Yahoo SAMOA Lab2 --Test Performance of Non-parallel Naive Bayes Classifier

Li Huang

Computer Engineering and Computer Science

J.B. Speed School of Engineering

University of Louisville

[l0huan08@gmail.com](mailto:l0huan08@gmail.com)

2014.4.3

Abstract:

A Naive Bayes Classifier was implemented under Yahoo SAMOA platform. It is a non-parallel algorithm. This lab test the performance, including speed and correct rate, of this algorithm under different cluster setting: (1)single machine (2) two machine-cluster (3) three machine-cluster. The source data are from Kdd99[[1]](#footnote-2) and Movie Review[[2]](#footnote-3). Vertical Hoeffding Decision Tree and another version of Naive Bayes classifier are also tested and compared.

Contents

[1.Purpose: 3](#_Toc385712856)

[2.Test environment: 3](#_Toc385712857)

[3.Test settings 4](#_Toc385712858)

[3.1 Algorithm 4](#_Toc385712859)

[3.1.1 Introduction of Naive Bayes Classifier 4](#_Toc385712860)

[3.1.2 Algorithms used in test 5](#_Toc385712861)

[3.2 Test Data 9](#_Toc385712862)

[3.3 Cluster modes 13](#_Toc385712863)

[4. Test Result 13](#_Toc385712864)

[4.1 Final result 14](#_Toc385712865)

[5 Conclusion 17](#_Toc385712866)

[Appendix 18](#_Toc385712867)

[1. Detail of Test Result 18](#_Toc385712868)

[1.1 Result file 18](#_Toc385712869)

[1.2 log file 19](#_Toc385712870)

[1.3 Raw Test Result 20](#_Toc385712871)

[1.4 Average value of Test Result 24](#_Toc385712872)

# 1.Purpose:

To learn how to implement user-defined data mining algorithm under SAMOA platform, I tried to implement a Naive Bayes Classifier. The first version of my algorithm is tried to be as simple as possible, so it is non-parallel version. To test the characteristic and possibility of SAMOA, I test this algorithm on different cluster settings: Single-machine, Two-machine cluster, Three-machine cluster, to see if it is possible to get better performance when the cluster becomes larger(with more computing nodes). I also compared this algorithm with VHDT(Vertical Hoeffding Decision Tree) which has been integrated into SAMOA, as well as another version of my Naive Bayes classifier.

# 2.Test environment:

The test is running on a cluster composed by three computers. They are connected by a switch. The hardware and software configuration are shown below:

|  |  |
| --- | --- |
| **Hardware** | **Software** |
| node1: Pentium4 1.8Ghz, 576 MB RAM  node2: Pentium4 1.8Ghz, 768 MB RAM  node3: Pentium4 2.35Ghz, 495 MB RAM | Ubuntu Linux Desktop ver12.04 32-bit  Yahoo S4 ver0.6  Yahoo SAMOA ver0.0.1 |

# 3.Test settings

3 different test data are applied to the implemented Naive Bayes algorithm, the performances (speed and correct rate) are collected under 4 different cluster configuration.

## 3.1 Algorithm

### 3.1.1 Introduction of Naive Bayes Classifier

Consider the test data instance, X=(x1,x2,..xi,..xM), has M attributes a1,a2,a3,a4,a5,a6,a7,….ai,…aM. It may belong to K class C1,C2,…,Ck,…CK.

To decide which class it belongs, we need to calculate the probability that an instance X belongs to class k, P(Ck|X), and find the maximum one.

1. **Bayes formula**

Pk = P( Ck| X) = P(X|Ck)\*P(Ck)/P(X) ……………………..(1)

Comparing Pj and Pk:

Pj / Pk = [ P(X|Cj)\*P(Cj) ] / [ P(X|Ck)\*P(Ck) ] …………………..(2)

1. **Independent assumption**

Naive Bayes assumes all the attributes a1,a2,…,aM are independent, that P(X|Ck)=P(x1,x2,x3,…,xM|Ck)=P(x1|Ck)\*P(x2|Ck)\*….\*P(xi|Ck)\*…P(xM|Ck) …………..(3)

Pj / Pk = [ P(x1|Cj)\*P(x2|Cj)\*…P(xM|Cj) ] / [P(x1|Ck)\*P(x2|Ck)\*…P(xM|Ck)] \* P(Cj)/P(Ck) …………(4)

So the estimation class label

1. **Probability estimation**

We can estimate by counting the training data:

P(c) = N(c) / N. …………….(5)

P(xi |c) = N(xi,c)/ N(c). ……………(6)

where

N(c): number of instances in training data that belongs to class c.

