BAB 1

LANDASAN TEORI

Pada bab ini akan dijelaskan beberapa dasar teori dan *framework* yang akan digunakan dalam merancang dan membangun perangkat lunak untuk melakukan klasifikasi naive bayes pada sistem terdistribusi Hadoop. Dasar teori yang akan dijelaskan pada bab ini meliputi: (1) Penambangan data ; (2) *Big data* ; (3) Sistem terdistribusi hadoop ; (4) *Naive bayes classification* ; (5) *Spring framework, maven,* dan *thymeleaf.*

1.1 Penambangan Data

1.1.1 Definisi Penambangan Data

Data mining (penambangan data) adalah suatu proses untuk menemukan suatu pengetahuan atau informasi yang berguna dari data berskala besar. Sering juga disebut sebagai bagian proses KDD (Knowledge Discovery in Databases) [**?**]. *Data mining* akan melakukan pemecahan masalah dengan menganalisa data yang ada di dalam database dan sering juga didefinisikan sebagai proses menemukan pola dalam data, dimana proses tersebut harus otomatis atau semi-otomatis dan pola yang ditemukan harus bermakna [**?**].

Data mining secara umum adalah mengenai pembuatan sebuah model. Suatu model dari teknik *data mining* merupakan suatu algoritma atau sekumpulan aturan yang menghubungkan suatu input data set dengan hasil dari output yang diinginkan. Sebuah model yang dibuat menggunakan teknik *data mining* dapat memiliki tingkat akurasi yang tinggi dan juga tingkat akurasi yang rendah. Hal tersebut bergantung pada pemilihan data yang digunakan sebagai input dan output yang harus memiliki hubungan [**?**].

1.1.2 Tugas Yang Dapat Dilakukan Oleh *Data Mining*

- *Classification*
- *Estimation*
- *Estimation*
- *Affinity grouping*
- *Clustering*
- *Description and profiling*

Pada bagian ini, hanya akan dijelaskan mengenai teknik klasifikasi dan klustering.

1. *Classification*[**?**]
   Klasifikasi juga merupakan algoritma mesin learning yang bersifat terawasi (*supervised learning*). Dimana dalam pembuatan modelnya diperlukan data latih (umumnya disebut sebagai *data training*) terlebih dahulu untuk dijadikan acuan dalam melakukan klasifikasi data.

Algoritma klasifikasi adalah salah satu tugas data mining yang paling umum, tampaknya dewasa ini hal tersebut menjadi penting bagi perusahaan - perusahaan yang memiliki data transaksi. Disamping perusahaan, kita juga sebagai manusia secara terus-menerus mengklasifikasikan, mengkategorikan, dan melakukan penilaian terhadap sesuatu.

Klasifikasi memeriksa atribut dari objek baru yang diberikan dan dilemparkan ke dalam model standar klasifikasi yang sudah ada. Objek yang akan diklasifikasikan umumnya diwakili oleh *record* - *record* dalam tabel database atau file, dan tindakan klasifikasi terdiri dari menambahkan kolom (*field*) baru dengan kode kelas (*kelompok*) dari beberapa jenis kelas. Tugas kita adalah untuk membangun model dari beberapa jenis *dataset* yang dapat diterapkan pada data yang belum terklasifikasi untuk diklasifikasikan ke dalam kelompok yang sudah ada. Berikut merupakan beberapa contoh dari penerapan teknik klasifikasi:

- Klasifikasi calon pengguna kartu kredit yang memiliki resiko rendah, sedang, atau tinggi.
- Memilih konten yang akan ditampilkan pada halaman Web.
- Menentukan nomor telepon sesuai dengan mesin faks.
- Menentukan potensi dari klaim asuransi palsu.

2. *Clustering*[**?**]
Klustering juga merupakan algoritma mesin learning yang bersifat tidak terawasi (*unsupervised learning*). Dimana dalam pembuatan modelnya tidak diperlukan data latih (umumnya disebut sebagai *data training*) terlebih dahulu. Klustering akan mengelompokkan sejumlah data yang memiliki kemiripan yang tinggi. Berikut merupakan beberapa contoh dari penerapan teknik klustering:

   - Mencari distribusi penjualan suatu barang tertentu berdasarkan lokasi tempat tinggal konsumen.
   - Mencari distribusi penjualan suatu barang tertentu berdasarkan tingkat penghasilan konsumen.
   - Mengetahui rentang umur yang paling banyak melakukan transaksi terhadap suatu barang (melakukan pembelian barang tersebut) untuk dijadikan *targeted marketing*.

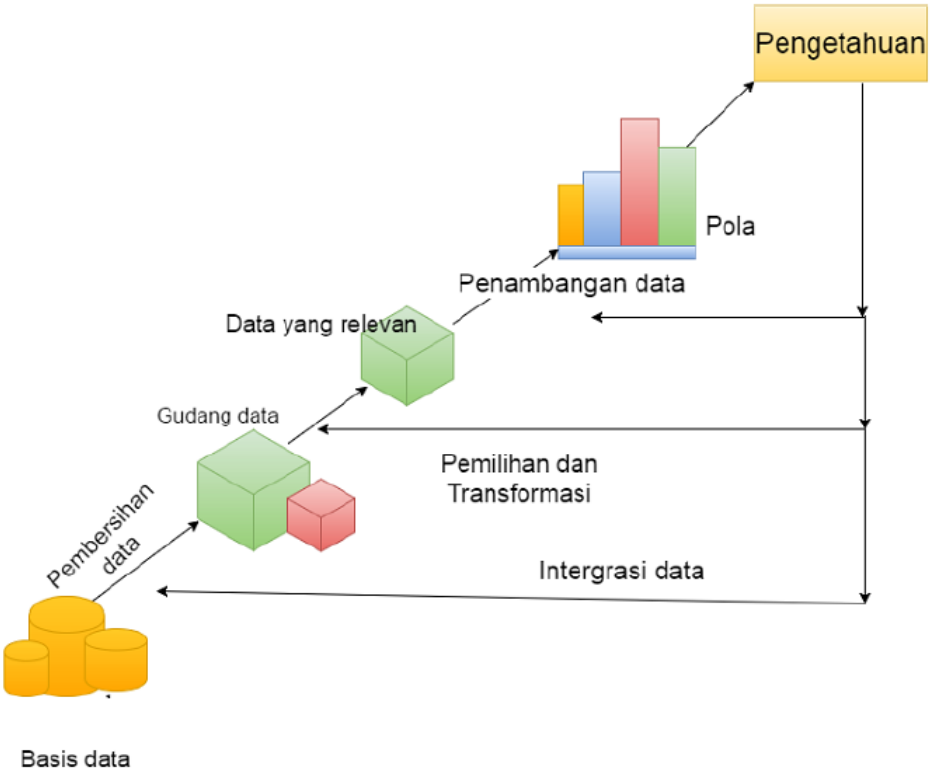
1.1.3 Langkah - langkah Penambangan Data

Gambar 2.1 menyatakan langkah-langkah untuk menemukan pengetahuan dan proses-proses pengolahan data. Gambar 2.1 juga menyatakan bahwa setelah meraih pengetahuan, proses pencarian pengetahuan lebih lanjut juga dapat dilakukan kembali. Lihat anak panah pada Gambar yang merujuk pada langkah-langkah yang dapat dilakukan kembali untuk mendapatkan pengetahuan yang lebih mendalam. Berikut adalah proses-proses yang harus dilakukan dalam mencari pengetahuan baru [**?**]:

- Pembersihan Data
  Pembersihan data adalah fase yang membuang data yang tidak berguna (noise) dan data yang tidak relevan.

- Integrasi data
  Pada fase ini, beberapa sumber data yang heterogen digabungkan menjadi satu.

- Pemilihan Data
  Fase ini adalah fase dimana data dipilih, data yang relevan akan dipilih dari koleksi data pada fase sebelumnya.



Gambar 1.1: Langkah Penambangan Data

- Transformasi Data
  Pada fase ini data yang dipilih ditransformasi sehingga memiliki bentuk yang layak untuk ditambang.

- Penambangan Data
  Fase ini adalah fase dimana teknik-teknik penambangan (termasuk Algoritma Apriori) digunakan untuk mencari pola yang memiliki potensi.

- Evaluasi Pola
  Pada fase ini, pola yang menarik akan diidentifikasi dan dianalisis sehingga dapat menghasilkan pengetahuan yang baru.

- Penyimpulan Pengetahuan
  Fase terakhir ini adalah fase dimana pengetahuan yang baru ditemukan itu direpresentasikan kepada pengguna.

1.2 *Big Data*

1.2.1 Definisi *Big Data*

Big Data merupakan suatu terminologi modern untuk sekumpulan data yang memiliki kesulitan tersendiri untuk diproses dengan cara tradisional (menggunakan satu buah komputer) [**?**]. Dewasa ini perusahaan - perusahaan tengah menghadapi tantangan - tantangan yang ada pada *Big Data*, karena pertumbuhan data yang sudah semakin cepat dan grafik jumlah pengguna internet yang semakin menaik setiap saat. Mereka mendapatkan akses pada data/informasi tersebut tetapi tidak tahu cara mengambil informasi yang berguna pada data tersebut karena format dari data tersebut yang semi bahkan ada yang tidak terstruktur dan juga ukuran-nya yang sangat besar ; Cukup kebingungan apakah data tersebut akan berguna apabila terus disimpan. Karena, dapat dapat menyebabkan *memory overload*.

Big Data tidak melulu berasal dari internet, di dalam kehidupan kita sehari - hari sering kali kita berurusan dengan data, seperti data pada sensor sidik jari ketika absensi, data pembelian pada supermarket, data sensor kelembaban udara pada 10 tahun terakhir untuk memprediksi cuaca, kenaikan dan penurunan harga saham, *bitcoin*, dsb. *Big Data* menjadi topik yang diminati karena dengan data yang begitu banyak, dapat diteliti pola yang terjadi pada data tersebut selama beberapa kurun waktu tertentu untuk digunakan dalam menganalisis data dan membuat keputusan serta memberikan prediksi kemunculan data berikutnya dengan tingkat akurasi yang tinggi berdasarkan data yang dipelajari. Perusahaan - perusahaan saat ini tengah memulai untuk mengumpulkan setiap data yang dapat mereka peroleh dari *customer* untuk melihat pola aktifitas *customer* mereka dan membuat keputusan yang dapat menguntungkan perusahaan berdasarkan hal tersebut. Tentu saja hal ini tidak dapat dilakukan menggunakan teknik komputasi yang tradisional (menggunakan satu buah komputer berteknologi tinggi), karena biaya dan waktu yang terlalu mahal dan lama.

1.2.2 Karakteristik *Big Data*

3 hal terpenting yang menjadi pokok permasalahan dalam *Big Data* adalah :

1. Mengolah data yang berjumlah sangat besar
   Ukuran data yang menjadi tantangan pada big data saat ini sangat besar, dari *Giga-Byte* bahkan bisa mencapai puluhan *PetaByte*.

2. Mengolah data yang memiliki tipe sangat bermacam - macam / variatif
   Tipe dari data pada *big data* sangat bermacam - macam. seperti: (1) angka, (2) tanggal, (3) *string*[1]

3. Mengolah data dengan performa yang optimal
   Tantangan yang terakhir pada *big data* adalah kebutuhan untuk mengolah *big data* dengan waktu dan sumber daya yang se-optimal dan se-efisien mungkin.

Berikut merupakan beberapa contoh *big data* dan pemanfaatannya diberbagai bidang [?]:

1. Pada bidang penerbangan dan transportasi, big data didapat dari data konsumsi bahan bakar dan pola lalu lintas di setiap armada secara nyata untuk meningkatkan efisiensi dan penghematan biaya.

2. Pada sektor kesehatan, big data didapat dari berbagai catatan kesehatan elektronik pasien dari berbagai sumber, perawatan, demografi dan pencatatan khasiat obat sehingga dapat memberikan proses pengembangan obat yang lebih efisien dan lebih cepat.

3. Pada sektor telekomunikasi, big data didapat dari penggunaan dan permintaan pengguna sehingga perusahaan telekomunikasi dapat menganalisis perilaku pengguna dan pola permintaan tersebut.

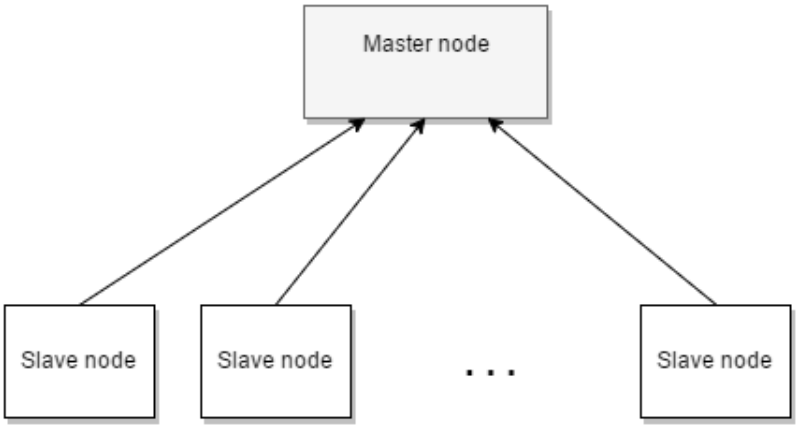
4. Pada sektor media dan hiburan, big data didapat dari penggunaan media dan hiburan sehingga dapat dimanfaatkan untuk membantu pengambilan keputusan dan analisis prediktif dari penggunaan media dan hiburan tersebut.

[1] *String* dalam pemrograman komputer adalah sebuah deret simbol. Tipe data string adalah tipe data yang digunakan untuk menyimpan barisan karakter

1.3 Sistem Terdistribusi Hadoop

1.3.1 Definisi Hadoop

Apache Hadoop merupakan platform *open source* yang didirikan pada tahun 2006 untuk melakukan analisis pada big data. Hadoop merupakan software framework open source yang dapat menangani pertumbuhan jumlah data yang tinggi tanpa mempengaruhi performa kinerja-nya dengan sistem komputasi yang terdistribusi pada beberapa mesin yang dimiliki oleh Hadoop secara efisien [**?**]. Hadoop terinspirasi dari MapReduce milik Google[**?**] dan Google File System (GFS) [**?**].
Hadoop cluster[2] terdiri dari 2 node [3] yaitu *Master Node* yang terdiri dari *Namenode* dan *Jobtracker daemon*[4] dan *Slave Node* yang terdiri dari *Datanode* dan *Task Tracker*.



Gambar 1.2: Arsitektur master-slave

1.3.2 Fitur - fitur dari *Hadoop* [**?**]

1. *Cost Effective System.* Hadoop dapat diimplementasikan pada beberapa komputer biasa yang tidak memiliki spesifikasi terlalu tinggi.

2. *Support Large Cluster or Node.* Jumlah node yang digunakan dapat mencapai ratusan bahkan ribuan node.

3. *Support Parallel Processing Data.* Proses yang dijalankan oleh hadoop dapat berjalan secara bersamaan pada cluster. Sehingga, kebutuhan akan waktu pengerjaan akan berkurang sebanding dengan banyaknya node yang dipakai.

4. *Distributed Data.* Hadoop akan menangani pendistribusian data kepada setiap node pada cluster dan melakukan replikasi data untuk seluruh cluster.

5. *Automatic Failover Management.* Jika cluster/node mengalami kerusakan (*fail*), maka hadoop secara otomatis akan membebankan proses yang dikerjakan oleh cluster/node yang mengalami kerusakan tersebut kepada node node baru yang siap dan melakukan replikasi seluruh konfigurasi dan data dari node yang mengalami kerusakan tersebut ke node node baru.

6. *Data Locality Optimization.* Menggunakan konsep *move-code-to-data.* Daripada menggunakan metode yang biasa digunakan, memasukan seluruh input ke dalam suatu kode,

[2] cluster merupakan sekelompok server
[3] node merupakan istilah teknik yang digunakan untuk menjelaskan suatu mesin atau komputer
[4] Daemon merupakan isitlah teknis yang digunakan untuk menjelaskan suatu proses yang berjalan secara background pada mesin linux

yang akan mengakibatkan *bottleneck* pada jaringan transfer karena *bandwidth* yang dibutuhkan tidak mampu melakukan transfer data yang sangat besar, lebih baik jika data tetap disimpan pada tempatnya,lalu kode kita yang dipindahkan ke tempat data tersebut berada. Karena, besaran file kode pasti akan jauh lebih kecil daripada data yang berukuran sangat besar. Sehingga, akan sangat menghemat waktu proses transfer yang berjalan.

7. *Heterogeneous Cluster.* Suatu *cluster, dapat terdiri dari komputer yang berbeda - beda spesifikasi, merek, maupun OS. Hal ini dapat memudahkan kita untuk membangun suatu *commodity hardware.*

8. *Scalability.* Hadoop memiliki kemampuan untuk dapat menambah ataupun mengurangi node pada suatu cluster tanpa membuat server yang sedang berjalan *down.*

1.3.3 Cara Kerja Hadoop

Hadoop menggunakan model pemrograman yang cukup sederhana untuk memproses data - data yang berukuran sangat besar melewati beberapa cluster mesin dan tempat penyimpanan memori yang terdistribusi. Oleh karena cara kerja hadoop yang menggunakan beberapa cluster mesin secara terdistribusi, besar kemungkinan untuk terjadi kegagalan dalam cluster tersebut pada saat ada proses yang sedang berjalan. Tetapi, kita tidak perlu khawatir dan mempersiapkan mekanisme penanganan untuk mengatasi hal tersebut, karena Hadoop sudah secara otomatis menangani hal tersebut agar apabila terjadi kegagalan (*failure*) pada 1 atau lebih cluster/node, Hadoop akan mendistribusikan proses beserta seluruh sumber daya yang dibutuhkan oleh proses tersebut kepada mesin lain yang siap (*available*). Hadoop sudah memiliki skema khusus untuk melindungi metadata dari kumpulan - kumpulan dataset yang berjumlah besar agar tidak hilang secara tidak sengaja, sehingga sangat aman digunakan (memiliki toleransi tinggi terhadap *node failures*). Tahapan cara kerja hadoop adalah [**?**] :

1. Membagi data input ke dalam bagian yang lebih kecil dan menyimpan setiap bagian dari data tersebut ke dalam node yang berbeda pada cluster.

2. Mereplikasi setiap bagian data tersebut kepada beberapa node yang berada di cluster yang sama maupun yang berbeda. Secara default direplikasi sebanyak 3 kali (*default replication factor is 3*).

3. Menangani jalannya proses komunikasi diantara cluster. Seperti untuk mengakses replikasi data, jika dibutuhkan, pada cluster yang berbeda.

1.3.4 Elemen dari Hadoop

Hadoop memiliki 2 elemen penting, HDFS (*Hadoop Distributed File System*) dan MapReduce. Keduanya sama - sama memiliki peran penting yang saling berhubungan untuk melakukan *distributed computing.*

1.3.4.1 HDFS (Hadoop Distributed File System)

HDFS merupakan sebuah sistem penyimpanan terdistribusi yang menyediakan akses throughput data yang tinggi. Akses throughput data yang tinggi memiliki arti bahwa HDFS dapat melakukan proses baca tulis data dalam skala yang besar. HDFS menciptakan beberapa replika dari setiap blok data dan mendistribusikannya pada seluruh cluster komputer untuk memungkinkan akses data yang dapat diandalkan dan cepat [**?**]. Dapat diandalkan dalam konteks ini bermaksud apabila sebuah node pada cluster Hadoop mengalami kegagalan yang menyebabkan data pada node tersebut corrupt, maka blok data tersebut tidak hilang karena selain node tersebut ada beberapa node lain yang menyimpan replika dari blok

data tersebut. HDFS dapat melakukan komputasi dengan performa tinggi pada skala komputasi yang sangat besar *(high-bandwidth computation)* dan biaya kapasitas penyimpanan terdistribusi yang dibutuhkan cukup rendah. HDFS didesain untuk data yang berukuran sangat besar GB to TB. HDFS memiliki default block size yang cukup besar, yakni 64 MB.

HDFS terdiri dari 3 komponen utama yang berjalan secara *background*, diantaranya adalah:

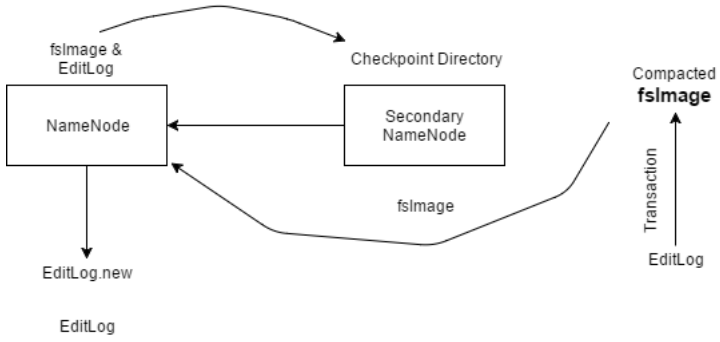
1. *NameNode*

   NameNode dapat disebut juga sebagai *Master Node*. MasterNode biasanya hanya terdiri dari satu node saja. Tugas pertama dari MasterNode adalah untuk menyimpan seluruh metadata milik HDFS, seperti file name, file permission, file ownership, file location, dan id block pada file system. Tugas kedua dari MasterNode adalah untuk mengalokasikan 1GB pada RAM untuk melakukan pelacakan terhadap kurang lebih 1 juta file. Dan juga melakukan penyimpanan kepada disk, untuk backup data.

2. *DataNode*

   DataNode dapat disebut juga sebagai *Slave Node*. Slave Node terdiri dari banyak node. SlaveNode bertanggung jawab untuk melakukan penyimpanan dan pengambilan data sesuai instruksi yang diperoleh dari NameNode. Secara periodik, SlaveNode memberikan status dari dirinya sendiri dan semua block file yang disimpan melalui **Heartbeat*. Slave Node menyimpan beberapa replikasi dari setiap file yang ada di dalam HDFS.

