UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

**Hak cipta dan penggunaan kembali:**

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

**Copyright and reuse:**

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

# BAB II

# LANDASAN TEORI

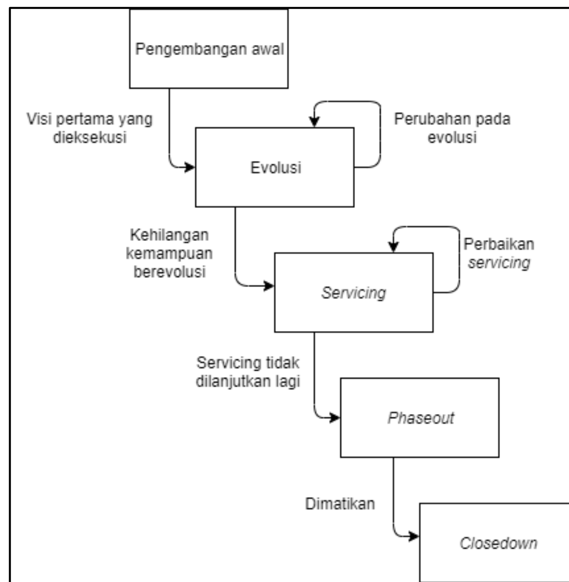
## 2.1 Rekayasa Kebutuhan

*Requirements engineering* (rekayasa kebutuhan) adalah proses mengumpulkan, menganalisa, mendokumentasikan, dan mengelola kebutuhan apa saja yang dibutuhkan untuk keperluan pengembangan piranti lunak (Aurum, 2011). *Requirements engineering* selalu berkaitan dengan menafsirkan dan memahami tujuan, kebutuhan, bahkan apa yang dipercayai oleh *stakeholder* (Aurum, 2011). Menurut Sommerville (2016), dalam rekayasa kebutuhan, ada beberapa hal yang perlu dilakukan yaitu studi fisibilitas, menemukan kebutuhan (pengumpulan dan analisa), mengubah kebutuhan ke dalam bentuk standar (spesifikasi), dan memastikan apakah kebutuhan sesuai dengan yang user butuhkan (validasi). Namun pada kenyataannya, rekayasa kebutuhan adalah proses berulang, aktifitas tersebut saling disisipkan (*interleaved*) (Sommerville, 2016).

Menurut Laplante (2017) dalam proses pengumpulan kebutuhan, *client* dilibatkan untuk menentukan ruang lingkup aplikasi, *services* apa saja yang tersedia di sistem, dan batasan operasional dari sistem. Pengumpulan kebutuhan mungkin dapat melibatkan pengguna, manajer, perekayasa (*engineer*), yang terlibat dalam *maintenance* piranti lunak, dan sebagainya, orang-orang tersebut sering disebut sebagai *stakeholders* (Laplante, 2017). Di semua sistem, kebutuhan dapat berubah, orang-orang yang terlibat, mengembangkan pemahaman yang lebih baik tentang

apa yang mereka inginkan dalam piranti lunak yang dibuat, seperti perubahan pada perangkat keras, perangkat lunak, dan lingkungan sistem (Sommerville, 2016). Proses memahami dan mengontrol perubahan terhadap kebutuhan sistem disebut juga sebagai manajemen kebutuhan (Sommerville, 2016). Selain itu, umpan balik pengguna juga bisa menyebabkan perubahan pada kebutuhan (Godfrey dan German, 2008; Morales-Ramirez, 2013).

Sommerville (2016) menyatakan bahwa pengembangan piranti lunak tidak berhenti setelah sistem telah dirilis, melainkan tetap berlanjut sepanjang masa hidup dari sistem tersebut. Masa hidup piranti lunak terdiri dari 5 tahapan yaitu pengembangan awal (*initial development*), perekayasa perangkat lunak membangun versi pertama dari sistem yang berfungsi, kemudian evolusi (*evolution*), kemampuan dan fungsionalitas sistem diekspansi untuk memenuhi kebutuhan pengguna, kemudian *servicing*, sistem diperbaiki dari kerusahan minor dan perubahan sederhana pada fungsionalitas, kemudian *phaseout*, pemilik sistem memutuskan untuk tidak melakukan *servicing* dan menghasilkan pemasukan dari sistem selama mungkin, dan yang terakhir adalah *closedown*, pemilik menarik sistem dari pasar dan mengarahkan pengguna untuk menggunakan sistem yang baru jika telah dibuat (Rajlich dan Bennet, 2000).