N: totally number of training instances.

N(xi,c): number of instances has value xi in attribute i, and belongs to class c.

*Smoothing:*

In practice, P(xi |c) = N(xi,c)/ N(c) is too harsh and maybe can be divided by 0. So “smoothing” is applied:

P(xi |c) = ( N(xi,c)+l ) / ( N(c)+l\*J ), ………….(7)

that l and J are user-defined constants, usually l=1 and J=|ai|=number of distinct values of attribute i.

*Deal with real attribute values:*

However, counting N(xi,c) only works for nominal attribute and enumerate and small integer attribute. If attribute i are real values, we cannot count N(xi,c). So in this situation we must use other method to estimate P(xi|c). One way is assuming the values in attribute i among the instances belong to class Ck are in a specific distribution, that P(xi|c) can be calculate directly as same as the probability density f(xi) among the training data belongs to class Ck.

\sigma^2_c : the variance of the values in x associated with class *c*

\mu_c : the mean of the values in x associated with class *c*

1. **Class labeling**

After calculating every Pj while j=1,2,…K, we can find the max one, Pk, then we can output the class label of the test instance X is Ck.

### 3.1.2 Algorithms used in test

Three algorithms are applied and compared in this test.

1. **Naive Bayes Classifier - my version 4 (NB-4)**

Several different versions of Naive Bayes algorithm were designed, and each version use different strategy and parallel topology. The “NB-4” is the 4th version of Naive Bayes algorithm I designed.

NB-4 is a non-parallel Naive Bayes Classifier implemented by myself. It useS a single “ModelProcessor” to train the model and apply the model to calculate classification result. The core data structure is the statistic information for each attribute of training data:

A “matrix” to record n(xi, Ck) and a vector to record n(Ck).

**Advantage**: Fast and simple data structure

**Disadvantage**: Unfortunately, this algorithm can only handle data with nominal attributes, or enumerate(small integer) attributes. When the attribute are continuous real values or large integers, this algorithm will crash and cause error. So latter I designed NB-5 algorithm.

1. **Naive Bayes Classifier - my version 5 (NB-5)**

This algorithm is designed to handle real values and large integer attributes in input data, which is a problem in “NB- 4”. The code is totally different from NB-4, that NB-5 is copy and modified from the source code of Naive Bayes Classifier in MOA(Massive Online Analysis)[[3]](#footnote-4). This algorithm consider the values in a specific numeric attribute(integer or real) within a specific class as Gauss(Normal) distribution, that it can calculate:

P(xi| Ck) = probability density(x) in values belongs to class Ck = f(x|Ck)

= 
f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{ -\frac{(x-\mu)^2}{2\sigma^2} }
. ……………………(8)

The parameter μ in this definition is the [mean](http://en.wikipedia.org/wiki/Mean) or [expectation](http://en.wikipedia.org/wiki/Expected_value) of the distribution (and also its [median](http://en.wikipedia.org/wiki/Median) and [mode](http://en.wikipedia.org/wiki/Mode_(statistics))). The parameter σ is its [standard deviation](http://en.wikipedia.org/wiki/Standard_deviation);

When encounter new instance, this algorithm update the  μ [i,k] and  σ [i,k], which is corresponding to the distribution of attribute i’s values within class k.

**Advantages:**

Simple; Able to process both numeric attributes and nominal attributes.

**Disadvantages**:

Not designed for parallel computing.

