BAB 1

ANALISIS

Bab ini akan membahas mengenai permasalahan umum yang dihadapi, melakukan studi kasus pada algoritma *naive bayes classifier*, dan melakukan analisis pemodelan sistem yang akan dibangun.

1.1 Deskripsi Masalah

Salah satu algoritma teknik *data mining* yang digunakan pada skripsi ini adalah algoritma *naive bayes classifier*. Tingkat akurasi pada algoritma ini dapat dipengaruhi oleh beberapa faktor. Salah satu faktor pentingnya adalah faktor volume data. Pada algoritma *naive bayes* yang diimplementasikan secara *standalone*, (non-*MapReduce*, tidak menggunakan komputasi secara paralel) tidak dapat mengolah proses menggunakan data yang sangat besar, karena adanya keterbatasan memori yang dimiliki oleh perangkat tersebut.

Apache hadoop digunakan untuk menangani hal meliputi big data dengan sistem yang terdistribusi. *Framework hadoop* dapat digunakan untuk membantu algoritma *naive bayes classifier* dalam menangani jumlah data yang sangat banyak dengan mengesampingkan batasan memori. Rancangan program *naive bayes* yang dibuat perlu berbasiskan *MapReduce* pada Hadoop. Dengan begitu, tingkat akurasi yang dimiliki oleh algoritma ini akan sangat maksimal dengan memberikan fasilitas untuk mengolah data yang sangat banyak dan beragam (*big data*). Waktu yang dibutuhkan untuk mengeksekusi program tersebut juga diharapkan akan sangat cepat sebanding dengan jumlah komputer/node yang digunakan pada sistem terdistribusi hadoop(mendelegasikan pekerjaan kepada tiap komputer/node yang terintegrasi pada sistem secara bersamaan).

1.2 Kebutuhan Pemilihan Data Masukan

Data yang dapat digunakan pada perangkat lunak yang dibuat adalah data tidak terstruktur yang menggunakan *comma-separated values*[1]. Data tidak terstruktur (*unstructured data*) adalah data yang tidak memiliki format pasti. Data tidak terstruktur biasanya merupakan data text yang berukuran sangat besar dan format dari isi datanya juga dapat memiliki format yang bermacam - macam, seperti: tanggal ; angka ; suatu kejadian ; dsb. Data - data yang digunakan bisa saja berupa data pencatatan pembelian selama 3 tahun terakhir dari suatu perusahaan, data penjualan mobil dengan spesifikasi kriteria yang rinci dari suatu perusahaan mobil, dsb. Selain itu, data yang digunakan juga perlu memiliki ukuran yang cukup besar (supaya manfaat dari penggunaan framework hadoop akan lebih terlihat signifikan). Seperti pada contoh data berikut mengenai penentuan seseorang akan bermain tenis atau tidak jika diberikan beberapa fakta yang terjadi terkait faktor lingkungan dan waktu :

```
1 Outlook,Temperature,Humidity,Windy,Play,Rand,Hour
```

[1] CSV (comma-separated values) merupakan data tabular yang berada di dalam plain text. Setiap baris dari data tersebut menyatakan sebuah record. Setiap record memiliki 1 atau lebih field yang dipisahkan oleh koma

```
Rainy,Hot,High,FALSE,No,3.5,12:00:00
Rainy,Hot,High,TRUE,No,12,14:00:00
Overcast,Hot,High,FALSE,Yes,11,,16:00:00
Sunny,Mild,High,FALSE,Yes,4,,18:00:00
Sunny,Cool,Normal,FALSE,Yes,2,09:00:00
Sunny,Cool,Normal,TRUE,No,1.9,17:00:00
Overcast,Cool,Normal,TRUE,Yes,6.4,20:00:00
Rainy,Mild,High,FALSE,No,10,07:00:00
Rainy,Cool,Normal,FALSE,Yes,9,06:00:00
```

Pada kolom pertama, ke-dua, ke-tiga, ke-empat, dan ke-lima merupakan data yang bertipe diskrit dan pada kolom ke-enam dan ke-tujuh merupakan data yang bertipe numerik. Tetapi, pada kolom ke-tujuh, perlu diberikan penanganan lebih lanjut karena data tersebut perlu dikonversi terlebih dahulu dari yang berbentuk jam ke bentuk numerik yang sederhana.

1.3 Kebutuhan Pra-pengolahan Data

Pada teknik data mining, diperlukan fase pra-pengolahan data terlebih dahulu sebelum melakukan mining dengan teknik tertentu, agar data yang masuk ke dalam perangkat lunak memiliki format yang pasti. Pada skripsi kali ini, fase pra-pengolahan akan diperlukan untuk mendeteksi dan menangani terjadinya *missing values*[2] pada data. Pendekatan yang digunakan untuk mengatasi terjadinya *missing-values* yang dapat menyebabkan analisis berjalan tidak lancar ini adalah metode *listwise deletion*. *Listwise deletion* merupakan salah satu metode dalam cabang ilmu statistika untuk mengatasi terjadi *missing-values* dengan cara mengabaikan seluruh record - record yang memiliki *missing-values* [14].

User	Device	OS	Transactions
A	Mobile	~~NA~~	5
B	Mobile	Android	3
C	~~NA~~	iOS	2
D	Tablet	Android	1
E	Mobile	iOS	4

Gambar 1.1: Missing-values [15]

1.4 Perhitungan Manual Dengan Data Studi Kasus

1.4.1 Studi kasus pembuatan model *naive bayes classifier*

Misal kita memiliki dataset yang menunjukan seseorang akan bermain tenis atau tidak berdasarkan dari data kelembaban dan pemandangan yang terjadi seperti pada tabel 1.4.1 berikut:

Tabel 1.1: Contoh Dataset (atribut kelas = **Play**)

Humidity	Outlook	**Play**
60	Rainy	No
78	Rainy	No
80	Sunny	Yes
75	Sunny	No
85	Sunny	Yes

[2] *Missing-values* merupakan keadaan dimana jumlah field pada suatu record tidak memenuhi jumlah field yang seharusnya

Langkah pertama yang perlu dilakukan untuk membuat algoritma naive bayes classifier adalah membuat table frekuensi untuk setiap atribut prediktor terhadap atribut kelas yang bertipe Diskrit. Pada contoh tabel diatas, diasumsikan bahwa atribut Humidity dan Outlook merupakan atribut prediktor, lalu untuk atribut Play merupakan atribut kelas.

Tabel 1.2: Table frekuensi atribut Outlook

	Play		
	Yes	No	*sum*
Sunny	2	1	**3**
Rainy	0	2	**2**
sum	**2**	**3**	

Pada table frekuensi untuk atribut Outlook, dapat dilihat bahwa $P(X = Rainy|C = Yes) = 0$. Naive bayes classifier tidak dapat mengatasi frekuensi yang nilainya 0. Karena dapat menyebabkan seluruh perhitungan menjadi 0(karena berapapun bilangannya, jika dikalikan dengan 0 akan selalu menghasilkan nilai 0), sehingga menjadi tidak relevan. Pendekatan yang perlu digunakan untuk mengatasi hal tersebut adalah dengan menggunakan metode Laplacian Correction (->Pustaka.Bib). Pada metode tersebut, dikatakan bahwa kita perlu menambah nilai 1 kepada seluruh nilai pada table frekuensi untuk mengatasi masalah *zero-frequency problems*. Asumsikan training database (D) itu sangat besar, dimana menambahkan frekuensi sebanyak 1 ke setiap jumlah perhitungan yang kita perlukan tidak akan memberikan pengaruh yang besar terhadap nilai kemungkinan akhir (Laplacian correction / Laplace estimator) [14]. Perubahan nilai atribut dapat dilihat pada table 3.

Tabel 1.3: Table frekuensi atribut Outlook

	Play		
	Yes	No	*sum*
Sunny	3	2	**5**
Rainy	1	3	**4**
sum	**4**	**5**	

Langkah kedua adalah membuat table kemungkinan dari table frekuensi yang telah dibuat :

Tabel 1.4: Table kemungkinan atribut Outlook

	Play		
	Yes	No	*sum*
Sunny	3/4	2/5	**5/9**
Rainy	1/4	3/5	**4/9**
sum	**4/9**	**5/9**	

Karena atribut Humidity bertipe numerik, atribut tersebut perlu diubah ke dalam kategori mereka masing - masing agar perhitungan dalam pembuatan model dapat tepat. Konversi atribut yang bertipe numerik bisa menggunakan distribusi variabel numerik untuk dapat menebak frekuensi-nya dengan mengasumsikan distribusi normal untuk variabel numerik. Rumus yang digunakan adalah :

Mencari mean (rata - rata)

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1.1}$$

Mencari Standard Deviation

$$\sigma = [\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2]^{0.5} \tag{1.2}$$

Normal Distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1.3}$$

Berikut merupakan table rata - rata dan standar deviasi dari atribut Humidity yang bertipe numerik :

Tabel 1.5: Table rata - rata dan standar deviasi atribut Humidity

Play Golf ?	Yes	80	85	
	No	60	75	78

Tabel 1.6: Table Distribusi

	Mean	StDev
Yes	82.5	3.5
No	71	9.6

Dari table distribusi tersebut didapatkan formula untuk menghitung klasifikasi untuk atribut Humidity adalah:

$$f(x|play = yes) = \frac{1}{\sqrt{2\pi(3.5)}} e^{-\frac{(x-82.5)^2}{2(3.5)^2}} \tag{1.4}$$

$$f(x|play = no) = \frac{1}{\sqrt{2\pi(9.6)}} e^{-\frac{(x-71)^2}{2(9.6)^2}} \tag{1.5}$$

Cara ini dapat diberlakukan juga pada atribut yang bertipe kontinu.

Setelah semua model dari naive bayes classifier telah jadi, maka klasifikasi sudah dapat dilakukan dengan model diatas.

1.4.2 Studi kasus melakukan klasifkasi menggunakan model *naive bayes classifier* yang telah dibuat sebelumnya

Dimisalkan kita memiliki 2 buah dataset yang akan diuji menggunakan model klasifikasi yang telah dibangun sebelumnya, seperti berikut :

1. $X = Humidity = 50, Outlook = Sunny$
2. $Y = Humidity = 90, Outlook = Sunny$

Untuk dataset X dan Y, akan dicari peluang kelas yang paling tinggi.

$$(C_{MAP} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{argmax} P(d|c)P(c))$$

Untuk dataset X dengan $P = Yes$:

Menghitung peluang untuk atribut $Outlook = Sunny$ dengan $P = Yes$

$$P(Outlook = Sunny|Yes) = 3/4 = 0.75$$

Menghitung peluang untuk atribut $Humidity = 50$ dengan $P = Yes$

$$P(Humidity = 50|Yes) = \frac{1}{\sqrt{2\pi(3.5)}} e^{-\frac{(\mathbf{50} - 82.5)^2}{2(3.5)^2}} = 4.031$$

Untuk dataset X dengan $P = No$:

Menghitung peluang untuk atribut $Outlook = Sunny$ dengan $P = No$

$$P(Outlook = Sunny|No) = 2/5 = 0.4$$

Menghitung peluang untuk atribut $Humidity = 50$ dengan $P = No$

$$P(Humidity = 50|No) = \frac{1}{\sqrt{2\pi(9.6)}} e^{-\frac{(\mathbf{50} - 71)^2}{2(9.6)^2}} = 0.011$$

Kesimpulan untuk *dataset X*

Dari perhitungan di atas, didapat bahwa :

Untuk kelas $Play = Yes$
$P(Play = Yes|X)$
$= P(Outlook = Sunny|Play = Yes) * P(Humidity = 50|Play = Yes) * P(Yes)$
$= 0.75 * 4.031 * 4/9$
$= 1.343$

Untuk kelas $Play = No$
$P(Play = No|X)$
$= P(Outlook = Sunny|Play = No) * P(Humidity = 50|Play = No) * P(No)$
$= 0.4 * 0.011 * 5/9$
$= 0.002$

Setelah itu, lakukan normalisasi terhadap nilai - nilai berikut:
$P(Play = Yes|X) = 1.343/(1.343 + 0.002)$
$= 0.998(99.8\%)$
$P(Play = No|X) = 0.002/(1.343 + 0.002)$
$= 0.002(0.2\%)$

Karena, $P(Play = Yes|X) > P(Play = No|X)$, maka hasil klasifikasi untuk *dataset X* ialah kelas $Play = Yes$.

Untuk dataset Y dengan $P = Yes$:

Menghitung peluang untuk atribut $Outlook = Sunny$ dengan $P = Yes$

$$P(Outlook = Sunny|Yes) = 3/4 = 0.75$$

Menghitung peluang untuk atribut $Humidity = 90$ dengan $P = Yes$

$$P(Humidity = 90|Yes) = \frac{1}{\sqrt{2\pi(3.5)}} e^{-\frac{(\mathbf{90} - 82.5)^2}{2(3.5)^2}} = 0.021$$

Untuk dataset Y dengan $P = No$:

Menghitung peluang untuk atribut $Outlook = Sunny$ dengan $P = No$

$$P(Outlook = Sunny|No) = 2/5 = 0.4$$

Menghitung peluang untuk atribut $Humidity = 90$ dengan $P = No$

$$P(Humidity = 90|No) = \frac{1}{\sqrt{2\pi(9.6)}} e^{-\frac{(\mathbf{90} - 71)^2}{2(9.6)^2}} = 0.018$$

Kesimpulan untuk *dataset Y*

Dari perhitungan di atas, didapat bahwa :

Untuk kelas $Play = Yes$
$P(Play = Yes|Y)$
$= P(Outlook = Sunny|Play = Yes) * P(Humidity = 90|Play = Yes) * P(Yes)$
$= 0.75 * 0.021 * 4/9$
$= 0.007$

Untuk kelas $Play = No$
$P(Play = No|Y)$
$= P(Outlook = Sunny|Play = No) * P(Humidity = 90|Play = No) * P(No)$
$= 0.4 * 0.018 * 5/9$
$= 0.004$

Setelah itu, lakukan normalisasi terhadap nilai - nilai berikut:
$P(Play = Yes|Y) = 0.007/(0.007 + 0.004)$
$= 0.64(64\%)$
$P(Play = No|Y) = 0.004/(0.007 + 0.004)$
$= 0.46(46\%)$

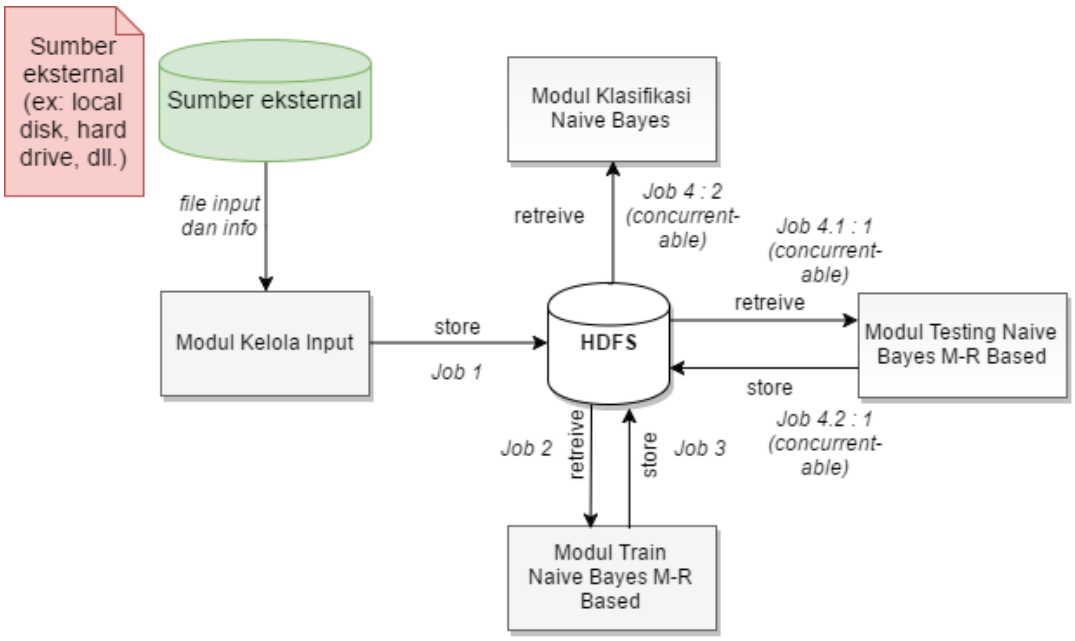
Karena, $P(Play = Yes|Y) > P(Play = No|Y)$, maka hasil klasifikasi untuk *dataset Y* ialah kelas $Play = Yes$.

