Bayesian Model Averaging for Improving Performance of the Naive Bayes Classifier

Ga Wu

Supervised by: Dr. Scott Sanner

Master of Computing at

The Department of Computer Science

Australian National University
Acknowledgements

I would like to express my very great appreciation to my supervisor Dr. Sanner for his patient guidance and enthusiastic encouragement. Under his supervising, I successfully completed the research project which I could not imagine before and believe this experience will change my life in the future.

I would also like to thank the course coordinator Prof. Liang for his advice and assistance on report writing and presentation preparation. The writing tips he shared are truly treasures which are not only limited in this project report.

My thanks and appreciations also go to my friends in computer science major who discuss machine learning algorithms, share materials and struggle together in the computer lab day and night.
Abstract

Feature selection has proved to be an effective way to reduce the model complexity while giving a relatively desirable accuracy, especially, when data is scarce or the acquisition of some feature is expensive. However, the single selected model may not always generalize well for unseen test data whereas other models may perform better. Bayesian Model Averaging (BMA) is a widely used approach to address this problem and there are several methods for implementing BMA. In this project, a formula appropriate for applying BMA on Naive Bayesian classifier is derived and the corresponding algorithm is also developed. We expect this modified classifier can always do better than Naive Bayes classifier and is competitive with other linear classifiers such as SVM and Logistic Regression, so a series of performance comparison on several different classifiers are made. To provide a comprehensive evaluation, we build all of the test classifiers under same framework API, use UCI data sets as data sources to train classifiers and apply cross-validation to assess statistical significance of the results. The results indicate three property of our modified classifier. Firstly, the prediction accuracy of our modified classifier is either better than Naive Bayes or, at least, equal to that of Naive Bayes and, in some cases, even better than SVM and Logistic Regression. Secondly, the running speed is faster than SVM and Logistic Regression. Thirdly, time complexity of the modified classifier is linear with respect to number of data as well as number of features.

Keywords — Bayesian Model Averaging, Naive Bayes, Feature Selection.
# CONTENTS

## Contents

1 Introduction

2 Mathematic Derivation
   2.1 Bayesian Model Averaging
   2.2 Naive Bayes classifier
   2.3 BMA for Naive Bayes Algorithm
   2.4 Automatic Feature Selection Mechanism
   2.5 Hyper-Parameter and Penalty Factor

3 Implementation
   3.1 Formula in Practice
   3.2 WEKA API and Framework
   3.3 Continuous Feature Handling
   3.4 Robustness and Missing Value Handling
   3.5 Pseudocode and Explanation

4 Experiment and Evaluation
   4.1 Cross-Validation
   4.2 UCI Dataset
   4.3 Control Group
   4.4 Experiment Results

5 Conclusion

A Appendix
   A.1 Notations
      A.1.1 General Notations
      A.1.2 Notation In Equations
LIST OF FIGURES

List of Figures

1 Automatic Model Weighting and Feature Selection 12
2 Value selection pattern of $C = H^{N+1}$ 14
3 Value selection pattern of $C = \left(Z \pm \frac{1}{2}H\right)^{N+1}$ 14
4 Gaussian Distribution Assumption for Feature with Numerical Type 17
5 Random Resampling Cross Validation 22
6 Example of Automatic Model Weighting and Feature Selection 27
7 Accuracy based on number of Features 28
8 Accuracy based on size of Dataset 29
9 Example of Hyper Parameter Tuning Affection 1 31
10 Example of Hyper Parameter Tuning Affection 2 32

List of Tables

1 Feature Selection Example 12
2 Experimental DataSets 24
3 95% Confidence Interval for true error 26
4 Time Consuming Table measured in $\mu$s 30
1 Introduction

When building prediction model, there are always several plausible models. Normally, only one model with relatively reliable prediction on testing dataset is selected to do later classification task. Clearly, manual model selection is hard and time consuming. To improve efficiency, a lot of works have been done to develop automatic model selection methods. However, the single selected model may not always generalize well for previously unseen dataset whereas other unselected models may perform better. Since uncertainty of single model can significantly impacts on decision making, especially for some classification problems which require high performance with little bias, plenty of research works have been done to reduce the impact of uncertainty.