The streaming processing topology (in SAMOA terminology) of NB-4 and NB-5 are shown below:



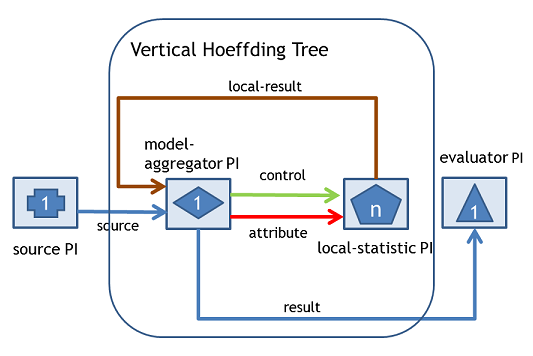
1. **Vertical Hoeffding Decision Tree (VHT)**

This algorithm is the default classifier which has been implemented by SAMOA. It is a parallel algorithm that its speed can scale-up when the number of computing nodes increases.

VHDT is tested as a baseline to compare with my own Naive Bayes classifier. This algorithm is a parallel algorithm that would speed up with more computing nodes.

The relevant document could be access at <https://github.com/yahoo/samoa/wiki/Vertical%20Hoeffding%20Tree%20Classifier>

The topology diagram is shown below:



1. **Bagging**

Bagging[[4]](#footnote-5) is a ``bootstrap''[[5]](#footnote-6) ensemble method that creates individuals for its ensemble by training each classifier on a random redistribution of the training set. The relevant introduction of Bagging algorithm could be found at <http://people.cs.pitt.edu/~milos/courses/cs2750-Spring04/lectures/class23.pdf> and <http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume11/opitz99a-html/node3.html> .

Yahoo SAMOA has already contained a default Bagging algorithm (so called OzaBag, see <http://moa.cms.waikato.ac.nz/details/classification/classifiers-2/> )

The topology of Bagging algorithm in SAMOA is below:



In this lab, I tested Bagging algorithm with 3 different “Learner” as mentioned before: NB-4, NB-5, VHDT.

**Prequential Evaluation Task**

All the algorithms mentioned above were integrated in the “Prequential Evaluation Task” of SAMOA. This task is a basic framework of classify online data with user-defined classifiers. The task reads online data and send each instance to the learner(classifier) for training, and also classify this instance with this classifier; finally the classification results are output to a performance evaluator, and the classification performance (such as correct rate, F-measure, etc…) are showed.

The document can be found at <https://github.com/yahoo/samoa/wiki/Prequential%20Evaluation%20Task> , and its topology is:



In conclusion, the algorithms selected to compare are:

1. Naive Bayes- my version 4 (NB-4)
2. Niave Bayes- my version 5 (NB-5)
3. Vertical Hoeffding Decision Tree (VHT)
4. Bagging NB-4 (BagNB4)
5. Bagging NB-5 (BagNB5)
6. Bagging VHT (BagVHT)

## 3.2 Test Data

1. **Movie Review data**

Movie Review Data are collections of movie-review documents labeled with respect to their overall sentiment polarity (positive or negative) and sentences labeled with respect to their subjectivity status (subjective or objective) or polarity. Basically, a movie review dataset contains many instances, and each instance is a movie review and a class label, “good movie” or “bad movie”, of this review.

In this lab, I only use the preprocessed data from “polarity dataset 2.0”, which contains 1000 positive and 1000 negative processed reviews. The preprocessed data is created by my classmate, Wen Long Sun. Moreover, I shuffled the order of instances that the “pos” and “neg” instances occur alternately. This data contains 2000 instances; each instance has about 1000 attributes and 1 class label, like below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Movie Review** | | | | | | | | | | | | | |
| **Attribute** | & | \* | - | 1 | 10 | 2 | … | acc | adam | … | unit | visit | @@class@@ |
| line 1 | 1 | 0 | 0 | 0 | 1 | 0 | … | 0 | 0 | … | 1 | 1 | pos |
| line 2 | 0 | 0 | 1 | 0 | 0 | 0 | … | 1 | 0 | … | 0 | 1 | neg |
| line 3 | 0 | 1 | 0 | 0 | 0 | 1 | … | 0 | 0 | … | 0 | 0 | pos |
| … | … | … |  |  |  |  |  |  |  |  |  |  |  |

Each instance stands for a review document, and each attribute is the occurrence of a word in this document. For example, if the word “adam” occurs in this document, then the value of attribute “adam” should set to 1, otherwise set to 0. The last attribute “@@class@@” shows the sentiment polarity of each review document.

