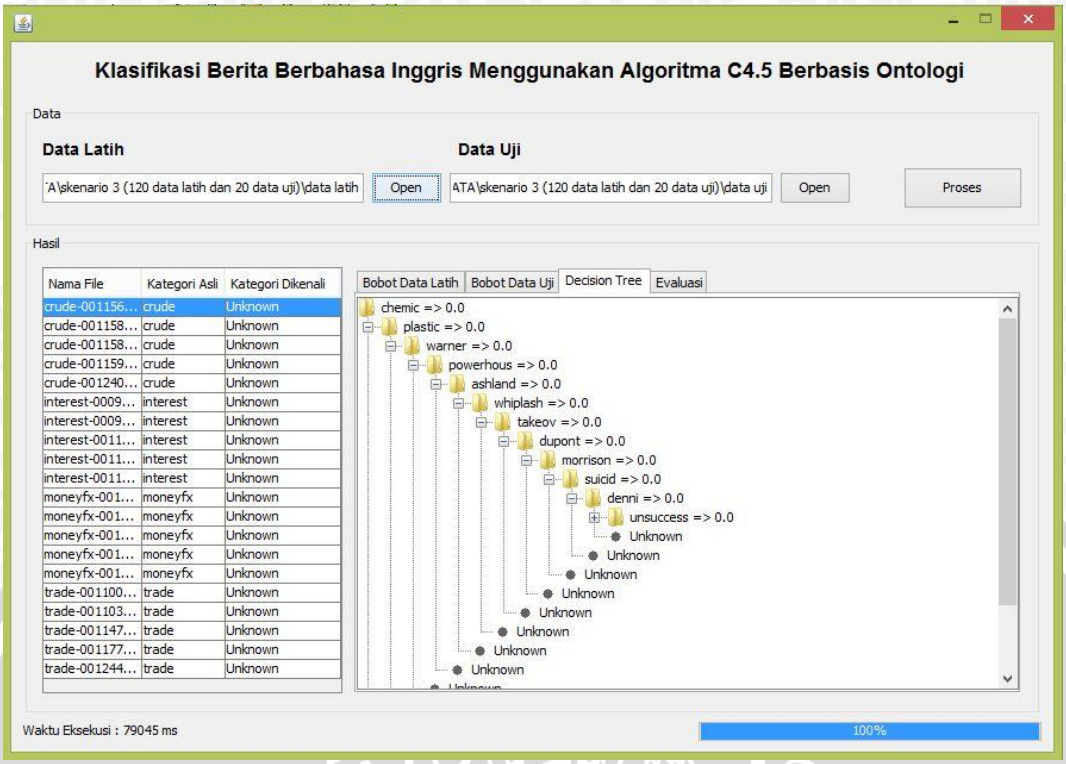
5.3 Analisa Hasil Pengujian

Analisis hasil pengujian dilakukan untuk menganalisa hasil pengujian jumlah dokumen uji dengan jumlah tetap dan bervariasi, serta hasil pengujian dengan menggunakan *K-Fold*. Analisis hasil pengujian perlu dilakukan untuk mengetahui faktor-faktor yang mempengaruhi hasil akurasi. Proses pengujian ada 3 macam, yaitu pengujian dokumen uji dengan jumlah tetap, pengujian dokumen uji dengan jumlah bervariasi, dan pengujian menggunakan *K-Fold*.