3. *Secondary NameNode*

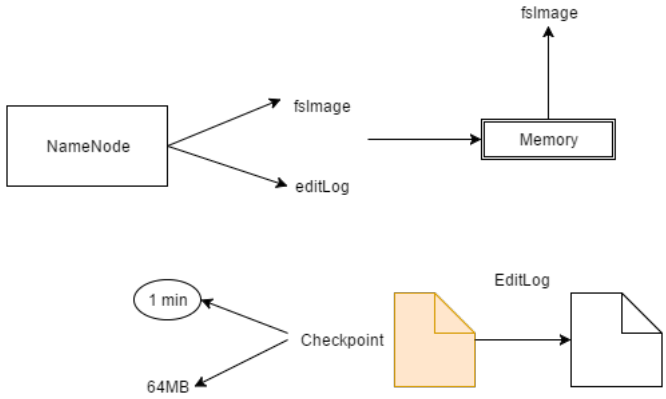


Gambar 1.3: Alur kerja Secondary NameNode dan NameNode

Secondary NameNode merupakan MasterNode cadangan yang selalu aktif mencatat seluruh kegiatan dari NameNode untuk menjadi backup apabila server masternode yang utama mengalami kendala. Setiap transaksi yang dilakukan akan dicatat di dalam file editLog [5]. Secondary NameNode akan memeriksa NameNode untuk menyimpan transaksi terbaru pada file editlog yang baru. Secondary NameNode akan membuat salinan dari fsImage [6] dan editLog pada direktori *checkpoint*. Setelah itu, ia akan menyimpan transaksi terbaru yang terjadi pada file editLog dan menyimpan seluruh informasi yang terbaru ke dalam *compacted fsImage* baru. Secondary NameNode mengirim fsImage yang baru tersebut ke NameNode dan NameNode akan memilih fsImage yang baru tersebut. Proses ini akan berjalan setiap 1 jam sekali atau ketika besaran dari file editLog sudah mencapai 64MB. Seperti yang tertera pada gambar 2.2. Ketika NameNode baru dijalankan, ia akan membuat file fsImage dan file editLog dari disk. Lalu menuliskan semua transaksi ke dalam metadata dari editLog yang telah di salin ke dalam RAM. Setelah selesai, versi baru dari fsImage akan dikembalikan dari memori RAM kepada memori disk.

[5] File editLog adalah file yang menyimpan seluruh modifikasi yang dilakukan terhadap metadata

[6] fsImage adalah file yang berisi snapshot lengkap mengenai metadata. Menyimpan semua blok yang dimiliki oleh suatu file dan file system property

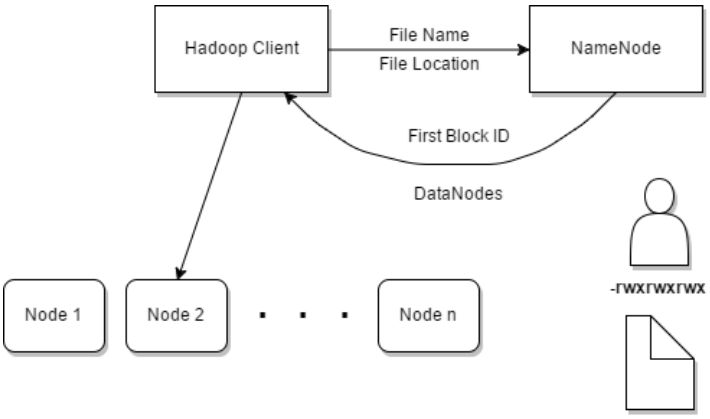


Gambar 1.4: FileSystem Metadata pada NameNode dan Secondary NameNode

HDFS memiliki 2 operasi yang penting, yaitu operasi baca dan operasi tulis dari dan ke HDFS. Berikut merupakan penjelasan lebih lanjut mengenai operasi baca dan operasi tulis dari dan ke HDFS.

Membaca data dari HDFS

Untuk dapat membaca data dari HDFS, Hadoop Client membutuhkan "'Hadoop Client Library"' dan configurasi dari cluster. Mekanisme operasi baca data dari HDFS adalah sebagai berikut : (Ilustrasi pada gambar 2.4) :



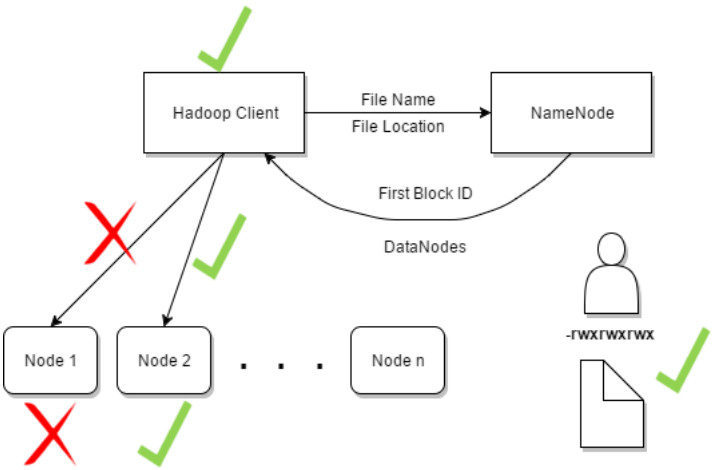
Gambar 1.5: Client membaca data dari HDFS

1. Client menghubungi NameNode dan memberikan nama file lokasi pada file yang ingin dibaca.
2. NameNode memvalidasi client untuk memeriksa permission yang dimiliki oleh user tersebut terhadap file yang diminta.
3. NameNode memberikan respon kembali kepada client dengan memberikan *first block ID* [7] dengan seluruh DataNode yang memiliki salinan/replikasi dari file yang diminta tersebut.
4. Setiap DataNode yang memiliki salinan akan diurutkan berdasarkan yang terdekat sebelum dikirimkan kepada client.
5. Setelah informasi - informasi di atas diterima oleh client, client dapat menghubungi secara langsung DataNode yang berhubungan dan membaca file nya.

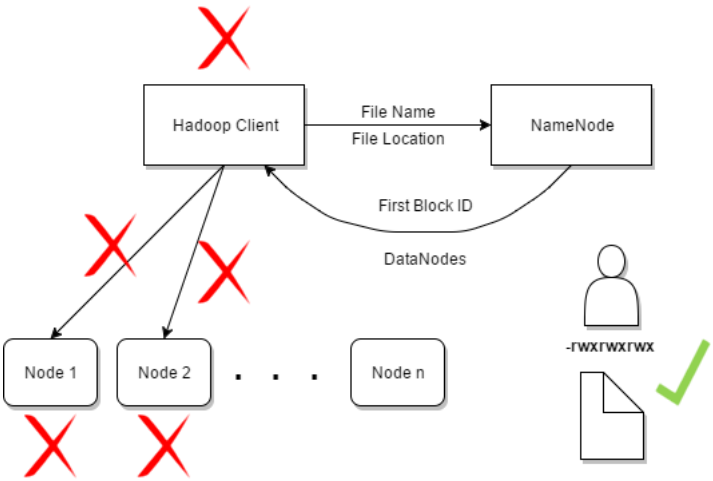
[7] First block ID merepresentasikan tempat penyimpanan 64MB pertama dari data yang diminta tersebut. Memberikan informasi seperti pada rak dan DataNode yang menyimpan blok pertama pada file yang diminta.

6. Jika DataNode yang digunakan pada saat melakukan operasi baca ke HDFS mengalami kerusakan, maka client akan langsung mengarahkan pembacaan ke DataNode yang lainnya yang memiliki replikasi dari data tersebut. Ilustrasi pada gambar 2.5.

7. Jika replikasi yang dibutuhkan pada DataNode lainnya tidak ada, maka operasi baca akan mengalami kagagalan *fail*. Ilustrasi pada gambar 2.6.



Gambar 1.6: Failure takeover (success scheme)



Gambar 1.7: Failure takeover (failure scheme)

Menulis data ke HDFS

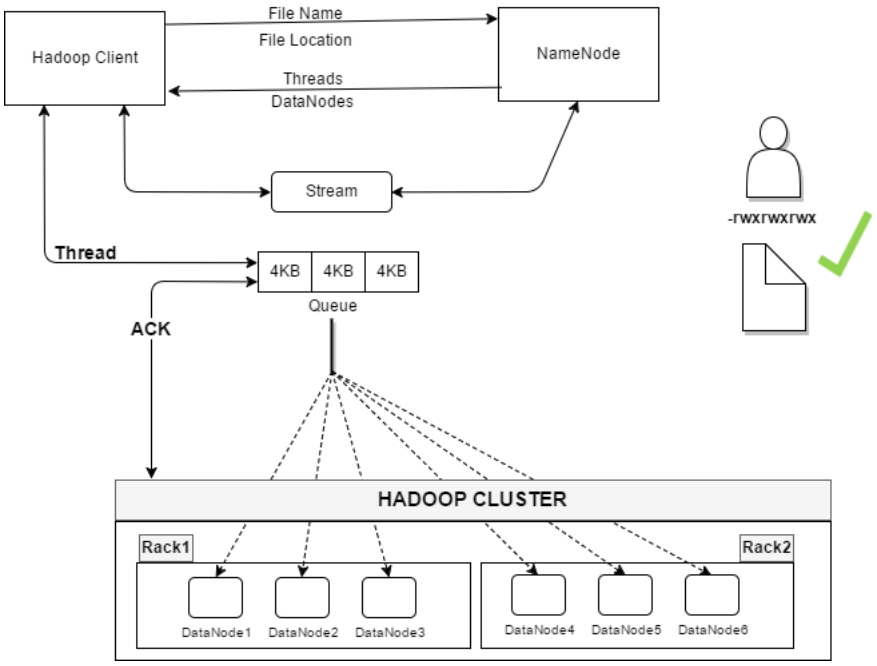
Gambar 2.6 Mekanisme operasi tulis ke dalam HDFS :

1. Client akan menghubungi NameNode dan memberikan nama file dan lokasi yang diinginkan dari file yang akan di tulis.

2. NameNode memvalidasi client untuk memeriksa permission yang dimiliki oleh user tersebut terhadap lokasi untuk penulisan file tersebut.

3. NameNode akan membukakan sebuah stream untuk client melakukan operasi tulis. Data yang ditulis pada stream akan dipecah menjadi bagian - bagian kecil yang berukuran 4KB dan disimpan ke dalam queue.

4. Client membuka thread yang berbeda yang akan bertanggung jawab untuk menuliskan data dari queue ke dalam HDFS.



Gambar 1.8: Client menulis data ke HDFS

5. Thread akan menghubungi NameNode untuk meminta list dari DataNode yang dibutuhkan untuk menyalin replikasi dari data baru ini.

6. Client akan menghubungi secara langsung DataNode pertama dan melakukan penulisan hingga sukses.

7. Setelah sukses, tahap tersebut akan diulangi untuk setiap node yang mereplikasi data tersebut yang diberikan dari NameNode.

8. Setelah selesai, ACK akan diberikan kepada client untuk memberi informasi bahwa penulisan telah selesai dan berhasil.

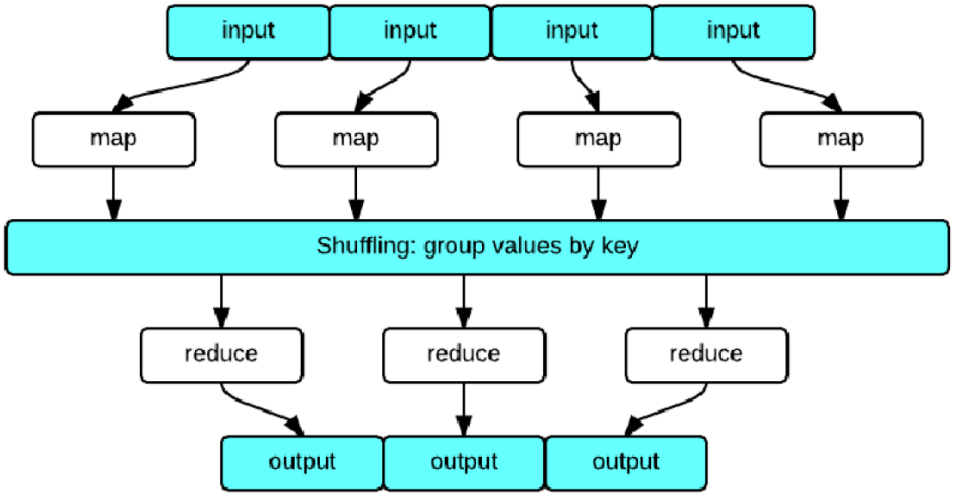
9. Jika ketika sedang penulisan telah mencapai maksimum dari block size, maka client akan kembali menghubungi NameNode untuk meminta kumpulan DataNode berikutnya yang dapat dilakukan operasi tulis.

10. Jika telah selesai, maka client akan menutup stream, lalu queue akan dibersihkan kembali, dan metada dari NameNode akan diupdate.

11. Jika pada saat penulisan terjadi kegagalan, maka data yang dikirimkan setelah ACK terakhir yang diterima oleh client akan dikembalikan ke queue. Lalu didelegasikan ke DataNode yang baru dengan block id yang baru.

1.3.4.2 MapReduce

MapReduce merupakan framework yang dirancang untuk memproses data secara paralel terdistribusi yang memiliki performa dan efisiensi yang sangat tinggi. Jenis pekerjaan pada fase map dan reduce yang dapat dikerjakan oleh framework ini merupakan jenis pekerjaan yang tidak memiliki hubungan berkesinambungan di antara tiap proses-nya, sehingga dapat berjalan secara bersama - sama (*concurrent*). Hadoop membagi/memecah seluruh dataset yang ada ke dalam beberapa partisi dan mendistribusikannya ke dalam kelompok/*cluster*. MapReduce memproses data di setiap server terhadap blok - blok data yang sudah dibagikan sebelumnya, sehingga akan sangat menghemat waktu pekerjaan yang dihabiskan [**?**].
Terdapat 3 fase utama pada MapReduce, yaitu fase *map*, fase *shuffle*, dan fase *reduce*.



Gambar 1.9: Ilustrasi framework MapReduce

1. Pada fase *map*, melakukan *convert* tiap partisi dari input kedalam pasangan *key/value* (seperti pada struktur data HashMap) lalu menggabungkan setiap *value* yang memiliki *key* yang sama.

2. Pada fase shuffle, hasil keluaran dari fase Map akan di sort berdasarkan key dan pasangan *key/value* tersebut akan di kirimkan ke reducer node yang menerima pasangan *key/value* yang sesuai.

3. Pada fase *reduce*, algoritma menerima sebuah pasangan key dengan himpunan dari value yang memiliki hubungan dengan key tersebut, lalu melakukan suatu proses yang nantinya akan menjadi keluaran dari program MapReduce.

Beberapa komponen utama dari *MapReduce* terdiri dari *JobTracker* dan *TaskTracker*.

- *JobTracker* berperan sebagai *master* dari *MapReduce*. *JobTracker* mengelola pekerjaan dan sumber daya dalam *cluster* (*TaskTracker*). *JobTracker* berusaha untuk menjadwalkan proses setiap *map* dan *reduce* pada *TaskTracker* sedekat mungkin dengan *DataNode* yang memiliki blok data yang diproses.

- *TaskTracker* adalah slave yang ada pada setiap node. *TaskTracker* bertanggung jawab untuk menjalankan proses *map* dan *reduce* seperti yang diperintahkan oleh *JobTracker*.

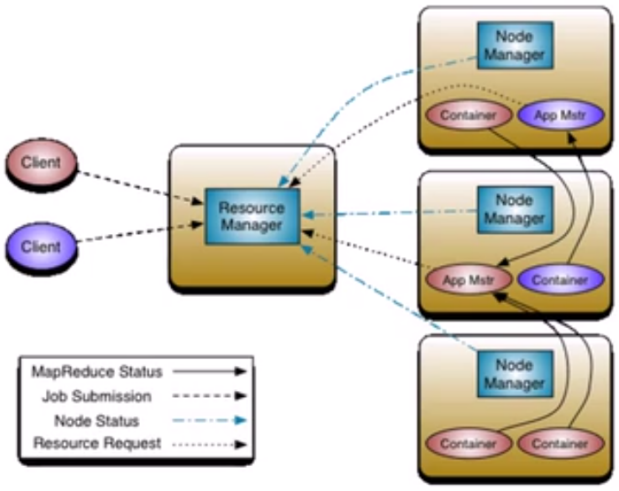
Komponen utama lainnya setelah release versi terbaru hadoop 2 dari MapReduce adalah Apache YARN (*Yet Another Resource Negotiator*). YARN bertanggung jawab untuk mengawasi *resource* yang tersedia pada seluruh node dan memantau status dari setiap *TaskTracker* yang ada pada setiap node dan status dari pekerjaannya. YARN sebagai arsitektur baru dari Apache hadoop 2 membagi dua fungsi utama dari JobTracker/TaskTracker pada MapReduce menjadi beberapa entitas yang terpisah, diantaranya adalah:

- *Resource Manager* di *node master*, yang bertugas untuk mengawasi dan mengatur seluruh *resource* yang tersedia dan digunakan pada seluruh node.

- *Application Master* di setiap aplikasi, yang berfungsi untuk memantau status dari setiap *TaskTracker* yang ada pada setiap node dan status dari pekerjaannya dan juga untuk negosiasi resource dengan *ResourceManager* dan kemudian bekerja sama dengan *NodeManager* untuk mengeksekusi dan memonitor *tasks*.

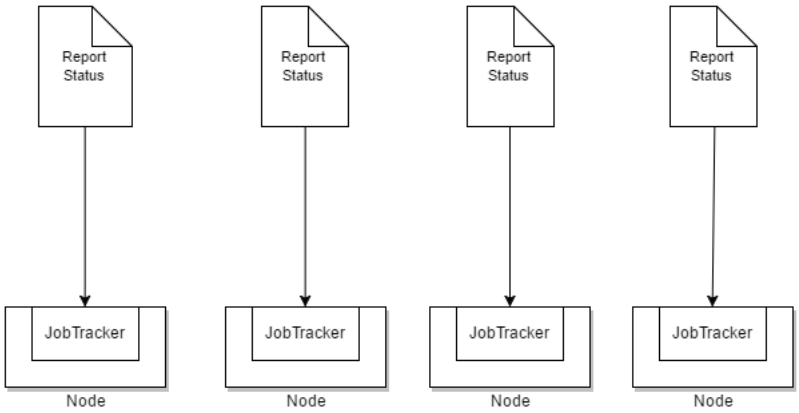
- *NodeManager* di *Agen-Framework* setiap *node slave*, yang bertanggung jawab terhadap *container*, dengan memantau penggunaan resource/sumber daya dari container(cpu, memori, disk, jaringan) dan melaporkannya pada ResourceManager.

- *Container* di setiap aplikasi yang jalan di *NodeManager*, sebagai wadah penyimpanan data/file.



Gambar 1.10: Arsitektur YARN-1

Pada YARN setiap pekerjaan akan memiliki JobTracker-nya masing - masing dan pada suatu cluster dapat memiliki beberapa JobTracker yang sedang bekerja. Setiap JobTracker pada node yang berbeda akan bisa berjalan pada software yang berbeda. Hal ini yang mendukung model dari *Heterogeneous Cluster*.



Gambar 1.11: Arsitektur YARN : Heterogenous Cluster YARN

Map-Reduce ada pada setiap DataNode pada cluster Hadoop. Setiap program Map-Reduce mengerjakan atau mengolah data - data yang terdapat pada DataNode-nya. Map-Reduce ditulis dengan bahasa Java, sehingga untuk setiap abstraksi dan aturan pada pembuatan kode mengikuti bahasa Java. Selain itu, Map-Reduce milik Hadoop dibuat untuk memudahkan programmer sehingga hanya perlu berfokus pada dua buah fase yang digunakan saja, yaitu fase map dan reduce. Sedangkan untuk fase Shuffle dan Sort sudah ditangani secara otomatis oleh framework Hadoop. Berikut merupakan penjelasan lebih lanjut mengenai mapper dan reducer yang digunakan pada fase map dan fase reduce.

Mapper

Mapper berfungsi untuk memetakan data yang diberikan kepada MapReduce. Dalam Hadoop, Mapper berada pada setiap node dan bekerja menggunakan data yang berada pada

masing-masing node. Hal ini dapat mengurangi lalulintas data yang terjadi pada cluster Hadoop karena tidak ada perpindahan data antar node. Mapper membaca data dalam bentuk pasangan key dengan value dan mengeluarkan nol atau lebih pasangan key dengan value dalam bentuk list yang disimpan pada penyimpanan local di DataNode (bukan dalam bentuk HDFS).

Reducer

Reducer berfungsi untuk mengurangi data yang tidak diperlukan (misal : data yang berulang) atau menyatukan data yang dapat disatukan dari hasil Mapper. Jumlah Reducer pada sebuah MapReduce pada sebuah node dapat lebih dari satu. Reducer menerima sebuah list pasangan key dengan value dari Mapper yang terurut berdasarkan keynya. Hasil keluaran dari Reducer berupa nol atau lebih pasangan key dan value yang sudah final. Hasil tersebut sudah disimpan di HDFS. Sebelum diterima oleh Reducer, seluruh hasil dari Mapper melalui sebuah tahap yang dinamakan shuffle and sort. Tujuan dari tahap ini adalah memastikan bahwa semua value dengan key yang sama masuk ke reducer yang sama dan list pasangan key dan value yang diterima terurut pada berdasarkan key.

Cara kerja MapReduce

1. MapReduce akan membaca setiap line dari input yang akan dijadikan sebagai input dari fase Map. Setiap line yang dibaca oleh fase Map merupakan pasangan *key/value* dari line yang menjadi input tersebut (sebagai value) dan offset dari text tersebut terhadap awal file (sebagai key).

2. Fase map akan dijalankan sesuai apa yang kita perintahkan/implementasikan pada program Map yang dibuat. Keluaran dari fase Map merupakan pasangan *key/value* yang telah diproses.

3. Keluaran dari fungsi map sebelum dijadikan sebagai hasil input dari fungsi reduce akan diproses oleh fase *Shuffle and Sort*. Pada fase Shuffle and Sort, hasil keluaran dari fungsi map akan diakumulasi dan pengurutan. Untuk setiap key yang sama akan dikelompokan kedalam sebuah pasangan *key/value* baru yang isinya merupakan key dari input tersebut dan value nya merupakan list dari seluruh pasangan *key/value* yang memiliki key sama.

4. Setelah output fungsi map melewati proses shuffle and sort, maka output tersebut akan terurut berdasarkan key. Sehingga, reducer akan dengan tepat melihat key dan seluruh value yang bersangkutan, yang sudah ditetapkan untuk diproses pada reducer node yang akan menjalankan tugasnya.

5. Pada fase reduce, akan dijalankan fungsi reduce yang sudah diimplementasikan sebelumnya untuk memproses pasangan dari *key/list-of-value-key* menjadi hasil yang dibutuhkan.

1.4 *Naive Bayes Classifier*

Naive Bayes merupakan salah satu metode mesin learning yang digunakan pada teknik data mining menggunakan metode perhitungan peluang. Konsep dasar yang digunakan oleh Naive bayes adalah Teorema Bayes, yaitu teorema dalam statistika untuk menghitung peluang suatu kejadian dari beberapa kejadian lainnya. Bayes Optimal Classifier menghitung peluang dari satu kelas dari masing-masing kelompok atribut yang ada dan menentukan kelas

mana yang paling optimal. Algoritma Naive Bayes melakukan klasifikasi berdasarkan pada teorema Bayes' seperti berikut :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.1}$$

Teori bayes memiliki asumsi bahwa probabilitas P(A|B) atau peluang kejadian A bila B terjadi tidak saling berhubungan dengan setiap kemungkinan dari nilai B yang diberikan (*naive*). Hal ini disebut sebagai *class conditional independence* . Sehingga memudahkan perhitungan yang dilakukan pada saat klasifikasi. Kemungkinan terjadinya kejadian A bila diberikan kejadian B dapat dihitung dengan menggunakan rumus diatas, yaitu mengalikan peluang dari kejadian B jika diberikan kejadian A dikalikan dengan peluang seluruh kejadian A dan dibagi dengan peluang dari seluruh kejadian B.

