BAB 2
TINJAUAN PUSTAKA

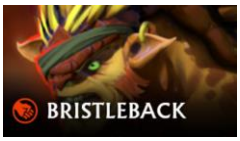
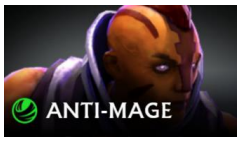
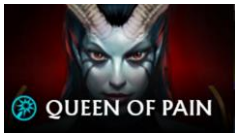
2.1 Dota 2

Dota 2 merupakan sekuel dari *game* terdahulunya *Defense of the Ancients* (DotA), yang dikembangkan oleh komunitas pecinta *game Warcraft III Blizzard Entertainment: Reign of Chaos*. Pengembangan Dota 2 dimulai pada tahun 2009 ketika Valve mempekerjakan *IceFrog* (nama samaran orang yang merancang DotA), untuk membuat versi modern dari DotA menggunakan *Source Game Engine*. Dota 2 memiliki kancah yang besar dalam dunia *esports*, dengan tim-tim dari belahan dunia bermain pada berbagai turnamen profesional. *The international*, merupakan turnamen tahunan utama Dota 2 dengan total hadiah hingga 40 juta dollar atau sekitar 579 milyar rupiah, menjadikannya turnamen *e-sport* dengan hadiah terbesar hingga saat ini.

2.1.1 Karakter Dota 2 (*Hero*)

Dota 2 memiliki 121 karakter disebut *hero*, setiap *hero* memiliki kelebihan dan kelemahan masing-masing. *Hero* dibagi menjadi dua peran utama sebagai *core* dan *support*. *Core* memulai permainan dalam kondisi lemah, namun seiring waktu menjadi lebih kuat sehingga nantinya dapat memenangkan pertandingan untuk timnya. *Support* umumnya tidak dapat memberikan *damage* besar, sebagai gantinya lebih berfungsi untuk mendukung *core* seperti memberi *healing* dan *buffs*. Semua *hero* memulai permainan dari level satu dan seiring permainan dapat mencapai level 30. Setiap *hero* memiliki beberapa kemampuan disebut "*skill*" dengan *skill* utama disebut *ultimate*. Semua *hero* memiliki tiga atribut: *strength*, *intelligence*, dan *agility*. Setiap *hero* memiliki satu atribut utama yang mempengaruhi *health point* (HP), *mana points*(MP), dan *attack speed* (AS), contoh seperti pada Tabel 2.1.

Tabel 2.1 Contoh *Hero* dengan Atribut dan Perannya [5]

Hero			
Jenis	STR	AGI	INT
Strength	22	23	18
Agility	17	24	22
Intelligence	14	12	25
Peran	*Carry, Durable*	*Carry, Escape*	*Nuker, Escape*

Pemain memilih *hero* pada fase pemilihan *hero*, dimana mereka dapat mendiskusikan strategi dan kecocokan *hero*. Cara pemilihan *hero* dapat ditentukan oleh *game mode* yang digunakan, diantaranya seperti berikut.

1. *All Pick*, tidak ada batasan dalam memilih *hero*. *All Pick* sendiri dibagi menjadi *Ranked All Pick* dan *Normal All Pick*.
2. *All Random*, memilih satu *hero* secara random untuk setiap pemain.
3. *Captain's Mode*, seorang pemain memilih seluruh *hero* untuk timnya.
4. *Single Draft*, memilih satu dari tiga *hero* yang disediakan.

2.1.2 Gameplay

Dota 2 dimainkan dengan dua tim terdiri dari 5 pemain, setiap pemain memainkan satu *hero*, berkompetisi untuk menghancurkan bangunan utama disebut "*ancient*", yang berada di tengah markas musuh. *Game* ini dimainkan secara *real-time* pada sebuah peta dengan perspektif *three-dimensional isometric* (perepresentasian objek tiga dimensi ke dalam teknik penggambaran dua dimensi) seperti pada Gambar 2.1. Kedua tim diberi nama sebagai *Radiant* dan *Dire*, bermarkas pada pojok peta yang saling berlawanan. Daerah *Radiant* dan *Dire* dipisahkan oleh sebuah sungai yang dihubungkan oleh tiga jalan. Setiap jalan dijaga dengan tower pertahanan yang menyerang setiap *unit* lawan dalam jangkauannya.

Gambar 2.1 Peta Dota 2

Ketika permainan dimulai, sekumpulan *unit* lemah disebut "*creep"* melintas sepanjang jalan ke arah *ancient* musuh dan menyerang setiap *unit* musuh yang ditemui. *Creep* tersebut diciptakan setiap menitnya, dari bangunan bernama "*barrack*" yang terletak pada markas. Peta permainan diisi dengan hutan dan kabut serta memiliki siklus waktu siang-malam. Pada bagian peta tersebut tepatnya pada hutan, terdapat "*camp*" yang berisi *creep* netral yang menyerang baik *hero* dari tim *radiant* maupun *dire*. *Creep* terkuat bernama "*Roshan*", dapat dikatakan sebagai bos dari semua *creep*, yang apabila dikalahkan menghasilkan *item* bernama "*aegis of immortal*" yang dapat membangkitkan *hero* satu kali ketika terbunuh.

Pemain dapat menaikkan level *hero* dan membeli *item* dari tempat disebut "*Shop*", guna membuat *hero* menjadi lebih kuat. Untuk menaikkan level *hero*, pemain dapat membunuh *creep* baik itu *creep* lawan maupun netral, membunuh *roshan*, menghancurkan bangunan lawan, atau membunuh *hero* lawan. Hasilnya pemain akan mendapatkan *experience points* (XP) serta *gold* yang dapat digunakan untuk membeli *item*. Sebuah tim yang merasa memiliki *hero* yang lebih kuat, biasanya akan mencoba menyerang *hero* lawan atau menghancurkan bangunan lawan. Ketika sebuah *hero* kehabisan HP dan mati, *hero* tersebut dinonaktifkan selama waktu tertentu ,dan kemudian hidup kembali di markas dengan mengalami

pengurangan *gold*. Pada akhirnya tim yang berhasil menghancurkan "*ancient*" musuh terlebih dahulu akan keluar sebagai pemenang.