Bayesian Model Averaging provides a way to address this uncertainty caused by model selection. It average over many competitive models with different weights to do conclusive prediction. There are many research works prove that BMA has been applied successfully to many prediction models such as Linear Regression [Raftery et al. 1997], Logistic Regression [Hoeting et al. 1999]. Furthermore, complete BMA implementation package has been built and shared online. In this work, encouraged by the success of previous works, we try to implement Bayesian Model Averaging on Naive Bayes classifier.

There are three reasons to choose Naive Bayes classifier. The first one is that Naive Bayes classifier is very efficient to train, evaluate and interpret. Secondly, Naive Bayes classifier is easy to update in an online classification task [Godec et al. 2010]. Thirdly, it is known to work well in practice, especially, if combined with feature selection [Fung et al. 2011].

Challenges of this attempt come from computational complexity, numerical precision issue and hyper-parameter selection as well as concerns about whether BMA for Naive Bayes classifier can provide a better performance. For BMA on Naive Bayes classifier, the models need to be averaged can be expressed as feature vectors. That is because the only difference of the models is feature combination. Since a complete model averaging requires to take every possible model into account, so for an inference or prediction task with K features, there are $2^K$ possible models need to be considered. Clearly, naive application of BMA has extended complexity in number of features, which makes it intractable for large feature sets. In this work, we investigate whether it is possible to do BMA for the Naive Bayes classifier within linear complexity with respect to number of features as well as number of data. Furthermore, a numerical precision issue is raised. Based on conditional independence assumption, to combine all possible candidate models, nested product operations

\[\text{http://cran.r-project.org/web/packages/BMA/index.html} \]
which may involve underflow risks is inevitable. How to handle it is still challenge. Another task is how to adjust weighting methods for model combination to pursue higher performance. Solving this task focuses on assignment of priori probability of models with given proper hyper-parameters. The last concern is whether BMA for Naive Bayes classifier could generate overall better prediction and, furthermore, under which condition, BMA is always expected to do better. Even though some former research works have proved that this approach can somehow improve accuracy of logistic regression classifier, if it can also improve Naive Bayes algorithm is still unproved.

In this work, a new mathematical model of BMA on Naive Bayes classifier is derived and mathematically proved to be linear. Since traditional model weighting method of BMA is determined by likelihood of prediction models, it is natural for discriminative classifier like logistic regression to recursively modify its parameter but hard to implemented on Naive Bayes. Therefore, how to design a different model selection mechanism is critical in this work. The derived mathematical model is implemented in Java and compared with other linear classifiers. To effectively implement automatic model selection, two hyper parameter tuning methods are introduced in this work. Experiments are performed on UCI datasets and some text classification problems from online news documents. We analyze time consuming, prediction accuracy, hyper-parameter affection of this implementation of BMA and show time consuming table, confidence interval tables, linear plots of hyper-parameter affection as well as classifiers versus number of features and number of data plots separately.

The report is organized as follows. We will introduce a formula derived for implementing BMA for NB in Section 2. Then, we will discuss hyper-parameter selection and how to weight models. In Section 3, we will give some detail about construction of the implemented program, underflow issues, missing value handling as well as brief framework introduction. Experiment, evaluation methods as well as results can be found in Section 4. Conclusions are in Section 5.
2 Mathematic Derivation

2.1 Bayesian Model Averaging

Given a classification problem, which category one certain data belongs to is, based on theory of Bayesian averaging model, determined by weighted summation of posterior probability that each candidate model provides as prediction for each class. The weight of model is in the form of probability which measure importance of that model given training set. To illustrate this more clearly, let’s assume D represent training sets and M is set of competitive models. Correspondingly, \( d_i \) represents \( i^{th} \) training data and \( m_k \) is \( k^{th} \) model or sometimes called hypothesis. Then, we can write down the following equation:

\[
\Pr(y|x, D) = \sum_{k=1}^{K} \Pr(y|x, m_k) \Pr(m_k|D) \tag{1}
\]

where \( \Pr(y|x, m_k) \) refers to posterior probability of model \( k \) and \( \Pr(m_k|D) \) is the weight of model \( K \).