Berdasarkan teori Bayes, untuk dataset d dan sebuah kelas c, didapatkan :

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{1.2}$$

- $P(c|d)$ merupakan peluang dari kemunculan suatu kelas/kelompok tertentu jika diberikan suatu dataset d
- $P(d|c)$ merupakan peluang suatu dataset tertentu jika diberikan suatu kelas c
- $P(c)$ merupakan probabilitas dari kelas
- $P(d)$ merupakan probabilitas dari dokumen/dataset

Untuk menghitung nilai $P(c|X)$ (X *merupakan dataset yang digunakan sebagai predictor*) dengan mengasumsikan bahwa masing - masing atribut tidak saling bergantung dengan atribut lainnya *class conditional independence*, maka didapat :

$$P(c|X) = P(x_1|c)P(x_2|c)...P(x_n|c)P(c) \tag{1.3}$$

Sebuah dataset dapat ditentukan klasifikasi-nya dengan algoritma naive bayes setelah dihitung semua kemungkinan dari nilai $P(c_k|X)$ dan untuk nilai keluaran yang paling besar akan dipilih menjadi kelas yang paling optimal dari dataset tersebut.

MAP Maximum A Posteriori

$$C_{MAP} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{argmax} P(d|c)P(c) \tag{1.4}$$

1.5 Framework Yang Digunakan Dalam Membangun Perangkat Lunak

1.5.1 *Spring Framework*

Spring Framework adalah salah satu framework[8] untuk aplikasi berbasis Java yang dapat digunakan untuk membangun sebuah aplikasi berskala besar. Model yang digunakan pada perangkat lunak yang dibangun adalah *spring framework web application* yang memiliki ekstensi untuk membuat server pada aplikasi berbasis web dengan J2EE (Java 2 Enterprise

[8]framework adalah tools yang bisa digunakan untuk mengembangkan cakupan luas dari arsitektur-arsitektur yang berbeda [?]

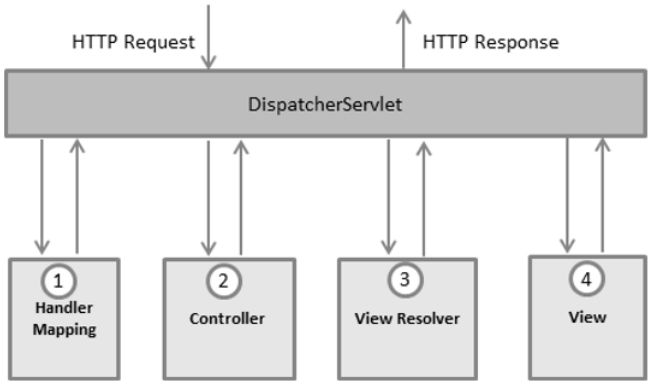
Edition) [?]. Dengan Spring, kita bisa mengembangkan aplikasi enterprise dan berbasis web. Spring termasuk portable karena aplikasi yang dikembangkan dapat berjalan pada JVM manapun. Terdapat beberapa cara yang disediakan oleh spring untuk melakukan deploy aplikasi kita, seperti:

1. Melakukan deploy aplikasi dengan menggunakan mode standalone (menggunakan server bawaan dari *spring*),
2. Melakukan deploy aplikasi pada aplikasi server (menggunakan aplikasi web server dari luar, seperti *Apache Tomcat Web Application*),
3. Melakukan deploy aplikasi dengan menggunakan *cloud PaaS*[9] (*Platform as Service*, seperti *Pivotal Web Service*).

Spring menyediakan model pemrograman terbuka yang komprehensif, kohesif, mudah dipahami serta memiliki *library* yang lengkap untuk melakukan integrasi ke *service* - *service* lain seperti *Hadoop-client Library*, *Mysql-connector Library*, dsb.

Inti dari framework ini lebih kepada untuk membangun aplikasi web, mengatur manajemen transaksi, akses data, messaging, pengujian dsb. Kita bisa mengembangkan aplikasi web berbasis MVC dan *Web-Service* framework *REST-ful*[10] (*Representational State Transfer*).

Design pattern yang digunakan pada perangkat lunak yang dibuat akan menggunakan konsep MVC (*model view controler*). Spring memiliki fitur khusus untuk membuat web berbasiskan *design pattern* MVC. Berikut merupakan ilustrasi[11] dari konsep MVC milik *spring*:



Gambar 1.12: Spring MVC

1.5.2 *Maven* [?]

Apache Maven adalah software build tools / project management yang dibangun dibawah *Apache Software Foundation* yang digunakan untuk melakukan proses building project. Jadi, ketika project akan dibuild menggunakan Maven, project tersebut bisa kita buka menggunakan IDE lain.

Selain itu, keuntungan menggunakan maven adalah mendukung dependency management. Artinya, ketika kita membutuhkan suatu library ke dalam project, kita tidak perlu

[9] *Platform as Service merupakan sebuah platform untuk mengembangkan, menjalankan, dan mengatur aplikasi kita tanpa melewati serangkaian konfigurasi yang rumit layaknya dedicated server pada umumnya*

[10] REST merupakan *standard* dalam arsitektur web yang menggunakan Protocol HTTP untuk pertukaran data. Konsep REST menekankan bahwa komunikasi yang terjadi antara *client* dan *server* hanya sebatas melakukan *request* dan memberikan *response* saja.

[11] Gambar diambil dari https://www.tutorialspoint.com/spring/images/spring_dispatcherservlet.png

mendownloadnya manual, kemudian dimasukkan ke dalam project. Kita tinggal memasukkan ependency-nya, dan maven akan menanganinya. Dengan begini, ukuran project menjadi lebih kecil karena tidak memasukkan library ke dalam project.

Untuk mendownload Apache Maven bisa langsung ke websitenya di http://maven.apache.org/. Versi terbaru dari Apache Maven adalah 3.5.0.

1.5.3 *Thymeleaf* [?]

Thymeleaf merupakan *template-engine open source* yang dapat melakukan *rendering* HTML pada *server-side*. Thymeleaf adalah Java XML template engine / XHTML / HTML5 yang dapat bekerja baik di web (*Servlet-based*) maupun lingkungan yang bukan web. Hal ini lebih cocok untuk melayani XHTML / HTML5 pada *layer* tampilan aplikasi berbasis web MVC, tetapi dapat memproses file XML bahkan di lingkungan offline. *Template engine* ini menyediakan integrasi penuh dengan *Spring Framework*.

Dalam aplikasi web Thymeleaf bertujuan untuk menjadi pengganti lengkap untuk JSP, dan menerapkan konsep Natural Template: file template yang bisa langsung dibuka di browser dan yang masih menampilkan dengan benar sebagai halaman web.

Sebagai *framework open source*, *Thymeleaf* memiliki lisensi Apache 2.0

BAB 2

ANALISIS

Bab ini akan membahas mengenai permasalahan umum yang dihadapi dan melakukan analisis pemodelan sistem yang akan dibangun.

2.1 Deskripsi Masalah dan Solusi Umum

2.1.1 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. Hal tersebut dikarenakan minimnya memori yang bisa diberikan oleh *hardware* pada sistem yang berjalan hanya pada 1 mesin/komputer.

2.1.2 Solusi Umum

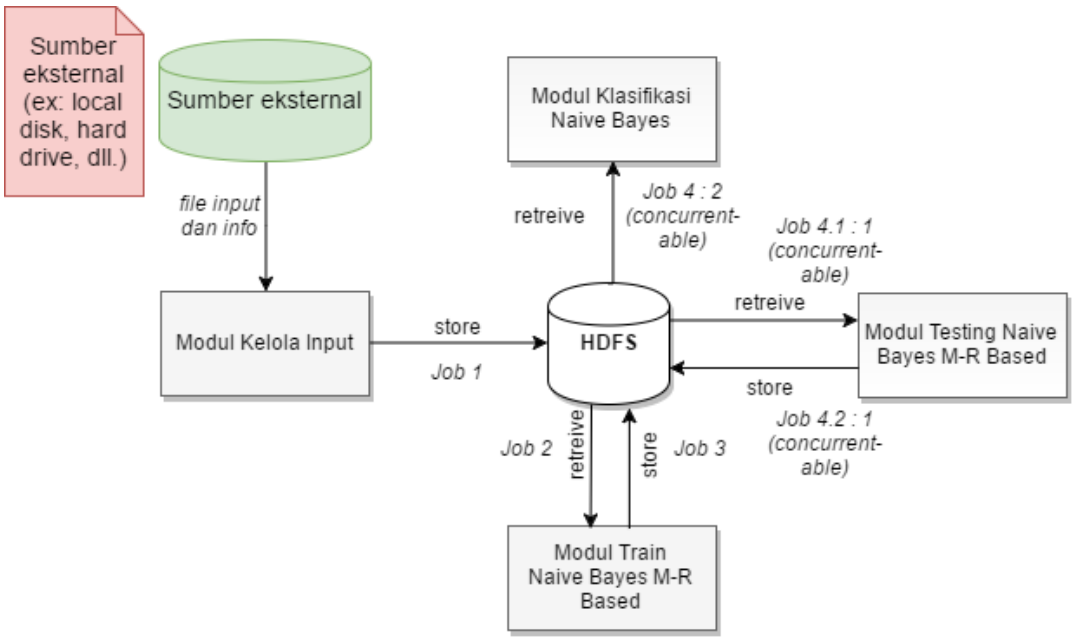
Apache hadoop digunakan untuk menangani hal meliputi big data dengan sistem yang terdistribusi. Framework ini dapat memfasilitasi sebuah program yang berjalan dengan menggunakan beberapa mesin/komputer sekaligus untuk menjadi sebagai tempat penyimpanan data sekaligus pemroses tugas dari program tersebut, dengan protokol komunikasi antar mesin/komputer yang sudah secara otomatis diatur oleh framework tersebut. *Framework hadoop* dapat digunakan untuk membantu algoritma *naive bayes classifier* dalam menangani jumlah data yang sangat banyak dengan bergantung pada banyaknya node yang digunakan. Sehingga, banyaknya node yang digunakan pada lingkungan hadoop yang akan dipakai akan mempengaruhi *scalability* dari algoritma *naive bayes classifier*. Tentu saja untuk mendapatkan keuntungan tersebut dalam menjalankan program *naive bayes classifier*, diperlukan rancangan program yang dibuat berbasiskan *MapReduce* pada Hadoop. Skripsi ini akan membangun perangkat lunak yang menerapkan algoritma klasifikasi *naive bayes* pada sistem terdistribusi hadoop.

2.2 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*.

2.2.1 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



Gambar 2.1: Rancangan Keseluruhan Modul Program

besar modul tersebut harus berjalan secara berurutan dan saling bergantung satu dengan 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

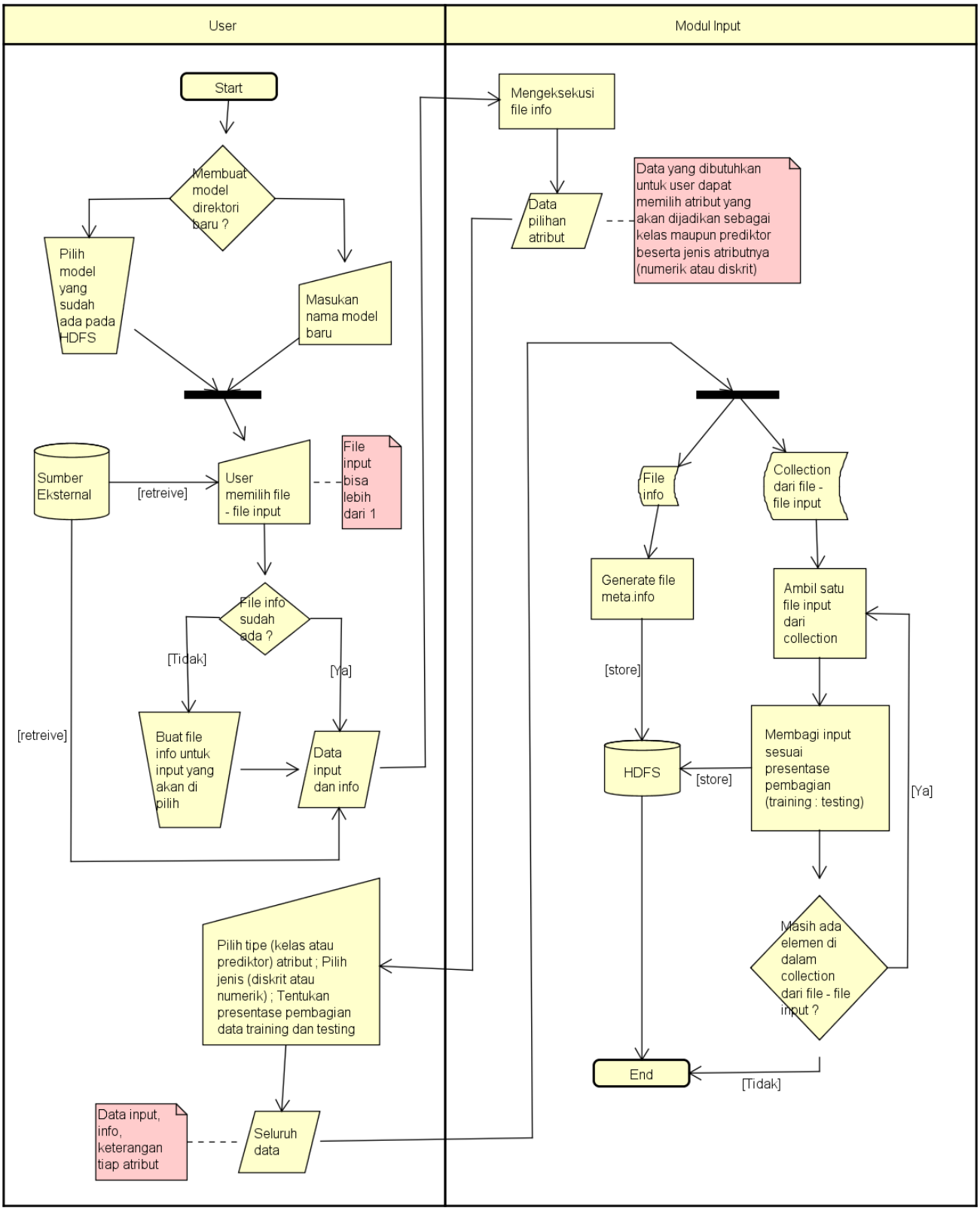
Gambar 2.2: Modul Specification

2.2.1.1 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 2.3: Flow Chart Modul Input

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 :

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

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* [1].

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 2.4: Missing-values [2]

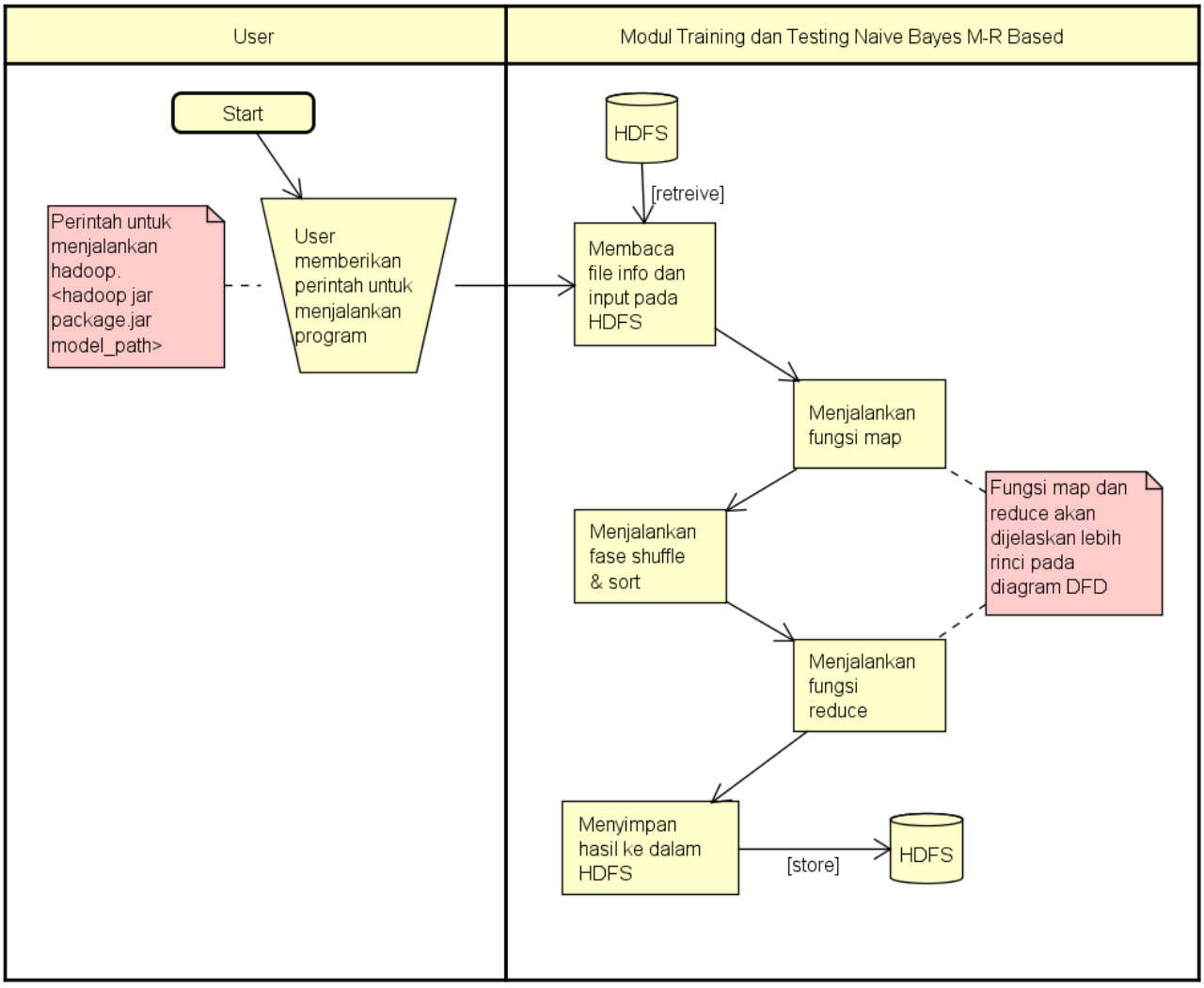
[1] CSV (comma-separated values) merupakan data tabular yang berada di dalam plain text. Setiap baris dari data tersebut menyatakan sebuah record. Setiap record memiliki 1 atau lebih field yang dipisahkan oleh koma

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

2.2.1.2 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*:

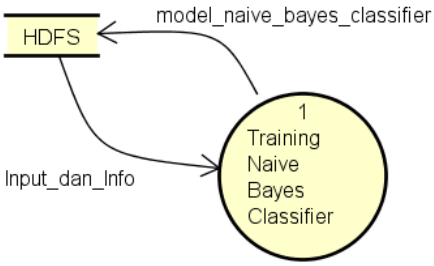


Gambar 2.5: Flow Chart Modul Training

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 2.6: *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
```

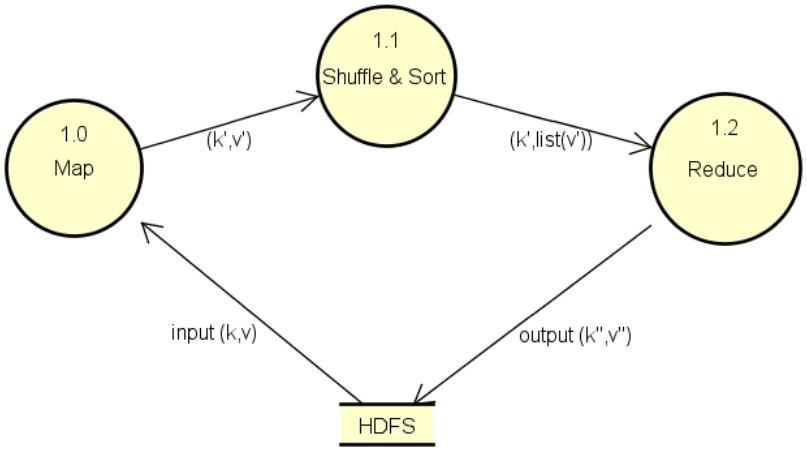
2. Data `input_dan_info`

   - 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 2.7: DFD level 1 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'))`

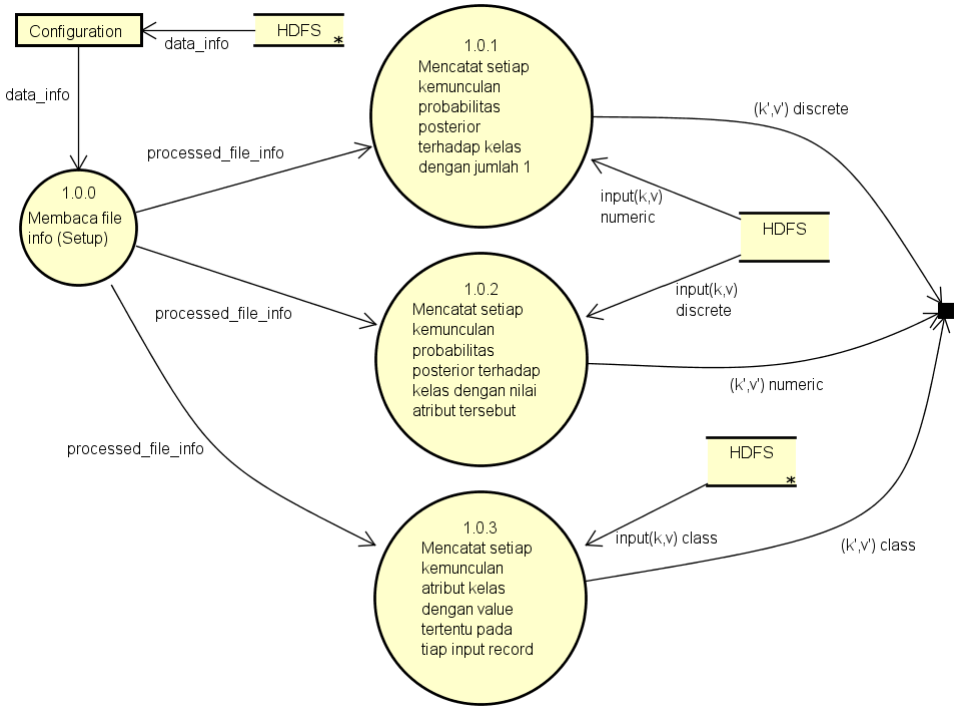
```
key                              list_of_value
   |_class|Play,Yes              [1,1,1,1]
   |disc|Humidity,High,Play,No       [1,1]
   |cont|Rand,Play,Yes             [34.2,23.3,15.0]
```

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 2.8: 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 2.9: 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 2.10: 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

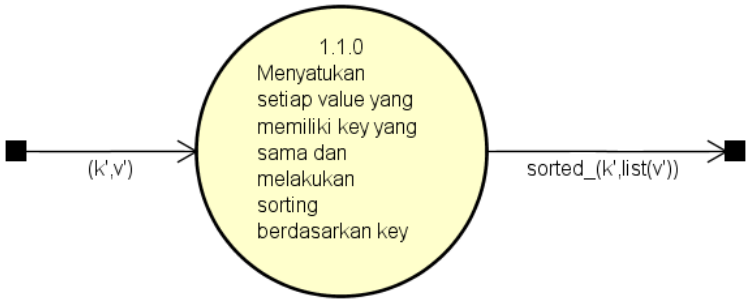
Gambar 2.11: P-Spec training map: pada proses 1.0.2

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 2.12: P-Spec training map: pada proses 1.0.3

DFD *level* 2: pada proses 1.1



Gambar 2.13: 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]
```