2.2 Sistem Rekomendasi

2.2.1 Pengertian

Sistem Rekomendasi adalah perangkat lunak dan teknik yang memberikan saran berupa *item* (barang atau jasa), yang berguna bagi *user* (pengguna). Saran yang diberikan terkait berbagai proses pengambilan keputusan, seperti barang apa hendak dibeli, musik apa yang cocok didengarkan, atau berita apa yang hendak dibaca [6].

Ide utama dari sistem rekomendasi ialah untuk memanfaatkan berbagai sumber data dalam menyimpulkan ketertarikan atau kebutuhan *user*. Sehingga analisis dalam sistem rekomendasi biasanya berdasarkan interaksi sebelumnya antara *user* dan *item*, karena ketertarikan dan kebiasaan pada masa lampau merupakan indikator untuk keputusan pada masa yang akan datang [7].

Sistem rekomendasi umumnya ditujukan bagi individu yang kurang pengalaman atau kemampuan, dalam mengevaluasi sejumlah potensial alternatif dari *item* yang direkomendasikan sistem rekomendasi, misal yang ditawarkan oleh sebuah situs web. Contoh seperti sistem rekomendasi buku, yang membantu *user* dalam memilih buku yang hendak dibaca. [6].

2.2.2 Model Dasar Sistem Rekomendasi

Model dasar sistem rekomendasi bekerja dengan dua jenis data, yaitu (i) informasi interaksi *user-item*, seperti *rating* atau perilaku pembelian, dan (ii) informasi tentang atribut *user* dan *item*, seperti profil atau kata kunci yang relevan. Metode yang menggunakan data pertama disebut *collaborative filtering* sedangkan yang menggunakan data kedua disebut *content-based*. Sementara sistem rekomendasi *knowledge-based* merekomendasikan berdasarkan informasi yang secara eksplisit disampaikan oleh *user*, dari pada menggunakan data lampau [7].

2.2.2.1 Model *Collaborative Filtering*

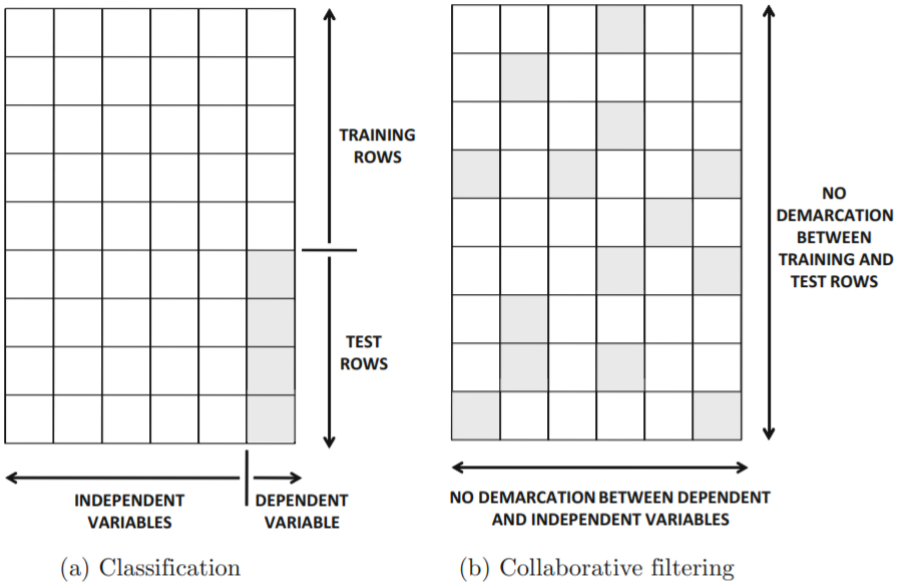
Model *collaborative filtering* menggunakan kolaborasi *rating* dari banyak *user* untuk membuat rekomendasi. *Rating* yang dispesifikasi disebut sebagai *observed rating*, sedangkan yang tidak dispesifikasi disebut *unspecified* atau *missing rating*. Ide dasarnya, *unspecified rating* dapat digantikan nilainya karena *observed rating* memiliki korelasi tinggi terhadap *user* dan *item* lainnya [7].

Ada dua tipe metode yang sering digunakan pada *collaborative filtering* yaitu metode *memory-based* dan *model-based* :

1. Metode *memory-based*: Disebut juga sebagai *neighborhood*-based *collaborative filtering*. Merupakan algoritma terdahulu, dimana *rating* dari *user-item* dikombinasikan untuk memprediksi nilai kedekatannya. Nilai kedekatan tersebut dapat didefinisikan dengan dua cara yaitu *user-based collaborative filtering* dan *item-based collaborative filtering*. Keuntungan dari metode *memory-based* terletak pada kemudahan implementasi dan hasil rekomendasi biasanya mudah dijelaskan.
2. Metode *model-based:* Pada metode ini, *machine learning* dan *data mining* digunakan dalam konteks prediksi model. Beberapa contoh metode model-based termasuk *decision trees*, metode *bayesian*, model *latent factor*, dan model *rule-based*.
    a. *Rule-based collaborative filtering* memanfaatkan *Association Rules*
    Association rules utamanya sangat berguna dalam konteks matriks *rating unary* (mengisi *rating* dilihat sebagai nilai positif, dan tidak ada mekanisme untuk menunjukkan nilai negatif). Tahap awalnya dimulai dari penggalian *association rule* dengan nilai minimum *support* dan minimum *confidence* yang ditentukan. Minimum *support* dan minimum *confidence* dapat dilihat sebagai parameter, yang menentukan akurasi prediksi. Kumpulan *rule* tersebut merupakan acuan yang digunakan untuk merekomendasikan *item* kepada *user* [7].

Collaborative filtering dapat dilihat sebagai generalisasi dari pemodelan klasifikasi dan regresi. Pada model klasifikasi dan regresi, variabel dependen / *class* dapat

dilihat sebagai atribut dengan nilai kosong. Kolom lain dapat dianggap sebagai variabel independen / fitur. Pada *collaborative filtering* setiap kolom diperbolehkan memiliki nilai kosong daripada hanya dependen variabel / *class.* Relasi antara kedua permasalahan tersebut dapat dilihat pada Gambar 2.2 [7].