1.5 Analisis Perangkat Lunak

Pada bagian ini akan dijelaskan menenai analisis perancangan perangkat lunak yang mencakup aliran proses dan gambaran secara umum diagram kelas untuk melakukan skema algoritma *naive bayes classifier* berbasis *map reduce*.

1.5.1 Analisis Skema Algoritma *Naive Bayes Classifier* Berbasis *Map Reduce*

Keseluruhan program yang akan menjalankan pelatihan maupun pengujian klasifikasi naive bayes berbasis mapreduce pada Hadoop yang dibuat akan memiliki 4 buah modul. sebagian besar modul tersebut harus berjalan secara berurutan dan saling bergantung satu dengan



Gambar 1.2: Rancangan Keseluruhan Modul Program

lainnya dalam menjalankan tugasnya. Berikut adalah spesifikasi ringkas dari modul beserta urutan yang perlu dijalankan terlebih dahulu:

Modul	Data Retreive From External Source	Data Retreive From HDFS	Data Store To HDFS	Order
Kelola Input	data input dari disk local user	-	data input yang akan dianalisis oleh perangkat lunak	1
Train Naive Bayes M-R Based	-	data yang akan dijadikan input modul	*naive bayes classifier model*	2
Testing Naive Bayes M-R Based	-	data testing yang akan dikadikan input modul	*confusion matrix* dan perhitungan error rate	3
Klasifikasi Naive Bayes	-	*naive bayes classifier model*	hasil klasifikasi dengan menggunakan model	3

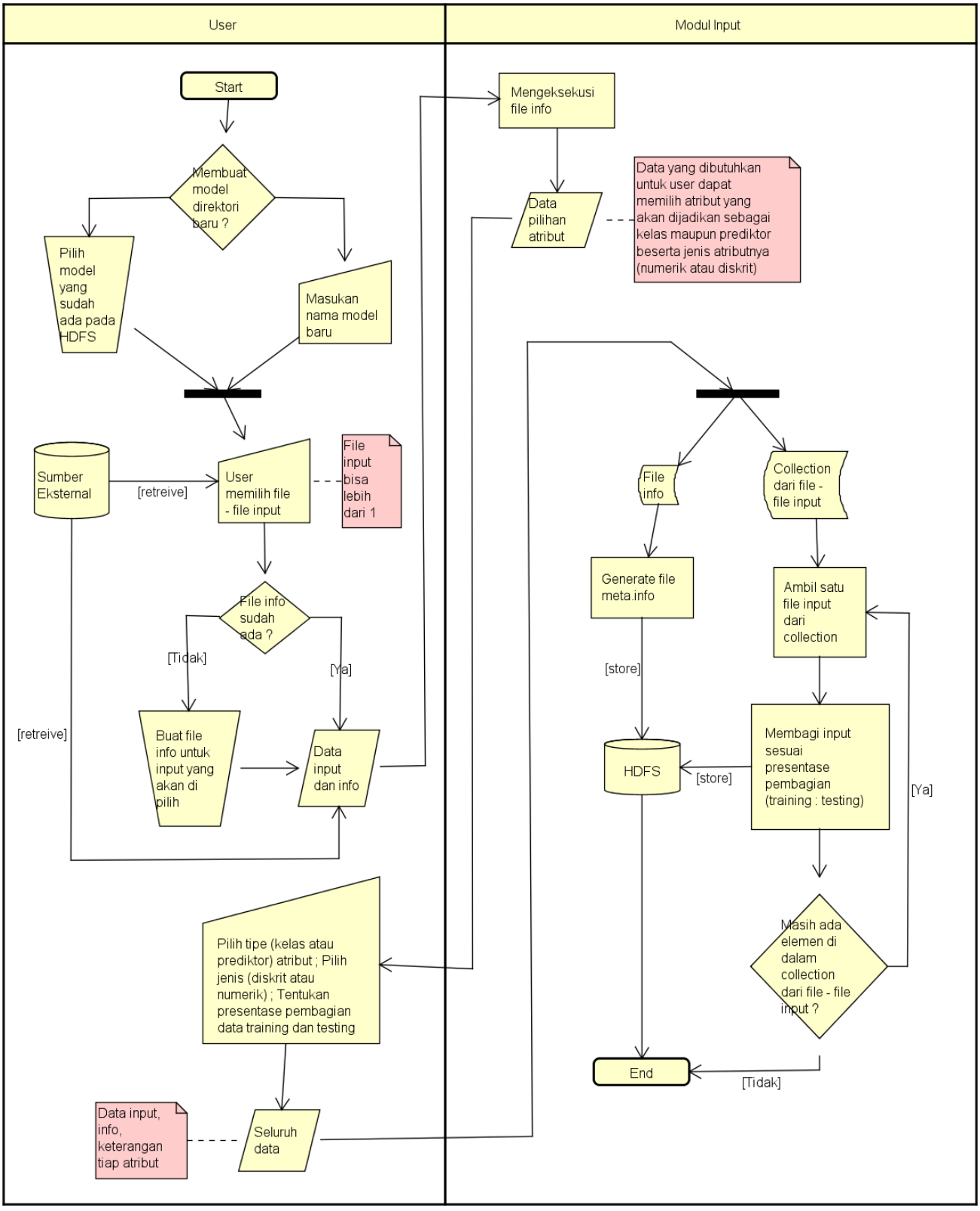
Gambar 1.3: Modul Specification

1.5.1.1 Modul Kelola *Input*

Pada Modul Input, program akan menerima input file dari pengguna berupa data yang akan dijadikan pelatihan untuk pembuatan model klasifikasi naive bayes. Pengguna diberikan pilihan untuk menentukan atribut mana saja yang akan dijadikan kelas dan yang dijadikan sebagai atribut prediktor dan memilih tipe konten dari atribut yang digunakan (mis: *diskrit atau numerik*). Selain itu, penguna juga diberikan pilihan untuk membagi presentase seluruh data input yang akan dijadikan sebagai *data training* dan *data testing*. Program pada modul ini akan meminta akses kepada server master Hadoop untuk melakukan proses tulis pada HDFS dengan meng-import library Hadoop Client API pada program. Pada modul ini terdapat file tambahan yang akan dimasukan ke dalam HDFS, yaitu file yang bernama meta.info. File meta.info ini akan berisi kumpulan dari atribut prediktor yang pengguna

pilih dan atribut kelas yang penggna pilih beserta dengan masing - masing tipe kontennya.

Berikut merupakan diagram *flow chart* untuk modul input:

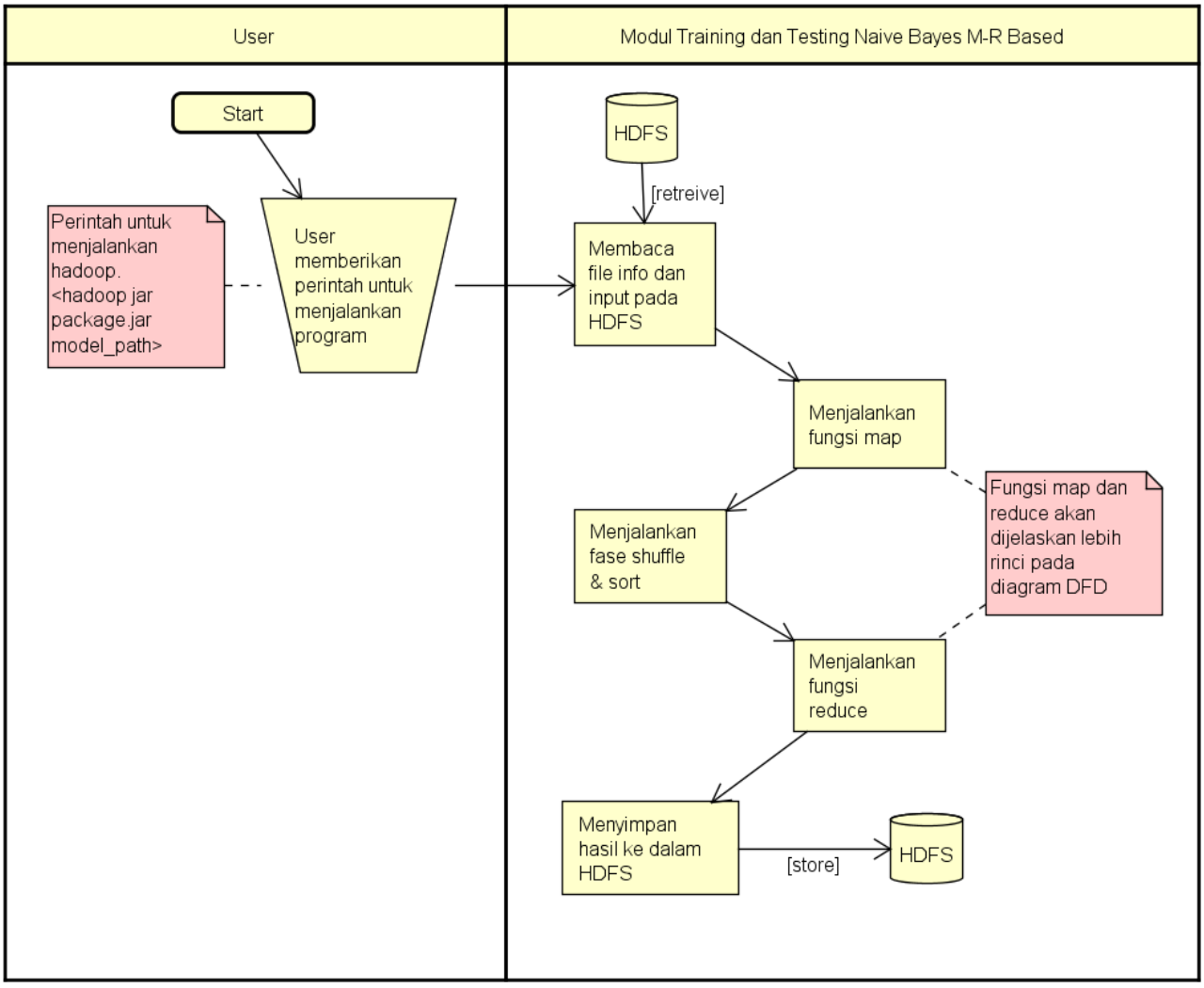


Gambar 1.4: Flow Chart Modul Input

1.5.1.2 Modul *Train Naive Bayes M-R Based*

Sebelum modul ini dijalankan, proses pada modul Input haruslah terlebih dulu selesai, karena file yang menjadi input pada modul ini merupakan hasil dari salinan file yang dijalankan pada proses dalam modul Input. Pada modul ini akan dijalankan proses train dalam pembentukan model klasifikasi naive bayes. Program *training* klasifikasi naive bayes dibuat di atas framework mapreduce yang akan dijalankan pada Hadoop. Terdapat 2 pengecekan yang akan dilakukan pada modul ini, yaitu untuk menghitung atribut yang bertipe diskrit dan numerik(kontinu).
Program pada modul ini akan memisahkan cara perhitungan yang digunakan dalam membangun sebuah model *naive bayes classifier*. Algoritma Naive Bayes yang akan diimplementasikan pada program akan menerima input berupa dataset dan info mengenai dataset tersebut. Info yang akan diberikan meliputi atribut yang digunakan untuk melakukan pembuatan model classifier, tipe dari tiap atribut yang akan digunakan, dan atribut yang akan menjadi kelas-nya.

Berikut merupakan diagram *flow chart* untuk modul *training*:

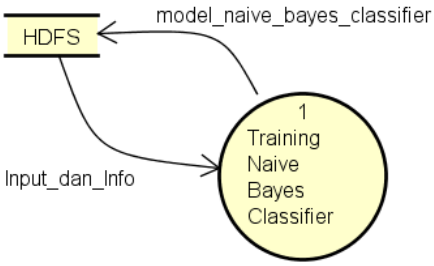


Gambar 1.5: Flow Chart Modul Training

Untuk proses yang berbasis *MapReduce* pada modul ini perlu digambarkan menggunakan DFD, agar bisa tergambarkan lebih rinci mengenai detail proses tersebut. Berikut merupakan *context diagram*[3] dan DFD untuk proses *MapReduce* pada modul *training*:

[3] *Context Diagram biasa disebut juga sebagai DFD level 0.*

Context Diagram Modul *Training*



Gambar 1.6: *Context diagram* modul Training

Data Dictionary Context Diagram Modul *Training*

1. Data `model_naive_bayes_classifier`

   - *Class Name* = [A..Z|a..z] **required*
   - *Class Value* = [A..Z|a..z] **required*
   - *Atribute Type* = [A..Z|a..z] **required*
   - Frekuensi kemunculan = [0..9] **required*
   - *Predictor Name* = [A..Z|a..z]
   - *Predictor Value* = [A..Z|a..z]
   - *Mean* = [0..9]
   - *Sigma (standard deviation)* = [0..9]

   Contoh data `model_naive_bayes_classifier`:

```
Play,Yes,2.0|CLASS
Play,No,3.0|CLASS
Humidity,Play,Yes ;82.5|3.5|NUMERIC
Humidity,Play,No ;71.0|9.6|NUMERIC
Outlook,Sunny,Play,Yes,2.0|DISCRETE
Outlook,Sunny,Play,No,1.0|DISCRETE
Outlook,Rainy,Play,No,2.0|DISCRETE
```