Pada pengujian menggunakan dokumen uji dengan jumlah tetap yang dilakukan pada 6 kali percobaan diperoleh hasil bahwa nilai *f-measure* tertinggi adalah 60.24% yaitu terjadi ketika data latih berjumlah 40 dan data uji berjumlah 20. Pada pengujian menggunakan dokumen uji dengan jumlah bervariasi yang dilakukan pada 6 kali percobaan diperoleh hasil bahwa nilai *f-measure* tertinggi adalah 50.48% yaitu terjadi ketika data latih berjumlah 80 dan data uji berjumlah 120. Sedangkan pada pengujian menggunakan *K-Fold* yang dilakukan dengan tiga kombinasi diperoleh hasil bahwa nilai *f-measure* tertinggi adalah 48.17% yaitu terjadi pada kombinasi ke-1 dan uji coba ke-2 ketika data latih berjumlah 40 dan data uji berjumlah 20. Jika dilihat dari ketiga pengujian tersebut, hasil *f-measure* tidak tergantung pada banyaknya jumlah dokumen latih dan dokumen uji, tetapi tergantung pada frekuensi *term* dokumen. Hal ini dapat dibuktikan ketika data latih berjumlah 120 dan data uji berjumlah 20 pada pengujian skenario 1 dan juga terjadi ketika data latih berjumlah 180 dan data uji berjumlah 20 pada pengujian skenario 2. Banyak *term* pada satu dokumen yang tidak dimiliki oleh dokumen lain, sehingga dokumen dengan nilai frekuensi *term* = 0 lebih banyak. Nilai frekuensi *term* berpengaruh pada nilai pembobotan *term*, dimana jika nilai frekuensi *term* = 0, nilai pembobotan *term* juga akan bernilai 0. Hal ini akan berpengaruh pada proses pembentukan *tree* yaitu pada saat proses *splitting*, dimana proses *splitting attribute* ini menggunakan nilai pembobotan *term* yang

telah di-*sorting* dan dihitung nilai mediannya untuk menentukan *split-point* yang tepat saat proses percabangan. Kasus ini terjadi misalnya nilai pembobotan *term* rata-rata sama pada setiap dokumen misal=0, maka setelah di-*sorting* juga akan menghasilkan nilai median yang sama dan *split-point* yang terpilih adalah 0, jadi terdapat cabang ≤ 0 dan > 0. Karena nilai pembobotan pada semua dokumen rata-rata sama bernilai 0, sehingga *tree* akan terus melakukan percabangan ke arah kiri yaitu pada cabang ≤ 0. Hal ini mengakibatkan pada cabang > 0 tidak terdapat dokumen yang diproses atau dokumen kosong. Pada kondisi ini *tree* pada cabang > 0 akan didefinisikan sebagai *unknown* atau tidak dikenali pada kategori manapun, sehingga jika terdapat data uji yang mengalami penelusuran ke arah cabang > 0 , akan didefinisikan sebagai *unknown*. Pada hasil pengujian ini, hal tersebut banyak terjadi, dokumen uji lebih banyak didefinisikan sebagai *unknown* karena permasalahan nilai pembobotan yang rata-rata sama, sehingga mengalami penurunan nilai evaluasi. Untuk pengujian menggunakan *K-Fold*, pada jumlah kombinasi data latih yang sama tetapi uji coba yang berbeda menghasilkan nilai *f-measure* yang berbeda karena setiap data latih pada setiap uji coba mempunyai dokumen latih yang berbeda-beda. Nilai *f-measure* tertinggi terjadi pada kombinasi ke-1 dan uji coba ke-2, artinya isi berita pada 20 dokumen uji banyak mempunyai kemiripan dengan isi berita pada 40 dokumen latih yang terdapat pada kombinasi ke-1 dan uji coba ke-2.