According to Bayes’ Theorem, the weight can be further calculated as posterior probability which is determined by product of priori probability of model \( k \) and probability of datasets \( D \) given model \( k \). And since \( \Pr(d_i|m_k) \) is independent and identically distributed, \( \Pr(D|m_k) \) can be written as product of every single sample in training set. That is:

\[
\Pr(m_k|D) \propto \Pr(m_k) \prod_{i=1}^{N} \Pr(d_i|m_k) \tag{2}
\]

Therefore, how Bayesian model averaging is able to automatically detect better models and assign good weights to them depends on two components: \( \Pr(m_k) \) and \( \Pr(d_i|m_k) \). \( \Pr(m_k) \), priori probability of model \( k \), can be adjusted to improve the performance of automatic detection such as penalizing more complex models. \( \Pr(D) \) is the probability of a training set which is common denominator and can be removed from consideration. But what is \( \Pr(d_i|m_k) \) suppose to be? Note that training data \( d_i \) consists of label \( y_i \) and descriptive attributes set \( x_i \). According to this conception, \( \Pr(d_i|m_k) \) refers to prediction accuracy of model \( k \) on data \( i \). That means if the descriptive attribute set \( x_i \) and model \( k \) is given, \( \Pr(d_i|m_k) \) is the rate that the model can provide correct label \( y \). Therefore,

\[
\Pr(d_i|m_k) = \Pr(y_i|x_i, m_k) \tag{3}
\]

is introduced here.
Thus, equation 1.2 and 3 can be combined together in equation 4.

$$\Pr(y|x, D) \propto \sum_{k=1}^{K} \Pr(y|m_k) \prod_{i=1}^{N} \Pr(y_i|x_i, m_k)$$

(4)

### 2.2 Naive Bayes classifier

In this section, we are going to describe basic conception of Naive Bayes classifier. Naive Bayes classifier is well known for its efficiency to train, evaluate and interpret [Zhang 2004]. And some research works show that the performance of Naive Bayes classifier can be improved significantly by using preprocess approaches such as feature selection [Fung et al. 2011].

Naive Bayes classifier is a probabilistic classifier, which generate posterior probability of class label $y$ given data $\bar{x}$ for all classes and choose the class label with the largest posterior probability as output as shown in equation 5.

$$\arg\max_{y} \Pr(y|\bar{x}) = \arg\max_{y} \left( \frac{\Pr(y) \Pr(\bar{x}|y)}{\Pr(\bar{x})} \right)$$

(5)

where $\Pr(y)$ is prior probability and $\Pr(\bar{x}|y)$ is likelihood probability. The prior probability $\Pr(y)$ refers to proportion of classes in training data set and the likelihood $\Pr(\bar{x}|y)$ represent occurrence probability of $\bar{x}$ if label is $y$. Both of them can be obtained from training process. Since the classifier only concerns which class get the largest probability, common denominator $\Pr(\bar{x})$ can be dropped.

Another important property of Naive Bayes classifier is conditional independence assumption, which means the occurrence probability of one feature given class label $y$ is independent with other features as shown in equation 6:

$$\Pr(\bar{x}|y) = \Pr(x_1|y, x_2, x_3..x_K) \Pr(x_2|y, x_3, x_4..x_K) .. \Pr(x_K)$$

$$= \Pr(x_1|y) \Pr(x_2|y) .. \Pr(x_K|y)$$

$$= \prod_{k=1}^{K} \Pr(x_k|y)$$

(6)

Then, the complete Naive Bayes classifier in log form can be expressed as equation 7.

$$\arg\max_{y} \log(\Pr(y|\bar{x})) \propto \arg\max_{y} \left( \log(\Pr(y)) + \sum_{k=1}^{K} \log(\Pr(x_k|y)) \right)$$

(7)
2.3 BMA for Naive Bayes Algorithm

Consider that if we directly use the previous talked formula in equation 4 to combine every possible Naive Bayes models with different feature combination, there are $2^K$ models need to combine for dataset with $K$ features and, additionally, weights also need to be calculated respectively. It seems to be an unavoidable time-consuming approach. The time complexity is about $O(2^K \log(K)N)$, where $N$ is number of training data and $K$ is number of features. It’s time complexity is clearly not linear and it is disaster for dataset with large number of features. Therefore, a new weighting mechanism is needed.