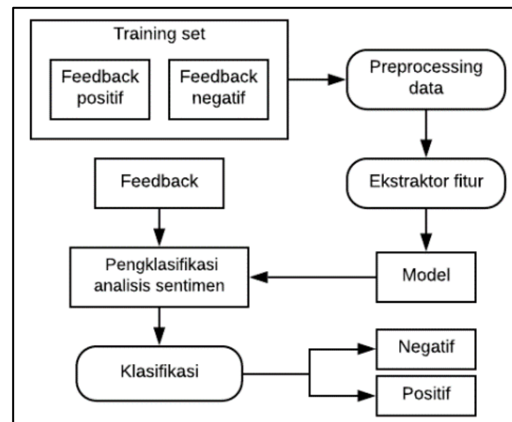
Gambar 2.1 Gambar Masa Hidup Piranti Lunak (Rajlich dan Bennet, 2000)

## 2.2 User Feedback

*User feedback* atau umpan balik penguna adalah informasi bermakna yang diberikan oleh pengguna dengan tujuan memberikan saran dan kritik untuk peningkatan kualitas piranti lunak (Morales-Ramirez, 2013). Berdasarkan tujuannya, ada beberapa alasan diberikannya suatu *feedback* yaitu bisa menjadi *corrective* atau negatif, *encouraging* atau positif, mempromosikan perilaku strategis, ataupun menyediakan klarifikasi tambahan (Morales-Ramirez, 2013). Jenis *feedback* negatif adalah umpan balik yang mencoba menetralisir gangguan, sedangkan umpan balik *corrective* adalah umpan balik yang menyediakan informasi mengenai seberapa bagus suatu tugas yang dikerjakan (Hattie dan Timperley, 2007; Brun dkk., 2009). Kebalikan dari jenis *feedback* ini adalah *encouraging* atau positif. Jenis *feedback* yang mempromosikan perilaku strategis adalah umpan balik yang membantu dalam memberikan pilihan lain untuk mencapai proses tertentu, sedangkan jenis umpan balik yang menyediakan

klarifikasi tambahan adalah umpan balik yang mengandung informasi tambahan seperti rincian penting yang membuat tujuan lebih jelas (Mory, 2004).

## 2.3 Analisa Sentimen




Gambar 2.2 Gambar Struktur Analisis Sentimen (Putri Wiratama, 2019)

Analisa sentimen adalah proses otomatisasi mining dan klasifikasi pendapat, pandangan, emosi dan sentimen dataset teks yang tidak terstruktur dengan bahasa mesin dan pemrograman komputer (Fiarni dkk., 2016). Dalam analisa sentimen, teks dikelompokkan ke dalam kategori seperti "positif" atau "negatif". Secara umum, struktur analisis sentimen *user feedback* dapat dilihat pada Gambar 2.2.

Pada penelitian ini, label yang digunakan adalah label "positif", "negatif", dan "netral". Proses yang dilakukan untuk analisis sentimen sama seperti pada Gambar 2.2. Hanya saja, jumlah label yang akan diprediksi bertambah satu, yaitu label "netral".

## 2.4 Word Embedding

Menurut Ahmad H. Abdullah (2018), komputer bisa mempelajari suatu karakter dari data melalui *feature extraction*. Beragam jenis *feature* diambil dari

dataset, lalu komputer mempelajari *feature* tersebut. Dalam penelitian ini, *feature* yang akan diekstraksi adalah *word similarity* atau kemiripan makna pada suatu kata.