Gambar 2.2 Perbandingan Klasifikasi dengan *Collaborative Filtering* [7]

2.2.2.2 Sistem Rekomendasi *Content-Based*

Pada sistem rekomendasi *content-based*, atribut deskripsi dari *item* digunakan untuk membuat rekomendasi. Pada metode ini, *rating* dan kebiasaan *user* dikombinasikan dengan informasi *content* yang tersedia dari *item*. Metode *content-based* memiliki beberapa kelebihan dalam membuat rekomendasi *item* baru, ketika data *rating* yang dimiliki tidak mencukupi. Model akan menggunakan *rating* tersedia yang dikaitkan dengan atribut suatu *item*, dalam membuat rekomendasi, meskipun *item* tersebut tidak memiliki catatan *rating* [7]. Metode *content-based* memiliki beberapa kelemahan seperti :

1. Pada beberapa kasus, metode ini memberikan rekomendasi yang mudah ditebak akibat penggunaan kata kunci atau *content*.
2. Walaupun metode ini efektif dalam memberi rekomendasi untuk *item* baru, metode *content-based* tidak efektif memberi rekomendasi untuk *user* baru.

2.2.2.3 Sistem Rekomendasi *Knowledge-Based*

Sistem rekomendasi *knowledge-based* sangat baik pada konteks *item* yang jarang dibeli atau digunakan. Contohnya seperti *real estate,* mobil mewah, layanan finansial, atau barang mewah. Pada kasus tersebut, *rating* yang cukup mungkin tidak tersedia dalam proses rekomendasi. Permasalahan tersebut dapat diselesaikan dengan metode *knowledge-based*, dimana *rating* tidak digunakan dalam membuat rekomendasi. Sebaliknya, proses rekomendasi dibuat berdasarkan kedekatan antara kebutuhan *user* dan deskripsi *item,* atau penggunaan batasan yang menggambarkan kebutuhan *user*. Teknik *knowledge-based* memiliki keunikan dibanding dua metode sebelumnya, dimana sistem ini memperbolehkan *user* secara eksplisit menentukan apa yang mereka inginkan [7]. Perbedaan ini dapat dilihat pada Tabel 2.2.

Tabel 2.2 Tujuan Konseptual beberapa Sistem Rekomendasi Dasar [7]

Metode	Tujuan Konseptual	Input
Collaborative	Membuat rekomendasi berdasarkan pendekatan kolaborasi antara banyak *user*.	*User rating* + *community rating*
Content-based	Membuat rekomendasi berdasarkan konten (atribut) dari *item user* pada masa sebelumnya.	*User rating*+ atribut *item*
Knowledge-based	Membuat rekomendasi berdasarkan spesifikasi eksplisit jenis konten (atribut) yang diinginkan *user*.	Spesifikasi *user* + atribut *item* + *domain knowledge*

Model sistem rekomendasi yang digunakan pada penelitian ini, yaitu model *collaborative filtering* dengan metode *model based*, yang disebut dengan ***rule-base collaborative filtering.***

2.2.3 Evaluasi Sistem Rekomendasi

Salah satu isu terkait praktik penggunaan sistem rekomendasi yaitu kebutuhan untuk mengevaluasinya. Evaluasi *offline* dilakukan dengan melakukan pengukuran performa dari beberapa algoritma yang digunakan, pada dataset yang sama. Evaluasi semacam ini biasanya dilakukan pada data publik yang sudah ada atau jika tidak dengan pengumpulan data.

Evaluasi *online* dapat dilakukan ketika sistem rekomendasi sudah diluncurkan. Evaluasi dilakukan terhadap *user* sesungguhnya untuk menganalisis *log* sistem guna meningkatkan kinerjanya. Jenis evaluasi lain dapat berfokus pada studi *user* apabila evaluasi *online* tidak memungkinkan dilakukan atau terlalu berisiko. Pada jenis evaluasi ini, eksperimen yang terkontrol dapat direncanakan, dimana sebuah grup *user* berukuran cukup kecil, diminta untuk melakukan berbagai tugas guna menguji sistem rekomendasi [6].

2.3 Data Mining

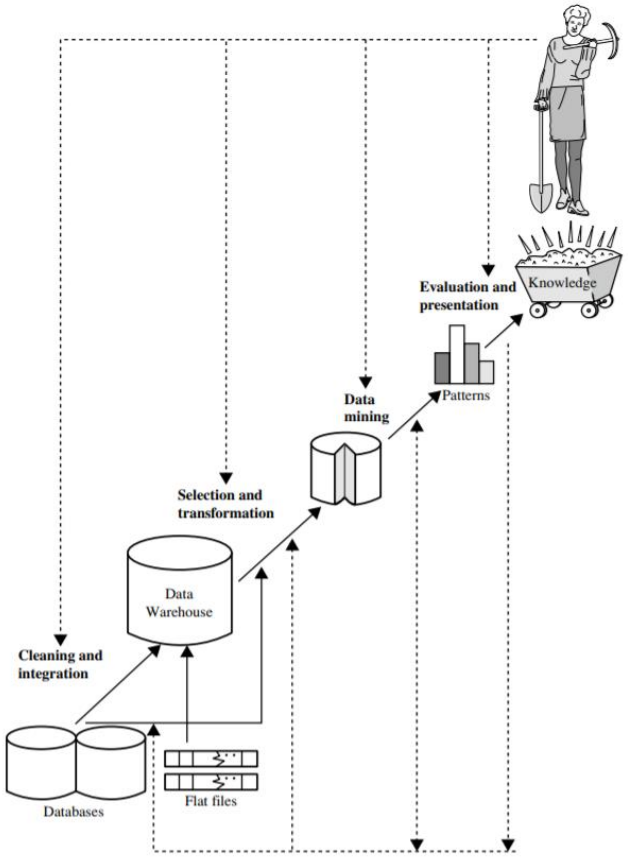
2.3.1 Pengertian *Data Mining*

Data mining sebagai ilmu antardisiplin, sering diartikan dengan berbagai pemahaman. Bahkan istilah *data mining* sesungguhnya tidak merepresentasikan gambaran utamanya. Untuk menggali emas dari bebatuan, kita mengatakan *gold mining* daripada *rock mining*. Sehingga secara analogi "*data mining*" seharusnya disebut "*knowledge mining from data*", yang sayangnya terlalu panjang untuk disebutkan. Namun jika menggunakan istilah "*knowledge mining*", justru tidak merepresentasikan penggalian dari kumpulan data. Jadi, penggunaan kata "*data*" dan "*mining*" menjadi pilihan yang paling terkenal [8].