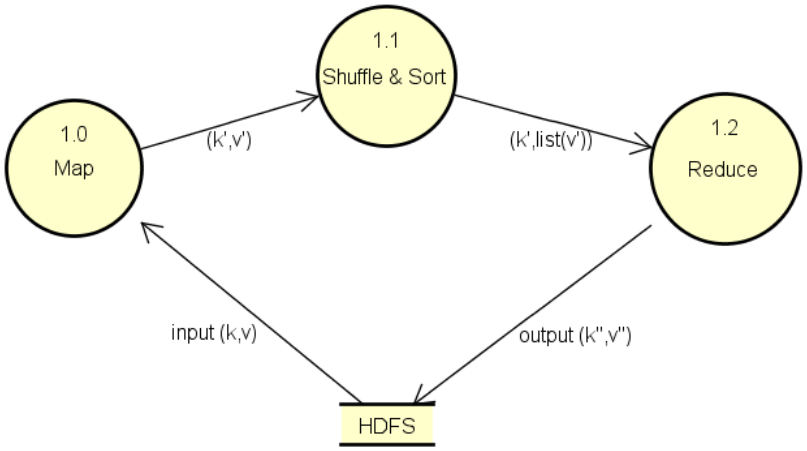
2. Data `input_dan_info`

   - Data nilai tiap field = [A..Z|a..z|0..9] **required*
   - Nama - nama field = [A..Z|a..z] **required*

   Contoh data `input_dan_info`:

```
<- Data input ->
Sunny,Mild,Normal,FALSE,Yes,5
Rainy,Mild,Normal,TRUE,Yes,4.5
Overcast,Mild,High,TRUE,Yes,3.1
Overcast,Hot,Normal,FALSE,Yes,8.2
Sunny,Mild,High,TRUE,No,3
<- Data info ->
Outlook,Temperature,Humidity,Windy,Rand
```

DFD *level* 1



Gambar 1.7: DFD level 0 modul Training

Data Dictionary pada DFD *level* 1

1. Data `input(k,v)`
   - Key: *NULL*. Karena, memang pada pertama kali data diambil dari HDFS, keynya belum terdefinisi.
   - Value: Nilai dari tiap field yang ada = [A..Z|a..z|0..9] **required*

   Contoh data `input_dan_info`:

```
key     value
      Sunny,Mild,Normal,FALSE,Yes,5
      Rainy,Mild,Normal,TRUE,Yes,4.5
      Overcast,Mild,High,TRUE,Yes,3.1
      Overcast,Hot,Normal,FALSE,Yes,8.2
      Sunny,Mild,High,TRUE,No,3
```

2. Data `(k',v')`
   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
     (c) *Attribute Type* = [A..Z|a..z] **required*
     (d) *Predictor Name* = [A..Z|a..z]
     (e) *Predictor Value* = [A..Z|a..z|0..9]
   - *Value* memiliki 3 jenis format yang berbeda untuk tiap jenis atribut, diantaranya adalah:
     (a) Nilai dari atribut numerik dan *kelas* = [0..9]
     (b) Diskrit: Frekuensi kemunculan = [1] (frekuensi kemunculan untuk satu probabiltas posterior pasti bernilai 1)

   Contoh data `(k',v')`:

```
key                          value
|_class|Play,Yes             1
|disc|Humidity,High,Play,No      1
|cont|Rand,Play,Yes           34.2
```

3. Data `(k',list(v'))`

   - Format data untuk variabel *key*, masih sama dengan format variabel *key* pada data `(k,v)`.
   - Untuk variabel *value* juga demikian, tetapi tipe-nya berubah menjadi list.

   Contoh data `(k',list(v'))`

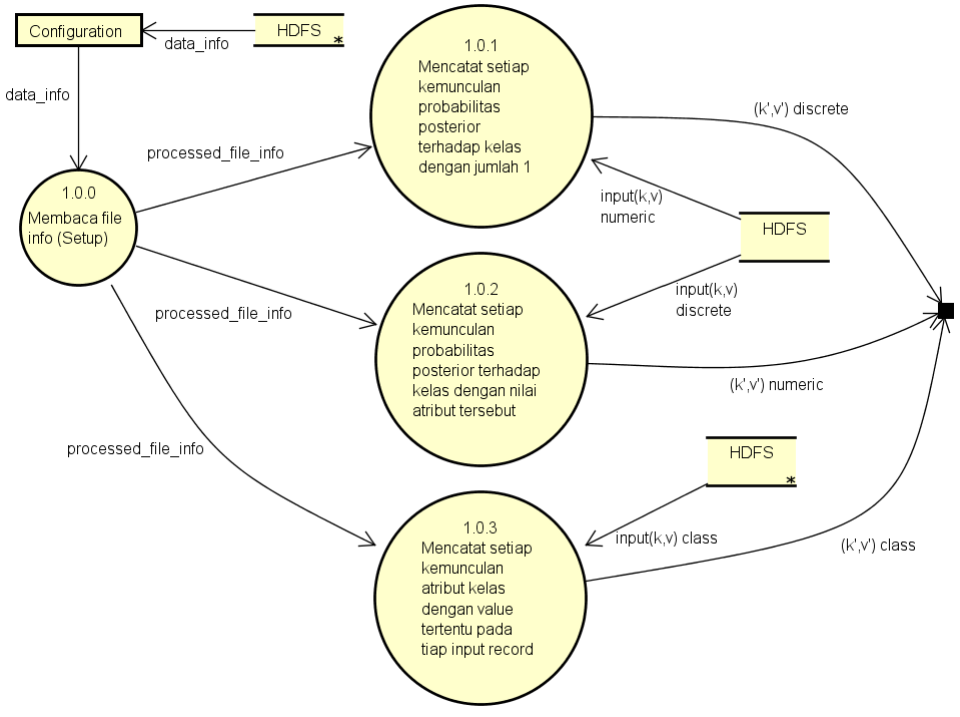
```
key                              list_of_value
   |_class|Play,Yes              [1,1,1,1]
   |disc|Humidity,High,Play,No       [1,1]
   |cont|Rand,Play,Yes             [34.2,23.3,15.0]
```

4. Data `input(k",v")`

   - Format data untuk variabel *key*, sama dengan format variabel *key* pada data `(k,v)`, tetapi untuk atribut yang bertipe diskrit dan kelas, ditambahkan dengan jumlah frekuensi kemunculan pada tiap probabilitas posterior yang muncul.
   - Format atribut *value* untuk tiap jenis:
     (a) Diskrit: *NULL*
     (b) Kelas: *NULL*
     (c) Numerik: *mean*, *sigma*, dan tipe atribut(numerik) = [A..Z|a..z|0..9]

```
key                                value
Play,Yes,5.0|CLASS                    (empty-string)
Rand,Play,No                       ;6.85|4.247|NUMERIC
Humidity,High,Play,No,3.0|DISCRETE    (empty-string)
```

DFD *level* 2: pada proses 1.0



Gambar 1.8: DFD level 2: proses 1.0

***Data Dictionary* pada DFD *level* 2: proses 1.0**

1. Data `data_info`

   - Nama field atribut kelas yang dipakai pada *training* = [A..Z|a..z]
   - Nama field atribut prediktor yang dipakai pada *training* = [A..Z|a..z]
   - Nomor indeks dari atribut kelas = [0..9]
   - Nomor indeks dari atribut prediktor = [0..9]
   - Tipe jenis atribut prediktor = [A..Z|a..z]
   - Jumlah field yang ada pada data *input* = [0..9]

   Setiap atribut pada prediktor akan dipisahkan menggunakan karakter titik-koma(;)*.
   Contoh data `data_info`

```
<- prediktor ->
Outlook,0,DISCRETE;Temperature,1,DISCRETE;Windy,3,DISCRETE;Rand,5,NUMERICAL
<- kelas ->
Play,4
<- jumlah field ->
6
```

2. Data `processed_file_info` Isi dari data `processed_file_info` sama dengan data `data_info`. Tetapi, format dan jenis tipe datanya dibedakan sedikit. Contoh data `data_info`

```
<- prediktor ->
[
   {Outlook,0,DISCRETE},
   {Temperature,1,DISCRETE},
   {Windy,3,DISCRETE},
   {Rand,5,NUMERICAL},
]
<- kelas ->
[
   {Play,4},
]
<- jumlah field ->
6
```

3. Data `input(k,v)numeric`
   Format pada data ini akan memiliki format sama dengan data pada `data_input`. Pengecekan akan dilakukan oleh sistem yang dibuat untuk mengenali tipe atribut dari tiap field yang akan diperiksanya.

4. Data `input(k,v)discrete` Format pada data ini akan memiliki format sama dengan data pada `data_input`. Pengecekan akan dilakukan oleh sistem yang dibuat untuk mengenali tipe atribut dari tiap field yang akan diperiksanya.

5. Data `(k',v')class`

   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
   - *Value* terdiri dari:
     (a) Frekuensi kemunculan atribut kelas tersebut = [1](bernilai selalu 1)

6. Data `(k',v')discrete`
   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
     (c) *Attribute Type* = [A..Z|a..z] **required*
     (d) *Predictor Name* = [A..Z|a..z] **required*
     (e) *Predictor Value* = [A..Z|a..z|0..9] **required*
   - *Value* terdiri dari:
     (a) Frekuensi kemunculan dari probabilitas posterior = [1] (bernilai selalu 1).
7. Data `(k',v')numeric`
   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
     (c) *Attribute Type* = [A..Z|a..z] **required*
     (d) *Predictor Name* = [A..Z|a..z] **required*
   - *Value* terdiri dari:
     (a) Nilai dari atribut numerik tersebut = [0..9]

P-Spec (*Process Specification*) pada DFD *level* 2: pada proses 1.0

P-Spec 1.0.0 Membaca file info (Setup)

Deskripsi	Proses ini akan melakukan pembacaan file info dan menyimpan disimpan dalam variabel
Data In	Data info dari kelas konfigurasi milik *hadoop*
Data Out	Data info yang sudah diproses
Proses	• Mengambil data info dari entitas eksternal konfigurasi • Memproses data info agar sesuai dengan kebutuhan sistem

Gambar 1.9: P-Spec training map: pada proses 1.0.0

P-Spec 1.0.1 Mencatat setiap kemunculan probabilitas posterior terhadap kelas dengan jumlah 1

Deskripsi	Untuk atribut bertipe diskrit, proses ini akan melakukan pencatatan setiap kemunculan probabilitas posterior terhadap kelas dengan jumlah 1
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan nama atribut terhadap kelas • Value = jumlah kemunculan = 1
Proses	• Memerika apakah atribut diskrit atau bukan • Jika ya, maka akan dilakukan pencatatan probabilitas posterior tiap data input terhadap kelas dengan jumlah 1

Gambar 1.10: P-Spec training map: pada proses 1.0.1

P-Spec 1.0.2 Mencatat setiap kemunculan probabilitas posterior terhadap kelas dengan nilai atribut tersebut

Deskripsi	Untuk atribut bertipe numerik, proses ini akan melakukan pencatatan untuk setiap kemunculan probabilitas posterior terhadap kelas dengan nilai atribut tersebut
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan nama atribut terhadap kelas • Value = nilai atribut tersebut (ex: 23.52)
Proses	• Memerika apakah atribut numerik atau tidak • Jika ya, maka akan dilakukan pencatatan untuk setiap kemunculan probabilitas posterior terhadap kelas dengan nilai atribut tersebut

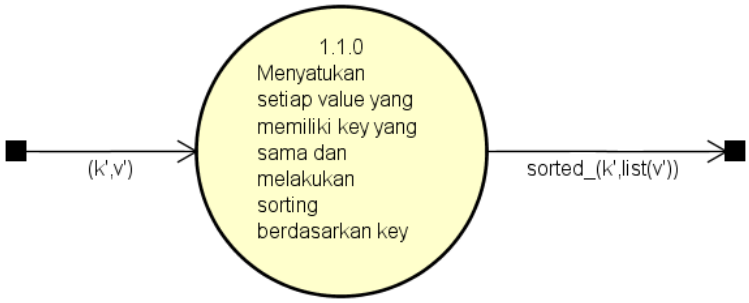
Gambar 1.11: P-Spec training map: pada proses 1.0.2

P-Spec 1.0.3 Mencatat setiap kemunculan atribut kelas dengan value tertentu pada tiap input record

Deskripsi	Untuk atribut bertipe kelas, proses ini akan mencatat setiap kemunculan atribut kelas tersebut pada tiap input record
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan nama atribut kelas dan value dari atribut kelas tersebut • Value = jumlah kemunculan = 1
Proses	• Memerika apakah atribut kelas atau bukan • Jika ya, maka akan dilakukan pencatatan frekuensi kemunculan = 1

Gambar 1.12: P-Spec training map: pada proses 1.0.3

DFD *level* 2: pada proses 1.1



Gambar 1.13: DFD level 2: proses 1.1

***Data Dictionary* pada DFD *level* 2: proses 1.1**

1. Data `(k,v)`
   - *Key* terdiri dari:

(a) *Class Name* = [A..Z|a..z] **required*
(b) *Class Value* = [A..Z|a..z] **required*
(c) *Attribute Type* = [A..Z|a..z] **required*
(d) *Predictor Name* = [A..Z|a..z]
(e) *Predictor Value* = [A..Z|a..z|0..9]

- *Value* terdiri dari:
  (a) Frekuensi kemunculan dari atribut kelas = [1]
  (b) Frekuensi kemunculan dari atribut prediktor = [1]
  (c) Nilai dari atribut numerik = [0..9]

Contoh data `(k,v)`

```
1  key                          value
2  |_class|Play,Yes             1
3  |cont|Rand,Play,Yes            32.5
4  |disc|Humidity,High,Play,No   1
```

2. Data `sorted_(k',list(v'))` Format dari variabel *key* dan *value* sama dengan data pada `(k,v)`. Hanya tipe pada variabel *value* diubah menjadi list.
Contoh data `sorted_(k',list(v'))`

```
1  key                          list_of_value
2  |_class|Play,Yes             [1,1,1,1]
3  |cont|Rand,Play,Yes            [32.5,24.5]
4  |disc|Humidity,High,Play,No   [1,1]
```

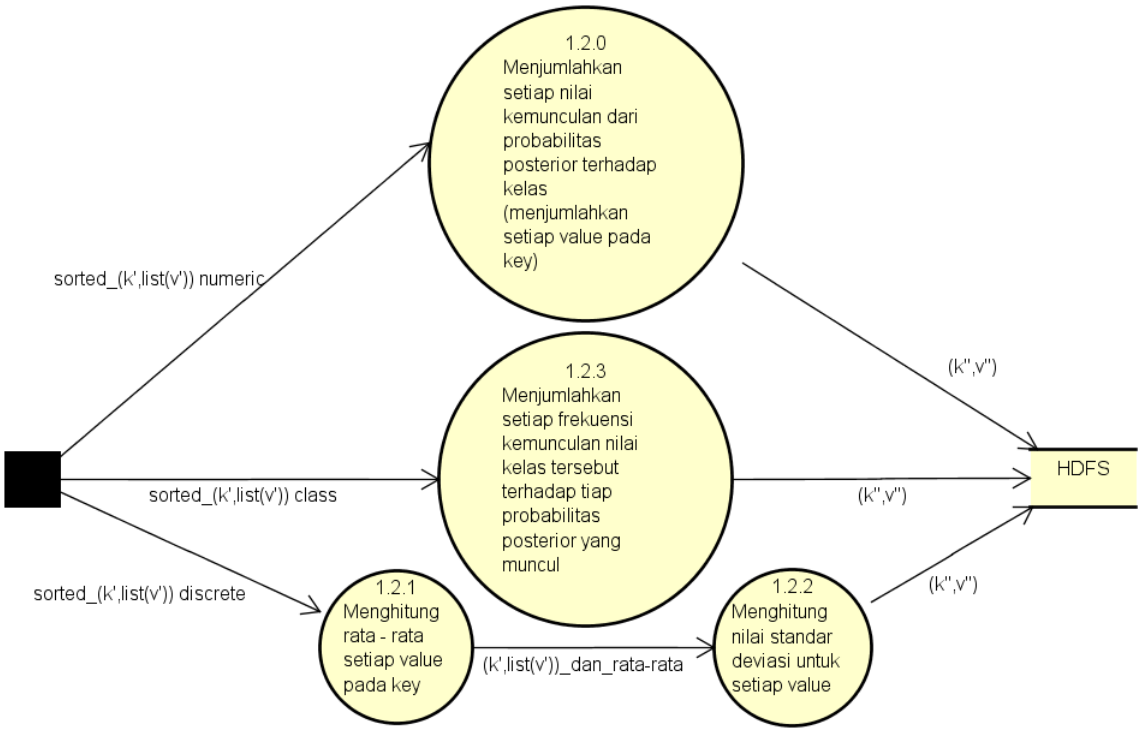
P-Spec (*Process Specification*) pada proses 1.1

P-Spec 1.1.0 Menyatukan setiap value yang memiliki key yang sama dan melakukan sorting berdasarkan key

Deskripsi	Proses ini akan menyatukan setiap value yang memiliki key yang sama dan melakukan sorting berdasarkan key
Data In	1. Pasangan key dan value yang telah diproses pada map
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan nama atribut kelas dan value dari atribut kelas tersebut • Value = Untuk atribut diskrit dan kelas, maka kumpulan dari frekuensi pada tiap record. Untuk atribut numerik, maka nilai dari atribut itu sendiri
Proses	• Akan menyatukan setiap value yang memiliki key sama dan melakukan sorting pada tiap pasangan key dan list of value yang sudah disatukan.