Satu masalah yang dihadapi, yaitu komputer hanya bisa mengolah data berupa angka. Apabila objek yang diterima berupa teks, maka diperlukan proses ekstraksi teks menjadi sebuah vektor numerik yang mempresentasikan tiap katanya. Oleh karena itu, untuk mencari *text similarity* diperlukan metode *word embedding* yang sederhananya merupakan proses konversi sebuah teks menjadi angka.

Contoh sederhana mengubah kata menjadi vektor angka, misalnya diberikan sebuah kalimat *Word Embedding are Word Converted into numbers*. Sebuah kamus akan berisi list seluruh kata yang unique, sehingga kamus yang terbentuk adalah [*Word, Embeddings, are, Converted, into, number*]. Menggunakan metode one-hot encoding akan menghasilkan vektor dimana 1 merepresentasikan posisi kata tersebut berada, dan 0 untuk kata lainnya. Vektor representasi dari kata numbers mengacu pada format kamus diatas adalah [0,0,0,0,1] dan kata *Converted* adalah [0,0,0,1,0,0].

Metode *word embedding* mempelajari representasi vektor dari kosakata yang konstan yang berasal dari kumpulan teks. Teknik pembelajaran dari metode ini adalah menggunakan *neural network* model untuk suatu task seperti klasifikasi dokumen, atau dengan *unsupervised learning* menggunakan statistik dokumen. Menurut Marwa Naili (2017), secara umum ada 3 model yang sering digunakan untuk melakukan *word embedding*, yaitu Latent Semantic Analysis (LSA), Word2Vec, dan GloVe.

## 2.5 FastText

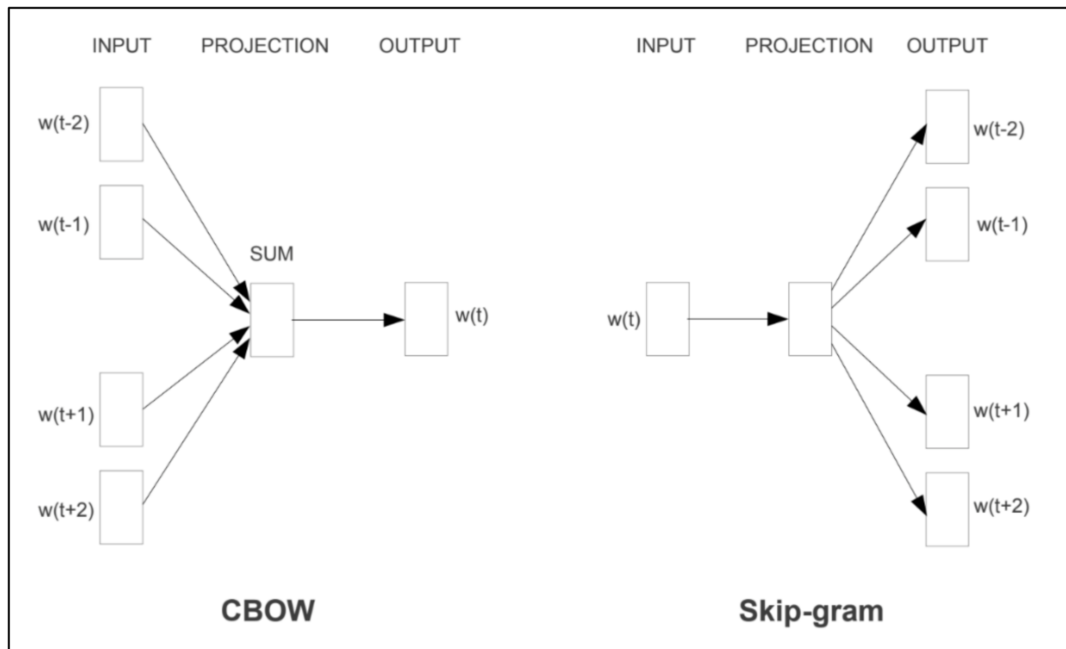
FastText adalah *library* yang dikeluarkan oleh Facebook yang dapat digunakan untuk *word embedding* (Rehurek, 2019). FastText sebenarnya merupakan pengembangan lebih lanjut dari Word2Vec yang dikembangkan oleh Google. FastText dan Word2Vec sama-sama bisa digunakan untuk mendeteksi *word similarity* secara semantik ataupun sintaksis.