**Size:**

2000(instances) x 1172(attributes)

1000 instances are “pos”, and 1000 are “neg”.

1. **Kdd99 data**

Kdd99 data can be found at <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

Kdd99 data is created to build a predictive model (i.e. a classifier) capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections.

The data contains about 5 Million instances, each with 41 attributes and 1 class label. Each instance is a connection record, including features such as “length (number of seconds) of the connection”, “type of the protocol”, “normal or error status of the connection”, etc. The class label is “normal” connection or 22 types of attacks, such as “buffer\_overflow”, “guess\_passwd”, etc.

The 22 types of attacks fall into four main categories:

* DOS: denial-of-service, e.g. syn flood;
* R2L: unauthorized access from a remote machine, e.g. guessing password;
* U2R: unauthorized access to local superuser (root) privileges, e.g., various ``buffer overflow'' attacks;
* probing: surveillance and other probing, e.g., port scanning.

The data is like:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Kdd99** | | | | | | | | | |
| **Attribute** | duration | protocol\_type | service | flag | src\_bytes | dst\_bytes | … | dst\_host\_srv\_rerror\_rate | class |
| line 1 | 0 | tcp | http | SF | 215 | 45076 | … | 0 | normal |
| line 2 | 0 | tcp | http | SF | 162 | 4528 |  | 0 | normal |
| line 3 | 0 | tcp | http | SF | 236 | 1228 |  | 0 | smurf |
| line 4 | 0 | tcp | http | SF | 233 | 2032 |  | 0 | neptune |
| … | … | … |  |  |  |  |  |  | … |

It’s **highly unbalanced** data that most of the instances are labeled “smurf”.

**Size:**

4,898,431(instances) x 42(attributes)

class distribution: Total 23 classes. 87% are “smurf”(DOS attack), 12% are “neptune”(DOS attack), 0.48 % are “normal”, as shown below:



1. **NSL Kdd99 data**

NSL Kdd99 data could be found at <http://nsl.cs.unb.ca/NSL-KDD/> . It is an improvement of KDD99 that it is more efficient for training classifiers. It remove redundant records; it transforms class labels into only two types: “normal” and “anomaly”; it select the records to make it more balance.

**Size:**

125,973(instances) x 42(attributes)

class distribution: 86% anomaly, 14% normal



Finally, the three datasets are transformed into “ARFF”[[6]](#footnote-7) file format to let SAMOA be able to read. They are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **data** | **file** | **total instance number** | **attribute number** | **class number** | **note** |
| Movie Review data | m.arff | 2000 | 1172 | 2 |  |
| NSL Kdd99 data | n.arff | 125,973 | 42 | 2  (unbalance 86% anomaly, 14% normal) | NB-4 cannot process because it contains real value attribute. |
| Kdd99 data | k.arff | 4,898,431 | 42 | 23  (unbalance, 87% smurf, 12% neptune, 0.48% normal) | NB-4 and VHT cannot process because it contains error values and real value attribute. |

All the data could be downloaded from my Google drive:

<https://drive.google.com/folderview?id=0B0k3wDoweGSZaWQ4c2IyaHhGSEk&usp=sharing>

## 3.3 Cluster modes

I test the different algorithms with different data, and I also need to test them in different cluster settings to see the parallel-computing scalability of the algorithms.

1. **Local mode (Local)**

Only run task on node 3, in local mode without S4.

1. **S4 mode, single node (S4-1)**

Run task on only node 3, with S4 platform.

1. **S4 mode, two nodes (S4-2)**

Run task on cluster with 2 computing nodes (node 3 and node 2), with S4 platform.

1. **S4 mode, three nodes (S4-3)**

Run task on cluster with 3 computing nodes (node 3, node2 and node 1), with S4 platform.

In conclusion, the test configurations can format as the combination table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data:  **movieReview, NSLKDD, KDD99** | | | | |
| Algorithm/Cluster | **Local mode(single node)** | **S4, 1 node** | **S4, 2 node** | **S4, 3 node** |
| **NB-4** |  |  |  |  |
| **NB-5** |  |  |  |  |
| **VHT** |  |  |  |  |
| **BagNB4** |  |  |  |  |
| **BagNB5** |  |  |  |  |
| **BagVHT** |  |  |  |  |

# 4. Test Result

The test result are collected from: (1)result files (2) log file (3) log(debug) information on the screen. Please see the middle output in Appendix. Here I only show the final rearranged result.