For solving above questions, we derive Bayesian Model Averaging for Naive Bayes as follow:

$$
\Pr(y|\vec{x}, D) \propto \sum_{k=1}^{K} \Pr(y|m_k) \Pr(m_k) \prod_{i=1}^{N} \Pr(d_i|m_k)
$$

(8)

where we could use a uniform prior: $\forall k \in \{1,2..K\}$

$$
\Pr(m_k) = \frac{1}{K}
$$

(9)

However, for giving models with higher dimension punishment to do automatic feature selection, we assume the distribution of models is not uniform.

With previous definition of $\Pr(d_i|m_k)$, equation 8 is transformed to equation 10. Since training data and testing data is in the exactly same structure, a combination can be made. Thus, entire function become the sum of predicted likelihood’s product multiplied with priori probability of model as shown in equation 11.

$$
\Pr(y|\vec{x}, D) \propto \sum_{k=1}^{K} \Pr(m_k) \Pr(y|m_k) \prod_{i=1}^{N} \Pr(y_i|\vec{x}_i, m_k)
$$

(10)

$$
\Pr(y|\vec{x}, D) \propto \sum_{k=1}^{K} \Pr(m_k) \prod_{i=1}^{N+1} \Pr(y_i|\vec{x}_i, m_k)
$$

(11)

where $y_{N+1}$ is original $y$ corresponding label of testing data and $\vec{x}_{N+1}$ refers to original $\vec{x}$ corresponding descriptive attributes set of testing data.

By previous interpretation of BMA formula, $\Pr(y_i|\vec{x}_i, m_k)$ represent posterior probability provided by model k. According to the Bayes’ theorem, this equation is further derived as shown in equation 12.

$$
\Pr(y|\vec{x}, D) \propto \sum_{k=1}^{K} \Pr(m_k) \prod_{i=1}^{N+1} \Pr(\vec{x}_i|y_i, m_k) \Pr(y_i|m_k)
$$

(12)
Since probability of label $y$ is independent with model $k$, the condition is removed later. It is worth to mention that models in Bayesian Model Averaging for Naive Bayes are only distinguished by using different feature combinations. It is natural to use feature set $\vec{f}$ to express model. We define:

$$m = \vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix}$$

(13)

and $f_k \in \{1, 0\}$. That is if $f_k = 1$, then feature $k$ is selected in the model $m$. Correspondingly, if $f_k = 0$, feature $k$ is not included in that model.

Therefore, the BMA formula is then transformed as follow:

$$\Pr(y|\vec{x}, D) \propto \sum_{f_1} \sum_{f_2} \ldots \sum_{f_k} \Pr(\vec{f}) \prod_{i=1}^{N+1} \Pr(\vec{x}_i|y_i, \vec{f}) \Pr(y_i)$$

(14)

Note that $\prod_{i=1}^{N+1} \Pr(y_i)$ is independent with feature set $\vec{f}$ at all. Therefore, according to associative property, $\prod_{i=1}^{N+1} \Pr(y_i)$ can be extracted from multi-level summation as shown in equation [15]

$$\Pr(y|\vec{x}, D) \propto \left( \prod_{i=1}^{N+1} \Pr(y_i) \right) \left( \sum_{f_1} \sum_{f_2} \ldots \sum_{f_k} \Pr(\vec{f}) \prod_{i=1}^{N+1} \Pr(\vec{x}_i|y_i, \vec{f}) \right)$$

(15)