Menurut Kamus Besar Bahasa Indonesia (KBBI), semantik berarti bagian struktur bahasa yang berhubungan dengan makna ungkapan atau struktur makna suatu wicara. Jadi, apabila ada *word similarity* secara semantik berarti ada kemiripan atau kedekatan dari segi makna. Sementara, kata sintaksis berasal dari kata *syntax* yang diadopsi dari bahasa Inggris yang memiliki arti yaitu struktur atau penulisan. Dengan kata lain, apabila ada *word similarity* secara sintaksis maka terdapat kemiripan atau kedekatan dari segi struktur penulisan.

*Input* kata akan direpresentasikan dalam bentuk vektor dan ditempatkan sedemikian rupa sehingga kata-kata yang bermakna sama akan muncul bersamaan, sedangkan yang berbeda akan terletak jauh dari vektor. Perbedaan paling mendasar antara FastText dengan Word2Vec adalah FastText mampu memproses *input* kata yang tidak terdapat dalam *vocabulary (Out of Vocabulary).*

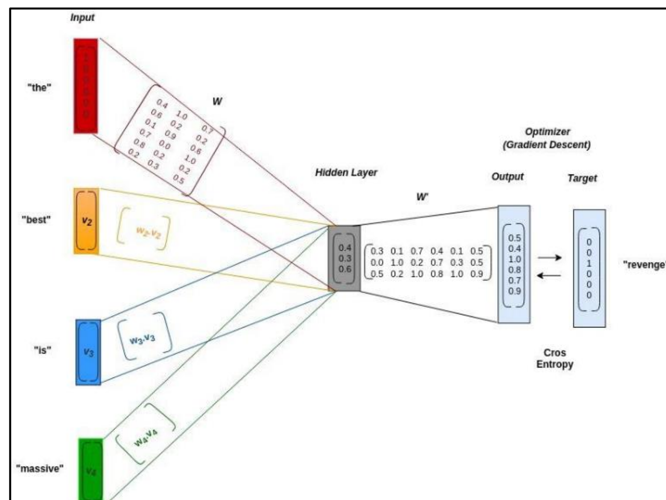
Mirip dengan model Word2Vec, terdapat dua jenis arsitektur pada FastText, yaitu Continuous Bag of Words (CBOW) dan Skip-Gram. CBOW memprediksi *current word* (sebagai target) dari konteks (sebagai input) di sekitarnya. Sedangkan, Skip-Gram merupakan kebalikan dari CBOW di mana *current word* (sebagai input)

digunakan untuk memprediksi konteks (sebagai target). Pada Gambar 2.3, terdapat visualisasi dari arsitektur CBOW dan Skip-Gram.



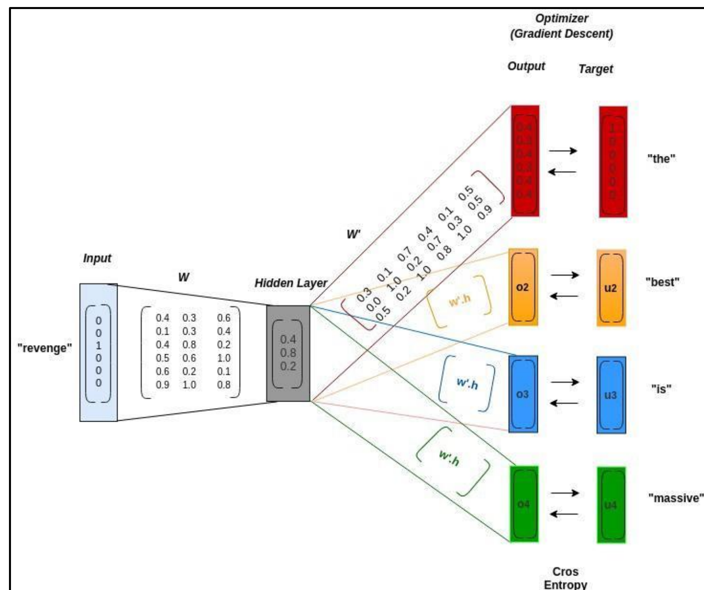

Gambar 2.3 Gambar Arsitektur CBOW dan Skip-gram (Mikolov, 2013)