Banyak orang yang menggunakan istilah *data mining* sebagai sinonim dari istilah lainnya *knowledge discovery from data* (KDD), sementara beberapa orang menganggap *data mining* hanya sebagai salah satu tahapan utama dari proses *knowledge discovery* [8]. Proses *knowledge discovery* seperti pada Gambar 2.3 merupakan langkah-langkah berurutan seperti berikut :

1. *Data cleaning* (menghilangkan *noise* dan data yang tidak konsisten)
2. *Data integration* (mengkombinasikan beberapa sumber data menjadi satu)

3. *Data selection* (pemilihan data yang relevan untuk proses analisis)
4. *Data transformation* (mentransformasi data ke dalam bentuk yang sesuai untuk proses *mining*)
5. *Data mining* (penerapan metode dalam mengekstrak pola dan informasi dari data)
6. *Pattern evaluation* (mengidentifikasi pola yang berguna berdasarkan ukuran yang ditentukan)
7. *Knowledge presentation* (menunjukkan hasil presentasi dan visualisasi dari pengetahuan yang diperoleh kepada *user*)



Gambar 2.3 *Data Mining* Sebagai Tahapan dalam *Knowledge Discovery*

Data mining umumnya tentang penyelesaian masalah dengan menganalisa data yang sudah ada pada database [9]. Pandangan tersebut merupakan salah satu tahapan dalam keseluruhan proses *knowledge discovery*. Namun, pada industri,

media, dan dunia riset, istilah *data mining* lebih sering digunakan untuk mengacu pada keseluruhan proses *knowledge discovery*. Sehingga secara garis besar dapat disimpulkan: *Data mining* merupakan proses menemukan pola dan pengetahuan yang menarik dari data yang berjumlah besar [8].

2.3.2 Metodologi *Data Mining*

Dalam *data mining* terdapat empat masalah fundamental dalam proses penggalian. Masalah-masalah tersebut meliputi klasifikasi dan regresi, *clustering*, *outlier analysis*, dan *association pattern mining*. *Data mining* secara keseluruhan ialah tentang menemukan relasi antara entri pada database yang berbentuk matriks, baik relasi antar kolom maupun antar baris. Oleh karena itu, empat masalah yang disebutkan cukup penting, karena sudah mencakup berbagai jenis representasi data matriks dengan relasi positif, negatif, *supervised*, atau *unsupervised* [10]. Metodologi data mining yang digunakan pada penelitian ini yaitu *association pattern mining*.

2.3.2.1 Association Pattern Mining

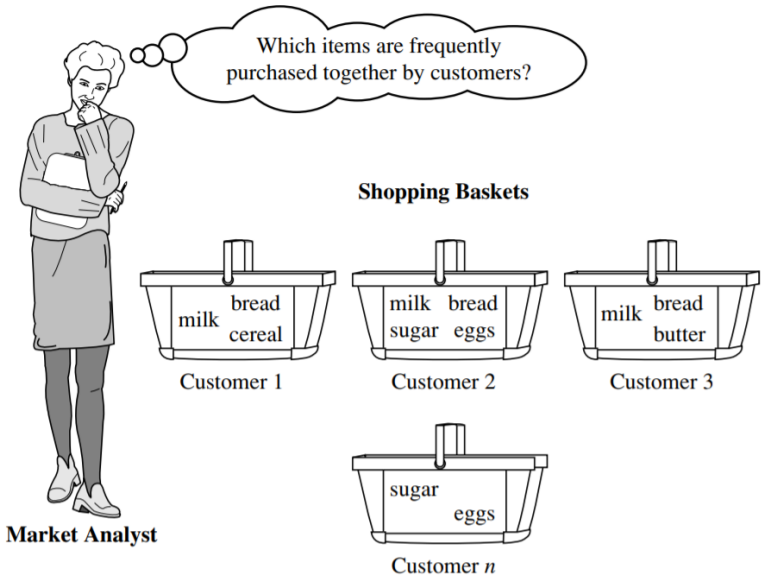
Pada bentuk paling primitifnya, *association pattern mining* didefinisikan dalam konteks database biner jarang, dimana matriks data memiliki entri bernilai 0/1, dan kebanyakan entri bernilai 0. Studi yang paling sering dilakukan dalam *association pattern mining* yaitu permasalahan *frequent pattern* dan *association rules* [10].

2.3.2.1.1 *Frequent Pattern* dan *Association Rules*

Frequent patterns adalah pola (baik itu *itemset*, *subsequences*, atau *substructures*) yang sering muncul pada kumpulan data. Sebagai contoh, susu dan roti yang sering muncul bersama dalam data transaksi pembelian disebut *frequent itemset*. Urutan pembelian, seperti membeli komputer, kemudian sebuah printer, kemudian *mouse*, jika sering terjadi pada sebuah database perbelanjaan, merupakan sebuah *frequent sequential pattern*. Sebuah *substructure* dapat mengacu pada bentuk seperti grafik dan *tree* yang dikombinasikan dengan *itemset* atau

subsequences. Apabila *substructures* sering terjadi maka disebut *frequent structured pattern* [8].

Penggalian *frequent pattern* mencari hubungan berulang pada sekumpulan data. Konsep dasarnya yaitu menemukan asosiasi dan korelasi antara *item* pada data transaksi atau data berelasi berukuran sangat besar. Contoh khas dari *frequent itemset mining* ialah *market basket analysis*. Proses ini menganalisis kebiasaan belanja pembeli, dengan menemukan asosiasi antara barang yang berbeda, yang ditempatkan di keranjang pembelian mereka (Gambar 2.4) [8].



Gambar 2.4 *Market Basket Analysis* [8]

Jika kita menganggap barang yang tersedia di toko sebagai semesta himpunan, maka setiap barang memiliki variabel *boolean* yang melambangkan ketersediaan atau tidak setiap barang. Setiap keranjang dilambangkan dengan nilai vektor *boolean* yang ditetapkan kepada tiap variabel tersebut. Vektor *boolean* ini dapat dianalisis kebiasaan pembelian yang menunjukkan barang-barang yang sering berasosiasi atau dibeli bersama. Pola ini dapat direpresentasikan dalam bentuk *association rules*. Sebagai contoh, orang yang membeli komputer biasanya cenderung membeli software antivirus pada waktu bersamaan, direpresentasikan dengan *association rules* [8] seperti berikut:

computer => antivirus_software [support = 2%, confidence = 60%] (2.1)

Rule support dan *confidence* merupakan dua tolak ukur dari aturan keterkaitan. Keduanya merefleksikan daya guna dan kepastian *rule* yang ditemukan. Pada *rule* (2.1), *support* 2% berarti 2% dari semua transaksi menunjukkan komputer dan antivirus dibeli bersama. *Confidence* 60% menunjukkan bahwa 60% orang yang membeli komputer juga membeli antivirus. Biasanya *association rules* dikatakan menarik jika memenuhi *minimum support threshold* dan *minimum confidence threshold*, yang dapat ditentukan oleh *user* atau seorang ahli. Semakin besar nilai *minimum support threshold* dan *minimum confidence threshold* akan semakin baik, namun harus disesuaikan dengan jumlah *rule* yang akan menjadi semakin sedikit [8].