Gambar 1.14: P-Spec training shuffle sort: pada proses 1.1.0

DFD *level* 2: pada proses 1.2



Gambar 1.15: DFD level 2: proses 1.2

***Data Dictionary* pada DFD *level* 2: proses 1.2**

1. Data `sorted_(k',list(v'))numeric`

   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
     (c) *Attribute Type* = [A..Z|a..z] **required*
     (d) *Predictor Name* = [A..Z|a..z] **required*
   - *Value* terdiri dari:
     (a) Nilai dari atribut numerik itu sendiri = [0..9]

   Contoh data `sorted_(k',list(v'))numeric`

```
1  key                    list_of_value
2  |cont|Rand,Play,Yes       [32.5,25.3]
3  |cont|Rand,Play,No        [40.21,54.3]
```

2. Data `sorted_(k',list(v'))discrete`

   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z] **required*
     (b) *Class Value* = [A..Z|a..z] **required*
     (c) *Attribute Type* = [A..Z|a..z] **required*
     (d) *Predictor Name* = [A..Z|a..z] **required*
     (e) *Predictor Value* = [A..Z|a..z] **required*
   - *Value* terdiri dari:

(a) Frekuensi kemunculan = [1]

Contoh data `sorted_(k',list(v'))discrete`

```
key                          list_of_value
|disc|Humidity,High,Play,No      [32.5,25.3]
|disc|Humidity,High,Play,Yes   [40.21,54.3]
```

3. Data `sorted_(k',list(v'))class`

- *Key* terdiri dari:
  (a) *Class Name* = [A..Z|a..z] **required*
  (b) *Class Value* = [A..Z|a..z] **required*
  (c) *Attribute Type* = [A..Z|a..z] **required*
- *Value* terdiri dari:
  (a) Frekuensi kemunculan = [1]

Contoh data `sorted_(k',list(v'))class`

```
key                 list_of_value
|_class|Play,No        [1,1,1,1]
|_class|Play,Yes    [1,1,1,1,1,1,1]
```

4. Data `sorted_(k',list(v'))_dan_rata-rata`

- *Key* terdiri dari:
  (a) *Class Name* = [A..Z|a..z] **required*
  (b) *Class Value* = [A..Z|a..z] **required*
  (c) *Attribute Type* = [A..Z|a..z] **required*
- *Value* terdiri dari:
  (a) Frekuensi kemunculan = [1]
- Rata - rata dari seluruh *list of value* tersebut = [0..9]

Contoh data `sorted_(k',list(v'))_dan_rata-rata`

```
key                          list_of_value rata-rata
|cont|Humidity,High,Play,No      [32.5,25.3]   28.9
|cont|Humidity,High,Play,Yes   [40.21,54.3] 47.255
```

5. Data `(k",v")`

- *Key* terdiri dari:
  (a) *Class Name* = [A..Z|a..z] **required*
  (b) *Class Value* = [A..Z|a..z] **required*
  (c) *Attribute Type* = [A..Z|a..z] **required*
  (d) *Predictor Name* = [A..Z|a..z]
  (e) *Predictor Value* = [A..Z|a..z]
  (f) Frekuensi kemunculan untuk atribut diskrit/kelas = [0..9]
- *Value* untuk atribut numerik terdiri dari:
  (a) *Predictor Value* = [0..9]
  (b) *Attribute Type* = [A..Z|a..z]
  (c) *Mean* = [0..9]

(d) *Sigma* = [0..9]

Contoh data `(k",v")`

```
key                          value
Play,No,3.0|CLASS            (empty-string)
Humidity,Play,Yes               ;82.5|3.5|NUMERIC
Outlook,Sunny,Play,Yes,2.0|DISCRETE (empty-string)
Outlook,Rainy,Play,No,2.0|DISCRETE (empty-string)
```

P-Spec (*Process Specification*) pada proses 1.2

P-Spec 1.2.0 Menjumlahkan setiap nilai kemunculan dari probabilitas posterior terhadap kelas (menjumlahkan setiap value pada key)

Deskripsi	Untuk setiap atribut diskrit, proses ini akan melakukan penjumlahan setiap nilai kemunculan dari probabilitas posterior terhadap kelas (menjumlahkan setiap value pada key)
Data In	pasangan key dan sebuah list dari value yang memiliki key sama, dimana: • key = keterangan nama atribut terhadap kelas • List<value> = kemunculan probabilitas tersebut
Data Out	Pasangan key dan value baru, dimana : • Key = keterangan nama dan tipe atribut terhadap kelas • Value = jumlah seluruh kemunculan
Proses	1. Periksa atribut apakah diskrit 2. Jika ya, maka lakukan penjumlahan terhadap tiap value pada list of value yang menjadi input 3. Masukkan nilai penjumlahan ke dalam value yang baru

Gambar 1.16: P-Spec training reduce: pada proses 1.2.0

P-Spec 1.2.1 Menghitung rata - rata setiap value pada key

Deskripsi	Untuk setiap atribut numerik, proses ini akan melakukan perhitungan rata - rata untuk setiap value dalam list
Data In	• list dari value yang diberikan dari input • rata - rata dari list of value tersebut
Data Out	Pasangan key dan value baru, dimana : • Key = keterangan nama dan tipe atribut terhadap kelas • Value = jumlah seluruh kemunculan
Proses	1. Periksa atribut apakah numerik 2. Jika ya, maka lakukan perhitungan rata rata untuk setiap value di dalam list of value yang menjadi input

Gambar 1.17: P-Spec training reduce: pada proses 1.2.1

P-Spec 1.2.2 Menghitung nilai standar deviasi untuk setiap value

Deskripsi	Proses ini akan melakukan perhitungan standar deviasi dari tiap value di dalam list of value dan rata - rata yang sudah dihitung sebelumnya
Data In	• list dari value yang diberikan dari input • rata - rata dari list of value tersebut
Data Out	Pasangan key dan value baru, dimana : • Key = keterangan nama dan tipe atribut terhadap kelas • Value = jumlah seluruh kemunculan
Proses	1. Lakukan perhitungan standar deviasi untuk setiap value di dalam list of value yang menjadi input dengan rata - rata yang sudah dihitung sebelumnya

Gambar 1.18: P-Spec training reduce: pada proses 1.2.2

P-Spec 1.2.3 Menjumlahkan setiap frekuensi kemunculan nilai kelas tersebut terhadap tiap probabilitas posterior yang muncul

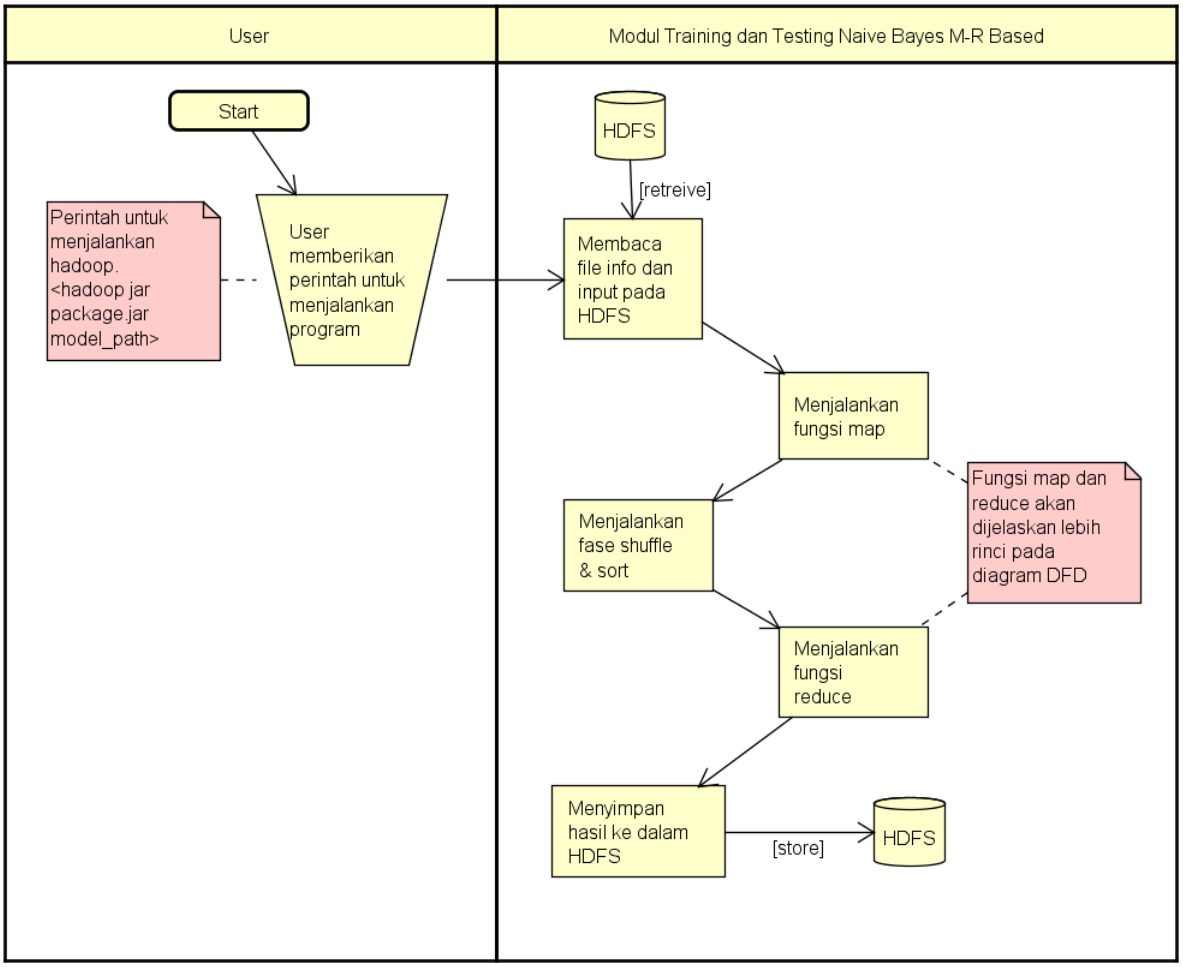
Deskripsi	Proses ini akan menjumlahkan setiap frekuensi kemunculan nilai kelas tersebut terhadap tiap probabilitas posterior yang muncul
Data In	1. Pasangan key dan value yang telah diproses pada map yang bertipe kelas
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan nama atribut kelas dan value dari atribut kelas tersebut • Value = jumlah frekuensi dari atribut nilai kelas tersebut
Proses	• Akan menyatukan setiap value yang memiliki key sama dan melakukan penjumlahan untuk setiap value pada key tersebut

Gambar 1.19: P-Spec training reduce: pada proses 1.2.3

1.5.1.3 Modul *Testing Naive Bayes M-R Based*

Pada modul ini, program akan memanfaatkan model klasifikasi naive bayes yang telah dibuat sebelumnya untuk melakukan klasifikasi pada data testing yang telah ada sebelumnya di HDFS (pada modul input) dan memberikan laporan analisis mengenai tingkat akurasi dan tingkat error yang dimiliki oleh model terhadap data tersebut.

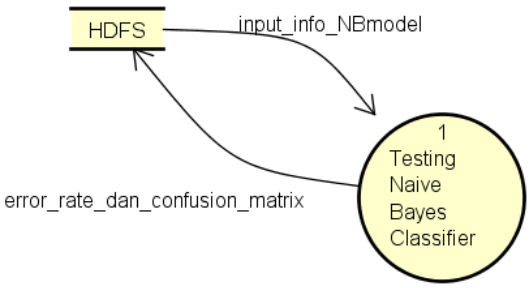
Berikut merupakan diagram *flow chart* untuk modul *testing*:



Gambar 1.20: Flow Chart Modul Testing

Sama seperti pada modul *training*, untuk proses yang berbasis *MapReduce* pada modul ini juga perlu digambarkan menggunakan DFD, agar bisa tergambarkan lebih rinci mengenai detail proses tersebut. Berikut merupakan *context diagram*[4] dan DFD untuk proses *MapReduce* pada modul *testing*:

Context Diagram Modul *Testing*



Gambar 1.21: *Context diagram* modul Testing

Data Dictionary Context Diagram Modul *Testing*

1. Data `input_info_NBmodel` terdiri dari input dan NBC (*Naive Bayes Classifier*) model.

[4] *Context Diagram biasa disebut juga sebagai DFD level 0.*

- *Input* terdiri dari:
  (a) Nilai dari atribut kelas pada input = [A..Z|a..z]
  (b) Nilai dari atribut prediktor bertipe diskrit pada input = [A..Z|a..z]
  (c) Nilai dari atribut prediktor bertipe numerik pada input = [0..9]
- NBC model terdiri dari:
  (a) *Class Name* = [A..Z|a..z|]
  (b) *Class Value* = [A..Z|a..z|]
  (c) *Attribute Type* = [A..Z|a..z|]
  (d) *Predictor Name* = [A..Z|a..z|]
  (e) *Predictor Value* = [A..Z|a..z|]
  (f) Frekuensi kemunculan untuk tiap atribut kelas = [A..Z|a..z|0..9]
  (g) Frekuensi kemunculan untuk tiap atribut prediktor diskrit = [A..Z|a..z|0..9]
  (h) Nilai mean dari atribut prediktor numerik = [0..9]
  (i) Nilai sigma/standard-deviasi dari atribut prediktor numerik = [0..9]