Pada Gambar 2.4, diberikan contoh *input* kalimat "*the best revenge is massive success*" dan dilakukan foward-backward training dengan arsitektur CBOW. Diasumsikan w(t-2) = "*the*", w(t-1) = "best", w(t+1) = "is", w(t+2) = "*massive*" sebagai input dan w(t) = "*revenge*" sebagai target.




Gambar 2.4 Ilustrasi foward-backward training CBOW (Mikolov, 2013)

Pada Gambar 2.5, terdapat visualisasi *foward-backward training* Skip-Gram dengan input yang sama seperti *foward-backward* CBOW sebelumnya. Diasumsikan w(t) = "*revenge*" sebagai input, dan w(t-2) = "the", w(t-1) = "*best*", w(t+1) = "is", w(t+2) = "*massive*" sebagai target.



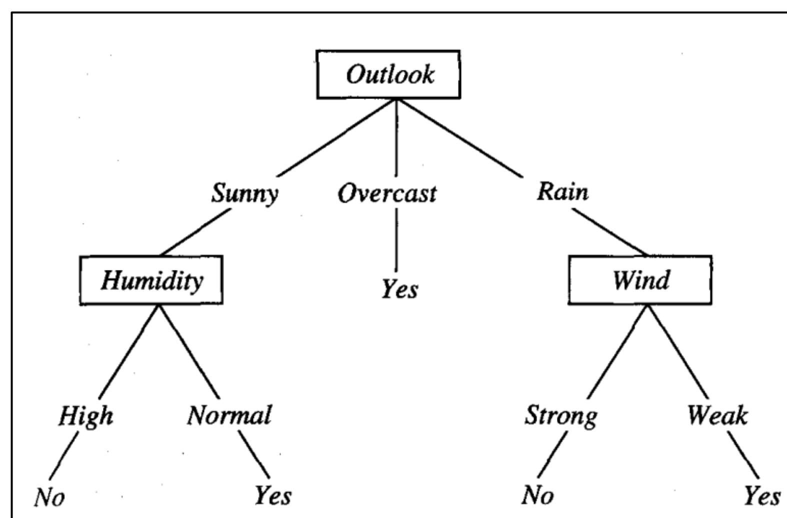

Gambar 2.5 Ilustrasi foward-backward training Skip-Gram (Mikolov, 2013)

Menurut Rehurek (2019), diperlukan *dataset* tersendiri untuk memperoleh hasil *dictionary* yang diinginkan. Rehurek (2019) mengistilahkan *dataset* yang digunakan untuk proses *training* menggunakan FastText sebagai *embedding dataset*. *Embedding dataset* yang telah di-*training* menggunakan FastText akan menghasilkan *vocabolary* yang memuat kosa kata yang bisa digunakan untuk mendeteksi *word similarity*. Hasil *training* yang dilakukan ini juga dikenal dengan istilah *pre-trained model*.

## 2.6 Decision Tree

Pohon keputusan adalah teknik *machine learning* untuk mempelajari dan mengambil keputusan dengan pendekatan fungsi target bernilai diskrit (Mitchell, 1997). Teknik ini dapat direpresentasikan dengan sekumpulan *if-then rules* supaya muda dipahami oleh manusia. Setiap pohon terdiri dari *leaf* (*nodes*) dan cabang. Setiap *leaf* mewakili atribut dalam kelompok yang akan diklasifikasikan dan setiap cabang mewakili nilai yang dapat diambil daun.