**Measurements:**

I measured 2 index for each test configuration [dataset + algorithm + cluster mode]:

1. **Speed** (seconds/X instances):

How many time the algorithm spends on train and testing every X data. X’s are different for different dataset. (MovieReview,X=1000; NslKdd,X=10,000; Kdd99,X=100,000)

1. **Correct rate**.

The percentage of test data that correctly classified (output class=target class).

## 4.1 Final result

The table below shows the average speed of different algorithms running on different cluster modes and different datasets. Some cells in the table are blank, because the algorithm failed to run in the situation corresponding to this cell (See Appendix).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Average Speed (seconds / x instances) | | | | | | | | | | | | |
| Algorithm/Dataset | **MovieRevew**  (seconds/1000 instances) | | | | **NslKdd**  (seconds/10,000 instances) | | | | **KDD99**  (seconds/100,000 instances) | | | |
| Cluster mode | local | 1 node | 2 nodes | 3 nodes | local | 1 node | 2 nodes | 3 nodes | local | 1 node | 2 nodes | 3 nodes |
| **NB-4** | 1 | 1 | 4 | 9 |  |  |  |  |  |  |  |  |
| **NB-5** | 2 | 3 | 8 | 12 | <1 | 2 | 3 | 8 | 20 | 28 | 39 | 118 |
| **VHT** | 6 | 6 | 9 | 15 | 1.5 | 4 | 2.7 |  |  |  |  |  |
| **BagNB4** |  |  |  |  |  |  |  |  |  |  |  |  |
| **BagNB5** |  |  |  |  | 3.5 | 16 | 18 | 41 | 155 | 255 | 285 | 808 |
| **BagVHT** |  |  |  |  |  |  |  |  |  |  |  |  |

We can draw the “speed graphs” from the tables above:

The vertical axis show the processing speed ( Training + Classification time of every unit frequency of data).

The correct rate of different algorithms are basically stable among different cluster mode, so we can draw the correct rates table of different algorithms with different datasets.

|  |  |  |  |
| --- | --- | --- | --- |
| Average Correct Rate (%) | | | |
| Algorithm/Dataset | **MovieRevew** | **NslKdd** | **Kdd99** |
| **NB-4** | 78.5 |  |  |
| **NB-5** | 72.6 | 90.0 | 99.5 |
| **VHT** | 49.4 | 67.1 |  |
| **BagNB4** | 84.3 |  |  |
| **BagNB5** | 86.9 | 90.1 | 99.7 |
| **BagVHT** | 49.3 | 58.5 |  |

# 5 Conclusion

Comparing the speed of the algorithms, NB-4 > NB-5 > VHT (> means faster), and BagNB4 > BagNB5 > BagVHT; Simple algorithm was much faster than Bagging-assembled algorithm.

Comparing the correct rate, we can find NB-4 > NB-5 > VHT, and BagNB5 > BagVHT; Bagging-assembled Naive Bayes algorithms were better(about 10%) than pure Naive Bayes algorithms only for MovieReview dataset(for about 1000 attributes), but Bagging did not improve VHT for all dataset, neither did it improve Naive Bayes for KDD99 and NslKDD99 datasets.

Comparing compatibility of the algorithms, NB-4 cannot process float-type numeric attributes and VHT sometimes raise bugs such as “deserialization error” and “null point exception”. Only NB-5 could run correctly with most of the data and cluster settings.

All the algorithms did not speed up with more computing nodes; instead, the performance decreased with more nodes. One reason is that NB-4 and NB-5 algorithms were not parallel structure. However, even the default VHT algorithm, which was claimed would “scale-up” by SAMOA developers, did not achieve better performance with more nodes.