Contoh data `input_info_NBmodel`

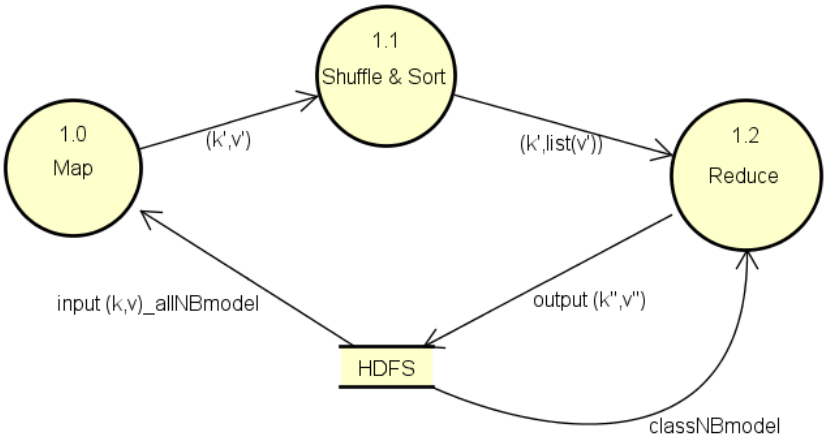
```
<- input ->
Sunny,Mild,Normal,FALSE,Yes,5
Rainy,Mild,Normal,TRUE,Yes,4.5
Overcast,Mild,High,TRUE,Yes,3.1
<- NBmodel ->
Play,Yes,2.0|CLASS
Play,No,3.0|CLASS
Humidity,Play,Yes ;82.5|3.5|NUMERIC
Humidity,Play,No ;71.0|9.6|NUMERIC
Outlook,Sunny,Play,Yes,2.0|DISCRETE
Outlook,Sunny,Play,No,1.0|DISCRETE
Outlook,Rainy,Play,No,2.0|DISCRETE
```

2. Data `error_rate_dan_confusion_matrix`

- Nama kelas = [A..Z|a..z]
- *Confusion Matrix* untuk tiap kelas = matrix $n * m$
- *Error rate* untuk *Accuracy* untuk tiap kelas = [0..9]
- *Error rate* untuk *Recall* untuk tiap *class value* = [0..9]
- *Error rate* untuk *Precision* untuk tiap *class value* = [0..9]
- *Error rate* untuk $F - Measure$ untuk tiap *class value* = [0..9]

```
@play
####
|     | no | yes |
| no  | 3  | 0     |
| yes | 0  | 2     |
####
Accuracy: 5/5 = 1.0
*For Value = no
Precision: -> 3 / 3 + 0 = 1.0
Recall: -> 3 / 3 + 0 = 1.0
*For Value = yes
Precision: -> 2 / 2 + 0 = 1.0
Recall: -> 2 / 2 + 0 = 1.0
F-Measure -> 0.8
```

DFD *level* 1



Gambar 1.22: DFD level 0 modul Testing

***Data Dictionary* pada DFD *level* 1**

1. Data `input(k,v)_allNBmodel`
   - *Key* pada `input(k,v)` = *NULL* (karena memang pada awal proses *mapreduce* key pada input belum terdefinisi)
   - *Value* pada `input(k,v)` terdiri dari:
     (a) Nilai tiap atribut prediktor diskrit pada input = [A..Z|a..z|]
     (b) Nilai tiap atribut prediktor numerik pada input = [0..9]
     (c) Nilai tiap atribut kelas pada input = [A..Z|a..z]
   - *allNBmodel* yang merupakan model dari NBC memiliki format sama dengan NBC model yang terdapat pada data `input_info_NBmodel`.
2. Data `(k',v')`
   - *Key* yang merupakan nama atribut kelas = [A..Z|a..z]
   - *Value* terdiri dari:
     (a) *Class Name* = [A..Z|a..z]
     (b) *Class Value Predicted* = [A..Z|a..z]
     (c) *Class Value Actual* = [A..Z|a..z]
     (d) *Percentage* = [0..9]

   Contoh data `(k',v')`

```
key        value
Play    Play|predicted=Yes|percentage=67.5%|actual=Yes
Play    Play|predicted=Yes|percentage=51.1%|actual=No
Play    Play|predicted=No|percentage=96.32%|actual=No
```

3. Data `(k',list(v'))`
   format key dan value pada data ini sama dengan data `(k',v')`. Hanya saja, untuk variabel *value*-nya dijadikan sebuah list untuk setiap nama variabel *key* yang sama.
   Contoh data `(k',list(v'))`

```
key         list_of_value
Play    [
            {Play|predicted=Yes|percentage=67.5%|actual=Yes},
            {Play|predicted=Yes|percentage=51.1%|actual=No},
            {Play|predicted=No|percentage=96.32%|actual=No},
        ]
```

4. Data `output(k",v")`
   - *Key* terdiri dari:
     (a) *Class Name* = [A..Z|a..z]
     (b) *Confusion Matrix* untuk tiap kelas = matrix $m * n$
   - *Value* terdiri dari:
     (a) *Error rate* untuk *Accuracy* untuk tiap kelas = [0..9]
     (b) *Error rate* untuk *Recall* untuk tiap *class value* = [0..9]
     (c) *Error rate* untuk *Precision* untuk tiap *class value* = [0..9]
     (d) *Error rate* untuk $F - Measure$ untuk tiap *class value* = [0..9]

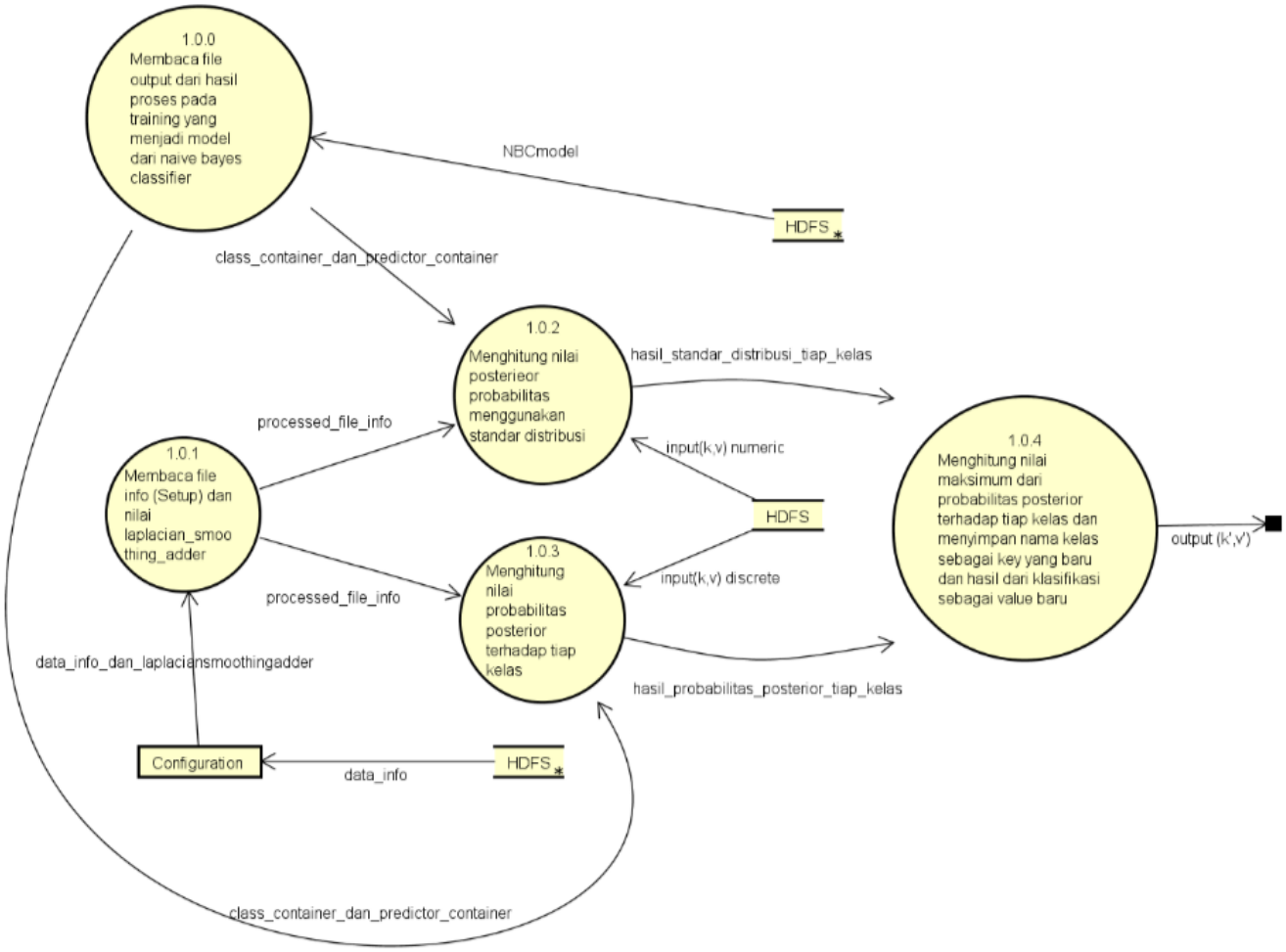
   Contoh data `output(k",v")`

```
<- Key ->
@play
####
|     | no | yes |
| no  | 3  | 0    |
| yes | 0  | 2    |
####
<- Value ->
Accuracy: 5/5 = 1.0
*For Value = no
Precision: -> 3 / 3 + 0 = 1.0
Recall: -> 3 / 3 + 0 = 1.0
*For Value = yes
Precision: -> 2 / 2 + 0 = 1.0
Recall: -> 2 / 2 + 0 = 1.0
F-Measure -> 0.8
```

5. Data `classNBmodel` yang merupakan model dari NBC memiliki format hampir sama dengan NBC model yang terdapat pada data `input_info_NBmodel`. Bedanya, data ini hanya mengambil model yang bertipe atribut kelas saja untuk digunakan dalam menghitung *confusion matrix*.
   Contoh data `classNBmodel`:

```
Play,Yes,2.0|CLASS
Play,No,3.0|CLASS
```

DFD *level* 2: pada proses 1.0



Gambar 1.23: DFD level 2: proses 1.0

P-Spec (*Process Specification*) pada proses 1.0

1. Data `NBCmodel` yang merupakan model dari NBC memiliki format sama dengan NBC model yang terdapat pada data `input_info_NBmodel` di *context diagram*.
2. Data `class_contanier_dan_predictor_container`
   - *Class Name* = [A..Z|a..z]
   - *Class Value* = [A..Z|a..z]
   - *Predictor Name* = [A..Z|a..z]
   - *Predictor Value* = [A..Z|a..z|0..9]
   - *Attribute Type* = [A..Z|a..z]
   - *Mean* = [0..9]
   - *Sigma* = [0..9]
3. Data `data_info`
   - Nama tiap field = [A..Z|a..z]
   - Nomor index tiap field = [0..9]
   - Tipe tiap field = [A..Z|a..z]

4. Data `data_info_dan_laplaciansmoothingadder`
   Data ini merupakan data yang sama pada data `data_info`, tetapi ditambahkan nilai *laplaciansmoothingadder* sebagai counter untuk penambahan tiap frekuensi probabilitas posterior untuk menghindari terjadinya permasalahan *zero-frequency*.

5. Data `processed_file_info` Data ini merupakan data yang terdapat pada data `data_info_dan_laplacia`
   hanya saja formatnya dibuat untuk memudahkan perangkat lunak yang nantinya dibuat membaca file info tersebut.

6. Data `input(k,v)numeric`

   - *Key* pada `input(k,v)numeric` = *NULL* (karena memang pada awal proses *mapreduce* key pada input belum terdefinisi)
   - *Value* pada `input(k,v)` terdiri dari:
     (a) Nilai tiap atribut prediktor numerik pada input = [0..9]
     (b) Nilai tiap atribut kelas pada input = [A..Z|a..z]

7. Data `input(k,v)discrete`

   - *Key* pada `input(k,v)discrete` = *NULL* (karena memang pada awal proses *mapreduce* key pada input belum terdefinisi)
   - *Value* pada `input(k,v)` terdiri dari:
     (a) Nilai tiap atribut prediktor diskrit pada input = [A..Z|a..z]
     (b) Nilai tiap atribut kelas pada input = [A..Z|a..z]

8. Data `hasil_standar_distribusi_tiap_kelas` merupakan nilai dari standard distribusi untuk probabilitas posterior tiap atribut numerik terhadap tiap kelas yang ada.

   - *Class Name* = [A..Z|a..z]
   - *Class Value* = [A..Z|a..z]
   - *Predictor Name* = [A..Z|a..z]
   - *Attribute Type* = [A..Z|a..z]
   - Nilai standar distribusi = [0..9]

9. Data `hasil_probabilitas_posterior_tiap_kelas` merupakan nilai hasil dari probabilitas posterior dari tiap atribut pada tiap atribut kelas yang ada.

   - *Class Name* = [A..Z|a..z]
   - *Class Value* = [A..Z|a..z]
   - *Predictor Name* = [A..Z|a..z]
   - *Predictor Value* = [A..Z|a..z]
   - *Attribute Type* = [A..Z|a..z]
   - Hasil nilai probabilitas posterior dari tiap atribut prediktor terhadap tiap kelas = [0..9]

10. Data `output(k',v')`

    - *Key* merupakan nama atribut kelas = [A..Z|a..z]
    - *Value* terdiri dari:
      (a) *Class Name* = [A..Z|a..z]
      (b) *Class Value Predicted* = [A..Z|a..z]
      (c) *Class Actual* = [A..Z|a..z]
      (d) *Percentage* = [0..9]

Contoh data `output(k',v')`:

```
key        value
Play    Play|predicted=Yes|percentage=67.5%|actual=Yes
Play    Play|predicted=Yes|percentage=51.1%|actual=No
Play    Play|predicted=No|percentage=96.32%|actual=No
```

P-Spec 1.0.0 Membaca file output dari hasil proses pada training yang menjadi model dari naive bayes classifier dan memasukkannya ke dalam kelas kontainer dan prediktor kontainer

Deskripsi	Proses ini akan membaca file output dari hasil proses pada training yang menjadi model dari naive bayes classifier dan untuk setiap tipe model memasukkannya ke kelas kontainer dan prediktor kontainer
Data In	1. Data model *naive bayes classifier* yang telah dibuat pada modul training
Data Out	1. *Class container* yang berisi mengenai seluruh atribut kelas beserta dengan frekuensi kemunculannya 2. *Predictor container* yang berisi mengenai seluruh detail dari atribut prediktor
Proses	• Membaca model *naive bayes classifier* pada HDFS • Melakukan pemilahan data untuk atribut prediktor dan kelas • Memasukkan model ke dalam kelas kontainer dan prediktor kontainer

Gambar 1.24: P-Spec training reduce: pada proses 1.0.0

P-Spec 1.0.1 Membaca file info (Setup)

Deskripsi	Proses ini akan melakukan pembacaan file info dan menyimpan disimpan dalam variabel
Data In	1. Data info dari kelas konfigurasi milik *hadoop* 2. Nilai *laplacian_smoothing_adder* yang akan digunakan untuk melakukan smoothing data
Data Out	Data info yang sudah diproses
Proses	• Mengambil data info dari entitas eksternal konfigurasi • Memproses data info agar sesuai dengan kebutuhan sistem