Mitchell (1997) mengatakan bahwa *decision tree* akan mengelompokan data dengan cara mengurutkan dari akar sampai ke beberapa simpul daun, yang menyediakan klasifikasi data. Setiap node dalam *tree* menentukan tes beberapa atribut data, dan setiap cabang menurun dari daun tersebut berhubungan dengan salah satu nilai yang mungkin untuk atribut ini. Data mulai diklasifikasikan mulai dari node yang berada di *root node*, melakukan tes pada atribut data di node tersebut dan melanjutkannya ke cabang yang lain. Proses ini dilakukan terus sampai *subtree* telah mencapai akhir pohon (*leaf node*) seperti yang terdapat pada Gambar 2.6.




Gambar 2.6 Contoh konsep Decision Tree (Mitchell, 1997)

Menurut Mitchell (1997) *decision tree* dapat dibagi menjadi tiga bagian berdasarkan Gambar 2.6.

1. Root Node

   Node ini merupakan node yang terletak paling atas dari suatu tree dan tidak memiliki cabang yang masuk ke dalamnya. Contohnya Node '*Outlook'*.

2. Internal Node

   Node ini merupakan node yang terdapat percabangan, hanya terdapat satu masukan dan memiliki minimal dua o*utput*. Contohnya Node '*Humidity*'.

3. Leaf Node

   Node ini adalah node akhir dan hanya memiliki satu masukan, serta tidak ada lagi output dalam node ini. Contohnya Node '*Yes*' dan '*No'*.

Han (2012) mengatakan bahwa pemilihan atribut dapat dilakukan dengan proses *Gini Index*. *Gini Index* adalah metrik yang mengukur seberapa sering kesalahan prediksi terhadap elemen yang dipilih dalam *tree* (Han, 2012). Dalam memilih atribut untuk memecahkan objek harus dipilih atribut yang menghasilkan *Gini Index* paling kecil. Atribut yang memiliki *Gini Index* terkecil akan dipilih sebagai *root* atau *internal node*. Han (2012) memberikan penjelasan persamaan yang dipakai untuk menemukan *Gini Index* dan *Gini Gain* seperti Persamaan 2.1, 2.2, dan 2.3.

$$Gini\ (D) = 1 - \sum_{i=1}^{m} P_i^2 \qquad (2.1)$$

$$Gini_A(D) = \sum_{i=1}^{v} \frac{|D_j|}{D} Gini(D_i) \qquad (2.2)$$

$$Gini\ Gain\ (A) = Gini(D) - Gain_A(D) \qquad (2.3)$$

Keterangan :

- D = Nilai Gini dari sampel data D
- M = Jumlah label yang ada dalam atribut
- Pi = Peluang dari label i atau rasio dari label i
- A = Atribut untuk memisahkan D ke dalam sub himpunan
- V = Sub himpunan
- $\frac{|D_j|}{D}$ = Bobot partisi ke-j
- Gini Gain (A) = Banyak cabang yang akan diperoleh A

Menurut Han (2012), langkah-langkah untuk membuat *decision tree* yang menggunakan Gini Index kurang lebih adalah sebagai berikut.

1. Menentukan *class* atau label yang akan menjadi *root* dalam *tree.*

   Dalam tahap ini, kolom pada *dataset* yang menjadi *class* atau *label* yang akan menjadi *root* pada *decision tree* yang akan dibuat. Menggunakan Persamaan 1, masing-masing peluang label dengan *constraint* yang telah ditentukan dihitung peluangnya dan dikuadratkan.

2. Menghitung *Gini Index* pada semua atribut atau fitur pada *dataset.*

   Kolom-kolom lain yang terdapat pada *dataset* yang digunakan untuk keperluan klasifikasi akan dihitung *Gini Index*-nya menggunakan Persamaan 2. Peluang masing-masing label secara keseluruhan akan

dikalikan dengan *Gini Index* masing-masing kolom dan diakumulasikan.