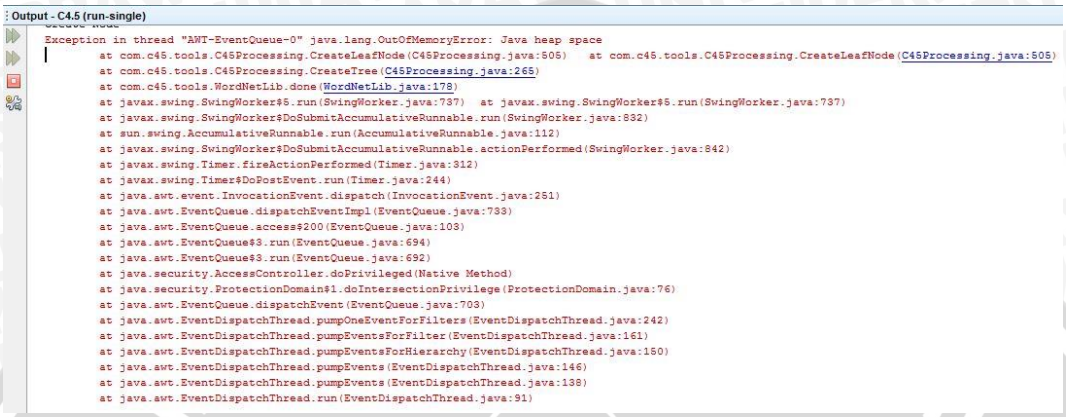
Penurunan nilai evaluasi disebabkan karena *domain* klasifikasi atau kelas-kelas yang digunakan pada penelitian ini berbeda jauh, artinya antara kategori *trade*, *crude*, *money-fx*, dan *interest* memiliki isi berita yang sangat berbeda, sehingga tiap kategori berita memiliki *term* unik yang tidak dimiliki kategori lain. Selain itu, penurunan nilai evaluasi juga dipengaruhi oleh hasil dari *preprocessing* yaitu proses *stemming* dimana terdapat *term* yang menghasilkan *root* kata yang tidak tepat sehingga tidak dapat dicari *synset term* tersebut pada *WordNet*. Gambar 5.4 menunjukkan pembentukan *tree* dimana semua cabang pada sisi kanan tidak dikenali atau *unknown*.



Gambar 5.4 Hasil Pembentukan *Tree* dengan Cabang Sisi Kanan *Unknown*

Jumlah dataset yang digunakan pada pengujian ini dibatasi maksimal 240 data latih dengan 60 dokumen untuk masing-masing kategori dikarenakan faktor *time consuming* dan *memory consuming*. *Time consuming* dapat ditunjukkan oleh waktu eksekusi pada tabel 5.4, tabel 5.5, dan tabel 5.6 dimana menunjukkan bahwa sistem membutuhkan waktu proses yang cukup lama dikarenakan proses pada saat *ontology extraction* dimana harus membandingkan setiap *term* untuk menemukan sinonim kata dan proses pada saat pembentukan *node tree* yang akan terus melakukan perulangan sebanyak *term* dokumen. Sedangkan *memory consuming* dikarenakan ketika dilakukan pengujian menggunakan 280 data latih dengan 70 dokumen untuk masing-masing kategori terjadi *out of memory*, yaitu permasalahan pada *java heap space*. Untuk menangani *out of memory* ini sudah dilakukan penambahan *java heap space* hingga batas maksimum yaitu 512MB, tetapi pembentukan *node tree* yang membutuhkan banyak memori untuk menyimpan puluhan sampai ratusan ribu *term* mengakibatkan penambahan pada *java heap space* tersebut tidak bisa mengatasi permasalahan *out of memory*. Selain itu hal ini juga bisa disebabkan spesifikasi *hardware* yang digunakan dimana

hardware yang digunakan dalam pembuatan sistem ini memiliki spesifikasi Processor Intel® Core™ i3 CPU M 330 @2.13GHZ dengan memory 2 GB. Gambar 5.5 menunjukkan hasil *out of memory* pada pengujian menggunakan 280 data latih.



Gambar 5.5 Hasil *Out of Memory* pada Pengujian 280 Data Latih

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil penelitian tentang pengklasifikasian berita berbahasa Inggris menggunakan algoritma C4.5 berbasis ontologi dapat disimpulkan bahwa:

1. Algoritma C4.5 berbasis ontologi dapat diterapkan pada proses pengklasifikasian berita berbahasa Inggris dimana dilakukan melalui beberapa tahap, yaitu *preprocessing* yang terdapat proses *ontology extraction* dan proses pembentukan *tree* yang dilakukan dengan menghitung nilai *information gain*. Hasil dari pembentukan *tree* adalah pembentukan *rule* dimana proses pengklasifikasian data uji akan dilakukan sesuai dengan *rule* yang terbentuk.
2. Hasil evaluasi sistem menunjukkan rata-rata *precision, recall,* dan *f-measure* tertinggi berturut-turut adalah 77.08%, 60%, dan 60.24% yang terjadi pada skenario 1 dengan jumlah data 40 untuk data latih dan 20 untuk data uji. Frekuensi *term* sangat berpengaruh terhadap hasil penelitian. Pada penelitian ini setiap data memiliki *term* unik yang tidak dimiliki data lain, sehingga frekuensi *term* hanya terpusat pada dokumen tertentu. Hal tersebut akan mempengaruhi hasil *splitting* pada proses pembentukan *tree* yang mengakibatkan banyak dokumen yang tidak dikenali atau *unknown* dikarenakan nilai pembobotan yang rata-rata sama.