Gambar 1.25: P-Spec training reduce: pada proses 1.0.1

P-Spec 1.0.2 Menghitung nilai posterieor probabilitas menggunakan standar distribusi

Deskripsi	Untuk atribut bertipe numerik, proses ini akan melakukan perhitungan nilai probabilitas posterior menggunakan standar distribusi
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Hasil standar distribusi untuk probabilitas posterior tiap kelas
Proses	• Memerika apakah numerikdiskrit atau bukan • Jika ya, maka akan dilakukan perhitungan probabilitas posterior menggunakan standar distribusi

Gambar 1.26: P-Spec training reduce: pada proses 1.0.2

P-Spec 1.0.3 Menghitung nilai probabilitas posterior terhadap tiap kelas

Deskripsi	Untuk atribut bertipe diskrit, proses ini akan melakukan probabilitas posterior terhadap tiap kelas
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Hasil probabilitas tiap kelas
Proses	• Memerika apakah atribut diskrit atau tidak • Jika ya, maka akan dilakukan perhitungan probabilitas posterior terhadap tiap kelas

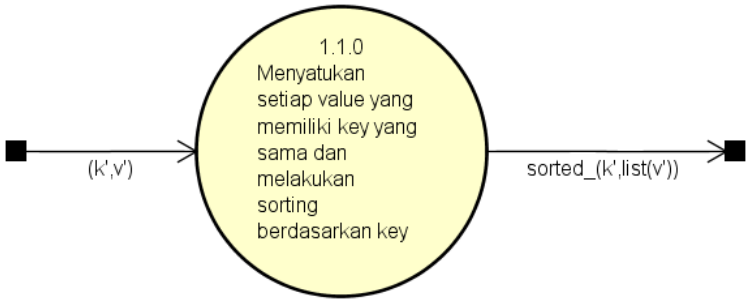
Gambar 1.27: P-Spec training reduce: pada proses 1.0.3

P-Spec 1.0.4 Menghitung nilai maksimum dari probabilitas posterior terhadap tiap kelas dan menyimpan nama kelas sebagai key yang baru dan hasil dari klasifikasi sebagai value baru

Deskripsi	Proses ini akan melakukan perhitungan nilai maksimum dari probabilitas posterior terhadap tiap kelas dan menyimpan nama kelas sebagai key yang baru dan hasil dari klasifikasi sebagai value baru
Data In	1. Data info yang telah diproses oleh proses sebelumnya 2. Data input berupa pasangan *key* dan *value*
Data Out	Pasangan *key* dan *value* dimana: 1. key = nama kelas 2. value = keterangan nilai kelas aktual, presentase, dan nilai kelas hasil prediksi
Proses	• Untuk setiap kelas, lakukan perkalian hasil standar distribusi dan seluruh hasil probabilitas posterior tiap kelas • Lalu, kalikan nilai tersebut dengan probabilitas prior kelas tersebut • Cari nilai maksimum untuk tiap kelas

Gambar 1.28: P-Spec training reduce: pada proses 1.0.4

DFD *level* 2: pada proses 1.1

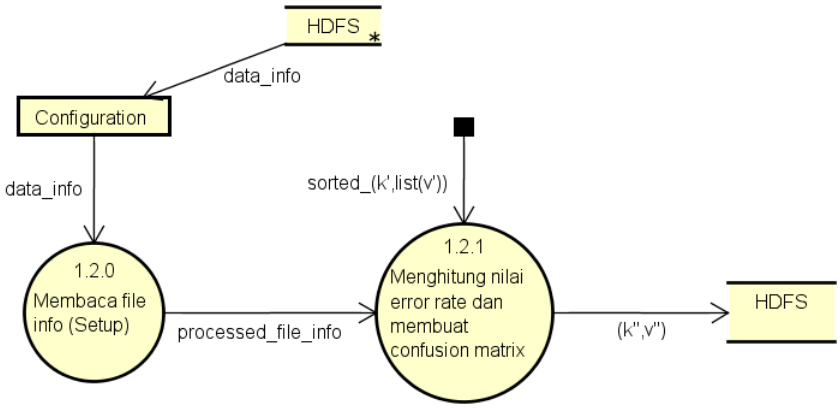


Gambar 1.29: DFD level 2: proses 1.1

Data Dictionary pada DFD *level* 2: proses 1.1

1. Data `(k',v')` memiliki format yang sama dengan data `output(k',v')` pada proses 1.0
2. Data `sorted(k',list(v'))` memiliki format yang sama dengan *value* pada data `output(k',v')`. Hanya saja value yang ini baru merupakan kumpulan dari *value* yang memiliki *key*(nama kelas) yang sama.

DFD *level* 2: pada proses 1.2



Gambar 1.30: DFD level 2: proses 1.2

Data Dictionary pada DFD *level* 2: proses 1.2

1. Data `data_info` memiliki format dan isi yang sama dengan data `data_info` pada proses 1.0
2. Data `processed_file_info` memiliki format dan isi yang mirip dengan data `processed_file_info` pada proses 1.0. Hanya saja tidak mengikutsertakan nilai *laplaciansmoothingadder*.
3. Data `sorted_(k',list(v'))` memiliki format dan isi yang sama dengan data `sorted_(k',list(v'))` pada proses 1.1.
4. Data `(k'',v'')`
   (a) *Key* terdiri dari:
   - *Class Name* = [A..Z|a..z]

- *Confusion Matrix* = matrix $n * m$

(b) *Value* terdiri dari:

- *Confusion Matrix* untuk tiap kelas = matrix $n * m$
- *Error rate* untuk *Accuracy* untuk tiap kelas = [0..9]
- *Error rate* untuk *Recall* untuk tiap *class value* = [0..9]
- *Error rate* untuk *Precision* untuk tiap *class value* = [0..9]
- *Error rate* untuk $F - Measure$ untuk tiap *class value* = [0..9]

Contoh data `output(k",v")`

```
<- Key ->
@play
####
|     |  no | yes |
| no  |  3  | 0     |
| yes | 0   | 2     |
####
<- Value ->
Accuracy: 5/5 = 1.0
*For Value = no
Precision: -> 3 / 3 + 0 = 1.0
Recall: -> 3 / 3 + 0 = 1.0
*For Value = yes
Precision: -> 2 / 2 + 0 = 1.0
Recall: -> 2 / 2 + 0 = 1.0
F-Measure -> 0.8
```

P-Spec (*Process Specification*) pada proses 1.0

P-Spec 1.2.0 Membaca file info (Setup)

Deskripsi	Proses ini akan melakukan pembacaan file info dan menyimpan disimpan dalam variabel
Data In	Data info dari kelas konfigurasi milik *hadoop*
Data Out	Data info yang sudah diproses
Proses	• Mengambil data info dari entitas eksternal konfigurasi • Memproses data info agar sesuai dengan kebutuhan sistem

Gambar 1.31: P-Spec testing reduce: pada proses 1.2.0

P-Spec 1.2.1 Menghitung nilai error rate dan membuat confusion matrix

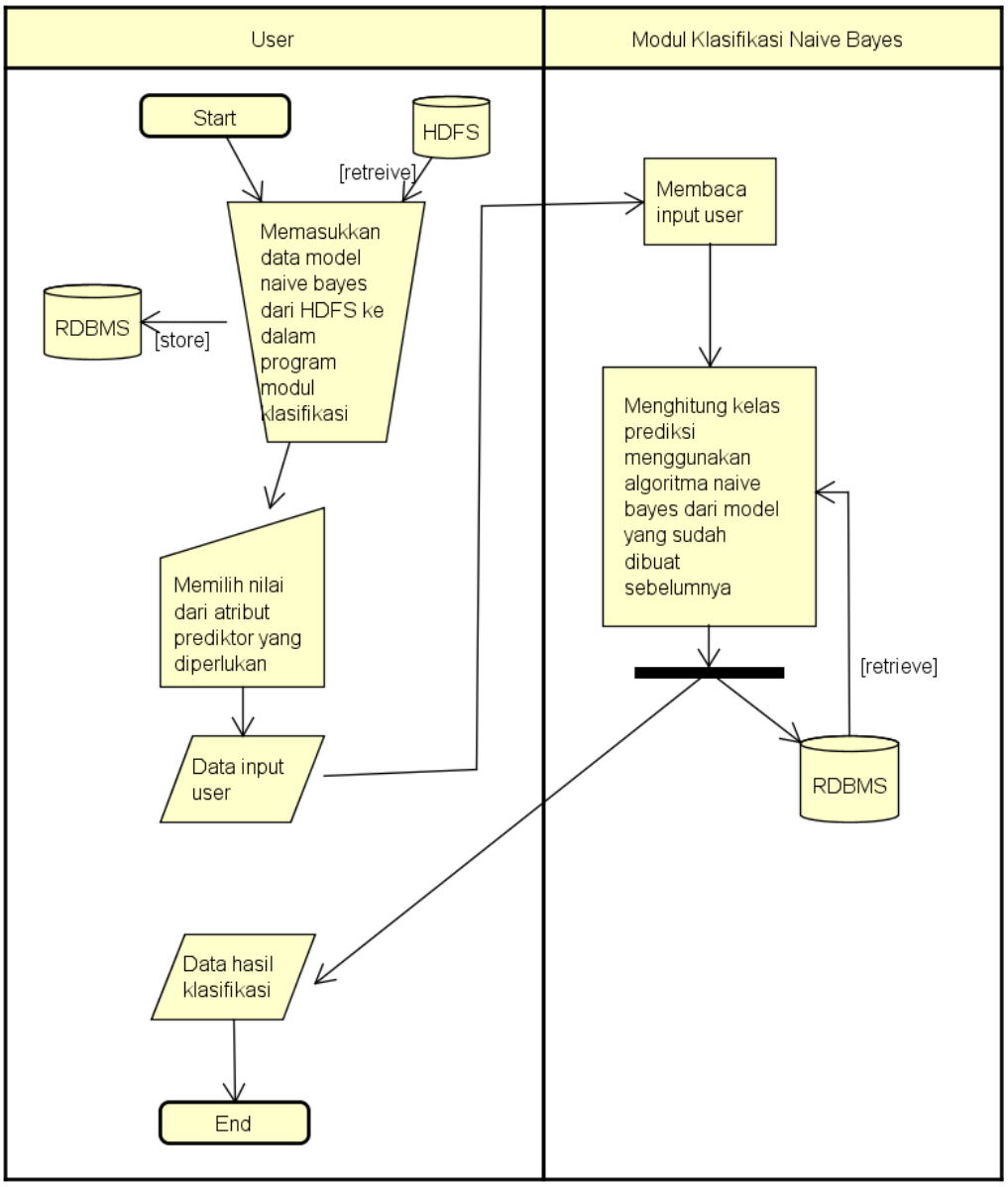
Deskripsi	Proses ini akan melakukan perhitungan nilai error rate dan membuat confusion matrix
Data In	• File info yang telah diproses • pasangan *key* dan *value*, dimana: key = nama kelas ; List<value> = kumpulan hasil keterangan prediksi terhadap kelas tersebut
Data Out	Pasangan key dan value baru, dimana : • Key = confusion matrix terhadap kelas tersebut • Value = perhitungan seluruh error rate dari hasil terhadap kelas tersebut
Proses	1. Untuk setiap list of value dalam key yang sama, akan dilakukan perhitungan confusion matrix 2. Melakukan perhitungan untuk error rate : Accuracy 3. Melakukan perhitungan untuk error rate : Precision untuk tiap nilai kelas 4. Melakukan perhitungan untuk error rate : Recall untuk tiap nilai kelas 5. Melakukan perhitungan untuk error rate : F-Measure untuk tiap nilai kelas

Gambar 1.32: P-Spec testing reduce: pada proses 1.2.1

1.5.1.4 Modul Klasifikasi *Naive Bayes*

Pada modul ini, program juga akan memanfaatkan model klasifikasi naive bayes yang telah dibuat sebelumnya untuk melakukan klasifikasi. Program pada modul ini dapat menerima 1 jenis input yang merupakan input manual secara satu - persatu atribut yang diperlukan untuk melakukan klasifikasi (*predict new case*).

Berikut merupakan diagram *flow chart* untuk modul klasifikasi:

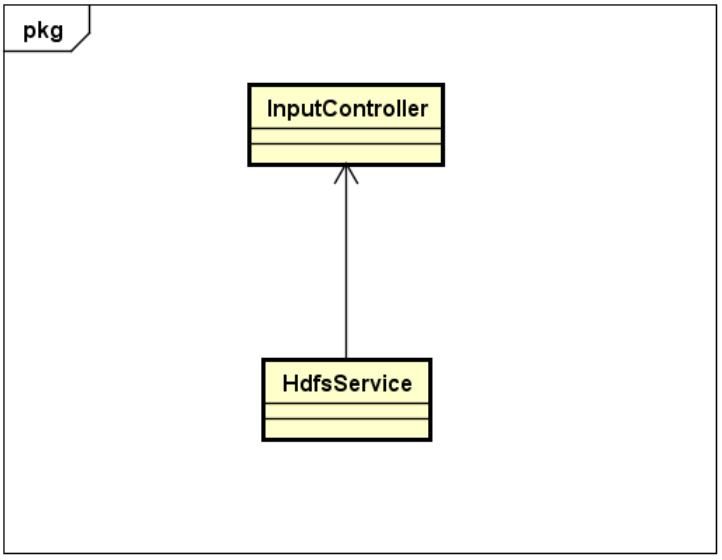


Gambar 1.33: Flow Chart Modul Klasifikasi

1.5.2 Diagram Kelas

Perangkat lunak yang dibangun akan mengikuti metode pemrograman berbasis objek (*Object Oriented Programming*). Sehingga, untuk melakukan pemodelan pada perangkat lunak yang dibuat akan menggunakan kelas yang memiliki beberapa atribut dan metode operasi. Berikut merupakan gambaran diagram kelas pada perangkat lunak untuk setiap modul.