Pada rule (2.1), dimisalkan berada pada sebuah data transaksi dilambangkan T, dengan *computer* dilambangkan dengan A dan *antivirus_software* dilambangkan dengan B. Maka rule $A => B$ pada suatu transaksi T memiliki support s, dimana s merupakan persentase pada data transaksi mengandung $A \cup B$, (pembelian yang mengandung A dan B). Nilai s diperoleh dari probabilitas $P(A \cup B)$ seperti pada persamaan (2.2). Rule $A => B$ memiliki confidence c pada data transaksi T, dimana c merupakan persentase pembelian A yang disertai dengan B. Nilai c diperoleh dari probabilitas kondisional $P(B|A)$ seperti pada persamaan (2.3) [8].

$$support(A => B) = P(A \cup B) \quad (2.2)$$

$$confidence(A => B) = P(B|A) \quad (2.3)$$

Rule yang memenuhi *minimum support threshold (min_sup)* dan *minimum confidence threshold (min_conf)* disebut *strong*. Sementara, suatu *itemset* yang memenuhi *min_sup* dan *min_conf* disebut *frequent itemset*. Jumlah pembelian yang mengandung *itemset* dapat disebut frekuensi, *support count*, atau *count*. Maka dengan menggunakan *support count*, persamaan (2.3) dapat diturunkan seperti berikut [8].

$$confidence(A => B) = \frac{support(A \cup B)}{support(A)} = \frac{P(A \cup B)}{P(A)} \quad (2.4)$$

Dari persamaan (2.4) dapat dilihat permasalahan penggalian *association rules* dapat dideduksi menjadi penggalian *frequent itemset*. Sehingga secara umum, penggalian *association rules* [8] dapat dilihat sebagai proses dua tahap:

1. *Menemukan semua frequent itemset*: Dalam arti, menemukan *itemset* dengan jumlah kejadian setidaknya memenuhi *min_sup* yang ditentukan.
2. *Menghasilkan strong association rules dari frequent itemset*: Dalam arti, *rule* yang diperoleh harus memenuhi *support* dan *confidence* minimal.

2.3.2.1.2 Metode Penggalian *Frequent Itemset*

Pada bagian ini akan menjabarkan metode penggalian *frequent itemset* tradisional seperti algoritma *Apriori* dan *FP-growth*.

1. Algoritma *Apriori* dengan Pembuatan Kandidat

Apriori menggunakan pendekatan iterasi dengan pencarian *level-wise*, dimana sebanyak k *itemset* digunakan untuk mengeksplorasi (k+1) *itemset*. Pertama, dilakukan *scanning* untuk mencari *frequent* 1-*itemset*, hasilnya disimpan sebagai L_1. Kemudian L_1 digunakan untuk mencari L_2 *frequent* 2-*itemset*, yang hasilnya digunakan untuk mencari L_3, dan seterusnya sampai pada *frequent* k-*itemset*. Pada pencarian L_k dibutuhkan *scan* penuh *database* [8].

2. Pendekatan *Pattern-Growth* untuk Penggalian *Frequent Itemset*

Seperti sudah dijabarkan sebelumnya, pembuatan kandidat pada algoritma Apriori merupakan proses yang mahal. Sebuah metode menarik, yang dapat membuat *frequent itemset* tanpa proses pembuatan kandidat disebut *frequent pattern tree* atau *FP-growth*. *FP-growth* mengadopsi strategi *divide-and-conquer*. Pertama, *database* yang merepresentasikan *frequent item* dikompres ke dalam *frequent pattern tree*, yang mempertahankan informasi keterkaitan dari *itemset*. Kemudian *database* yang dikompres dipecah menjadi kumpulan *database* kondisional, yang masing-masing terdiri dari satu *frequent pattern* dengan

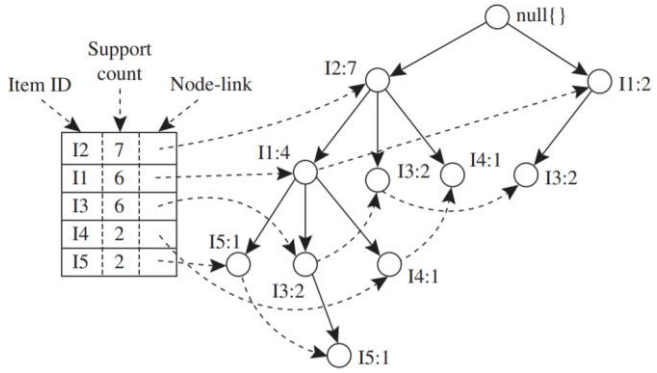
penggalian *database* yang terpisah disebut "*pattern fragment*". Setiap *pattern fragment* hanya memeriksa kumpulan data yang terkait dengannya. Sehingga, dengan pendekatan ini mengurangi intensitas pencarian pada kumpulan data [8].

Sebagai contoh, dari Tabel 2.3 dilakukan penggalian *frequent pattern* menggunakan pendekatan *frequent pattern growth*. Proses *scan database* pertama sama seperti *Apriori*, dengan menurunkan *frequent* 1-*itemset* dan frekuensinya (*support count*). Hasilnya diperoleh L = {{I1: 6}, {I2:7}, {I3:6}, {I4:2}, {I5:2}}.