In summary, NB-5 could achieve a correct rate of about 80% or more for selected datasets, which is much better than the default-VHT algorithm in SAMOA. In addition, its speed, 2s/1000 instances for 1000-attributes dataset, 20s/100,000 instances for 40-attributes dataset, is faster than VHT. Moreover, It can handle real(float) type numeric attributes with a “Normal Distribution” assumption. Some problems of SAMOA was found: (1) the default-Bagging algorithm in SAMOA did not improve correct rate but takes much more time than pure classifiers. (2) VHT did not speed up with more computing nodes.

In the future, I will try to improving NB-5 algorithm into parallel structure to make it speed up with more nodes.

# Appendix

# 1. Detail of Test Result

## 1.1 Result file

The test result is output to the file “resultModeAlgorithmData”, such as “resultLocalBagP4m” (Local,BagNB4,m) or “resultS42P5k”(S4-2,NB-5,k). Each result file records the correct rate and Kappa statistics of the classification result. For example, resultS42P5k file is:

|  |
| --- |
| evaluation instances,classified instances,classifications correct (percent),Kappa Statistic (percent),Kappa Temporal Statistic (percent)  100000.0,100000.0,99.64,97.38271583414786,85.28209321340947  200000.0,200000.0,99.73349999999999,99.16494079571832,96.39889196675895  300000.0,300000.0,99.65833333333333,98.56872445672649,93.08973235353612 |

We can translate it to a CSV table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| evaluation instances | classified instances | classifications correct (percent) | Kappa Statistic (percent) | Kappa Temporal Statistic (percent) |
| 100000 | 100000 | 99.64 | 97.38271583 | 85.28209321 |
| 200000 | 200000 | 99.7335 | 99.1649408 | 96.39889197 |
| 300000 | 300000 | 99.65833333 | 98.56872446 | 93.08973235 |

Because “Prequential Evaluation” task is online processor that train model, test(classify) the test instances, and evaluate the performance at the same time, the correct rate is changing while new instances come. This task can set an “evaluation frequency”, which means how many instances between two evaluation points. The table above shows 3 evaluation points at 100000,200000 and 30000 testing instances.

I choose proper evaluation frequencies for different datasets, they are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | m | n | k |
| **Evaluation frequency** | 1000 | 10,000 | 100,000 |

## 1.2 log file

Except the result file, the classification task also output log information to screen. The log information contains running status, error message, debug information, and the most important one--running time of the task. I copy the logs from screen and save them to files.

For example, the log file “logS41P5m” records the log information of running the NB-5 algorithm with MovieReview data on S4-single-node mode:

|  |
| --- |
| SAMOA: Scalable Advanced Massive Online Analysis Platform  Version: 0.0.1  Copyright: Copyright Yahoo! Inc 2013  Web: http://github.com/yahoo/samoa  17:03:36.880 [S4 platform loader] INFO c.y.l.s.l.classifiers.hl.NaiveBayes - ================================================  17:03:36.885 [S4 platform loader] INFO c.y.l.s.l.classifiers.hl.NaiveBayes - Begin init NaiveBayes Classifier topology.  17:03:36.901 [S4 platform loader] INFO c.y.l.s.l.classifiers.hl.NaiveBayes - Sucessfully initializing NaiveBayes classifier topology.  17:03:36.924 [S4 platform loader] INFO org.apache.s4.core.App - Init prototype [com.yahoo.labs.samoa.topology.impl.S4EntranceProcessingItem].  17:03:36.931 [S4 platform loader] INFO org.apache.s4.core.App - Init prototype [com.yahoo.labs.samoa.topology.impl.S4ProcessingItem].  17:03:36.937 [S4 platform loader] INFO org.apache.s4.core.App - Init prototype [com.yahoo.labs.samoa.topology.impl.S4ProcessingItem].  17:03:36.938 [S4 platform loader] INFO c.y.l.samoa.topology.impl.S4DoTask - Starting DoTaskApp... App Partition [0]  17:03:37.063 [STREAM-0\_PROCESSING-ITEM-0] INFO c.y.l.s.l.c.hl.NBModelProcessor - NBModelProcessor created, id = 0  17:03:37.164 [STREAM-0\_PROCESSING-ITEM-0] INFO c.y.l.s.l.c.hl.NBModelProcessor - K=2,A=1172  17:03:41.099 [STREAM-1\_PROCESSING-ITEM-1] INFO c.y.l.s.e.EvaluatorProcessor - **3 seconds for 1000 instances**  17:03:41.109 [STREAM-1\_PROCESSING-ITEM-1] INFO c.y.l.s.e.EvaluatorProcessor - evaluation instances = 1,000  classified instances = 1,000  classifications correct (percent) = 70.3  Kappa Statistic (percent) = 40.968  Kappa Temporal Statistic (percent) = 41.879 |

The underlined part shows the processing speed (training time + test time) of this classification algorithm, which is 3s/1000 instances.