6.2 Saran

Sistem yang telah dibangun pada penelitian ini masih memiliki kekurangan. Adapun beberapa saran untuk penelitian lebih lanjut adalah sebagai berikut:

1. Pengujian pada data latih yang terlalu banyak akan mengalami *out of memory* jika dijalankan pada *hardware* yang memiliki spesifikasi Processor Intel® Core™ i3 CPU M 330 @2.13GHZ dengan memory 2 GB, dikarenakan semakin banyak *term* yang diproses maka proses pembentukan *node tree* juga akan membutuhkan banyak memori. Sehingga untuk pengembangan selanjutnya jika memproses data latih yang banyak sebaiknya menggunakan sistem yang dijalankan pada *hardware* dengan spesifikasi yang lebih baik.

2. Sebaiknya dilakukan perbandingan dengan algoritma *text mining* yang lain sehingga dapat diperoleh hasil akurasi yang lebih baik.

DAFTAR PUSTAKA

[AGG-12] Aggarwal, Charu C. 2012. *A Survey of Text Classification Algorithms*. Chapter 6. University of Illinois. Urbana-Champaign.

[BAS-10] Basnur, P. W., dan Sensuse, D. I. 2010. *Pengklasifikasian Otomatis Berbasis Ontologi Untuk Artikel Berita Berbahasa Indonesia*, Makara, Teknologi, Vol. 14, No. 2.

[DES-09] Destuardi, I., dan Sumpeno, S. 2009. *Klasifikasi Emosi Untuk Teks Bahasa Indonesia Menggunakan Metode Naive Bayes*. Institut Teknologi Sepuluh Nopember. Surabaya.

[ELS-05] Elsayed, A.,dkk. 2005. *Applying Data Mining For Ontology Building.* Cairo University Giza. Egypt.

[FEL-07] Feldman, Ronen., dan James Sanger. 2007. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press. New York.

[FRA-10] Francis, Louise, et al. 2010. *Text Mining Handbook*. http://www.casact.org/pubs/forum/10spforum/Francis_Flynn.pdf. Diakses tanggal 11 Maret 2014.

[GUO-04] Guo, G., dkk. 2004. *An KNN Model-based Approach and Its Application in Text Categorization.* University of Ulster Newtownabbey.

[HAN-06] Han, Jiawei dan Kamber, Micheline. 2006. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers is an imprint of Elsevier. San Francisco.

[HOT-03] Hotho, A., Staab, S., dan Stumme, G. 2003. *Wordnet improves Text Document Clustering*. University of Karlsruhe. Germany.

[KAN-03] Kantardzic, M. 2003. *Data Mining, Concepts, Models, Methods, and Algorithms*. IEEE Press.

[KUS-09] Kusrini, dkk. 2009. *Perbandingan Metode Nearest Neighbor dan Algoritma C4.5 Untuk Menganalisis Kemungkinan Pengunduran*

Diri Calon Mahasiswa di STMIK AMIKOM Yogyakarta. STMIK AMIKOM Yogyakarta.

[LAN-10] Langgeni, D., Baisal, A., & Firdaus, Y. 2010. *Clustering Artikel Berita Berbahasa Indonesia dengan Menggunakan Unsupervised Feature Selection*. Seminar Nasional Informatika.

[LAT-12] Latifah, R. 2012. *Penerapan Neighbor-Weighted K-Nearest Neighbor Untuk Klasifikasi Corpus Teks Berbahasa Indonesia Yang Tidak Seimbang*. Universitas Brawijaya.