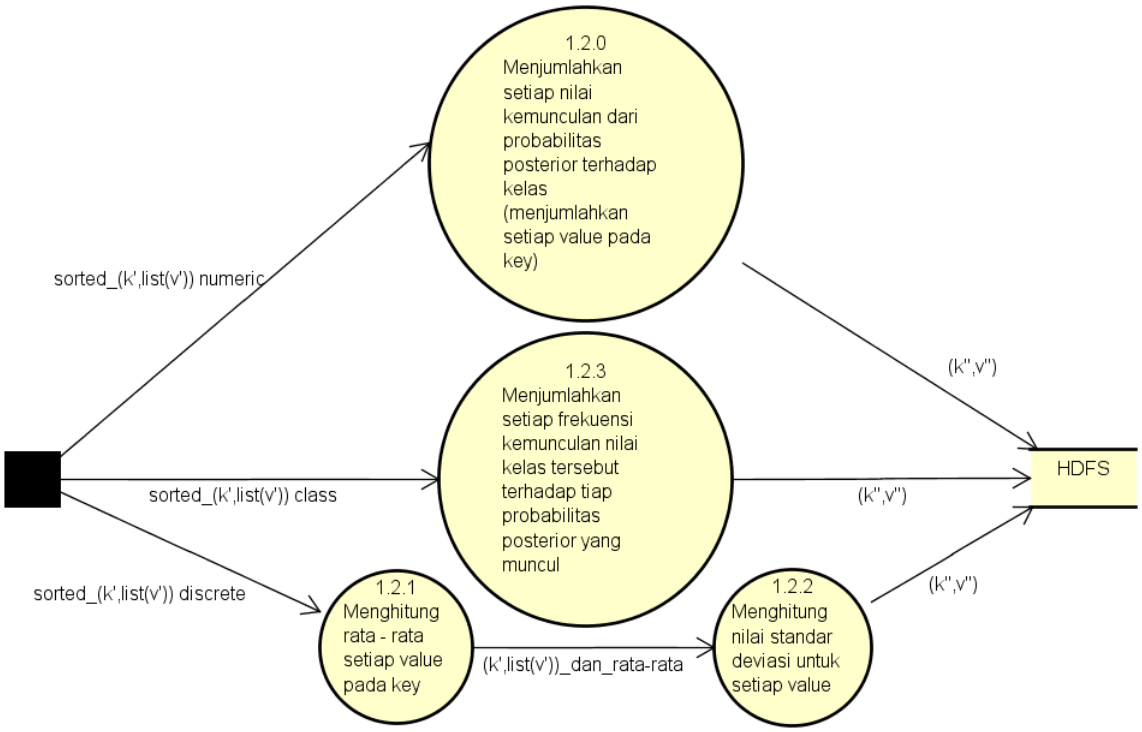
P-Spec (*Process Specification*) pada proses 1.1

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 2.14: P-Spec training shuffle sort: pada proses 1.1.0

DFD *level* 2: pada proses 1.2



Gambar 2.15: 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 2.16: 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 2.17: 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 2.18: 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

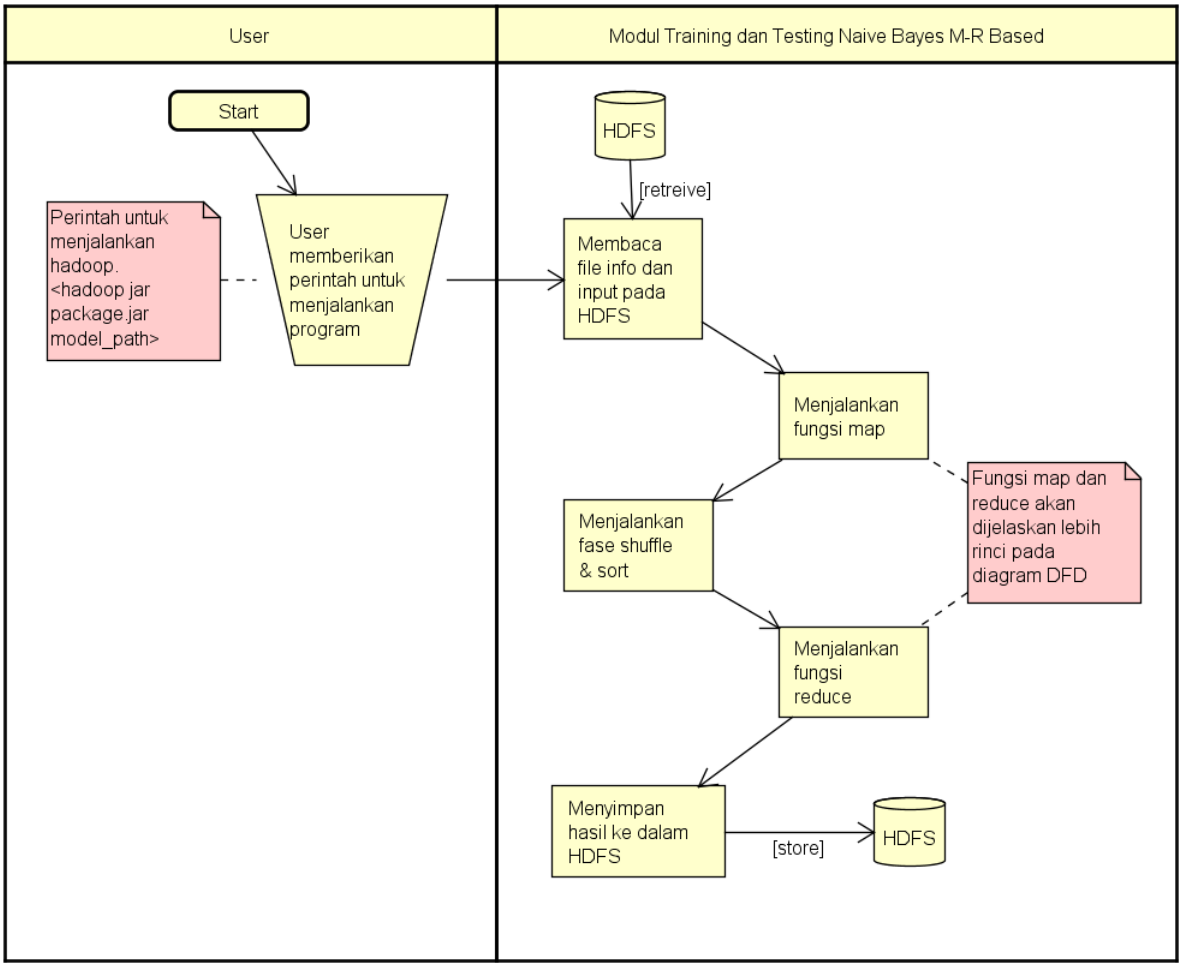
Deskripsi	Proses ini akan menjumlahkan setiap frekuensi kemunculan nilai kelas tersebut terhadap tiap probabilitas posterior yang muncul
Data In	1. Pasangan key dan value yang telah diproses pada map yang bertipe kelas
Data Out	Pasangan *key* dan *value* yang telah di proses dengan: • Key = keterangan 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 2.19: P-Spec training reduce: pada proses 1.2.3

2.2.1.3 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.

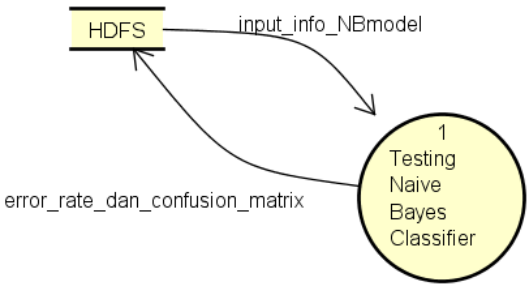
Berikut merupakan diagram *flow chart* untuk modul *testing*:



Gambar 2.20: 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 2.21: *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`

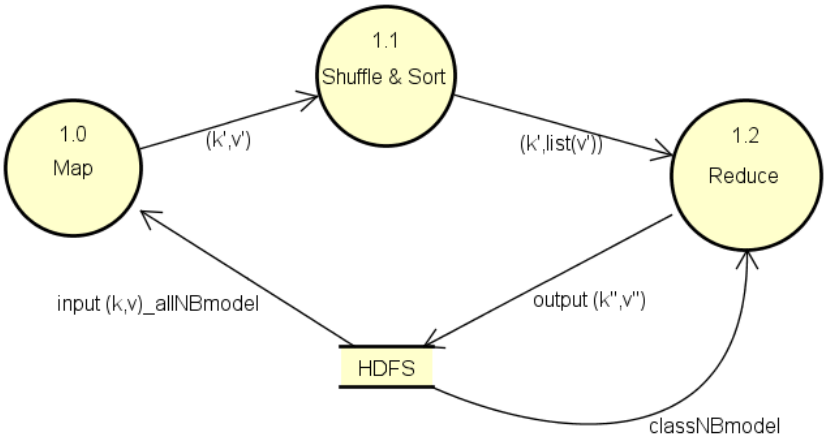
```
<- input ->
Sunny,Mild,Normal,FALSE,Yes,5
Rainy,Mild,Normal,TRUE,Yes,4.5
Overcast,Mild,High,TRUE,Yes,3.1
<- NBmodel ->
Play,Yes,2.0|CLASS
Play,No,3.0|CLASS
Humidity,Play,Yes ;82.5|3.5|NUMERIC
Humidity,Play,No ;71.0|9.6|NUMERIC
Outlook,Sunny,Play,Yes,2.0|DISCRETE
Outlook,Sunny,Play,No,1.0|DISCRETE
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 2.22: DFD level 1 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]

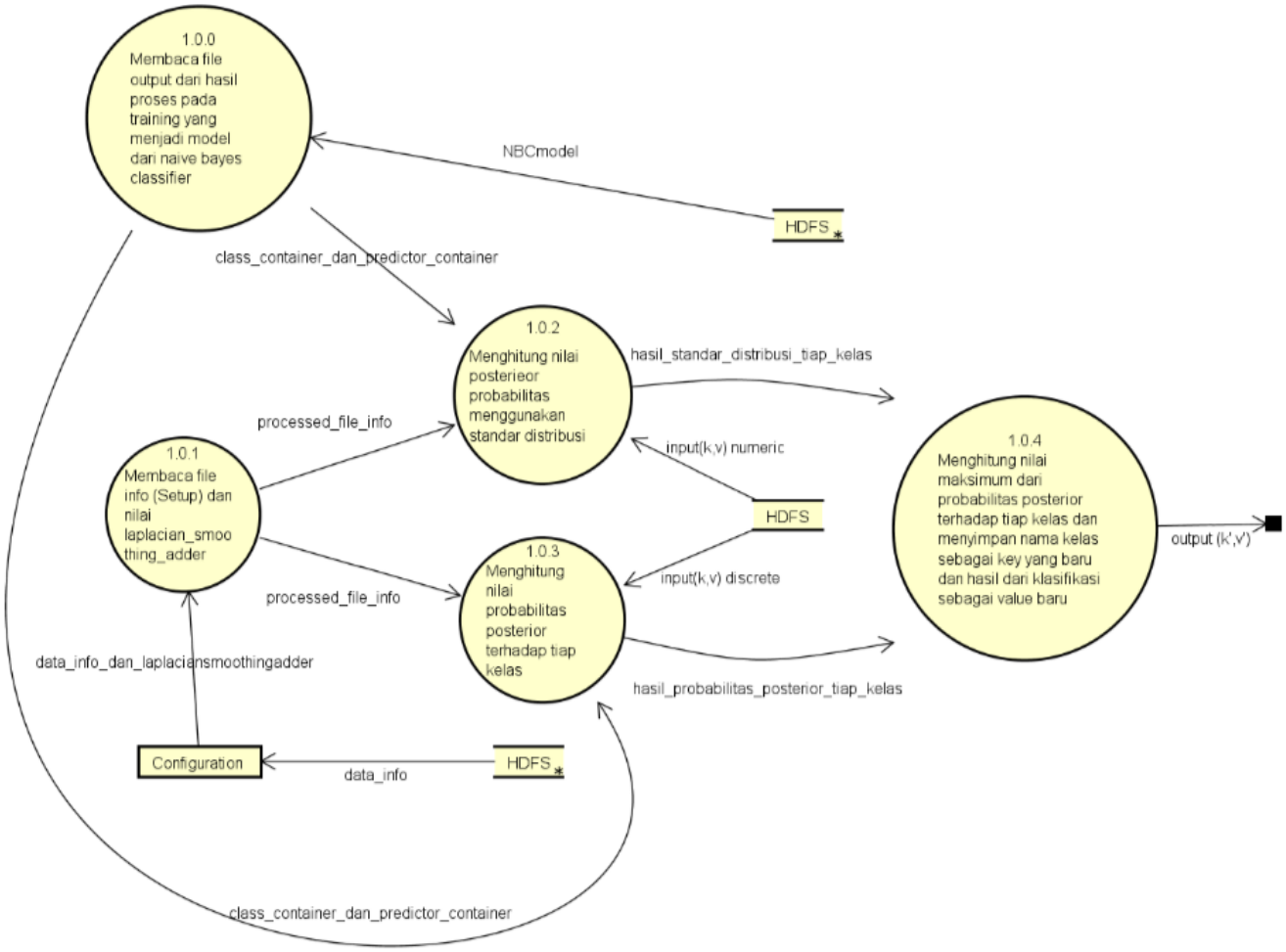
   Contoh data `output(k",v")`

```
<- 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 2.23: 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 2.24: 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 2.25: 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 2.26: 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

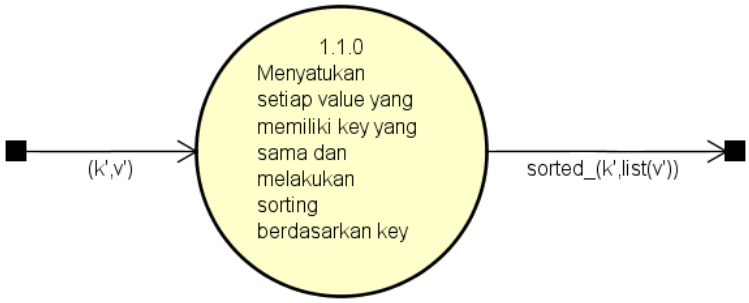
Gambar 2.27: P-Spec training reduce: pada proses 1.0.3

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 2.28: P-Spec training reduce: pada proses 1.0.4

DFD *level* 2: pada proses 1.1

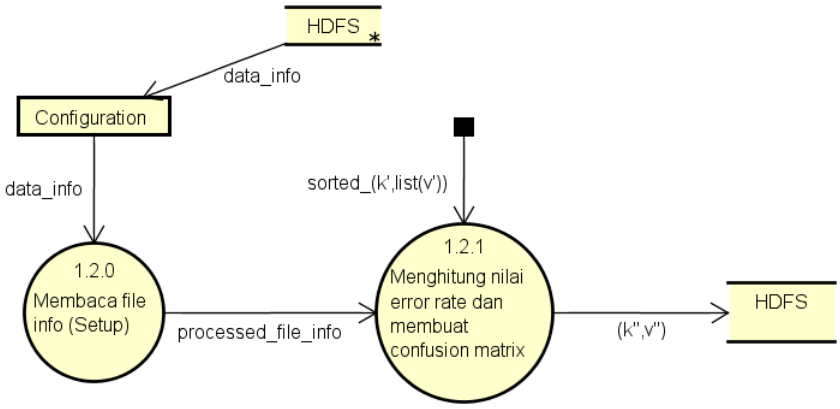


Gambar 2.29: DFD level 2: proses 1.1

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 2.30: 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 2.31: P-Spec testing reduce: pada proses 1.2.0

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

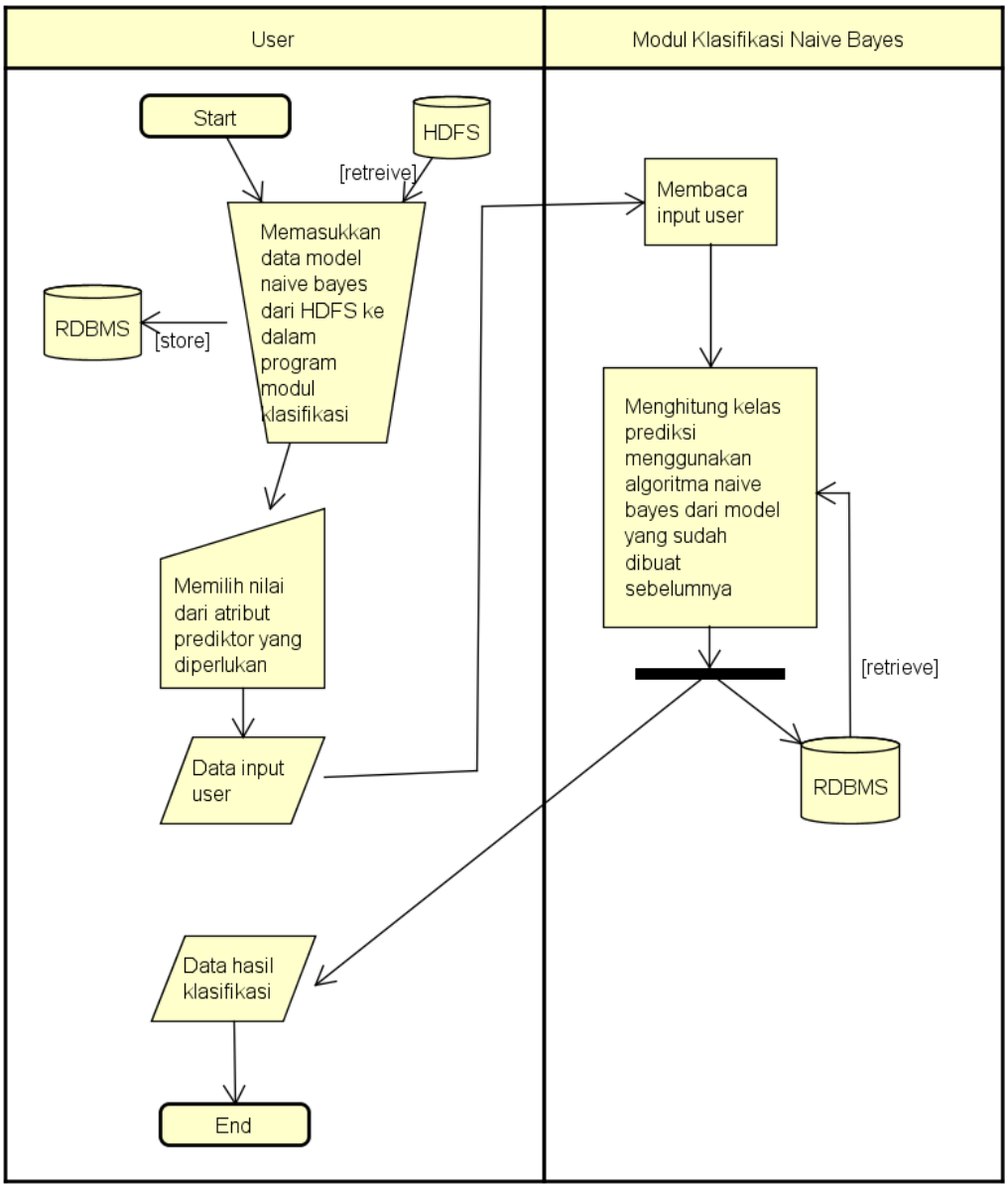
Deskripsi	Proses ini akan melakukan perhitungan nilai error rate dan membuat confusion matrix
Data In	• File info yang telah diproses • pasangan *key* dan *value*, dimana: key = nama kelas ; List<value> = kumpulan hasil keterangan prediksi terhadap kelas tersebut
Data Out	Pasangan key dan value baru, 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 2.32: P-Spec testing reduce: pada proses 1.2.1

2.2.1.4 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:

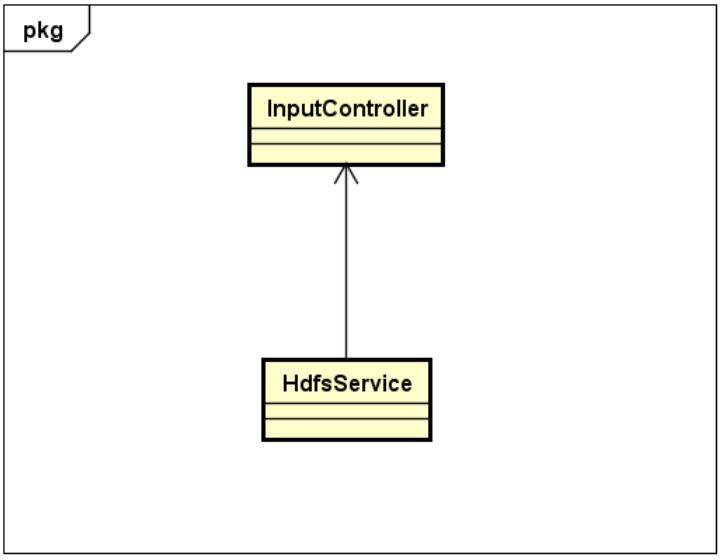


Gambar 2.33: Flow Chart Modul Klasifikasi

2.2.2 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.