Since the appearance of features is mutually independent, $\Pr(\vec{f})$ can be further factorized:

$$\Pr(\vec{f}) = \prod_{k=1}^K \Pr(f_k)$$

(16)

Correspondingly, with conditional independence assumption of Naive Bayes classifier, $\Pr(\vec{x}_i|y_i, \vec{f})$ is also can be factorized:

$$\Pr(\vec{x}_i|y_i, \vec{f}) = \prod_{k=1}^K \Pr(x_{ik}|y_i, f_k)$$

(17)

Therefore,

$$\Pr(y|\vec{x}, D) \propto \left( \prod_{i=1}^{N+1} \Pr(y_i) \right) \left( \sum_{f_1} \sum_{f_2} \ldots \sum_{f_k} \prod_{k=1}^K \Pr(f_k) \prod_{i=1}^{N+1} \prod_{k=1}^K \Pr(x_{ik}|y_i, f_k) \right)$$

(18)
After applied associative property again, the product symbol is abstracted as shown in equation [19]

\[
\Pr(y|\vec{x}, D) \propto \left( \prod_{i=1}^{N+1} \Pr(y_i) \right) \left( \prod_{k=1}^{K} \sum_{f_k \in \{0,1\}} \Pr(f_k) \prod_{i=1}^{N+1} \Pr(x_{ik}|y_i, f_k) \right)
\]

where \(x_{ik}\) is feature \(k\) of given data \(i\), \(f_k\) is the \(k^{th}\) feature of explanation set and \(N + 1\) is number of training set plus one tasting data.

In equation [19] \(\prod_{i=1}^{N+1} \Pr(y_i)\) means there are \(N\) training data and one testing data with assumed label \(y\). Since the values of priori probability of label \(y\) for training data are known, the product of those value is constant. And, we only care about for which class the posterior probability is largest. Therefore, \(\prod_{i=1}^{N} \Pr(y_i)\) is removed in equation [20]

\[
\Pr(y|\vec{x}, D) \propto \Pr(y) \times \left( \prod_{k=1}^{K} \sum_{f_k \in \{0,1\}} \Pr(f_k) \prod_{i=1}^{N+1} \Pr(x_{ik}|y_i, f_k) \right)
\]

According to our previous definition of \(f_k\), when \(f_k\) is 1, the feature \(k\) is selected. Therefore, \(\Pr(x_{ik}|y_i, f_k)\) is just \(\Pr(x_{ik}|y_i)\). But if \(f_k\) is equal to zero, there is no clear expression of what \(\Pr(x_{ik}|y_i, f_k)\) is. Then, some assumptions has been made and evaluated by experiment. The final assumption which is proved to be reasonable is that \(\Pr(x_{ik}|y_i, f_k = 0)\) equal to \(\Pr(x_{ik})\). That is:

\[
\Pr(x_{ik}|y_i, f_k) = \begin{cases} 
\Pr(x_{ik}|y_i) & \text{if } f_k = 1 \\
\Pr(x_{ik}) & \text{if } f_k = 0
\end{cases}
\]

This assumption is based on independence concern. Suppose \(x_k\) is independent with \(y\), then the left side of following equation is equal to 1. This assumption also support the feature weighting of automatic feature selection mechanism.

\[
\frac{\prod_{i=1}^{N} \Pr(x_{ik}, y)}{\prod_{i=1}^{N} \Pr(x_{ik}) \Pr(y)} = \frac{\prod_{i=1}^{N} \Pr(x_{ik}|y)}{\prod_{i=1}^{N} \Pr(x_{ik})} \geq 1
\]

As Naive Bayes classifier, \(\Pr(y|\vec{x}, D)\) is just posterior probability and we only concern about for which class the posterior probability is largest as shown in equation [23]

\[
\arg\max_y \Pr(y|\vec{x}, D) \propto \arg\max_y \left( \Pr(y) \times \left( \prod_{k=1}^{K} \sum_{f_k \in \{0,1\}} \Pr(f_k) \prod_{i=1}^{N+1} \Pr(x_{ik}|y_i, f_k) \right) \right)
\]
2.4 Automatic Feature Selection Mechanism