## 1.3 Raw Test Result

The tables below are raw performance results I recorded.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MovieReview**  **evaluation frequency: 1000** | | | | |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** |
| **NB-4** | **speed**  **(second/1000 instance):**  1  **correct rate:**  77.4%  77.7% | **speed(s/1000 instances)**:  1  **correct rate:**  77.7%  77.7% | **speed(s/1000 instances)**:  4  2  7  **Correct rate:**  78.1%  79.2%  80.0% | **speed(s/1000 instances)**:  7  9  10  9  9  **Correct rate**  77.8%  80.3%  81.0%  81.7%  81.5% |
| **NB-5** | **speed(s/1000 instances)**:  2  2  **correct rate:**  70.3%  73.8% | **speed(s/1000 instances):**  3  **correct rate:**  70.3% | **speed(s/1000 instances):**  13  6  6  **correct rate:**  70.7%  74.8%  76.6% | **speed(s/1000 instances):**  20  10  6  11  13  **correct rate:**  71.0%  75.8%  77.4%  78.7%  78.8% |
| **VHT** | 6  50.2% | 6  48.2% | 8  9  10  deserializer error  48.4%  49.5%  49.4% | 20  8  13  18  14  49.0%  50.9%  50.1%  50.9%  50.1% |
| **BagNB4** | 2  2  76.9%  77.2% | 7  88.1% | 11  6  6  86.7%  85.1%  84.5% | 38  22  24  87.5%  85.8%  85.0% |
| **BagNB5** | 0  0  0  0  92.2%  91.4%  90.9%  90.5% | 16  85.7% | 32  17  13  84.8%  82.8%  82.5% | Zookeeper session expired |
| **BagVHT** | 26  48.4% | 47  50.12% | OutOfMemoryError | Serialization error |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **NSL KDD99**  **evaluation frequency: 10,000**  **speed(second/10,000 instance)**  **correct rate** | | | | |  |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** | **Note** |
| **NB-4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **NB-5** | **speed**  0  0  0  0  **correct rate:**  90.1%  90.0%  89.9%  89.8% | 3  2  2  1  90.1%  89.9%  89.8%  89.8% | 4  3  2  3  3  3  90.1%  90.1%  89.8%  89.7%  89.8%  89.8% | 21  8  5  5  8  5  4  90.3%  90.1%  89.8%  89.8%  89.8%  89.8%  89.8% |  |
| **VHT** | 3  1  1  1  78.0%  85.1%  87.8%  89.3% | 9  3  3  2  3  69.5%  80.5%  84.1%  86.1%  87.5% | 3  1  1  1  1  9  deserializer error  53.0%  53.3%  53.2%  53.2%  53.0%  53.1% | deserializer error  53.0%  53.3%  53.1%  53.2%  53.0%  53.0%  53.0% |  |
| **BagNB4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **BagNB5** | 4  4  3  3  90.1%  90.0%  89.97%  89.9% | 18  15  15  15  90.2%  90.0%  89.9%  89.9% | 28  19  17  15  15  15  90.2%  90.0%  89.9%  89.8%  89.9%  89.9% | 56  40  41  36  31  90.1%  89.9%  89.9%  89.8%  89.8% |  |
| **BagVHT** | NullPointerException | NullPointerException  59.2%  60.7% | deserializer error  57.0% | deserializer error |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **KDD99**  **evaluation frequency: 100,000**  **speed(second/10,000 instance)**  **correct rate** | | | | |  |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** | **Note** |
| **NB-4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **NB-5** | **speed(second/10,000 instance)**  19,  20,  21  **correct rate:**  99.7%, 99.7%,  99.7%, 99.3% | **speed**  28,  28,  29  **correct rate:**  99.7%,  99.7%,  99.7% | 49,29,  99.6%, 99.7% | **speed**  274,  108,  69,  69,  70  **correct rate:**  99.7%, 99.1% ,  99.3%,99.3%, 98.5% | 808  99.8% |
| **VHT** | - | - | - | - | VHT cannot handle error values in dataset |
| **BagNB4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **BagNB5** | **speed**  133  147,  165,  175  **correct rate:**  99.7%,  99.7%,  99.7%,99.3% | **speed**  249,  261  **correct:**  99.7%, 99.8% | 302,  275,  279  99.8%, 99.8%, 99.7% |  |  |
| **BagVHT** | - | - | - | - | VHT cannot process error values in dataset |