2.2.2.1 Modul Kelola *Input*



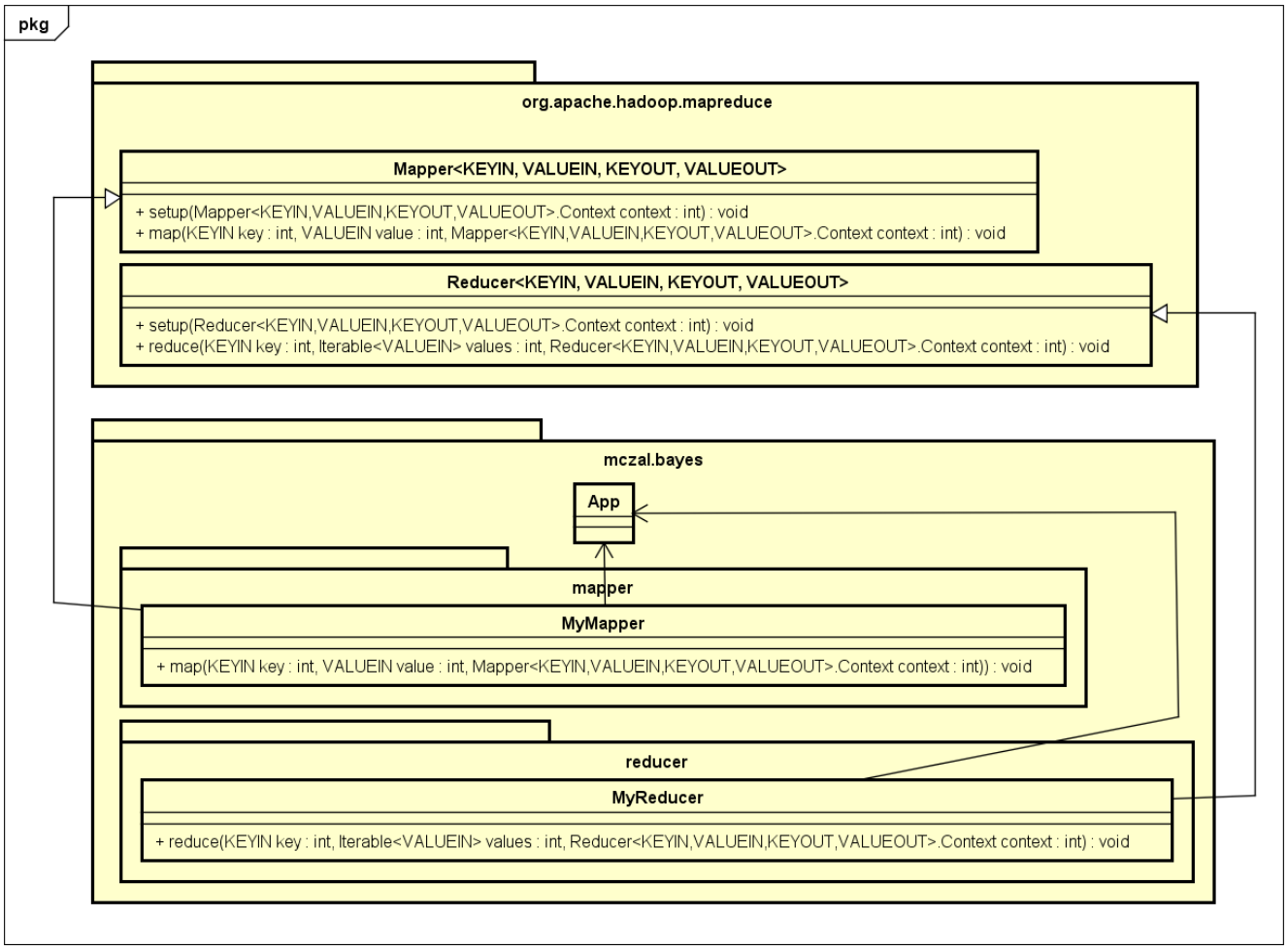
Gambar 2.34: Diagram kelas modul kelola input

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.

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



Gambar 2.35: Diagram kelas modul *training* dan *testing*

Pada modul ini, akan dibuatkan 3 kelas utama untuk menangani proses *training* dan *testing* berbasis *MapReduce*. Di dalam kelas tersebut terdapat operasi - operasi untuk menjalankan proses *MapReduce* pada modul training dan testing. Operasi - operasi ini telah dijelaskan sebelumnya pada 2.2.1.2 dan 2.2.1.3. Berikut merupakan pemetaan fungsi pada kelas diagram yang akan dibuat dan proses *MapReduce* pada DFD yang telah dibuat sebelumnya:

Tabel 2.1: Pemetaan Proses *MapReduce* Pada DFD Dengan Operasi Pada Diagram Kelas

Nomor	Proses DFD	Operasi pada kelas	Section
1	DFD level 1 modul *training* Proses 1.0 Map	MyMapper.map()	2.7
2	DFD level 1 modul *training*: Proses 1.2 Reduce	MyReducer.reduce()	2.7
3	DFD level 1 modul *testing*: Proses 1.0 Map	MyMapper.map()	2.22
4	DFD level 1 modul *testing*: Proses 1.2 Reduce	MyReducer.reduce()	2.22

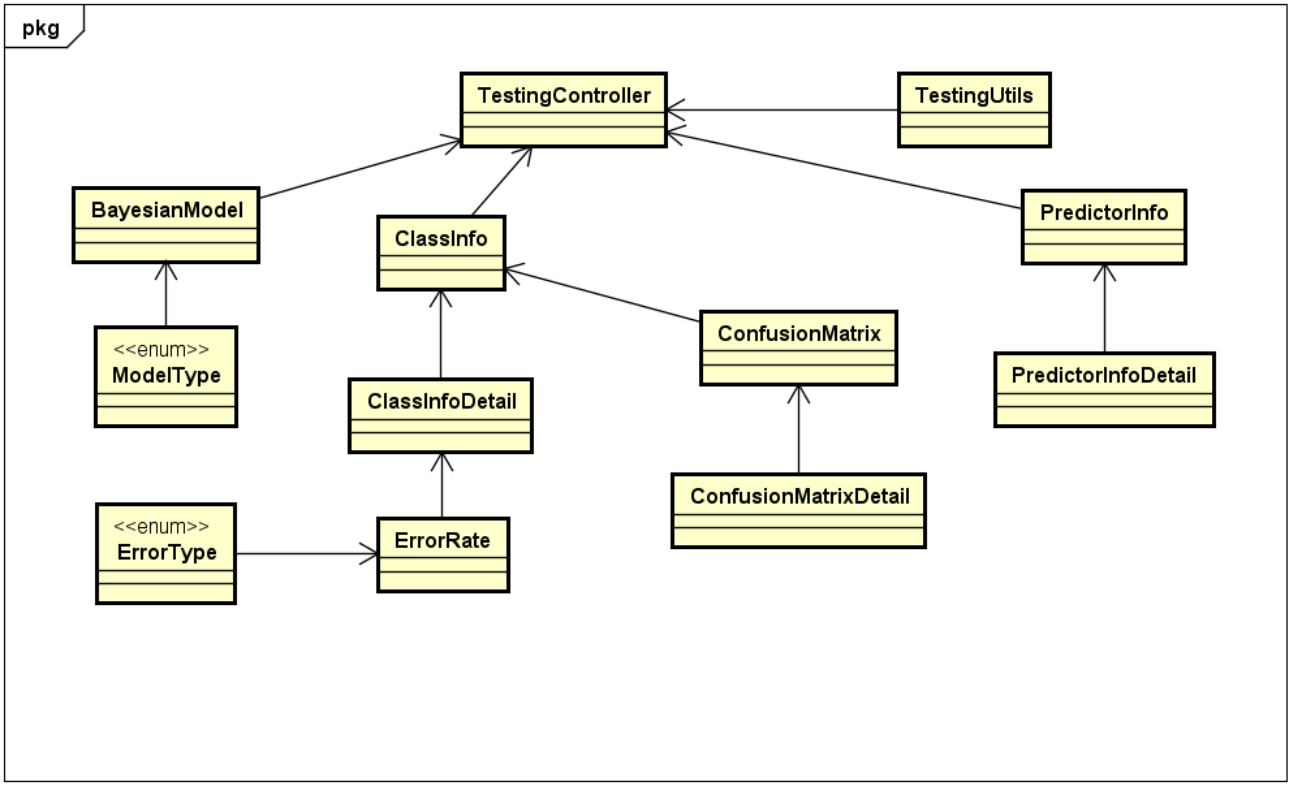
Berikut merupakan penjelasan 3 kelas 3 kelas utama untuk menangani proses *training* dan *testing* berbasis *MapReduce*:

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*.

2.2.2.3 Modul Klasifikasi *Naive Bayes*



Gambar 2.36: 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 3

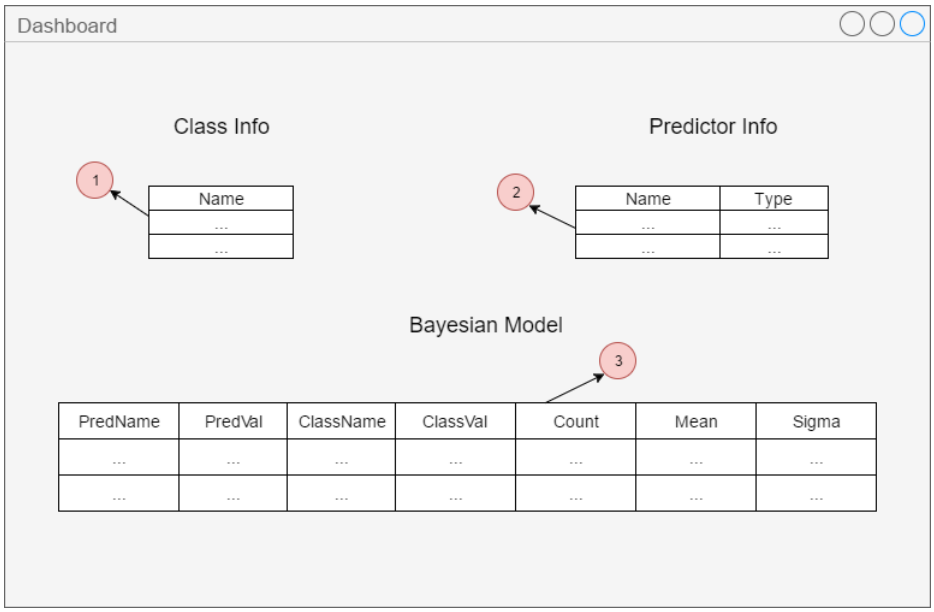
PERANCANGAN

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.

3.1 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:

3.1.1 *Dashboard*



Gambar 3.1: Dashboard

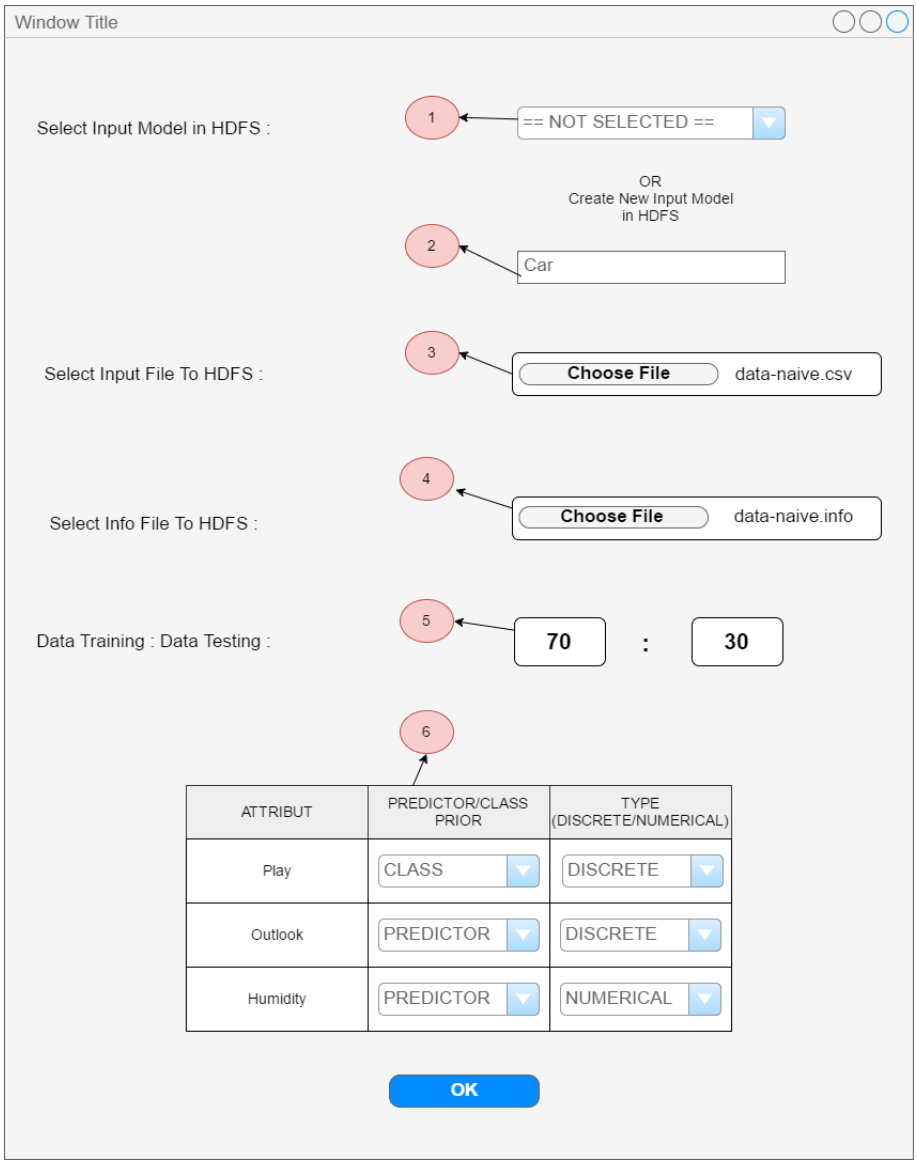
Dashboard dibuat untuk memudahkan user dalam memonitor model NBC yang telah dimasukkan ke dalam perangkat lunak yang dibangun. Berikut penjelasan lebih lanjut mengenai tiap komponen pada rancangan *dashboard* yang dibuat:

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.

3.1.2 *Input Set Manager*



Gambar 3.2: *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:

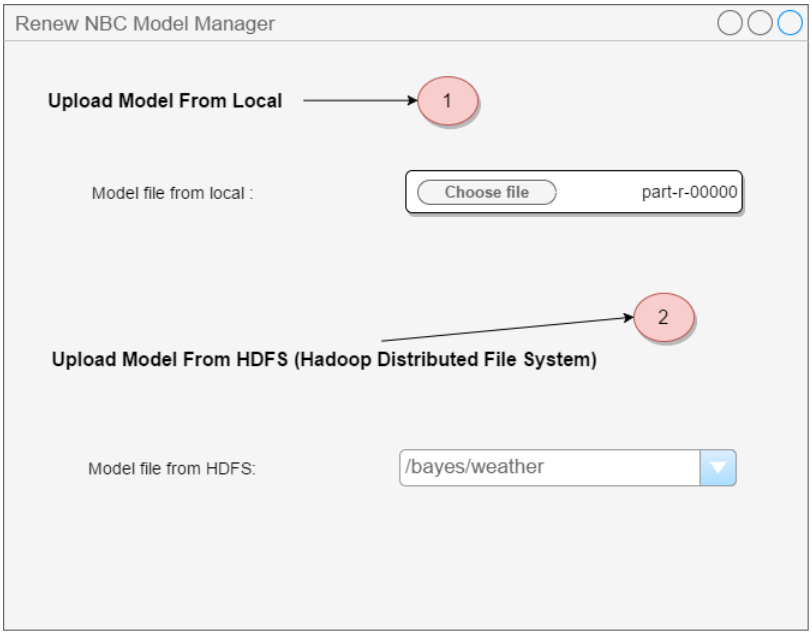
1. User dapat memilih tipe model input yang sudah ada dalam HDFS.

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).

3.1.3 *Renew Model Manager*

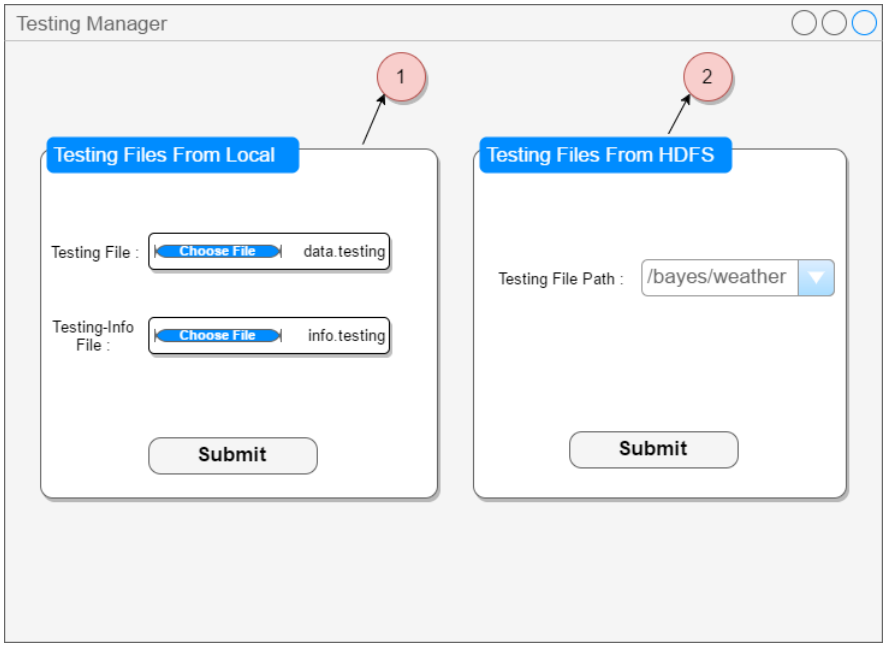


Gambar 3.3: *Renew Model Manager*

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.

3.1.4 *Testing Manager*

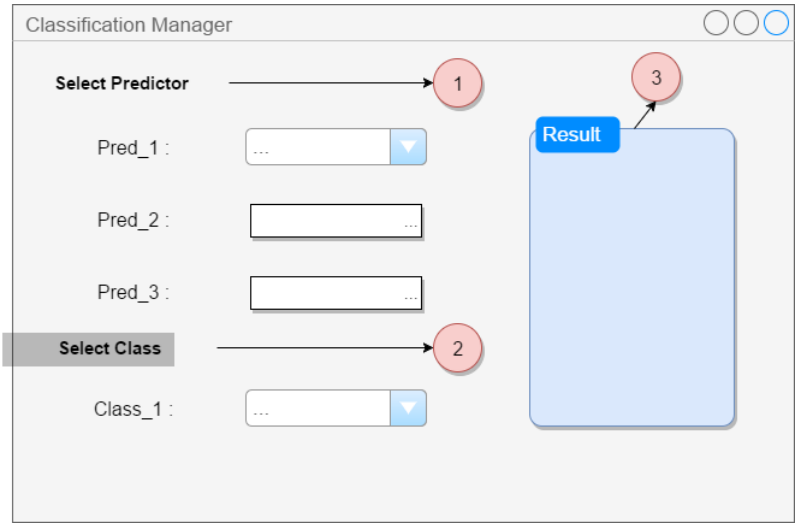


Gambar 3.4: *Testing Manager*

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.

3.1.5 *Classification Manager*

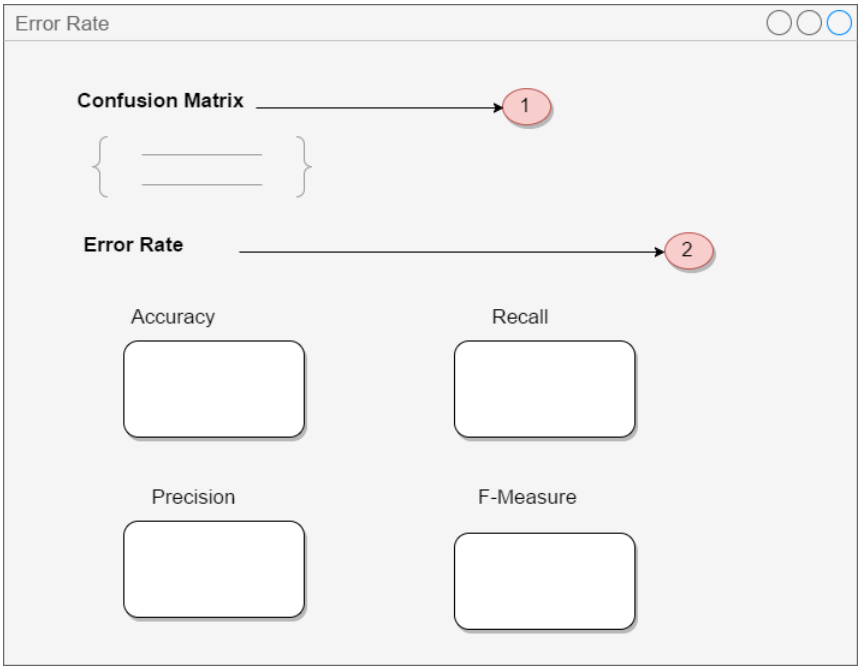


Gambar 3.5: *Classification Manager*

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.

3.1.6 *Error Rate Dashboard*



Gambar 3.6: *Error Rate Dashboard*

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

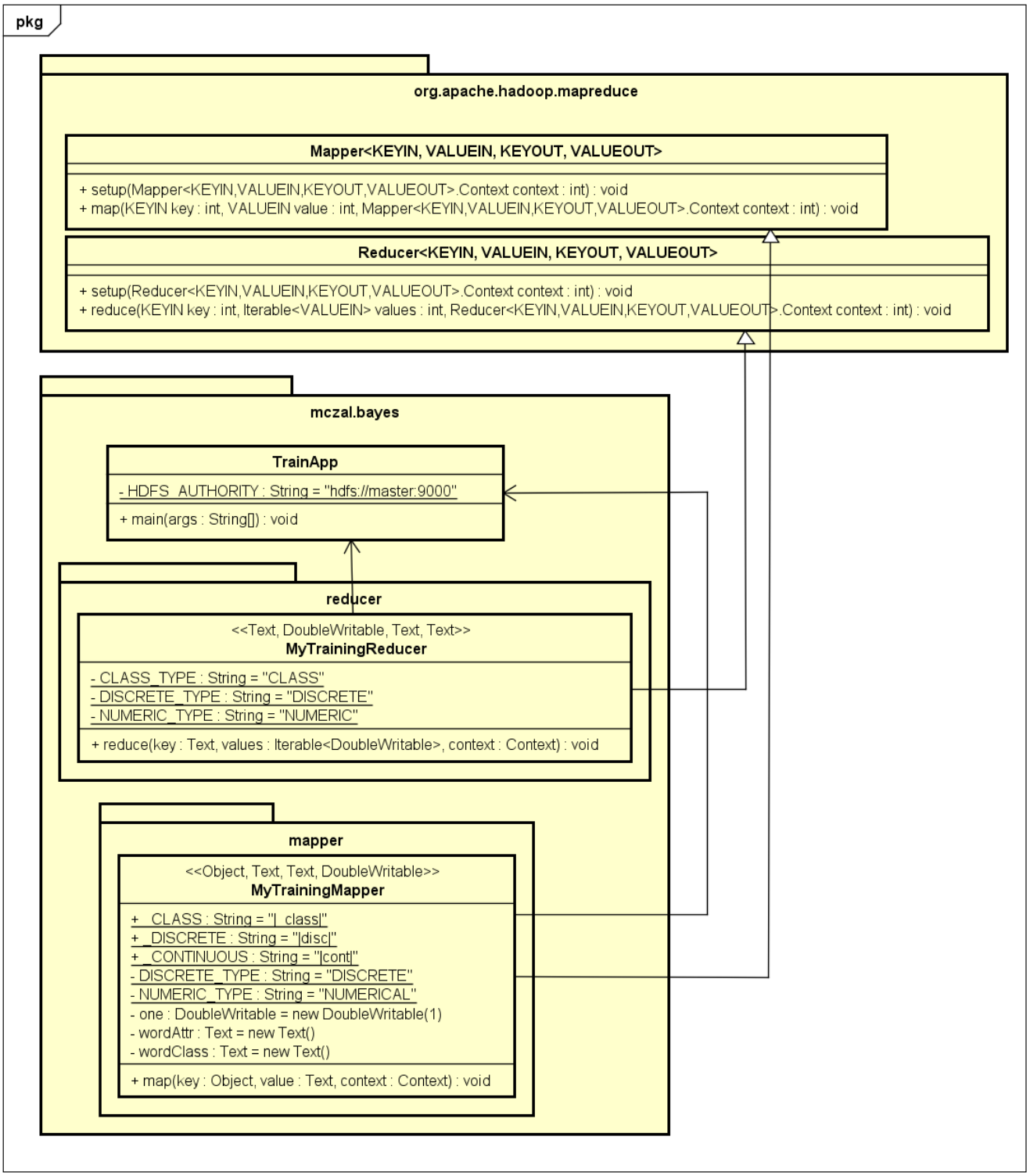
1. *Confusion matrix* untuk setiap atribut kelas.
2. Error rate yang akan dihasilkan setelah melakukan klasifikasi meliputi: (1)*Accuracy*; (2)*Precision*; (3)*Recall*; (4)$F - Measure$.