Since BMA combines models with weight, how to make the best model get higher weight, which also known as feature selection, is crucial to ensure accuracy of this algorithms. In equation 20, values of $f_k$ refers to whether that feature is selected or not. According to previous definition of $\Pr(x_{ik}|y_i, f_k)$, an interesting property can be found, if a feature has different distribution in different classes, then the result of $\prod_{i=1}^{N+1} \Pr(x_{ik}|y_i, f_k = 1)$ is always bigger than that of $\prod_{i=1}^{N+1} \Pr(x_{ik}|y_i, f_k = 0)$ as shown in equation 22. And larger variance of distribution in different classes cause larger result of the products. This property can be seen as feature weighting method. For example, suppose we have a classification problem dataset which has three features, A, B and C. Shown as below:

<table>
<thead>
<tr>
<th>Class</th>
<th>$A = 1$</th>
<th>$A = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>$F$</td>
<td>0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>$B = 1$</th>
<th>$B = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>$F$</td>
<td>0.4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>$C = 1$</th>
<th>$C = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>$F$</td>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>

It is clear that feature A is quite trivial, since there is no distributional difference at all in different classes. And feature C has the largest distributional variance between different classes. According to Naive Bayes algorithm, feature C afford the biggest contribution among those three features to do the classification task, which is exactly same as our result of product above. Thus, in this example, the models with feature combination ABC or BC obtain the largest weight. Note that, even A is trivial, its existing or not doesn’t hurt the final accuracy. As contrast, if a model only choose feature A, it should be the worst model, since it is no affection at all.
2.5 Hyper-Parameter and Penalty Factor

Hyper-Parameters are factors that control one or more prior probability in probability theory. In BMA, one hyper parameter is used to control $\Pr(f_k)$ to punish extra feature selection, which means the more features selected by a model, the harsher the punishment applied to that model. For example, in previous example, models which only select combination of ABC or BC are regards to be the best model, when there is no punishment for models with more features\footnote{The prior probability $\Pr(f_k)$ can be defined to be uniform distributed. That is $\Pr(f_k) = \frac{1}{2}$ for both 0 and 1.}. However, feature A is quite useless in the classification task. If the hyper-parameter is adjusted properly, then the model with largest weight will be BC only.

The Figure[1] shows the affection of the punishment. The left hand side represent model weighting without considering number of feature in models. And the right hand side refers to model weighting after applied punishment from hyper-parameter.

Currently, after normalizing, $\Pr(f_k)$ is defined as shown in equation \[24\]

$$\Pr(f_k) = \begin{cases} 
\frac{1}{C} & \text{if } f_k = 1 \\
1 & \text{otherwise}
\end{cases}$$

where $C$ is hyper parameter. Therefore, the final mathematic model of BMA for Naive Bayes classifier is derived as shown below.

$$\arg\max_y \Pr(y|x, D) \propto \arg\max_y \left( \Pr(y) \times \left( \prod_{k=1}^{K} \left( \prod_{i=1}^{N+1} \Pr(x_{ik}) + \frac{1}{C} \prod_{i=1}^{N+1} \Pr(x_{ik} | y_i) \right) \right) \right)$$

(25)

where $x_{ik}$ is feature $k$ of given data $i$, $C$ is hyper parameter and the $N + 1$th data refers to testing data.

Since $C$ is a real number and the range of $C$ can be $(-\infty, +\infty)$, it is hard and unnecessary to tune it step by step with relative small movements. Thus, $C$ needs to be designed more complicate for shrinking tuning range.

There are two methods to tune hyper parameter $C$. The first method selects values with equal intervals. And, the second one select values based on exponential expression which select more values that close to mean of selection range.