Tabel 2.3 Data Transaksi Pada AllElectronic

TID	***Daftar item_ID***
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3,
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Sebuah *tree* kemudian dibangun sebagai berikut. Pertama buat *root* dari *tree*, dan diberi label "*null*". *Scan database* untuk kedua kali. *Item* pada setiap transaksi diproses dengan urutan *descending* berdasarkan *support count* katakanlah L. Sebagai contoh, untuk transaksi pertama, "T100:I1,I2,I5" yang terdapat tiga *item* (I2,I1,I5 dalam urutan L), mengarah pada pembangunan ranting dengan tiga *node*, (I2:1), (I1:1), dan (I5:1), dimana I2 terhubung sebagai *child* ke *root*, I1 terhubung ke I2, dan I5 terhubung ke I1. Transaksi kedua T200, terdapat *item* I2 dan I4, dengan hasil ranting I2 terhubung ke *root* dan I4 terhubung ke I2. Demikian seterusnya hingga semua transaksi dilakukan *scan*, maka akan diperoleh *tree* seperti pada Gambar 2.5 [8].



Gambar 2.5 *FP-tree* yang dikompres dan informasinya

Penggalian *FP-tree* dimulai dengan mencari *frequent* 1-*itemset* (disebut sebagai *suffix pattern*), kemudian dibangun *conditional pattern base* serta *conditional FP-tree*-nya, dan dilakukan penggalian *recursive* pada *tree*. Pola yang bertumbuh, diperoleh dengan penggabungan *suffix pattern* dengan *frequent pattern* didapat dari *conditional FP-tree*. Ringkasan penggalian *FP-tree* dapat dilihat pada Tabel 2.4.

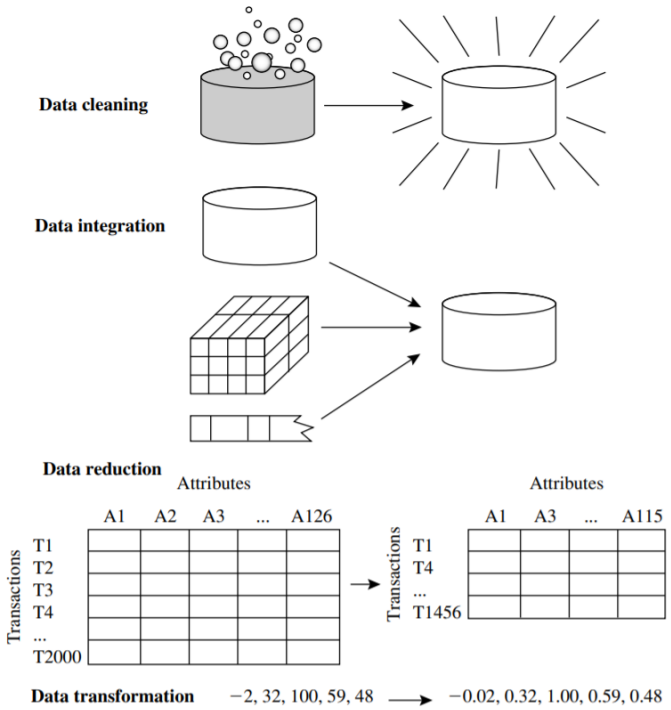
Tabel 2.4 Penggalian *FP-tree* Menggunakan *Conditional Pattern Base*

Item	***Conditional Pattern Base***	***Conditional FP-tree***	***Hasil Frequent Pattern***
I5	{{I2,I1:1},{I2,I1,I3:1}	(I2:2,I1:2,I2,I1:2)	{I2,I5:2},{I1,I5:5},{I2,I1,I5:2}
I4	{{I2,I1:1},{I2:1}}	(I2:2)	{I2,I4:2}
I3	{{I2,I1:2},{I2:2},{I1:2}}	(I2:4,I1:2),(I1:2)	{I2,I3:4},{I1,I3:4},{I2,I1,I3:2}
I1	{{I2:4}}	(I2:4)	{I2,I1:4}

Penelitian ini menggunakan metode pencarian *frequent pattern* dengan algoritma *FP-growth*, untuk menghasilkan *association rules* sebagai model yang akan digunakan untuk membuat sistem rekomendasi *model based*.

2.4 Preprocessing

Data pada dunia nyata biasanya harus diolah terlebih dahulu, agar layak digunakan pada proses *data mining* ataupun *machine learning* [6]. Ada beberapa teknik *preprocessing* yang umumnya digunakan, diantaranya *Data Cleaning, Data Integration, Data Reduction,* dan *Data Transformation* [8] seperti pada Gambar 2.6. Pada penelitian ini akan menerapkan beberapa teknik *data cleaning*, untuk menyiapkan dan membersihkan data yang digunakan.



Gambar 2.6 Bentuk *Data Preprocessing* [8]

2.4.1 Data Cleaning

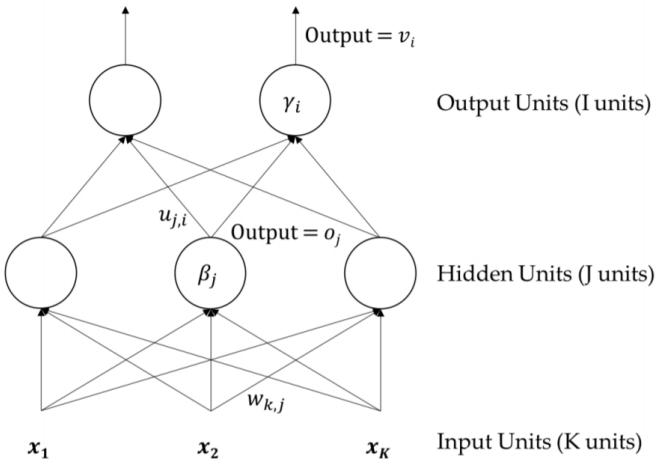
Data cleaning merupakan rangkaian proses membersihkan data dengan mengatasi *missing values*, menghaluskan *noisy data*, mengidentifikasi dan menghilangkan *outlier*, menghilangkan redudansi data, mengatasi data tidak konsisten, dan menyeimbangkan label data.

2.5 Machine Learning

Machine learning merupakan bidang ilmu dan seni dimana program komputer belajar dari data. Arthur Samuel pada bukunya, mengartikan *machine learning* sebagai bidang keilmuan yang memberikan komputer kemampuan untuk belajar tanpa diprogram secara eksplisit [11]. *Machine learning* meliputi banyak algoritma dan topik. Meskipun setiap algoritma *machine learning* tidak lain merupakan fungsi matematika yang menggunakan data tertentu, bentuk dari data dan tugas yang dikerjakan mungkin cukup bervariasi. Jenis *machine learning* yang digunakan pada penelitian ini yaitu *supervised learning* tepatnya menggunakan algoritma klasifikasi *multilayer perceptron*.