3.2 Diagram Kelas Lengkap dan Design Pattern

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

3.2.1 Diagram Kelas Modul *Train Naive Bayes M-R Based*

Pada diagram kelas ini terdapat terdapat 2 *package* utama yang akan menjadi inti dari modul ini, salah satunya merupakan *package* yang dimiliki oleh *library* dari *hadoop client* untuk dapat menjalankan proses *mapreduce*. Berikut adalah gambar dari *package* - *package* tersebut:



Gambar 3.7: *Diagram Kelas Train Naive Bayes M-R Based*

***Package** org.apache.hadoop.mapreduce*

Dalam *package* ini terdapat 2 kelas utama yang akan menjadi kelas *parent* dari kelas *reducer* dan *mapper* yang di-implementasikan.

1. Kelas *Mapper*
   Kelas ini memiliki 4 buah *generic types*[1] yang perlu di-implementasikan pada kelas

[1] A generic type is a generic class or interface that is parameterized over types [3]

yang mengimplementasi method ini. Parameter *generic types* tersebut antara lain adalah:

- *KEYIN*
  Kelas ini merupakan tipe kelas yang akan menjadi *key* masukan pada proses Map untuk program *Mapreduce* yang dibuat.
- *VALUEIN*
  Merupakan tipe kelas yang akan menjadi *value* dari masukan pada proses Map untuk program *Mapreduce* yang dibuat.
- *KEYOUT*
  Merupakan tipe kelas yang akan menjadi *key* keluaran pada proses Map untuk program *Mapreduce* yang dibuat.
- *VALUEOUT*
  Merupakan tipe kelas yang akan menjadi *value* keluaran pada proses Map untuk program *Mapreduce* yang dibuat.

2. Kelas *Reducer*
   Kelas ini memiliki 4 buah *generic types* yang perlu di-implementasikan pada kelas yang mengimplementasi method ini. Parameter *generic types* tersebut antara lain adalah:

   - *KEYIN*
     Kelas ini merupakan tipe kelas yang akan menjadi *key* masukan pada proses *Reduce* untuk program *Mapreduce* yang dibuat.
   - *VALUEIN*
     Merupakan tipe kelas yang akan menjadi *value* dari masukan pada proses *Reduce* untuk program *Mapreduce* yang dibuat.
   - *KEYOUT*
     Merupakan tipe kelas yang akan menjadi *key* keluaran pada proses *Reduce* untuk program *Mapreduce* yang dibuat.
   - *VALUEOUT*
     Merupakan tipe kelas yang akan menjadi *value* keluaran pada proses *Reduce* untuk program *Mapreduce* yang dibuat.

***Package** mczal.bayes*

Dalam *package* ini terdapat beberapa *package* lagi dan kelas - kelas yang akan dijadikan implementasi dari kelas *Mapper* dan *Reducer* pada package *org.apache.hadoop.mapreduce* untuk proses *MapReduce*.

1. Kelas *TrainApp*
   Kelas ini merupakan kelas utama(*main class*) yang akan dijalankan pertama kali program ini dieksekusi. Kelas ini memiliki 1 atribut *static* dan 1 method *main*.

   (a) Atribut `HDFS_AUTHORITY`
   Atribut ini bertipe *String* dan memiliki *modifier static* dan *final* agar nilainya tidak dapat diubah - ubah. Inisialisasi pertama dari atribut ini adalah: `"hdfs://master:9000"`. `"hdfs://master:9000"` merupakan url dari node master yang digunakan untuk dapat berkomunikasi dengan node master pada lingkungan *hadoop*. Atribut ini digunakan untuk *request* operasi baca file *meta.info* pada HDFS.

   (b) Operasi `main`
   Operasi ini akan menjadi operasi pertama yang dijalankan ketika melakukan eksekusi program mapreduce pada modul ini. Operasi ini akan men-*set* kelas *mapper* dan kelas *reducer* yang akan digunakan serta variabel - variabel yang perlu dikirimkan kepada tiap node yang dibutuhkan oleh mereka.

2. Kelas *MyTrainingMapper*
   Kelas ini merupakan kelas turunan dari kelas *Mapper* yang ada pada package `org.apache.hadoop.mapred`
   Generic type kelas parent dari kelas ini adalah:

   (a) Object merupakan tipe *key* masukan dari proses *map*.

   (b) Text merupakan tipe *key* masukan dari proses *map*. Karena, data masukan akan berupa *string*.

   (c) Text merupakan tipe *key* keluaran dari proses *map*. Karena, *key* pada hasil dari fase *map* akan mengeluarkan data yang bertipe *string*.

   (d) DoubleWritable merupakan tipe *value* keluaran dari proses *map*. Karena, *value* keluaran pada proses *map* akan berisi jumlah frekuensi atau nilai dari atribut prediktor-numerik yang bertipe numerik.

   Kelas ini memiliki 8 atribut. Diantaranya adalah:

   (a) `_CLASS` merupakan atribut bertipe *string* yang memiliki *modifier static* dan *final*. Atribut ini memiliki nilai inisialisasi awal = "|*_class*|" yang akan digunakan untuk membedakan bahwa data keluaran yang memiliki string berisi "|*_class*|" merupakan data keluaran yang akan digunakan untuk menghitung jumlah frekuensi dari kemunculan atribut bertipe kelas.

   (b) `_DISCRETE` merupakan atribut bertipe *string* yang memiliki *modifier static* dan *final*. Atribut ini memiliki nilai inisialisasi awal = "|*disc*|" yang akan digunakan untuk membedakan bahwa data keluaran yang memiliki string berisi "|*disc*|" merupakan data keluaran yang akan digunakan untuk menghitung jumlah frekuensi dari kemunculan atribut prediktor bertipe diskrit berdasarkan atribut kelas tertentu.

   (c) `_CONTINUOUS` merupakan atribut bertipe *string* yang memiliki *modifier static* dan *final*. Atribut ini memiliki nilai inisialisasi awal = "|*cont*|" yang akan digunakan untuk membedakan bahwa data keluaran yang memiliki string berisi "|*cont*|" merupakan data keluaran yang akan digunakan untuk mencatat nilai dari atribut numerik berdasarkan atribut kelas tertentu.

   (d) `DISCRETE_TYPE` merupakan atribut bertipe *string* yang memiliki *modifier static* dan *final*. Atribut ini memiliki nilai inisialisasi awal = "*DISCRETE*" yang akan digunakan untuk membedakan bahwa data input setelah displit dengan regex ',' pada indeks tertentu memiliki info bertipe diskrit.

   (e) `NUMERIC_TYPE` merupakan atribut bertipe *string* yang memiliki *modifier static* dan *final*. Atribut ini memiliki nilai inisialisasi awal = "*NUMERICAL*" yang akan digunakan untuk membedakan bahwa data input setelah displit dengan regex ',' pada indeks tertentu memiliki info bertipe numerik.

   (f) `one` merupakan atribut bertipe *DoubleWritable* yang memiliki *modifier static*. Atribut ini memiliki nilai inisialisasi awal = integer bernilai 1 yang akan digunakan untuk mencatat tiap kemunculan atribut prediktor terhadap atribut kelas tertentu yang sedang diperiksa dengan jumlah kemunculan sebanyak 1.

   (g) `wordAttr` merupakan atribut bertipe *Text* yang memiliki *modifier static*. Atribut ini akan digunakan untuk mencatat tiap atribut prediktor yang akan ditulis menjadi *output key* untuk perhitungan probabilitas posterior dari atribut tertentu pada atribut kelas tertentu.

   (h) `wordClass` merupakan atribut bertipe *Text* yang memiliki *modifier static*. Atribut ini akan digunakan untuk mencatat tiap kemunculan atribut kelas tertentu dan akan ditulis menjadi *output key* untuk perhitungan frekuensi atribut kelas.

   Kelas ini memiliki 1 buah operasi. Operasi tersebut adalah operasi *map(key : Object, value : Text, context : Context) : void* yang akan melakukan operasi pada fase

map untuk proses training pada modul ini. Seperti yang digambarkan pada DFD sebelumnya, proses ini akan melakukan perhitungan jumlah frekuensi kemunculan tiap atribut. Terkecuali untuk atribut prediktor yang bertipe numerik, operasi ini akan mengeluarkan kembali nilai yang dibaca dari input.

Algorithm 1 NBC Model Map Algorithm

```
1: procedure MAP(key, value, context)                          ▷ Map function
2:     _CLASS ← "|_class|"
3:     _DISCRETE ← "|disc|"
4:     _CONTINUOUS ← "|cont|"
5:     one ← 1
6:     NUMERIC_TYPE ← "NUMERICAL"
7:     DISCRETE_TYPE ← "DISCRETE"
8:     countColumn ← getInputColumnCount
9:     inputSplit[] ← value.split(',')
10:    if inputSplit.length != countCols then                  ▷ Ignoring missing values
11:        return
12:    classConf ← getClassInfo                                ▷ get class info from meta.info
13:    classSplitConf[] ← classConf.split(";")
14:    attrConf ← getAttributeInfo                             ▷ get predictor info from meta.info
15:    attrSplitConf[] ← attrConf.split(";")
16:    checkerClassPrior ← classSplitConf.length
17:    for i ← 0 to attrSplitConf.length do
18:        if attrSplitConf[i].split(",")[2].equals(DISCRETE_TYPE) then
19:            for j ← 0 to classSplitConf.length do
20:                currKey ←_ DISCRETE + attrSplitConf[i].split(",")[0] +
   "," + inputSplit[Integer.parseInt(attrSplitConf[i].split(",")[1])] + "," +
   classSplitConf[j].split(",")[0]+","+inputSplit[Integer.parseInt(classSplitConf[j].split(",")[1])];
21:                wordAttr.set(currKey);
22:                context.write(wordAttr, one)
```

3. Kelas *MyTrainingReducer*
   Kelas ini merupakan kelas turunan dari kelas *Reducer* pada *package org.apache.hadoop.mapreduce*. *Generic type* kelas parent dari kelas ini adalah:

   (a) Text merupakan tipe *key* masukan dari proses *reduce* yang dikirim dari proses sebelumnya yaitu proses *map*

   (b) DoubleWritable merupakan tipe *value* masukan dari proses *reduce* yang dikriim dari proses sebelumnya yaitu proses *map*.

   (c) Text merupakan tipe *key* keluaran dari proses *reduce*. Karena data keluaran berupa model NBC akan memiliki key berisi string.

   (d) Text merupakan tipe *value* keluaran dari proses *reduce*. Karena data keluaran berupa model NBC akan memiliki *value* berisi string.

   Kelas ini memiliki 3 atribut yang memiliki *modifier static* dan *final*. Atribut tersebut diantaranya adalah:

   (a) `CLASS_TYPE` merupakan atribut bertipe string yang memiliki nilai inisialisasi awal = *"CLASS"*. Atribut ini akan digunakan sebagai pembeda pada data keluaran yang merupakan atribut kelas.

   (b) `DISCRETE_TYPE` merupakan atribut bertipe string yang memiliki nilai inisialisasi awal = *"DISCRETE"*. Atribut ini akan digunakan sebagai pembeda pada data keluaran yang merupakan atribut prediktor bertipe diskrit.

(c) `NUMERIC_TYPE` merupakan atribut bertipe string yang memiliki nilai inisialisasi awal = *"NUMERIC"*. Atribut ini akan digunakan sebagai pembeda pada data keluaran yang merupakan atribut prediktor bertipe numerik.

Kelas ini memiliki 1 buah operasi. Operasi tersebut adalah operasi *reduce(key : Text, values : Iterable<DoubleWritable>, context : Context) : void* yang akan melakukan operasi pada fase reduce untuk proses training pada modul ini. Seperti yang digambarkan pada DFD sebelumnya, proses ini akan melakukan akumulasi dari tiap perhitungan jumlah frekuensi kemunculan tiap value yang memiliki key yang sama. Terkecuali untuk atribut prediktor yang bertipe numerik, operasi ini akan menghitung nilai rata - rata dari tiap value pada key tersebut dan lalu dilanjutkan menghitung nilai standar deviasi pada probabilitas posterior atribut numerik tersebut.

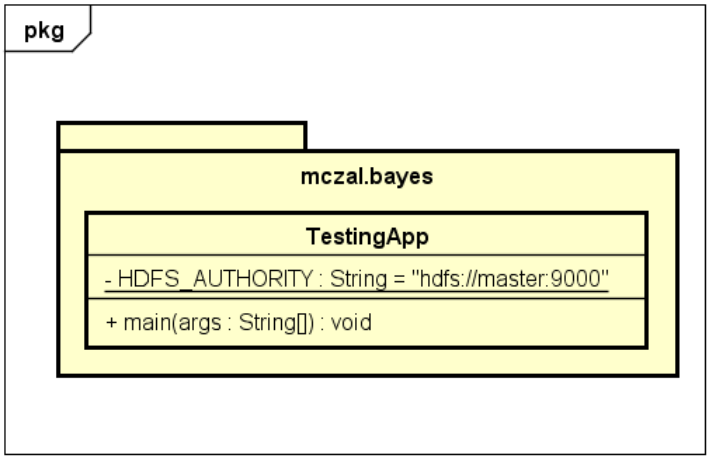
Algorithm 2 NBC Model Reduce Algorithm

```
1: procedure REDUCE(key, values[], context)                    ▷ Reduce function
2:     CLASS_TYPE ← "CLASS"
3:     DISCRETE_TYPE ← "DISCRETE"
4:     NUMERIC_TYPE ← "NUMERIC"
5:     sum ← 0
6:     count ← 0
7:     caches ← List<Double>
8:     for all value ∈ values do
9:         caches.add(value)
10:        sum += value
11:        count++
12:    if key.type ← discrete then
13:        splitter ← key.split("|")[2]
14:        result ← splitter + "," + sum + "|" +DISCRETE_TYPE
15:        write(result,"")
16:    else if key.type ← class then
17:        splitter ← key.split("|")[2]
18:        result ← splitter + "," + sum + "|" +CLASS_TYPE
19:        write(result,"")
20:    else if key.type ← contiuous then
21:        mean ← sum/count
22:        calcTemp ← 0.0
23:        for all c ∈ caches do
24:            calcTemp += (c − mean)^2
25:        calcTemp ← calcTemp * (1/count)
26:        sigma ← calcTemp^0.5
27:        result ← key.split("|")[2]
28:        write(result, ";" + mean + "|" + sigma + "|" +)NUMERIC_TYPE
```

3.2.2 Diagram Kelas Modul *Testing Naive Bayes M-R Based*

Pada diagram kelas ini terdapat 1 kelas utama(*main*) yang akan dijalankan pertama kali saat program dieksekusi dan 3 package utama yang merupakan implementasi dari modul *testing naive bayes*. Untuk kelas yang berfungsi sebagai *mapper* dan *reducer* pada modul ini, sama dengan diagram kelas pada modul "*Training Naive Bayes M-R Based*" merupakan kelas implementasi(kelas turunan) dari 2 kelas Mapper dan Reducer milik hadoop pada package `org.apache.hadoop.mapreduce`. Karena bentuk relasi-nya sama persis dengan yang ada pada modul "*Training Naive Bayes M-R Based*", pada modul ini tidak akan digambarkan relasi kelas mapper dan reducer dengan 2 kelas parent milik *library hadoop*.

3.2.2.1 Kelas *Main*



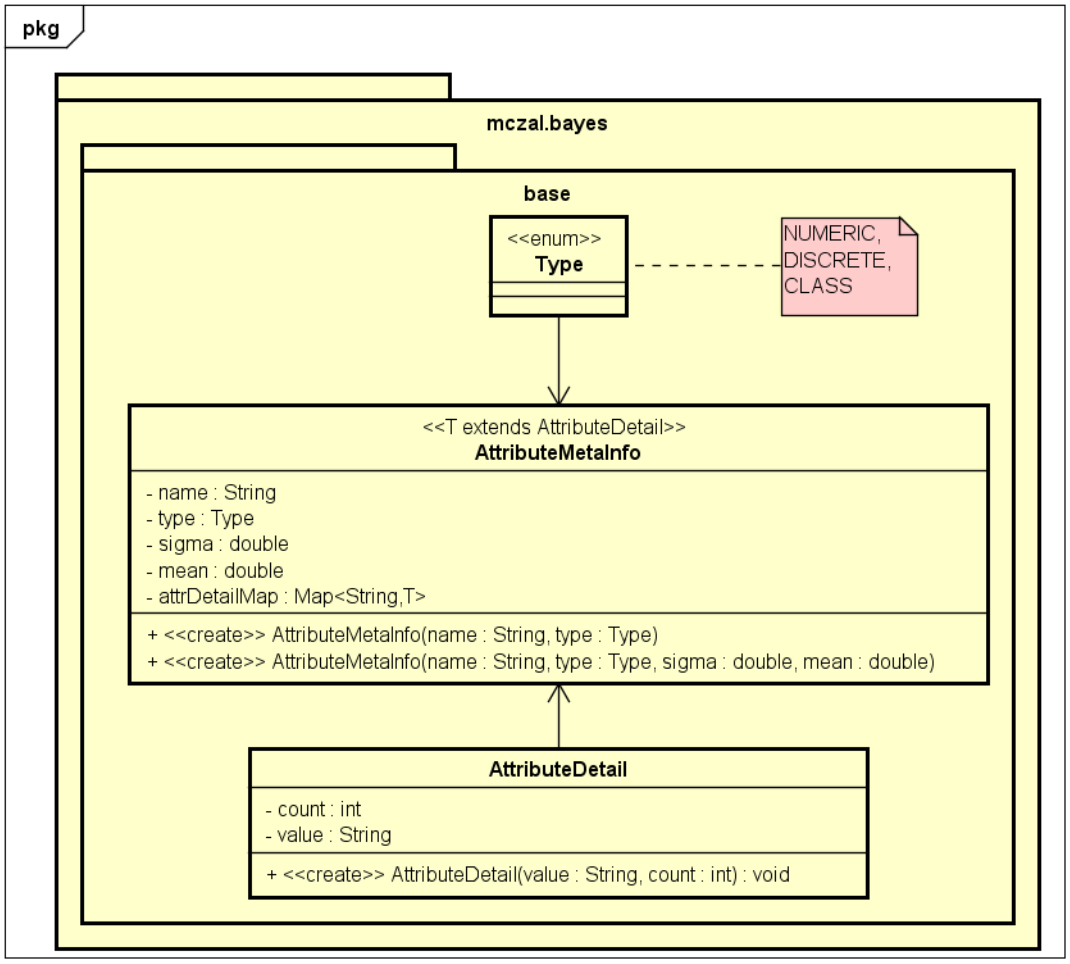
Gambar 3.8: Diagram Kelas Modul *Testing: Main*

Kelas *TestingApp* ini merupakan kelas utama(*main class*) yang akan dijalankan pertama kali program *MapReduce* ini dieksekusi. Kelas ini memiliki 1 atribut *static* dan 1 method *main*.

1. Atribut `HDFS_AUTHORITY`
   Atribut ini bertipe *String* dan memiliki *modifier static* dan *final* agar nilainya tidak dapat diubah - ubah. Inisialisasi pertama dari atribut ini adalah: `"hdfs://master:9000"`. `"hdfs://master:9000"` merupakan url dari node master yang digunakan untuk dapat berkomunikasi dengan node master pada lingkungan *hadoop*. Atribut ini digunakan untuk *request* operasi baca file *meta.info* pada HDFS.

2. Operasi `main`
   Operasi ini akan menjadi operasi pertama yang dijalankan ketika melakukan eksekusi program mapreduce pada modul ini. Operasi ini akan men-*set* kelas *mapper* dan kelas *reducer* yang akan digunakan serta variabel - variabel yang perlu dikirimkan kepada tiap node yang dibutuhkan oleh mereka.

3.2.2.2 *Package Base*



Gambar 3.9: Diagram Kelas Modul *Testing: Package Base*

Pada package ini terdapat 3 kelas utama yang akan menjadi kelas *parent*(*superclass*) dari kelas - kelas yang akan mengimplementasikan-nya. Kelas - kelas parent pada package ini dibuat untuk memenuhi konsep inheritance yang baik dan menghindari terjadinya duplikasi kode pada kelas - kelas terseubt. Kelas - kelas ini juga akan meminimumkan kode yang nantinya dibuat. Berikut adalah penjelasan lebih lanjut mengenai tiap kelas serta atribut dan operasi yang dimilikinya:

1. `Enum`[2] *Type*
   ini berisi:
   (a) NUMERIC
   (b) DISCRETE
   (c) CLASS

   Enum ini akan digunakan sebagai pengenal tipe dari tiap atribut yang ada pada model NBC.

2. Kelas `AttributeMetaInfo`
   Kelas ini memiliki 1 buah parameter *generic type* yang perlu diimplementasikan pada

[2]Enum adalah sebuah tipe data yang nilainya hanya terbatas dari pilihan nilai-nilai yang telah didefinisikan terlebih dahulu. *Enumeration* di Java baru diperkenalkan pada versi Java 5.

kelas yang akan menjadi turunan dari kelas ini. *Generic type* pada kelas ini harus merupakan turunan dari kelas `AttributeDetail`. Kelas ini akan mencatat seluruh atribut(jenis: prediktor,kelas;tipe: numerik,diskrit) yang ada pada model NBC. Atribut yang akan dimiliki oleh kelas ini merupakan atribut yang merepresentasikan atribut pada model NBC secara umum. Atribut - atribut pada kelas ini adalah:

(a) `name`
Atribut ini bertipe `string` yang akan merepresentasikan nama dari atribut pada model NBC.

(b) `type`
Atribut ini bertipe `Enum Type` yang akan merepresentasikan jenis dari atribut pada model NBC.