## 1.4 Average value of Test Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MovieReview**  **evaluation frequency: 1000**  **speed(second/1000 instance)**  **correct rate(low~high)** | | | | |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** |
| **NB-4** | 1  77.4%~77.7% | 1  77.7% | 4.3  78.1%~80.0% | 8.8  77.8%~81.7% |
| **NB-5** | 2  70.3%~73.8% | 3  70.3% | 8.3  70.7%~76.6% | 12  71.0%~78.8% |
| **VHT** | 6  50.2% | 6  48.2% | 9  48.4%~49.5%  deserializer error | 14.6  49.0%~50.9% |
| **BagNB4** | 2  76.9%~77.2% | 7  88.1% | 23  86.7%~84.5% | 28  87.5%~85.0% |
| **BagNB5** | 0  92.2%~90.5% | 16  85.7% | 20.7  84.8%~82.5% | Zookeeper session expired |
| **BagVHT** | 26  48.4% | 47  50.12% | OutOfMemoryError | Serialization error |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **NSL KDD99**  **evaluation frequency: 10,000**  **speed(second/10,000 instance)**  **correct rate** | | | | |  |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** | **Note** |
| **NB-4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **NB-5** | 0  90.1%~89.8% | 2  90.1%~89.8% | 3  90.1%~89.8% | 8  90.3%~89.8% |  |
| **VHT** | 1.5  78.0%~89.3% | 4  69.5%~87.5% | 2.7  deserializer error  53.0%~53.3% | deserializer error  53.0%~53.3% |  |
| **BagNB4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **BagNB5** | 3.5  90.1%~89.9% | 15.8  90.2%~89.9% | 18.2  90.2%~89.9% | 40.8  90.1%~89.8% |  |
| **BagVHT** | NullPointerException | NullPointerException  59.2%~60.7% | deserializer error  57.0% | deserializer error |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **KDD99**  **evaluation frequency: 100,000**  **speed(second/100,000 instance)**  **correct rate** | | | | |  |
| Algorithm/Cluster | **Local** | **S4-1** | **S4-2** | **S4-3** | **Note** |
| **NB-4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **NB-5** | 20  99.7%~99.3% | 28.3  99.7% | 39  99.6%~99.7% | 118  99.7% ~98.5% |  |
| **VHT** | - | - | - | - | VHT cannot handle error values in dataset |
| **BagNB4** | - | - | - | - | NB-4 cannot process real value  attributes |
| **BagNB5** | 155  99.7%~99.3% | 255  99.7%~99.8% | 285.3  99.8%~99.7% | 808  99.8% |  |
| **BagVHT** | - | - | - | - | VHT cannot process error values in dataset |

1. Kdd99 data: <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> [↑](#footnote-ref-2)
2. Movie Review data: https://www.cs.cornell.edu/people/pabo/movie-review-data/ [↑](#footnote-ref-3)
3. MOA: http://moa.cms.waikato.ac.nz/ [↑](#footnote-ref-4)
4. Breiman, L. 1996a. Bagging predictors . Machine Learning, 24(2), 123-140. [↑](#footnote-ref-5)
5. Efron, B. Tibshirani, R. 1993. An Introduction to the Bootstrap. Chapman and Hall, New York. [↑](#footnote-ref-6)
6. http://www.cs.waikato.ac.nz/ml/weka/arff.html [↑](#footnote-ref-7)