2.5.1 Multilayer Perceptron

Sebuah *Multilayer Perceptron* (MLP) terdiri dari satu *input layer*, satu atau lebih *hidden layer*, dan terakhir satu *output layer* [11]. Perhatikan ilustrasi pada Gambar 2.7, MLP secara literal memiliki beberapa layer. Masing-masing neuron terlihat terkoneksi satu dengan yang lain, konfigurasi ini disebut *fully connected*.



Gambar 2.7 *Multilayer Perceptron* [12]

Berdasarkan Gambar 2.7, kita dapat menggunakan persamaan (2.5) dan (2.6) untuk menghitung *output* dari *layer* yang berbeda, dengan u, w adalah *learning parameter*, β, γ melambangkan *noise* atau bias. K adalah banyaknya *input unit*, J

adalah banyaknya *hidden unit* dan σ merupakan fungsi aktivasi seperti *step function*, *sign function*, *rectifier (ReLu)* dan *sigmoid function*. Fungsi aktivasi *sigmoid* dapat dilihat pada persamaan (2.6) dan fungsi aktivasi *relu* dapat dilihat pada persamaan (2.7). Proses perhitungan ini disebut juga dengan *feedforward*.

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right) \quad (2.5)$$

$$S(z) = \frac{1}{1 + e^{-z}} \quad (2.6)$$

$$S(z) = \max(0, z)$$
$$S(z) = \begin{cases} z, jika\ z \geq 0 \\ 0, jika\ z < 0 \end{cases} \quad (2.7)$$

2.5.2 Backpropagation

Untuk melatih model MLP, algoritma umum yang digunakan ialah *backpropagation*. Cara kerjanya, yaitu memperbaharui parameter (*synapse weight*) secara bertahap dari *output* ke *input layer*, berdasarkan *error*/*loss* dengan *output*-nya dibandingkan dengan *desired output.* Intinya adalah mengoreksi *synapse weight* dari *output layer* ke *hidden layer*, kemudian *error* tersebut dipropagasi ke *layer* sebelum-sebelumnya [12].

Backpropagation untuk Gambar 2.7 dapat dilakukan penurunan persamaan untuk menentukan perubahan *weight*, β, dan γ. Proses perhitungan dari *input* ke *hidden* dan dari *hidden* ke *output* menggunakan persamaan (*2.5*). Proses perhitungan dari *output* ke *hidden* layer dapat dilihat pada persamaan (2.8) dan terakhir perhitungan dari *hidden* ke *input layer* pada persamaan (2.9), dimana *n* adalah *learning rate* [12].

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i o_j \quad (2.8)$$
$$\Delta \gamma_i = -\eta(t)\delta_i$$

$$\varphi_j = \sum_{i=1}^{I} \delta_i u_{j,i} o_j(1 - o_j)$$
$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k \quad (2.9)$$
$$\Delta \beta_j = -\eta(t)\varphi_j$$

Secara keseluruhan proses pelatihan MLP menggunakan algoritma *backpropagation* dapat dilihat seperti pada Gambar 2.8.

(2) Hidden to Output

$$v_i = \sigma\left(\sum_{j=1}^{J} o_j u_{j,i} + \gamma_i\right)$$

(3) Output to Hidden

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i o_j$$
$$\Delta \gamma_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right)$$

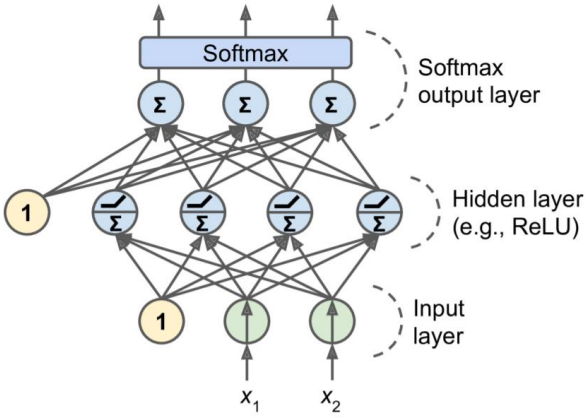
(4) Hidden to Input

$$\varphi_j = \sum_{i=1}^{I} \delta_i u_{j,i} o_j(1 - o_j)$$
$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$
$$\Delta \beta_j = -\eta(t)\varphi_j$$

Gambar 2.8 Proses Pelatihan MLP Menggunakan *Backpropagation*

2.5.3 Arsitektur Model MLP

MLP dapat mengatasi permasalahan klasifikasi biner multilabel. Sebagai contoh, MLP akan menggunakan dua *output neuron*, dimana keduanya menggunakan fungsi aktivasi *logistic*. Kemudian untuk *output* dibutuhkan satu neuron untuk setiap kelas *output*, serta menggunakan fungsi aktivasi *softmax* untuk semua *layer output*. Arsitektur seperti ini disebut klasifikasi *multiclass* (lihat Gambar 2.9).



Gambar 2.9 Klasifikasi Modern MLP menggunakan *ReLU* dan *Softmax* [11]

Untuk *loss function*, karena yang diprediksi adalah distribusi probabilitas, lebih tepat apabila menggunakan *cross-entropy* atau *log loss*. Ringkasan arsitektur klasifikasi MLP pada umumnya dapat dilihat pada Tabel 2.5.

Tabel 2.5 Arsitektur Umum Klasifikasi MLP

Hyperparameter	**Klasifikasi biner**	**Klasifikasi biner multilabel**	**Klasifikasi *multiclass***
# *input neuron*	1 per fitur	1 per fitur	1 per fitur
# *hidden layer*	1-5	1-5	1-5
# neuron per *hidden layer*	10-100	10-100	10-100
# *output neuron*	1	1 per label	1 per class
Hidden activation	*ReLU*	*ReLU*	*ReLU*
Output activation	*Logistic*	*Logistic*	*Softmax*
Loss function	*Cross-Entropy*	*Cross-Entropy*	*Cross-Entropy*

2.5.4 Penentuan Jumlah *Hidden Layer* dan *Neuron*

Ada dua keputusan yang harus dibuat terkait *hidden layer* : berapa banyak *hidden layer* digunakan dalam *neural network* dan berapa banyak neuron pada setiap *hidden layer*. Permasalahan yang menggunakan dua *hidden layer* pada *neural network* cukup jarang ditemui. Namun, *neural network* dengan dua *hidden layer*

dapat merepresentasikan fungsi dengan berbagai ukuran. Tidak ada alasan teori untuk menggunakan *neural network* dengan lebih dari satu *hidden layer*. [13]. Tabel 2.6 menunjukkan kapabilitas arsitektur *neural network* dengan nol sampai dua *hidden layer*.