1.5.2.1 Modul Kelola *Input*



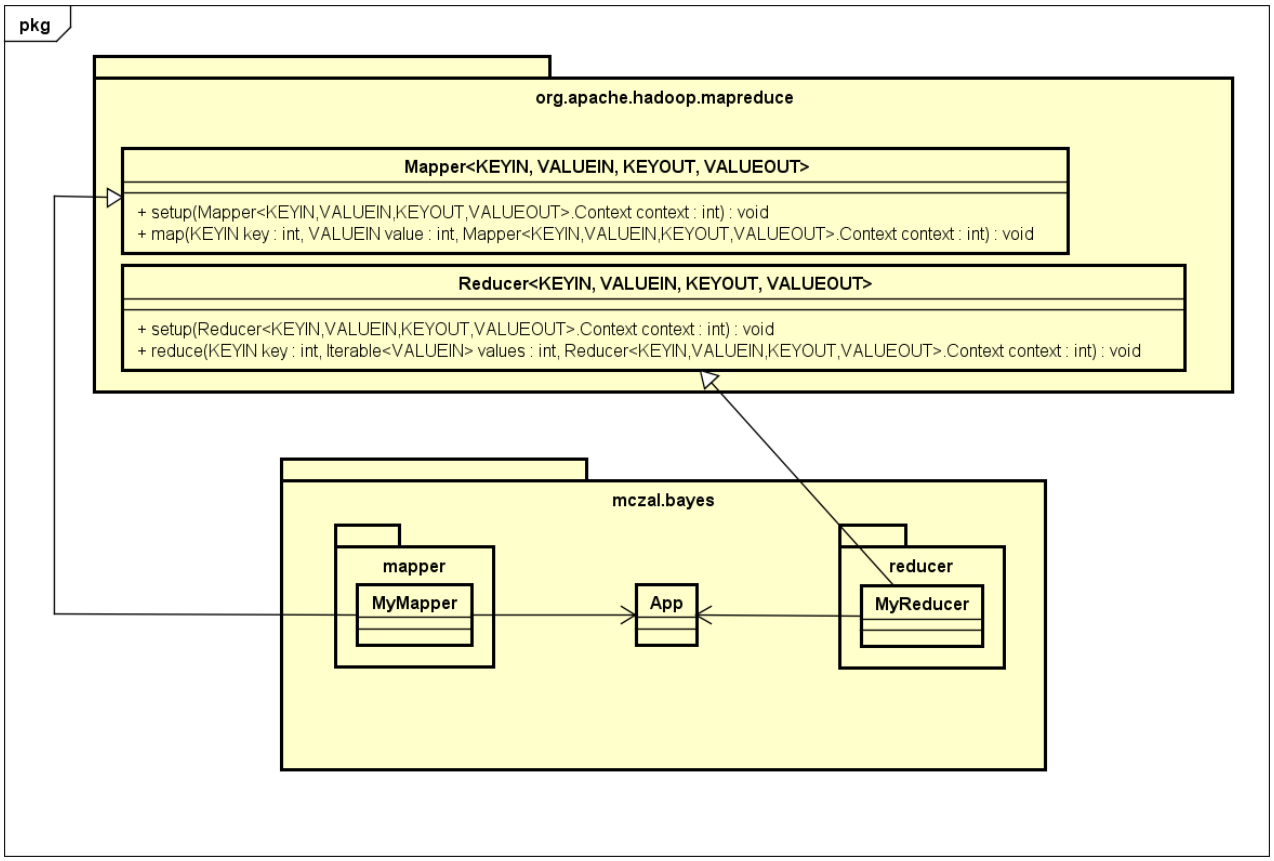
Gambar 1.34: Diagram kelas modul kelola input

Pada modul ini, akan dibuatkan 2 kelas utama untuk menangani proses memasukkan file input ke dalam HDFS. Kelas tersebut diantaranya adalah:

1. *InputController.class*
   Kelas ini akan menjadi sebagai kelas yang meng-enkapsulasi seluruh proses penting yang dibutuhkan untuk memasukan file input ke dalam HDFS. Kelas ini hanya membuatkan satu method untuk melakukan operasi input yang akan diakses oleh user. Kelas ini akan memiliki objek instansiasi dari kelas *HdfsService.class* dan memanggil beberapa method di dalamnya untuk melakukan operasi penulisan ke dalam HDFS.

2. HdfsService.class
   Kelas ini akan mengatur segala kebutuhan yang diperlukan untuk melakukan proses penulisan ke dalam HDFS. Kelas ini akan memiliki koneksi terhadap HDFS Master sebagai *hadoop client* untuk memerintahkan penulisan dan pendistribusian file baru yang akan dimasukkan ke dalam HDFS.

1.5.2.2 Modul *Training dan Testing Naive Bayes M-R Based*



Gambar 1.35: Diagram kelas modul *training* dan *testing*

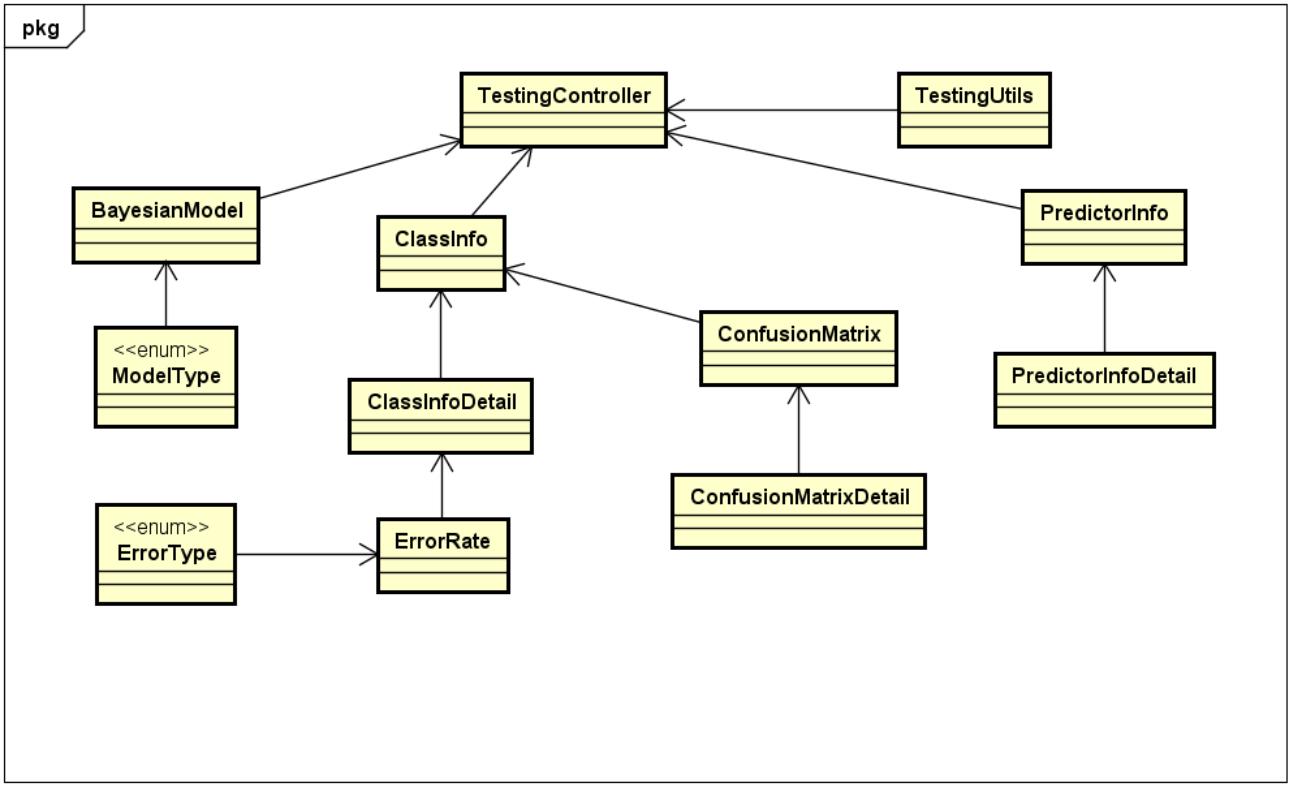
Pada modul ini, akan dibuatkan 3 kelas utama untuk menangani proses *training* dan *testing* berbasis *MapReduce*. Kelas tersebut diantaranya adalah:

1. App.class
   Kelas ini akan menjadi kelas utama yang akan menjalankan operasi *testing* maupun *training* yang berbasis *MapReduce*. Pada kelas ini akan ditentukan pula kelas mana saja yang akan dijadikan sebagai kelas mapper dan kelas reducer nya, begitu juga dengan pasangan *key* dan *value* untuk *input* dan untuk *output* di setiap kelas.

2. MyMapper.class
   Kelas ini akan menjalankan operasi pada fase map untuk proses training dan testing berbasis *MapReduce*.

3. MyReducer.class
   Kelas ini akan menjalankan operasi pada fase reduce untuk proses training dan testing berbasis *MapReduce*.

1.5.2.3 Modul Klasifikasi *Naive Bayes*



Gambar 1.36: Diagram kelas modul klasifikasi *naive bayes*

Pada modul ini, akan dibuatkan 10 kelas utama dan 2 kelas yang bertipe emum untuk menangani proses klasifikasi menggunakan model yang telah dibuat sebelumnya. Kelas tersebut diantaranya adalah:

1. *TestingController.class*
   Kelas ini merupakan kelas utama yang akan melakukan enkapsulasi seluruh proses penting yang akan dijalankan pada proses klasifikasi.

2. *TestingUtils.class*
   Kelas ini akan menjadi kelas-pembantu pada kelas *TestingController.class* untuk melakukan operasi - operasi yang dibutuhkan pada algoritma klasifikasi *naive bayes*. Seperti perhitungan probabilitas posterior dan normal distribusi

3. *BayesianModel.class*
   Kelas ini merupakan kelas utama untuk merepresentasikan model klasifikasi *naive bayes* yang telah dibuat sebelumnya.

4. *ModelType.enum*
   Enum ini akan menjadi tipe untuk setiap model yang ada pada kelas *BayesianModel*. Enum tersebut terdiri antara: DISCRETE, NUMERIC, dan CLASS.

5. *ClassInfo.class*
   Kelas ini akan merepresentasikan seluruh atribut kelas pada model klasifikasi *naive bayes* yang telah dibuat sebelumnya.

6. *ClassInfoDetail.class*
   Kelas ini merupakan ekstensi dari kelas *ClassInfo.class*. Kelas ini akan menyimpan seluruh detail mengenai atribut kelas tertentu pada model *naive bayes* yang sudah jadi.

7. *ErrorRate.class*
   Kelas ini akan merepresentasikan perhitungan *ErrorRate* yang dapat dihitung setelah melakukan testing terhadap model *naive bayes* yang sudah jadi sebelumnya.

8. *ErrorType.enum*
   Enum ini akan menjadi tipe untuk tiap error yang ada. Enum tersebut terdiri dari: $ACCURACY$, $PRECISION$, $RECALL$, dan $F_MEASURE$.

9. *ConfusionMatrix.class*
   Kelas ini akan merepresentasikan *ConfusionMatrix* yang akan diperoleh setelah menjalani testing/klasifikasi pada model *naive bayes* yang sudah jadi sebelumnya.

10. *ConfusionMatrixDetail.class*
    Kelas ini merupakan ekstensi dari kelas *ConfusionMatrix.class*. Kelas ini akan menyimpan seluruh detail yang dimiliki oleh tiap instansiasi dari kelas *ConfusionMatrix*.

11. *PredictorInfo.class*
    Kelas ini akan merepresentasikan sebagai seluruh atribut prediktor yang digunakan pada model *naive bayes*.

12. *PredictorInfoDetail.class*
    Kelas ini merupakan kelas ekstensi dari kelas *PredictorInfo.class*. Kelas ini akan menyimpan seluruh detail pad tiap instansiasi dari kelas *PredictorInfo*.

BAB 2

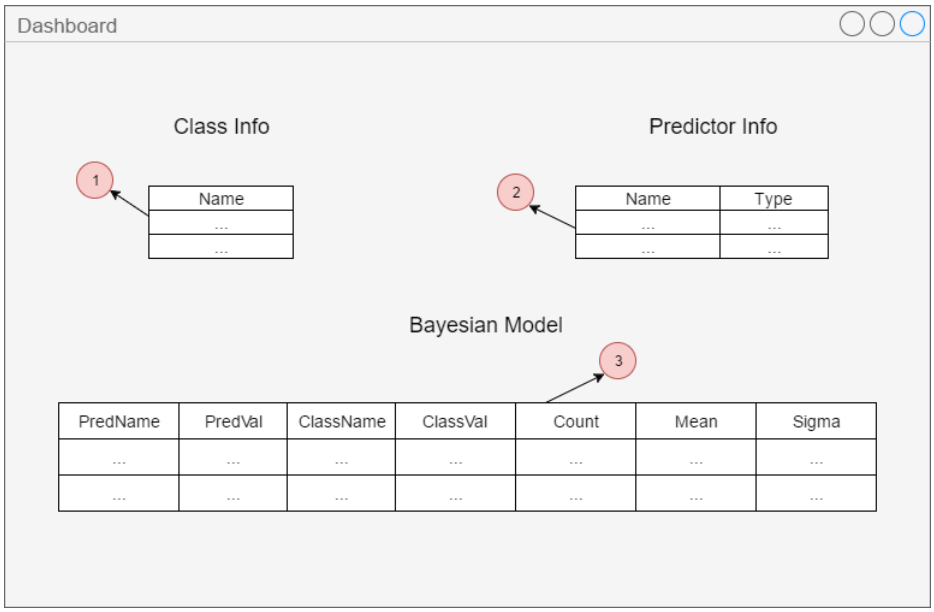
PERANCANGAN

Berdasarkan analisis yang telah dilakukan, terdapat beberapa hal yang perlu dirancang untuk pembangunan perangkat lunak naive bayes berbasis *hadoop mapreduce*. Pada bab ini akan dijelaskan perancangan yang diperlukan untuk membangun perangkat lunak yaitu perancang- an antarmuka, diagram kelas rinci, serta rincian metode.