[MAN-09] Manning, C. D., Raghavan, P., dan Schütze, H. 2009. *An Introduction to Information Retrieval.* Cambridge University Press, Cambridge. England.

[MIL-95] Miller, G. A. 1995. *WordNet: A Lexical Database for English*, Communication of the Acm, Vol. 38, No. 11.

[MOO-05] Mooney, R.J. 2005. *Text Mining with Information Extraction.* University of Texas. Austin.

[MUR-13] Murdianto, A.S.A. 2013. *Klasifikasi Berita Berbahasa Inggris Menggunakan Algoritma K-Nearest Neighbor (KNN) Berbasis Ontologi*. Universitas Brawijaya. Malang.

[POR-80] Porter, M. F. 1980. *An Algorithm for Suffix Stripping*. Cambrigde.

[POR-05] Porter, M. F. 2005. *The English (Porter2) Stemming Algorithm*. Diakses tanggal 11 Maret 2014.

[SAL-12] Saleh, A., dan Rizki, M. 2012. *Ekspresi Wajah 3D Berdasarkan Klasifikasi Teks Menggunakan Aplikasi Rainbow.* Politeknik Elektronika Negeri Surabaya. Surabaya.

[SAN-07] Santosa, Budi. 2007. *Teknik Pemanfaatan Data Untuk Keperluan Bisnis.* Graha Ilmu. Yogyakarta.

[SEB-02] Sebastiani, F. 2002. *Text Categorization*. Italy.

[SIL-09] Silvestro, L. D., Daria, S., dan Alessandro, T. 2009. *Automatic Classification of Legal Textual Documents using C4.5*. Istituto di Teoria e Tecniche dell'Informazione Giuridica Consiglio Nazionale delle Ricerche. Catania.

[SOU-03] Soucy, P., dan Mineau, G. W. 2003. *Beyond TFIDF Weighting for Text Categorization in the Vector Space Model*. Canada.

[SUM-05] Sumadiria, A. H. 2005. *Jurnalistik Indonesia: Menulis Berita dan Feature*. PT. Remaja Rosdakarya. Bandung.

[SUM-06] Sumathi, S & S.N. Sivanandam. 2006. *Introduction to Data Mining and its Applications*. Springer Science and Business Media Inc. New York.

[SWA-13] Swamy, M. N., dan M. Hanumanthappa. 2013. *Indian Language Text Representation and Categorization Using Supervised Learning Algorithm*. International Journal of Data Mining Techniques and Applications Vol. 02, Pages. 251-257. India.

[TRI-09] Triawati, Candra. 2009. *Pemodelan Berbasis Konsep Untuk Kategorisasi Artikel Berita Berbahasa Indonesia*. Seminar Nasional Aplikasi Teknologi Informasi (SNATI). Yogyakarta.

[XIA-09] Xiao-fei, Z., dan Ke-liang, Z. 2009. *KNN Text Categorization Algorithm Based on Semantic Centre*. International Conference on Information Technology and Computer Science. China.

[XIO-09] Xiaoliang, Z., dan Wang, J. 2009. *Research and Application of the improved Algorithm C4.5 on Decision Tree*. International Conference on Test Measurement. China.

[YAN-02] Yang, Y., dan Pedersen, J.O. 2002. *A Comparative Study on Feature Selection in Text Categorization*. USA.