Tabel 2.6 Menentukan Jumlah *Hidden Layer* [13]

Jumlah Hidden Layer	Hasil
tidak ada	Hanya mampu merepresentasikan fungsi linear terpisah
1	Dapat memperkirakan berbagai fungsi yang mengandung pemetaan kontinu dari satu ruang terbatas ke ruang lainnya.
2	Dapat merepresentasikan batas keputusan yang berubah-ubah ke akurasi yang berubah-ubah dengan fungsi aktivasi rasional dan dapat memperkirakan berbagai pemetaan halus ke berbagai akurasi.

Penentuan jumlah neuron pada setiap *hidden layer* merupakan bagian yang sangat penting, dalam menentukan arsitektur *neural network* secara keseluruhan. Baik jumlah *hidden layer* maupun jumlah neuronnya memiliki pengaruh yang sangat besar terhadap *output* [13].

Menggunakan terlalu sedikit neuron akan mengakibatkan *underfitting*. Sedangkan penggunaan neuron yang terlampau banyak akan mengakibatkan dua masalah. Masalah pertama yaitu *overfitting*, dan masalah kedua terjadi ketika data latih tidak cukup. Oleh karena itu, suatu titik temu perlu dicapai antara terlalu sedikit dan terlalu banyak neuron pada *hidden layer* [13].

Ada beberapa aturan untuk menentukan jumlah neuron yang tepat pada *hidden layer* [13], seperti berikut :

1. Jumlah neuron pada *hidden layer* harus diantara ukuran *input layer* dan *output layer*.
2. Jumlah neuron pada *hidden layer* harus 2/3 *input layer* ditambah *output layer*.

3. Jumlah neuron pada *hidden layer* harus kurang dari dua kali ukuran *input layer*.

Tiga aturan di atas merupakan titik awal yang dapat dipertimbangkan, agar tidak memulai secara acak dalam menentukan jumlah neuron pada *hidden layer*. Pada akhirnya, pemilihan arsitektur *neural network* akan bergantung pada *trial* and *error*, sampai didapat arsitektur dengan *output* terbaik [13].

2.5.5 Evaluasi Klasifikasi MLP

Performa klasifikasi MLP diukur berdasarkan *loss value* dan *accuracy value* dari pengujian model pada data *test*. Pelatihan model dilakukan dengan tujuan untuk mendapatkan model dengan *loss value* sekecil mungkin dan *accuracy value* semaksimal mungkin. Untuk mengurangi *loss value* dan menaikkan *accuracy* dapat dilakukan pengaturan *hyperparameter*. Fungsi *loss* yang digunakan oleh MLP dengan *library tensorflow/keras* yaitu *binary crossentropy* dengan rumus seperti pada persamaan (2.10).

$$\text{Loss} = -\frac{1}{\substack{\text{output} \\ \text{size}}} \sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \qquad (2.10)$$

2.6 Perangkat Lunak Pendukung

Terdapat perangkat lunak dan *library* yang digunakan untuk mengimplementasikan sistem rekomendasi diantaranya :

1. *Tensorflow* : merupakan *library open source* yang dikembangkan dan dipelihara oleh Google. *Tensorflow* dibangun pada struktur data disebut tensor. Tensor merupakan penyamaan ke dalam bentuk skalar dan vektor. *Tensorflow* menyediakan berbagai utilitas untuk komputasi *machine learning* dan *deep learning* [14].
2. *Keras* : merupakan API *deep learning* level tinggi yang dapat dengan mudah menjalankan komputasi *neural network*. Keras dapat dijalankan di atas *Tensorflow*, *Theano* atau *Microsoft Cognitive Toolkit* [11].

3. *Scikit-Learn* : merupakan, *module* python yang sangat mudah digunakan, namun dapat mengimplementasikan berbagai algoritma *machine learning* secara efisien, sehingga membuatnya menjadi pintu masuk yang baik dalam mempelajari *machine learning* [11].
4. *Pandas* : merupakan salah satu *library python* yang banyak digunakan pada bidang *data science*. *Pandas* memberikan metode struktur data dan analisis data yang memiliki performa tinggi. *Pandas* menyediakan *in-memory* tabel objek dua dimensi disebut *DataFrame*, yang bila dijadikan satu dimensi berbentuk *array* disebut *series*.
5. *Numpy* : merupakan singkatan dari *Numerical Python*, adalah *library* python yang mendukung implementasi *array* multidimensi seperti matriks. *Numpy* sendiri dibangun di lingkungan bahasa C, untuk memberikan komputasi matematika yang lebih superior daripada menggunakan python [14].
6. *Mlxtend* (*machine learning extensions*) : merupakan *library* Python yang terdiri dari berbagai alat untuk tugas *data science*. *Mlxtend* merupakan proyek *open source* dengan lisensi BSD [15].
7. *Anaconda* : merupakan distribusi *package manager* yang secara otomatis menginstall *Python*, *Jupyter*, dan berbagai *package* untuk *machine learning* dan *data science*. *Anaconda* dibuat sangat mudah dalam mengatasi berbagai versi *package* dan *update-nya* [14].
8. *Jupyter Notebook* : merupakan sebuah *tool* yang dapat memproses serta menunjukkan hasil setiap tahap-tahap dari proses *machine learning*. *Jupyter Notebook* dikenal dengan kemudahan dan dukungannya terhadap fitur, dan *platform* solusi pengembangan *machine learning* dan *deep-learning* saat ini [14].
9. *Django* : merupakan *framework* pengembangan web menggunakan Python yang paling populer. *Framework* ini ringan, kuat, skalabilitas tinggi, dan dirawat dan dikembangkan oleh komunitas yang cukup cepat dalam menutupi kelemahan *security* dan menambahkan fitur-fitur baru. *Django* memiliki sangat banyak fitur *built-in* yang memberikan metode dan penerapan yang sangat baik [14].