(c) `sigma`
Atribut ini bertipe `double` yang akan merepresentasikan nilai standar deviasi dari keseluruhan data atribut ini pada model NBC jika dan hanya jika tipe dari atribut ini merupakan numerik. Jika tidak, maka nilai atribut ini akan selalu dikosongkan.

(d) `mean`
Atribut ini bertipe `double` yang akan merepresentasikan nilai rata - rata dari keseluruhan data atribut ini pada model NBC jika dan hanya jika tipe dari atribut ini merupakan numerik. Jika tidak, maka nilai atribut ini akan selalu dikosongkan.

(e) `attrDetailMap`
Atribut ini bertipe `Map` dengan `key=string` dan `value=T(generic type)` yang akan merepresentasikan kumpulan dari semua nilai yang dimiliki oleh atribut ini jika dan hanya jika tipe dari atribut ini merupakan diskrit atau atribut kelas. Jika tidak, maka nilai atribut ini akan selalu dikosongkan.

Kelas ini menerima 2 buah operasi konstruktor untuk membuat instansiasi dan menginisialisasi nilai - nilai dari atribut di dalamnya. Konstruktor tersebut antara lain adalah:

(a) `AttributeMetaInfo(name : String, type : Type)`
Konstruktor ini digunakan untuk menginisialisasi atribut tanpa mengisi nilai dari atribut `sigma` dan `mean`. Konstruktor ini biasa digunakan untuk atribut bertipe *non-numerik*.

(b) `AttributeMetaInfo(name : String, type : Type, sigma : double, mean : double)`
Konstruktor ini digunakan untuk menginisialisasi atribut dengan mengisi seluruh atribut yang ada, kecuali `attrDetailMap`. Konstruktor ini biasa digunakan untuk atribut bertipe numerik.

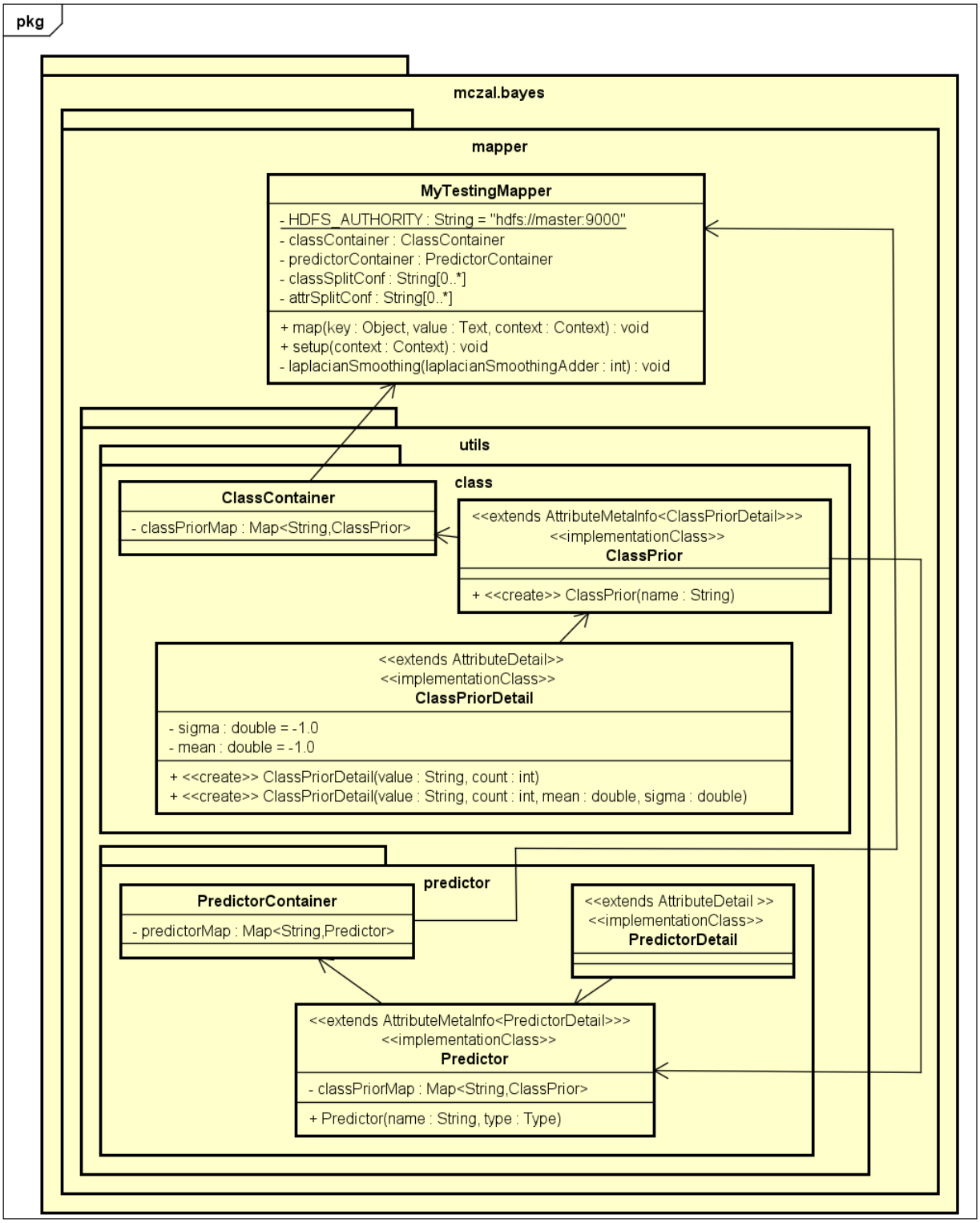
3. Kelas `AttributeDetail`
Kelas ini akan merepresentasikan tiap nilai diskrit dari atribut diskrit/kelas yang ada pada model NBC. Selain mencatat nama dari nilai diskrit atribut diskrit/kelas pada NBC, kelas ini juga akan mencatat jumlah frekuensi dari atribut ini pada model NBC. Kelas ini memiliki 2 atribut dan 1 operasi konstruktor, diantaranya adalah:

(a) Atribut `count` bertipe `integer` yang akan digunakan untuk mencatat jumlah frekuensi dari kemunculan nilai atribut ini.

(b) Atribut `value` bertipe `string` yang akan digunakan untuk mencatat nama dari nilai diskrit instansiasi dari kelas ini.

(c) Operasi konstruktor `AttributeDetail(value : String, count : int): void` akan menginisialisasi nilai awal seluruh atribut yang dimiliki oleh kelas ini pada saat instansiasi objek dari kelas ini.

3.2.2.3 *Package Mapper*



Gambar 3.10: Diagram Kelas Modul *Testing: Package Mapper*

Pada package ini terdapat 7 kelas utama yang akan digunakan untuk melakukan proses pada fase *map* pada modul testing. Berikut merupakan penjelasan tiap package dan kelas - kelas yang ada pada `Package mapper`:

1. Kelas `MyTestingMapper`
   Kelas ini merupakan kelas utama yang akan dijalankan pada proses map. Kelas ini

memiliki 5 atribut dan 3 operasi yang akan digunakan untuk dapat melakukan operasi pada fase map secara keseluruhan. Berikut merupakan penjelasan mengenai atribut dan operasi pada kelas ini:

- Atribut
  (a) `HDFS_AUTHORITY`
  Atribut ini bertipe *String* dan memiliki *modifier static* dan *final* agar nilainya tidak dapat diubah - ubah. Inisialisasi pertama dari atribut ini adalah: `"hdfs://master:9000"`. `"hdfs://master:9000"` merupakan url dari node master yang digunakan untuk dapat berkomunikasi dengan node master pada lingkungan *hadoop*. Atribut ini digunakan untuk *request* operasi baca file *meta.info* dan model NBC pada HDFS.
  (b) `classContainer`
  Atribut ini bertipe `ClassContainer` yang akan menyimpan seluruh model atribut NBC yang merupakan atribut kelas.
  (c) `predictorContainer`
  Atribut ini bertipe `PredictorContainer` yang akan menyimpan seluruh model atribut NBC yang merupakan atribut prediktor.
  (d) `classSplitConf`
  Atribut ini bertipe `array of string` yang menyimpan seluruh keterangan mengenai field - field yang merupakan atribut bertipe kelas yang digunakan pada model NBC. Keterangan yang disimpan pada atribut ini adalah: (1) nomor indeks; (2) name field; (3) jenis atribut.
  (e) `attrSplitConf`
  Atribut ini bertipe `array of string` yang menyimpan seluruh keterangan mengenai field - field yang merupakan atribut bertipe prediktor yang digunakan pada model NBC. Keterangan yang disimpan pada atribut ini adalah: (1) nomor indeks; (2) name field; (3) jenis atribut.
- Operasi
  (a) `map(` Operasi ini akan melakukan proses pada fase *map* untuk proses testing pada modul ini. Proses yang dilakukan akan meliputi perhitungan nilai probabilitas posterior untuk setiap input pada value. Proses perhitungan tersebut akan dibagi menjadi 2 proses yang terpisah untuk atribut numerik dan diskrit. Setelah diketahui hasilnya, lalu akan disimpan hasil-nya untuk dijadikan keluaran pada operasi ini. Berikut merupakan `pseudo-code` pada operasi ini:

Algorithm 3 NBC Testing algorithm

```
1: procedure MAP(key, value, context)                          ▷ Map function
2:     input ← value.toString().trim().split(",")
3:     allResults ← List<String>                          ▷ Format:[CN|CV|Res|Act]
4:     for all (classIdx, className) ∈ classSplitConf do
5:         currInClassValue ← input[classIdx]
6:         classPrior ← classContainer[className]
7:         allClassResult ← ListOfString                  ▷ seluruh hasil dari tiap kelas
8:         for all (classPriorDetail) ∈ classPrior do
9:             currClassAllPredictorResult ← 1.0
10:            flag ← 0
11:            outer :
12:            for all (attrIdx, attrName, attrType) ∈ attrSplitConf do
13:                currInAttrValue ← input[attrIdx]
14:                if attrType ←discrete then
15:                    predictor ← predictorContainer[attrName]
16:                    predDetail ← predictor.getDetail[currInAttrValue]
17:                    countDividend ← classPriorDetail.getCount
18:                    divClassPrior ← predidctor.getClassPrior[className]
19:                    divClassPriorDetail ← divClassPrior.getDetail[classPriorDetail]
20:                    divisor ← divClassPriorDetail.getCount
21:                    currRes ← (countDividend)/(divisor)
22:                    currClassAllPredictorResult∗ = currRes
23:                else if attrType ←numeric then
24:                    predictor ← predictorContainer[attrName]
25:                    mean ← classPriorDetail.getMean
26:                    sigma ← classPriorDetail.getSigma
27:                    divisor ← √(2.0 ∗ Π ∗ sigma)
28:                    powerDividend ← currInAttrValue − mean)^2 ∗ −1
29:                    powerDivisor ← 2.0 ∗ sigma/2
30:                    resPower ← powerDividend/powerDivisor
31:                    currRes ← (1/divisor) ∗ Math.E^resPower
32:                    currClassAllPredictorResult∗ = currRes
33:                currClassCount ← classPriorDetail.getCount()
34:                allCurrClassCount ← 0.0
35:                for all x ∈ classPrior.getDetail do
36:                    allCurrClassCount+ = x.getCount
37:                currClassAllPredictorResult∗ = (currClassCount/allCurrClassCount)
38:                allClassResult.add(currClassAllPredictorResult)
39:        maxClass ← ""
40:        checker ← Double.MIN_VALUE
41:        divisorNorm ← 0.0
42:        for all result ∈ allClassResult do
43:            divisorNorm+ = result.currentVal
44:            if checker < currentVal then
45:                checker = currentVal
46:                maxClass = s
47:        allResults.add(maxClass)
48:    for all result ∈ allResults do
49:        write(result)
```

(b) `setup(` Operasi ini akan melakukan pembacaan model NBC yang sudah dibuat pada proses sebelumnya pada modul training dalam HDFS serta mela-

kukan konversi data dari *plain text* model NBC pada HDFS ke dalam objek - objek kelas dan prediktor. Lalu, memasukkannya ke dalam kontainer - kontainer sesuai dengan tipe atribut tersebut.

(c) `laplacianSmoothing()` Operasi ini akan melakukan *smoothing* pada mode NBC yang diperoleh pada saat operasi `setup()` dijalankan. Proses ini akan melakukan penambahan tiap frekuensi pada atribut bertipe diskrit serta kelasnya sebanyak parameter yang diterima (`default=1`). Operasi ini dibuat untuk menangani terjadinya *zero-frequency problem* pada model NBC yang telah dibuat. Berikut merupakan *pseudo-code* pada operasi ini:

Algorithm 4 Laplacian Smoothing algorithm

```
1: for all (className, classPrior) ∈ classContainer do
2:     for all (classValue, classPriorDetail) ∈ classPrior do
3:         for all (predictorName, predictor) ∈ predictorContainer do
4:             totalAdditionForCurrClassDetail ← 0
5:             for all (predictorDetailName, predictorDetail) ∈ predictor do
6:                 classPriorDetailChecker ← predictorDetail.classPrior[className]
   .getDetail[classValue]
7:                 if classPriorDetailChecker ← null then
8:                     classPriorDetailChecker ← newClassPriorDetail
   (classValue,laplacianSmoothingAdder)
9:                     predictorDetail.classPrior[className]
   .putDetail(classValue,classPriorDetailChecker)
10:                    tmpClassPriorDetail ← classPrior.get[classValue]
11:                    tmpClassPriorDetail.setCount(tmpClassPriorDetail.getCount() +
   laplacianSmoothingAdder)
12:                else
13:                    classPriorDetailChecker.setCount(classPriorDetailModify.getCount()+
   laplacianSmoothingAdder)
14:                for i ← 0 to laplacianSmoothingAdder do
15:                    totalAdditionForCurrClassDetail.incrementAndGet()
16:            classPriorDetailSecondHandler ← predictor.classPrior[className]
   .getDetail[classValue]
17:            classPriorDetailSecondHandler.setCount( classPriorDetailSecondHandler
   .getCount()+totalAdditionForCurrClassDetail.get())
```

2. `package utils.class`
   Pada package ini terdapat 3 kelas utama yang akan digunakan sebagai model dari atribut kelas NBC sekaligus kelas kontainer untuk penyimpanannya. Berikut merupakan kelas - kelas yang ada pada `package utils.class`:

   (a) `ClassContainer`
   Kelas ini akan digunakan sebagai penyimpanan seluruh atribut kelas pada model NBC. Kelas ini memiliki 1 atribut yaitu `classPriorMap` yang bertipe `Map`. Atribut ini memiliki `key=String` dan `value=ClassPrior`. Atribut ini akan berisi kumpulan dari atribut kelas pada NBC.

   (b) `ClassPrior`
   Kelas ini akan merepresentasikan atribut kelas pada model NBC yang ada. Kelas ini merupakan turunan dari kelas `AttributeMetaInfo`. Sehingga, kelas ini secara tidak langsung memiliki atribut yang dimiliki oleh kelas `AttributeMetaInfo`.

   (c) `ClassPriorDetail`
   Kelas ini akan merepresentasikan nilai diskrit dari atribut kelas pada model NBC

yang ada. Kelas ini merupakan kelas turunan dari kelas `AttributeDetail`. Sehingga, kelas ini secara tidak langsung memiliki atribut yang dimiliki oleh kelas `AttributeDetail`. Karena kelas ini merepresentasikan atribut bertipe diskrit, maka nilai awal untuk atribut `mean` dan `sigma` pada kelas *parent*-nya akan di-*override* dan diinisialisasi dengan nilai -1.

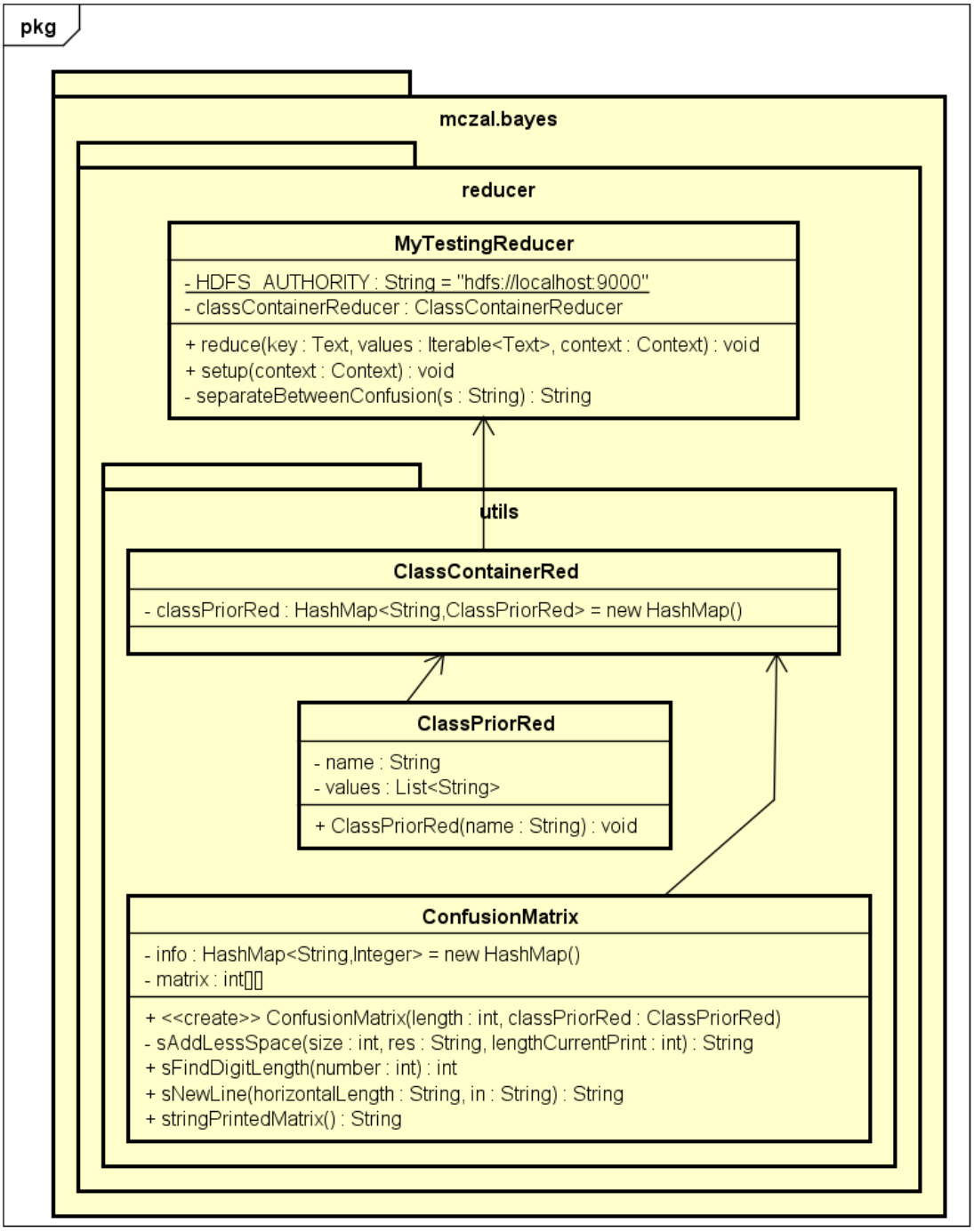
3. `package utils.predictor`
   Pada package ini terdapat 3 kelas utama yang akan digunakan sebagai model dari atribut prediktor NBC sekaligus kelas kontainer untuk penyimpanannya. Berikut merupakan kelas - kelas yang ada pada `package utils.predictor`:

   (a) `PredictorContainer`
   Kelas ini akan digunakan sebagai penyimpanan seluruh atribut prediktor pada model NBC. Kelas ini memiliki 1 atribut yaitu `predictorMap` yang bertipe `Map`. Atribut ini memiliki `key=String` dan `value=Predictor`. Atribut ini akan berisi kumpulan dari atribut predictor pada NBC.

   (b) `Predictor`
   Kelas ini akan merepresentasikan atribut prediktor pada model NBC yang ada. Kelas ini merupakan turunan dari kelas `AttributeMetaInfo`. Sehingga, kelas ini secara tidak langsung memiliki atribut yang dimiliki oleh kelas `AttributeMetaInfo`. Atribut tambahan yang dimiliki secara eksklusif oleh kelas ini merupakan atribut `classPriorMap`. Atribut ini akan berisi tentang seluruh kelas dari probabilitas posterior atribut prediktor tersebut.

   (c) `PredictorDetail`
   Kelas ini akan merepresentasikan nilai diskrit dari atribut prediktor yang bertipe diskrit pada model NBC yang ada. Kelas ini merupakan kelas turunan dari kelas `AttributeDetail`. Sehingga, kelas ini secara tidak langsung memiliki atribut yang dimiliki oleh kelas `AttributeDetail`.

3.2.2.4 *Package Reducer*



Gambar 3.11: Diagram Kelas Modul *Testing: Package Reducer*

Pada package ini terdapat 4 kelas utama yang akan digunakan untuk menjalankan proses *reduce* pada modul *testing*. Sebagian diantaranya berada di dalam *sub-package* yang berbeda. Berikut merupakan penjelasan tiap package dan kelas:

1. Kelas `MyTestingReducer` Kelas ini merupakan kelas utama yang akan dijalankan pada proses reduce. Kelas ini memiliki 2 atribut dan 3 operasi yang akan digunakan untuk dapat melakukan operasi pada fase reduce secara keseluruhan. Berikut merupakan penjelasan mengenai atribut dan operasi pada kelas ini:

    - Atribut

(a) `HDFS_AUTHORITY`
Atribut ini bertipe *String* dan memiliki *modifier static* dan *final* agar nilainya tidak dapat diubah - ubah. Inisialisasi pertama dari atribut ini adalah: `"hdfs://master:9000"`. `"hdfs://master:9000"` merupakan url dari node master yang digunakan untuk dapat berkomunikasi dengan node master pada lingkungan *hadoop*. Atribut ini digunakan untuk *request* operasi baca file *meta.info* dan model NBC pada HDFS.

(b) `classContainerReducer`
Atribut ini bertipe `ClassContainerReducer` yang akan digunakan untuk menyimpan seluruh atribut kelas pada NBC.