LAMPIRAN

Lampiran 1 Daftar *Stopwords*

able	beginnings	even	him	mainly	off	readily
about	begins	ever	himself	make	often	really
above	behind	every	his	makes	oh	recent
abst	being	everybody	hither	many	ok	recently
accordance	believe	everyone	home	may	okay	ref
according	below	everything	how	maybe	old	refs
accordingly	beside	everywhere	howbeit	me	omitted	regarding
across	besides	ex	however	mean	on	regardless
act	between	except	hundred	means	once	regards
actually	beyond	far	id	meantime	one	related
added	biol	few	ie	meanwhile	ones	relatively
adj	both	ff	if	merely	only	research
adopted	brief	fifth	i'll	mg	onto	respectively
affected	briefly	first	im	might	or	resulted
affecting	but	five	immediate	million	ord	resulting
affects	by	fix	immediately	miss	other	results
after	ca	followed	importance	ml	others	right
afterwards	came	following	important	more	otherwise	run
again	can	follows	in	moreover	ought	said
against	cannot	for	inc	most	our	same
ah	can't	former	indeed	mostly	ours	saw
all	cause	formerly	index	mr	ourselves	say
almost	causes	forth	information	mrs	out	saying
alone	certain	found	instead	much	outside	says
along	certainly	four	into	mug	over	sec
already	co	from	invention	must	overall	section
also	com	further	inward	my	owing	see
although	come	furthermore	is	myself	own	seeing
always	comes	gave	isn't	na	page	seem
am	contain	get	it	name	pages	seemed
among	containing	gets	itd	namely	part	seeming
amongst	contains	getting	it'll	nay	particular	seems
an	could	give	its	nd	particularly	seen
and	couldnt	given	itself	near	past	self
announce	date	gives	i've	nearly	per	selves
another	did	giving	just	necessarily	perhaps	sent
any	didn't	go	keep	necessary	placed	seven
anybody	different	goes	keeps	need	please	several
anyhow	do	gone	kept	needs	plus	shall
anymore	does	got	keys	neither	poorly	she
anyone	doesn't	gotten	kg	never	possible	shed

anything	doing	had	km	nevertheless	possibly	she'll
anyway	done	happens	know	new	potentially	shes
anyways	don't	hardly	known	next	pp	should
anywhere	down	has	knows	nine	predomina	shouldn't
apparently	downwards	hasn't	largely	ninety	ntly	show
approximately	due	have	last	no	present	showed
are	during	haven't	lately	nobody	previously	shown
aren	e	having	later	non	primarily	showns
arent	each	he	latter	none	probably	shows
arise	ed	hed	latterly	nonetheless	promptly	significant
around	edu	hence	least	noone	proud	significantly
as	effect	her	less	nor	provides	similar
aside	eg	here	lest	normally	put	similarly
ask	eight	hereafter	let	nos	que	since
asking	eighty	hereby	lets	not	quickly	six
at	either	herein	like	noted	quite	slightly
auth	else	heres	liked	nothing	qv	so
available	elsewhere	hereupon	likely	now	ran	some
away	end	hers	line	nowhere	rather	somebody
awfully	ending	herself	little	obtain	rd	somehow
	enough	hes	'll	obtained	re	someone
	especially	hi	look	obviously		
	et	hid	looking	of		
	et-al		looks			
	etc		ltd			
			m			
			mad			

somethan	take	therefore	thru	unlikely	whereas	within
something	taken	therein	thus	until	whereby	without
sometime	taking	there'll	til	unto	wherein	won't
sometimes	tell	thereof	tip	want	wheres	words
somewhat	tends	therere	to	wants	whereupon	world
somewhere	th	theres	together	was	wherever	would
soon	than	thereto	too	wasn't	whether	wouldn't
sorry	thank	thereupon	took	way	which	www
specifically	thanks	there've	toward	we	while	yes
specified	thanx	these	towards	wed	whim	yet
specify	that	they	tried	welcome	whither	you
specifying	that'll	theyd	tries	we'll	who	youd
state	thats	they'll	truly	went	whod	you'll
states	that've	theyre	try	were	whoever	your
still	the	they've	trying	weren't	whole	youre
stop	their	think	ts	we've	who'll	yours
strongly	theirs	this	twice	what	whom	yourself
sub	them	those	two	whatever	whomever	yourselves
substantially	themselves	thou	u	what'll	whos	you've
successfully	then	though	un	whats	whose	z
such	thence	thoughh	under	when	why	zero
sufficiently	there	thousand	unfortunatel	whence	widely	
suggest	thereafter	throug	y	whenever	willing	
sup	thereby	through	unless	where	wish	
sure	thered	throughout	unlike	whereafter	with	