2.1 Perancangan Antarmuka

Perangkat lunak *naive bayes classification* memiliki 6 buah tampilan untuk yang tidak berbasis *MapReduce*, yaitu: (1) *Dashboard* (2) *Input Set Manager* (3) *Renew Model Manager* (4) *Testing Manager* (5) *Classification Manager* (6) *Error Rate Dashboard*. Untuk program yang berbasis *MapReduce* tidak akan memiliki antarmuka yang khusus, karena program hanya perlu dijalankan dengan menggunakan CLI (*command line interface*). Berikut adalah penjelasan dan gambar dari tiap antarmuka yang dirancang:

2.1.1 *Dashboard*



Gambar 2.1: Dashboard

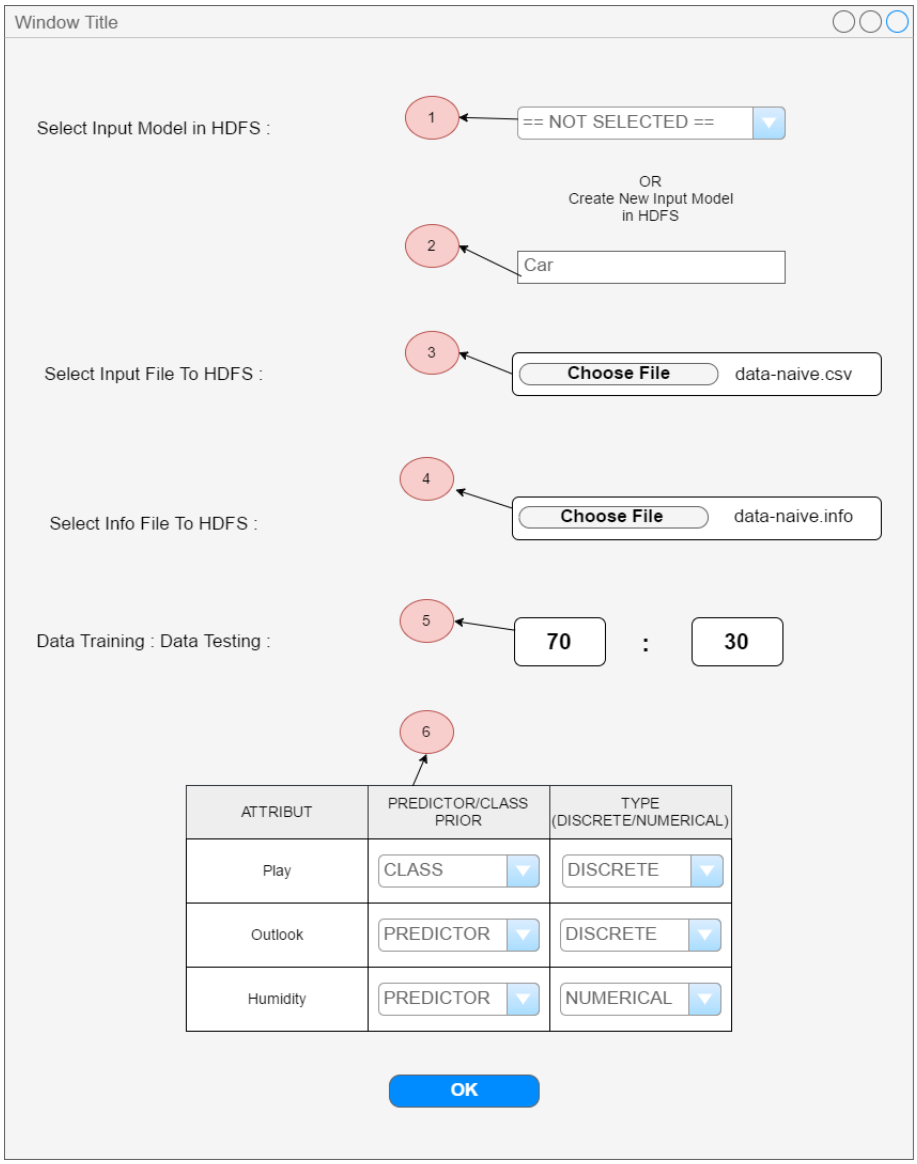
Dashboard dibuat untuk memudahkan user dalam memonitor model NBC yang telah dimasukkan ke dalam perangkat lunak yang dibangun. Berikut penjelasan lebih lanjut mengenai tiap komponen pada rancangan *dashboard* yang dibuat:

1. Berisi nama - nama atribut kelas dan total frekeunsi kemunculannya tiap nilai.

2. Berisi nama - nama atribut prediktor dan frekuensi kemunculannya untuk prediktor bertipe diskrit dan *mean & sigma* untuk yang bertipe numerik.

3. *Bayesian model* merupakan model dari NBC yang akan digunakan untuk testing dan klasifikasi. Model ini merupakan model yang langsung di-import dari hasil training di dalam HDFS.

2.1.2 *Input Set Manager*



Gambar 2.2: *Input Set Manager*

Input Set Manager dibuat untuk memudahkan user melakukan input data ke dalam HDFS menggunakan perangkat lunak yang dibuat. Berikut penjelasan lebih lanjut mengenai tiap komponen pada rancangan *Input Set Manager* yang dibuat:

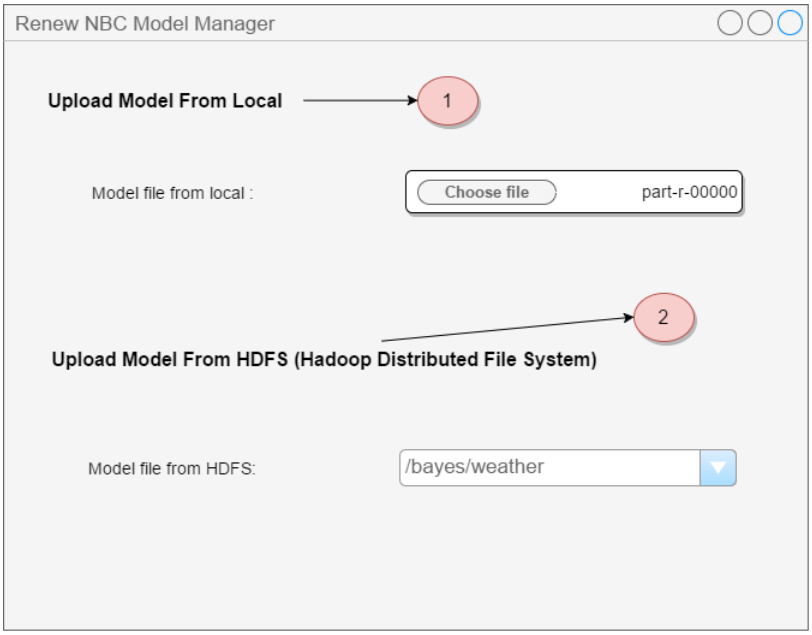
1. User dapat memilih tipe model input yang sudah ada dalam HDFS.

2. Jika ingin membuat tipe model input baru pada HDFS, maka user perlu mengisi kolom ini dan mengisi nama model yang diinginkan.

3. User dapat memilih file input yang akan dikirimkan ke dalam HDFS. User dapat memilih $>$ 1 file sekaligus.

4. User dapat memilih file info mengenai file input, yang dikirimkan ke dalam HDFS.
5. User dapat memilih presentase pembagian data antara data *training* dan data *testing* dari keseluruhan data input yang akan dimasukkan ke dalam HDFS.
6. Setelah memillih file info, user dapat memilih atribut mana saja yang akan digunakan untuk training. User juga dapat memilih tipe(diskrit/numerik) dari atribut tersebut beserta jenisnya (kelas/prediktor).

2.1.3 *Renew Model Manager*

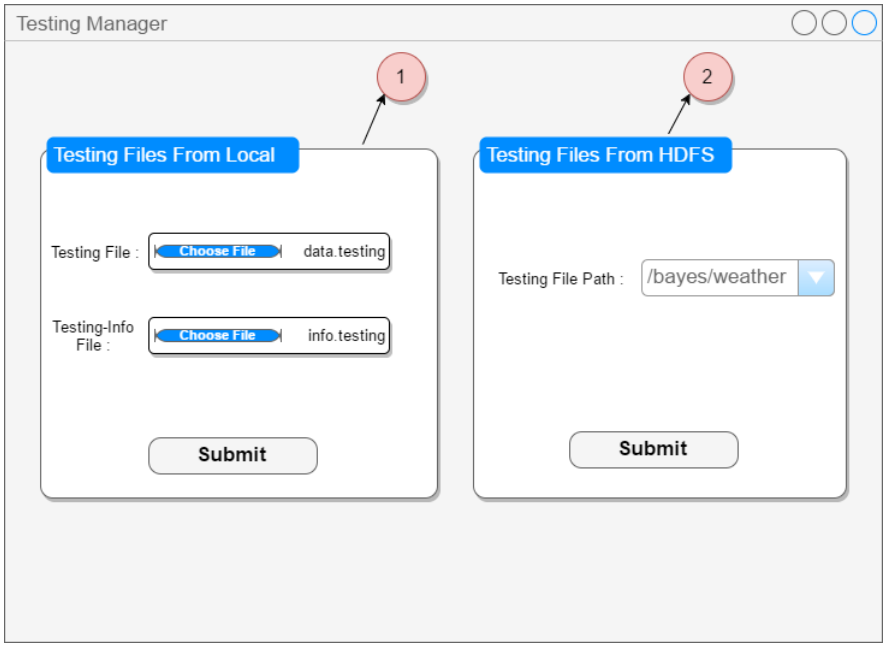


Gambar 2.3: *Renew Model Manager*

Renew Model Manager dibuat agar user selalu bisa memperbaharui model NBC pada perangkat lunak yang dibikin.

1. User dapat memilih file model NBC hasil dari training dari sistem penyimpanan *local*.
2. User dapat memilih file model NBC hasil dari training langsung dari HDFS.

2.1.4 *Testing Manager*

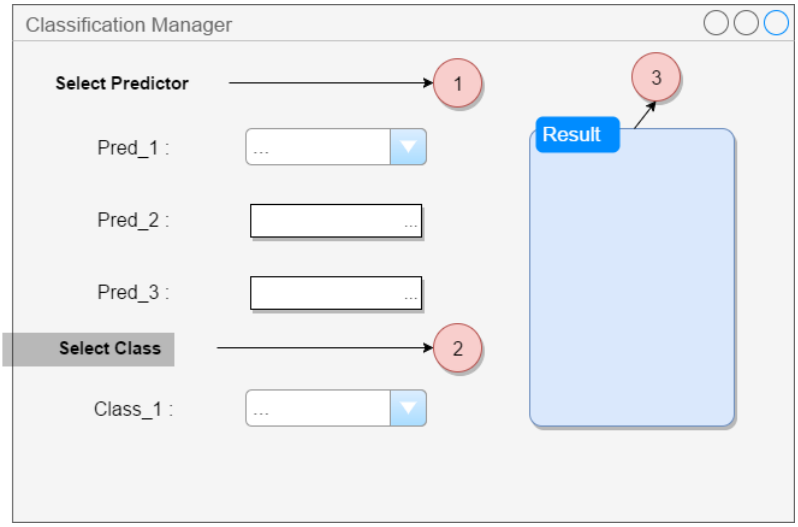


Gambar 2.4: *Testing Manager*

Testing Manager dibuat untuk melakukan testing pada model NBC yang sudah di-import ke dalam program sebelumnya.

1. User dapat memilih file input dan file info dari penyimpanan *local* milik user.
2. User dapat memilih file testing yang sudah ada di dalam HDFS dengan memilih model input direktori pada HDFS.

2.1.5 *Classification Manager*

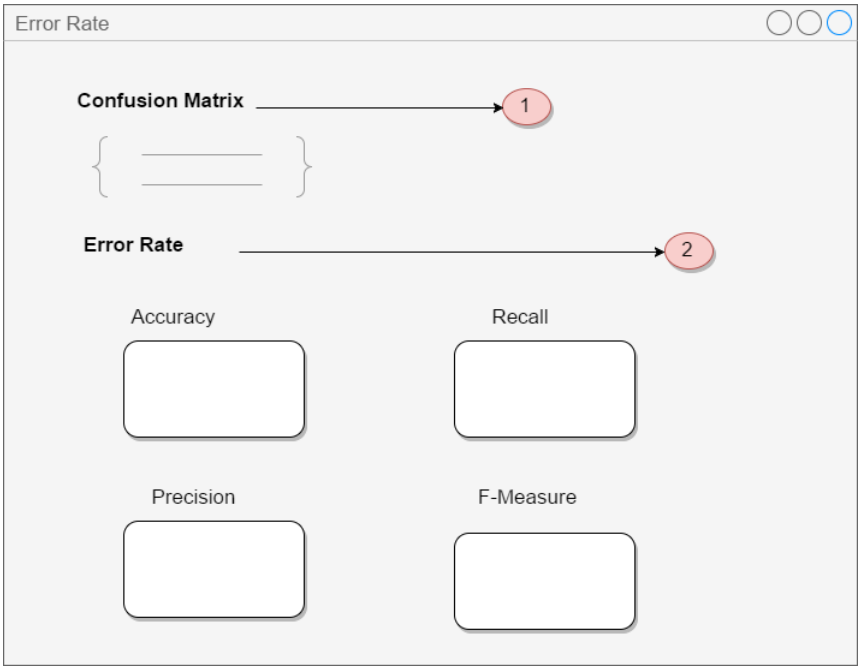


Gambar 2.5: *Classification Manager*

Classification Manager dapat digunakan untuk mengklasifikasi satu record input/kasus yang secara langsung diisi sendiri oleh user yang menggunakannya terhadap model NBC yang sudah ada pada perangkat lunak sebelumnya.

1. User memilih nilai prediktor untuk kasus baru (prediktor dapat berupa dropdown untuk yang bertipe diskrit dan *number* untuk yang bertipe numerik)
2. User dapat memilih kelas yang menjadi prediksi sebelumnya dari user untuk diperiksa kebenarannya jika menggunakan program setelah diklasifikasikan menggunakan model NBC yang sudah ada.
3. Hasil dari klasifikasi yang telah dijalankan.

2.1.6 *Error Rate Dashboard*



Gambar 2.6: *Error Rate Dashboard*

Error Rate Dashboard dibuat untuk memonitor hasil *error rate* yang sudah dihitung setelah menjalani proses testing.

1. *Confusion matrix* untuk setiap atribut kelas.
2. Error rate yang akan dihasilkan setelah melakukan klasifikasi meliputi: (1)*Accuracy*; (2)*Precision*; (3)*Recall*; (4)$F - Measure$.

2.2 Diagram Kelas Lengkap

Berikut adalah penjelasan dari kelas - kelas yang ada pada keempat modul yang dibuat dan beserta penjelasan setiap atribut dan operasi yang dimiliki oleh kelas - kelas.

2.2.1 Modul Kelola Input

2.2.2 Modul *Train Naive Bayes M-R Based*

2.2.3 Modul *Testing Naive Bayes M-R Based*

2.2.4 Modul Klasifikasi Naive Bayes

DAFTAR REFERENSI

[1] G. Piateski and W. Frawley, *Knowledge Discovery in Databases*. Cambridge, MA, USA: MIT Press, 1991.

[2] O. R. ZaÃrane, "Chapter I: Introduction to data mining." https://webdocs.cs.ualberta.ca/~zaiane/courses/cmput695/F07/slides/ch1-695-F07.pdf, 2015. [Online; diakses 14-September-2015].

[3] G. L. Michael J. A. Berry, *Data Mining Techniques. For Marketing, Sales, and Customer Support*. Verlag John Wiley And Sons, Inc, 1997.

[4] P. Zikopoulos, C. Eaton, *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[5] J. network, "Introduction to big data - infrastructure and networking considerations." http://www.one.com.vn/sites/default/files/file-attached/catalog/introduction_to_big_data_-_infrastructure_and_networking_considerations.pdf, 2012. [Online; diakses 09-April-2017].

[6] C. Lam, *Hadoop in Action*. Greenwich, CT, USA: Manning Publications Co., 1st ed., 2010.

[7] J. Dean and S. Ghemawat, "Communications of the acm," in *MapReduce: simplified data processing on large clusters*, pp. 107–113, 2004.

[8] H. G. S. Ghemawat and S. Leung, "Proceedings of the nineteenth acm symposium on operating systems principles," in *The Google file system*, (London, UK), 2003.

[9] A. Holmes, *Hadoop in Practice*. Greenwich, CT, USA: Manning Publications Co., 2012.

[10] E. B. Setiawan, "Pemilihan ea framework," in *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*, 2009.

[11] M. Toha, *Implementasi framework spring mvc untuk pembuatan sistem informasi manajemen e commerce*. PhD thesis, Universitas Sebelas Maret, 2010.

[12] E. Gunawan, "Mengenal apache maven." http://www.erikgunawan.com/mengenal-apache-maven/, 2015. [Online; diakses 09-April-2017].

[13] A. Cogoluègnes, "Introducing the thymeleaf template engine," 2013.

[14] J. L. Peugh and C. K. Enders, "Missing data in educational research: A review of reporting practices and suggestions for improvement," *Review of educational research*, vol. 74, no. 4, pp. 525–556, 2004.

[15] J. Joseph, "How to treat missing values in your data." http://www.datasciencecentral.com/profiles/blogs/how-to-treat-missing-values-in-your-data-1, 2016. [Online; diakses 09-April-2017].