3. Memilih fitur dengan Gini Index yang terendah.

   Fitur dengan Gini Index yang terendah akan menjadi *root* atau *internal node* dalam *decision tree.*

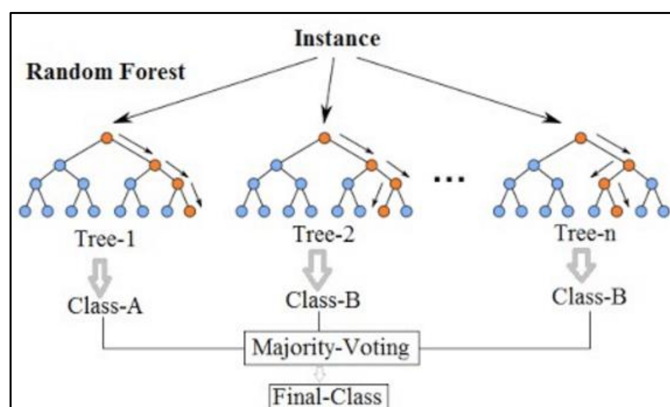
4. Di setiap percabangan, dilakukan rekursif dari langkah pertama.

   Pada setiap cabang, perlu dilakukan rekursif dari langkah pertama hingga menemukan *leaf* atau Gini Index pada cabang tersebut bernilai 0.

5. Menghitung Gini Gain.

   Gini Gain didapatkan menggunakan Persamaan 3 untuk menghitung selisih antara hasil dari Persamaan 1 dengan Gini Index dari masing-masing cabang pertama.

## 2.7 Random Forest Classifier




Gambar 2.7 Struktur Random Forest Classifier (Koerhsen, 2017)

Random Forest adalah algoritma *machine learning* yang memiliki kemampuan untuk melakukan *regression* dan klasifikasi pada suatu *task* (Tahira,

2017). Algoritma ini terdiri dari beberapa *decision tree* yang secara *random* dipilih dari *subset training set*. Hasil klasifikasi merupakan hasil akumulasi dari *votes* yang dilakukan oleh *decision tree* yang berbeda-beda. Pada Gambar 2.7, terdapat gambaran umum dari Algoritma Random Forest.

Menurut Hastie (2009), *Bagging* atau *bootstrap aggregation* adalah teknik yang umum digunakan dalam algoritma Random Forest. Teknik *bagging* berfungsi untuk mengurangi *variance* dari fungsi prediksi yang telah diestimasikan. Metode tersebut dinilai bekerja efektif untuk data yang memiliki variasi yang besar, dan prosedur yang memiliki tingkat bias rendah seperti *trees.* Dalam kasus klasifikasi, setiap *tree* akan menghasilkan hasil prediksi yang kemudian akan dihitung mayoritas hasilnya atau dengan kata lain akan dilakukan *majority vote* (Hastie, 2009).

Menurut Sieling (2015), proses pembuatan *tree* dapat dilakukan dengan *Gini* atau *Entropy. Gini* lebih diperuntukkan untuk atribut kontinyu, sedangkan *entropy* biasa digunakan untuk atribut diskrit yang terdapat pada setiap label. Pembentukan *tree* merujuk pada persamaan 2.1, 2.2, dan 2.3 sama dengan persamaan *decision tree* (Breiman, 1996). *Decision tree* dibuat secara rekursif sesuai dengan jumlah *tree* yang telah ditentukan. Breiman (1996) menyatakan bahwa umumnya optimal jumlah pohon yang dibentuk untuk proses klasifikasi adalah $\sqrt{p}$ dan untuk regresi adalah *p/3* yang mana *p* adalah jumlah *predictor* yang dipakai. *Output* dari setiap pohon dalam proses klasifikasi dikumpulkan dan akan dilakukan *majority vote* sesuai persamaan 2.4 (Hastie, 2009).

$$\hat{C}_{rf}^{B}(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B \qquad (2.4)$$

$\hat{C}_b(x)$ adalah class atau label yang akan menjadi target prediksi dari klasifikasi yang dilakukan oleh decision tree sejumlah b.