- Operasi

(a) `reduce()`
Operasi ini akan melakukan proses pada fase *reduce* untuk proses testing pada modul ini. Proses yang dilakukan akan meliputi perhitungan nilai - nilai pada `confusion matrix` untuk setiap atribut kelas berdasarkan hasil klasifikasi pada fase *map* sebelumnya. Setelah itu, proses ini juga akan melakukan kalkulasi untuk *error rate*. Variabel yang akan digunakan pada perhitungan *error rate* adalah: (1) `Accuracy` (2) `Precision` (3) `Recall` (4) `F-Measure`. Berikut merupakan *pseudo-code* untuk proses *reduce* pada modul *testing*:

Algorithm 5 NBC Testing algorithm

```
1: procedure REDUCE(key, values[], context)                    ▷ Reduce function
2:     HDFS_AUTHORITY ← "hdfs://master:9000"
3:     classContainerRed ← ClassContainerRed
4:     classPriorRed = classContainerRed.get[key]
5:     confusionMatrix ← ConfusionMatrix(classPriorRed.valueSize,classPriorRed)
6:     outKey ← "@" + key                                      ▷ class name
7:     for all s ∈ values do
8:         splitter ← s.split("
9: |")
10:        actual ← splitter[2].split("=")[1]
11:        predicted ← splitter[0].split("=")[1]
12:        predIndex ← confusionMatrix.get(predicted)
13:        actIndex ← confusionMatrix.get(actual)
14:        confusionMatrix.getMatrix()[actIndex][predIndex]++
15:    outKey ← confusionMatrix.stringPrintedMatrix()
16:    dividend ← 0                                            ▷ Accuracy
17:    divisor ← 0
18:    for i ← 0 to confusionMatrix.matrixLength do
19:        for j ← 0 to confusionMatrix.matrixLength do
20:            if i = j then
21:                dividend + = confusionMatrix.getMatrix()[i][j]
22:            divisor + = confusionMatrix.getMatrix()[i][j]
23:    accuracyOperator ← dividend + "/" + divisor
24:    accuracyResult ← dividend/divisor
25:    outVal + = accuracyOperator+"="+accuracyResult
26:    for all (className,index) ∈ confusionMatrix do
27:        currTP ← confusionMatrix.getMatrix()[index][index]
28:        currFP ← 0
29:        currFN ← 0
30:        for i ← 0 to confusionMatrix.matrixLength do
31:            if i! = index then
32:                currFP + = confusionMatrix.getMatrix()[i][index]
33:                currFN + = confusionMatrix.getMatrix()[index][i]
           //*                                                  ▷ Precision
34:        precisionOperation ← currTP + "/" + currTP + "+" + currFP
35:        precisionResult ← currTP/(currTP + currFP)
36:        outVal + = precisionOperation + "=" + precisionResult
         //*                                                    ▷ Recall
37:        recallOperation ← currTP + "/" + currTP + " + " + currFN
38:        recallResult ← currTP/ (currTP + currFN)
39:        outVal + = recallOperation + "=" + recallResult
         //*                                                    ▷ F-Measure
40:        α ← (2*precisionResult*recallResult) / (precisionResult+recallResult)
41:        fMeasureOperation ← 1/alpha1/P + (1 − alpha)1/R
42:        fMeasureResult ← 1/((alpha ∗ (1/P)) + ((1 − alpha) ∗ (1/R)))
43:        outVal + = fMeasureOperation + "=" + fMeasureResult
44:    write(outKey,outVal)
```

(b) `setup()`

Operasi ini akan melakukan pembacaan pada model NBC yang sudah dibuat sebelumnya dalam HDFS. Model NBC yang dibaca hanya yang untuk atribut yang bertipe kelas dan lalu memasukkannya ke dalam kelas kontainer pada

proses ini.

(c) `separateBetweenConfusionMatrix(s)`
Operasi ini akan melakukan penambahan pada *string parameter* dengan separator yang ditentukan di dalam operasi ini.

2. `package utils` Pada kelas *package* ini terdapat 3 kelas utama yang digunakan untuk menyimpan atribut kelas dari model NBC dan penyimpanan untuk `confusion matrix` yang akan digunakan untuk melakukan perhitungan *error rate*. Berikut merupakan penjeleasan kelas - kelas yang terdapat pada *package* ini:

   (a) `ClassContainerRed`
   Kelas ini akan digunakan sebagai penyimpanan seluruh atribut kelas pada model NBC. Kelas ini memiliki 1 atribut yaitu `classPriorRed` yang bertipe `HashMap`. Atribut ini memiliki `key=String` sebagai nama dari kelas tersebut dan `value=ClassPriorRed` sebagai kelas yang akan menyimpan detail dari kelas tersebut. Atribut ini akan berisi kumpulan dari atribut kelas pada NBC.

   (b) `ClassPriorRed`
   Kelas ini akan merepresentasikan atribut kelas pada model NBC yang ada. Kelas ini memiliki 2 atribut, diantaranya adalah:

      i. `name`
      Atribut ini akan berisi nama dari atribut kelas model NBC ini.

      ii. `values`
      Atribut ini akan berisi nilai - nilai diskrit dari atribut kelas model NBC ini.

   (c) `ConfusionMatrix`
   Kelas ini akan merepresentasikan nilai dari `matrix confusion` yang akan diisi dengan kesesuaian antara hasil prediksi dari algoritma NBC dengan nilai aktual milik record input tersebut. Nilai - nilai dari `matrix confusion` yang ada pada kelas ini juga akan digunakan untuk melakukan perhitungan *error rate*: (1) *Accuracy*; (2) *Precision*; (3) *Recall*; (4) *F-Measure*. Kelas ini memiliki 2 atribut dan 5 operasi, berikut merupakan penjelasan lebih lanjut:

   - Atribut

      i. `info`
      Atribut ini bertipe `HashMap` dengan `key=string` untuk menyimpan nama dari nilai diskrit pada atribut kelas tertentu dan `value=integer` untuk menyimpan nomor dari indeks pada matrix yang merepresentasikan kelas tersebut.

      ii. `matrix[][]`
      Atribut ini bertipe matrix 2 dimensi yang akan merepresentasikan matrix *confusion* dari atribut kelas tertentu.

   - Operasi

      i. `ConfusionMatrix()`
      Operasi ini merupakan konstruktor untuk mengisi nilai awal dari dari atribut - atribut pada kelas ini. Operasi ini akan menerima 2 buah parameter yang berisi (1) classPriorRed untuk mencatat tiap nilai diskrit dari atribut kelas tersebut dan (2) panjang dari matrix yang nantinya akan menjadi panjang dari matrix 2 dimensi yang dimiliki oleh kelas ini.

      ii. `sAddLessSpace()`
      Operasi ini melakukan penambahan spasi pada *string parameter input*. Operasi ini akan digunakan untuk men-*generate* keseluruhan matrix dalam bentuk string .

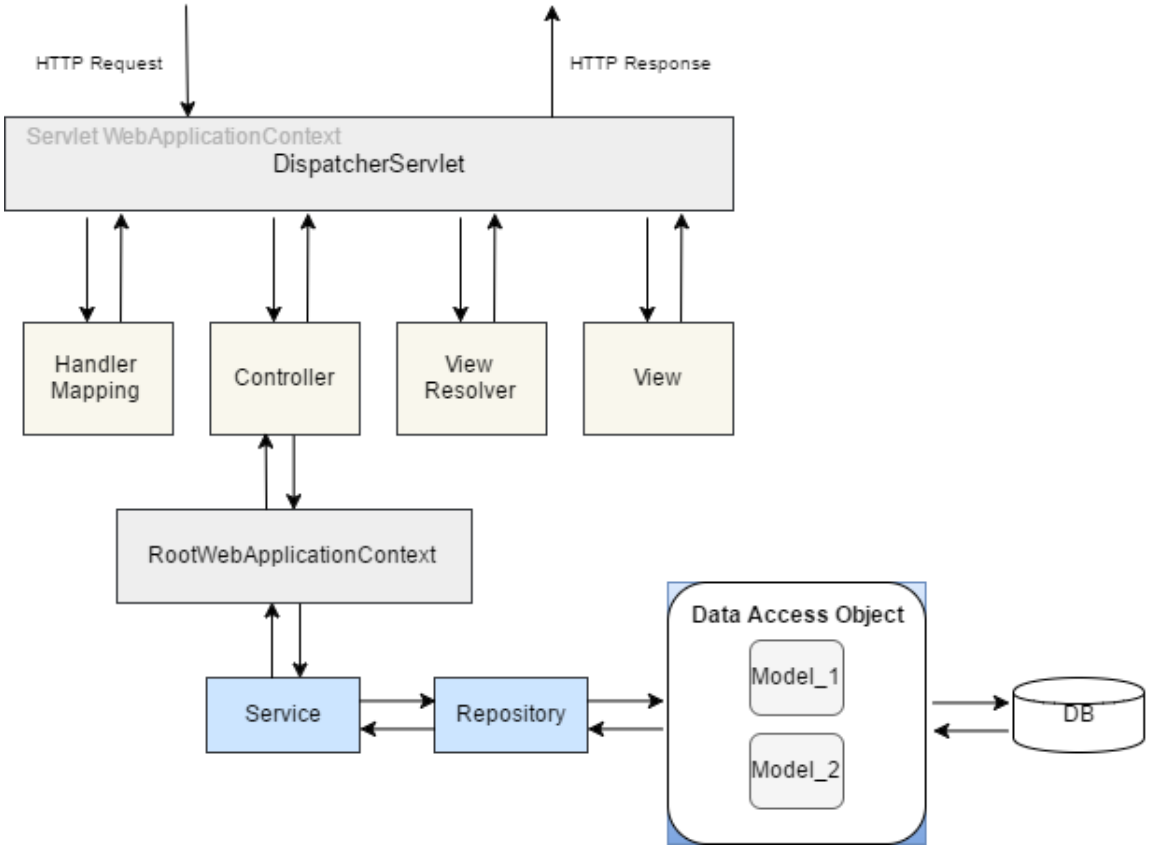
      iii. `sFindDigitLengthNumber()`
      Operasi ini melakukan pencarian panjang digit dari *integer* pada *parame-*

ter. Operasi ini akan digunakan untuk men-*generate* keseluruhan matrix dalam bentuk string.

iv. `sNewLine()`
Operasi ini penambahan baris baru pada *string parameter*. Operasi ini akan digunakan untuk men-*generate* keseluruhan matrix dalam bentuk string.

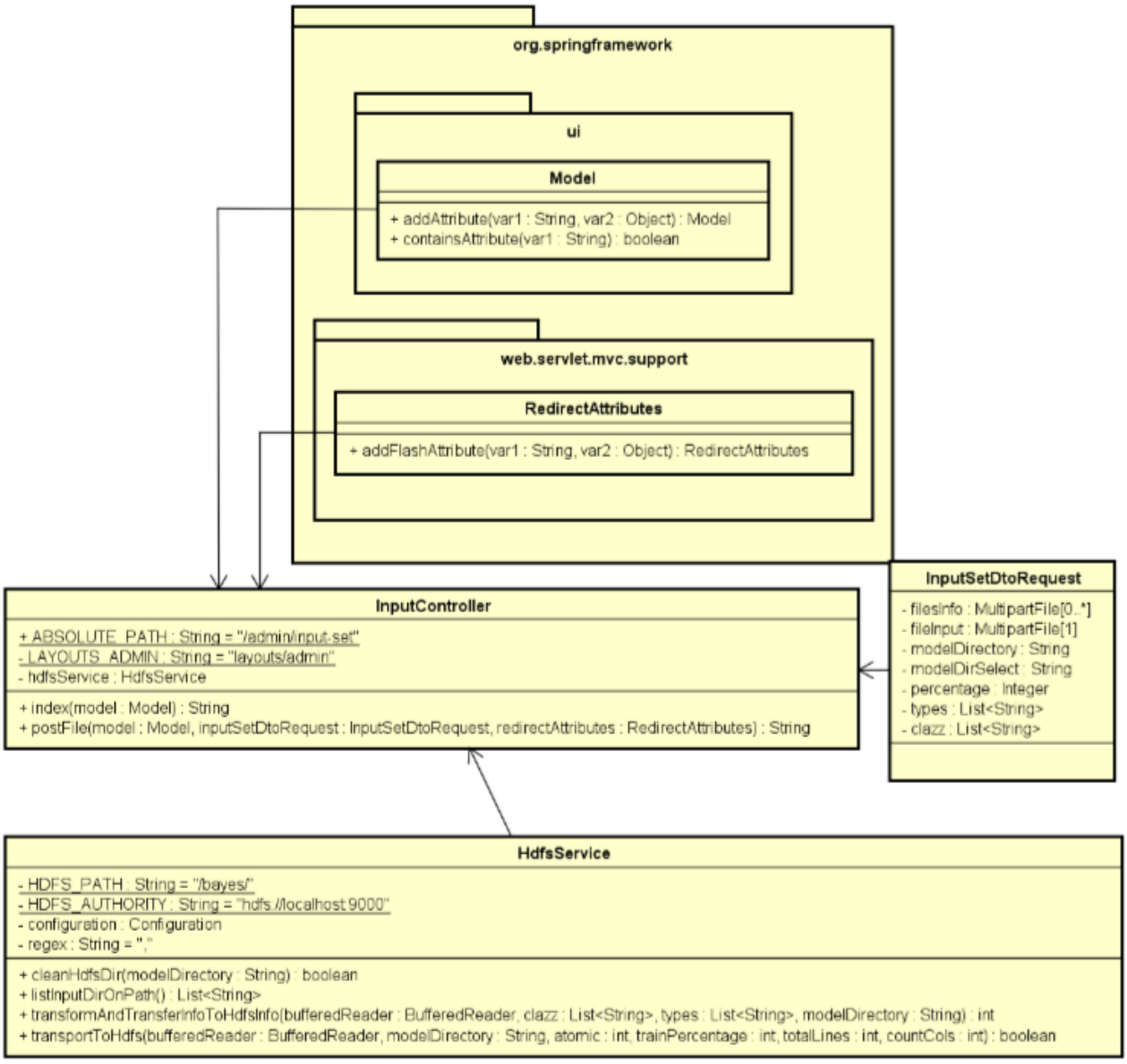
v. `stringPrintedMatrix()`
Operasi ini akan melakukan konversi dari matrix 2 dimensi milik atribut `matrix[][]` ke dalam bentuk `string` yang nantinya akan di-*print* menjadi keluaran proses ini.

3.2.3 *Design Pattern* Modul Kelola Input dan Modul Klasifikasi *Naive Bayes*



Gambar 3.12: Design Pattern Modul Klasifikasi

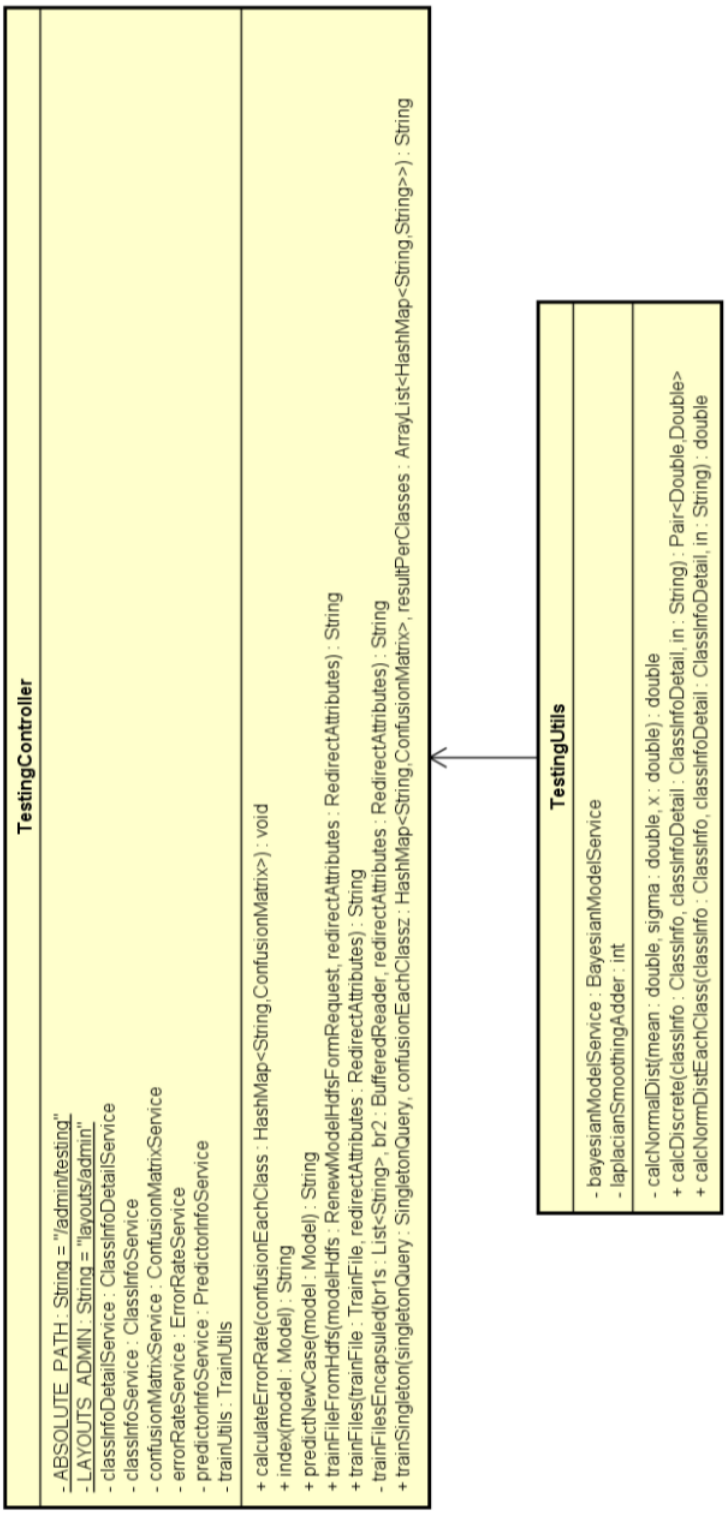
3.2.4 Diagram Kelas Modul Kelola Input



Gambar 3.13: Diagram Kelas Modul Kelola Input

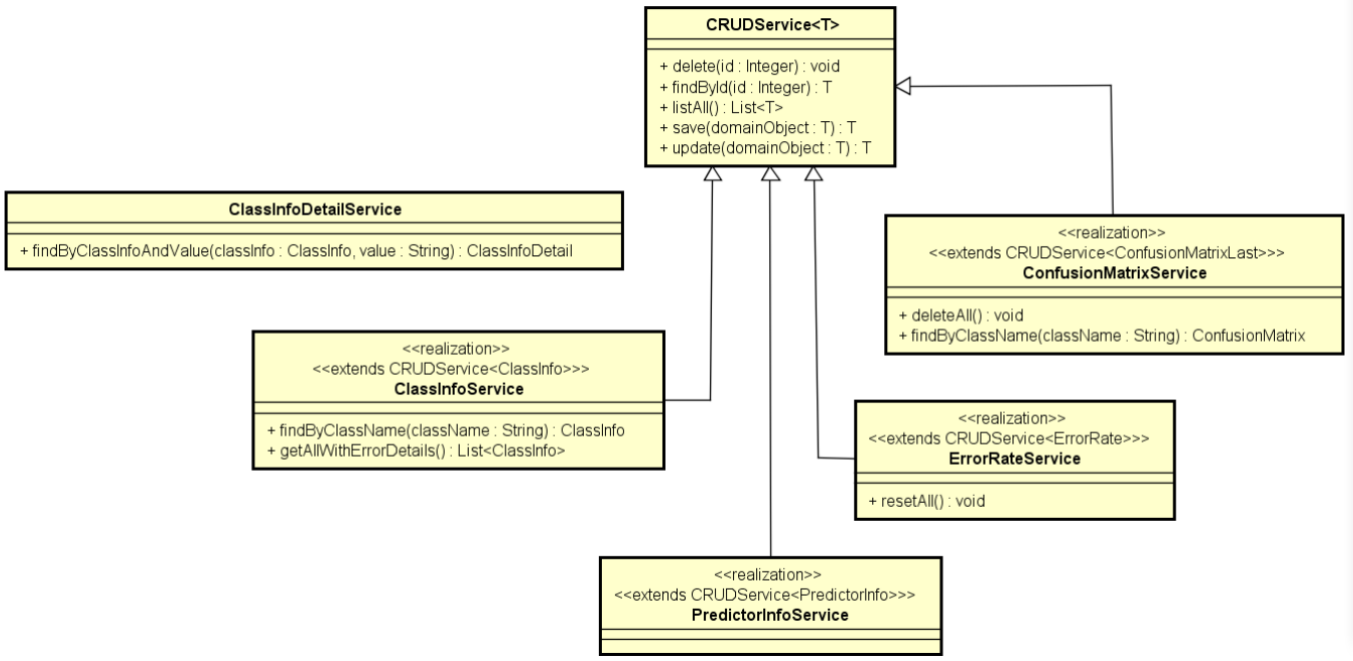
3.2.5 Diagram Kelas Modul Klasifikasi *Naive Bayes*

3.2.5.1 *Controller*



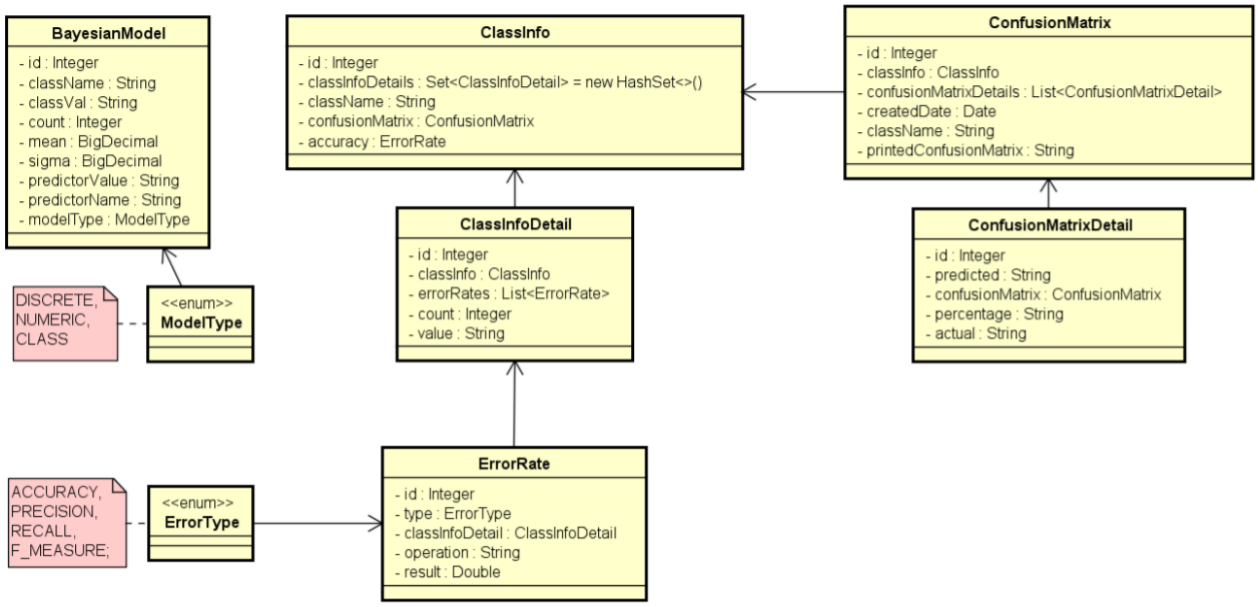
Gambar 3.14: Diagram Kelas Modul Kelola Input

3.2.5.2 *Service*



Gambar 3.15: Diagram Kelas Modul Kelola Input

3.2.5.3 *Model*



Gambar 3.16: Diagram Kelas Modul Kelola Input

DAFTAR REFERENSI

[1] 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.

[2] 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].

[3] Oracle, "Generic types." https://docs.oracle.com/javase/tutorial/java/generics/types.html, 2015. [Online; diakses 19-April-2017].