The first one is quite simple and straightforward, but still effective in most cases.

$$C = H^{N+1}$$

(26)
where $N$ is size of training data, $H \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$

\[ C = \left( Z \pm \frac{1}{2} H \right)^{N+1} \quad (27) \]

The second one is relatively sophisticated. It is considered to handle potential numerical bias and noise which may cause the left part of equation 22 is much larger than 1, even $x_{ik}$ is not really quite dependant with class $y$. The modified centre of tuning options is $Z$. And, there are more tuning options close to $Z$.

The centre of tuning options, $Z$, ensures $\frac{1}{Z^{N+1}} \prod_{i=1}^{N+1} \Pr(x_{ik}|y_i)$ equals to $\prod_{i=1}^{N+1} \Pr(x_{ik})$ when feature is completely no contribution but only noise and tuning arm $H$ is default value $+\infty$. $H$ and $Z$ are separately in range of

\[ H \in \{1, 2, 3, 4, 5, +\infty\} \quad (28) \]

\[ Z \in \{0.8, 0.9, 1, 1.1, 1.2\} \quad (29) \]

In this experiment, the first method is used to do experiment. The first method is sufficient to do most cases. The second method has 30 combinations which is time consuming and it is only proper to use when the simple method doesn’t work well on certain classification problem.
3 Implementation

3.1 Formula in Practice

If we notice that there are too many multiplication in the function which may involve underflow issue, it is obvious that our formula deducted in previous section is not ready for implementing in programming language. So, log form is again used here as equation (30) shown below:

$$\log(Pr(y|\bar{x}, D)) \propto \log(Pr(y)) \times \left( \sum_{k=1}^{K} \log(G(k = 1) + G(k = 0)) \right)$$  \hspace{1cm} (30)

where $G(k) = Pr(f_k) \prod_{i=1}^{N+1} Pr(x_{ik}|y_i, f_k)$

However, it is still insufficient to avoid underflow problem. When number of training data increasing, function $G(k)$ tend to be to zero. Therefore, a much simple question similar to this one is considered: How to transform $\log(A + B)$? It turns out if it is changed into the form shown in equation (31) the underflow problem can be solved.

$$\log(A + B) = \log(A) + \log\left(\frac{B}{A} + 1\right) \text{ if } A > B$$  \hspace{1cm} (31)

Correspondingly, if value of $G(k = 1)$ is much larger than that of $G(k = 0)$, $\log(G(k = 1) + G(k = 0))$ is very close to $\log(G(k = 1))$. The property of equation (31) is corresponding to automatic feature selection. If A represents that the feature is not selected and B represents that is selected, it is good to have results that either A or B, but not both or nothing.

3.2 WEKA API and Framework

In this work, I use Java an object-oriented programming language to implement the BMA algorithm under eclipse workbench. Since Java has the advantage of great encapsulation property as Object-oriented programming language, I build several package, classes and interfaces to meet OOP standard, which also give a good structure for following explanation. Eclipse JAVA standard IDE is also used in this project to provide a better package management and version control. Furthermore, the environment of experiment is macbook pro 721 with 16GB memory which installed mac OS 10.9 operating system. Note that Eclipse restrict memory usage by define eclipse.ini file in software package which I don’t change.

WEKA\textsuperscript{3}, which contents a lots of classes of models, is a tool used in my project to do final evaluation. Actually, WEKA is a complete suite of machine learning

\textsuperscript{3}WEKA: Waikato Environment for Knowledge Analysis
software which was written in Java. That means it is product that can directly be used in data analysis and data mining tasks. It is worth mentioning that analysis algorithm implementing codes of WEKA is an excellent way to learn machine learning. This also proves the predictive model it provided is quite stable, popular and also meet the requirement of being control algorithms. Note that we only import and use function Library of WEKA but not graphic interface.

The control group of the experiments consist of Logistic Regression, traditional Naive bayes as well as support vector machine. Naive Bayes classifier is a in-built WEKA class with pre-processing setting options, and SVM as well as Logistic Regression are abstracted from Liblinear package. To provide fair comparison, my implementation is also made to be able to call those pre-processing functions as well as setting options. Note that my implement of Bayesian model averaging is not built on Naive bayes of WEKA, since some functions cannot be effectively extend from it. Therefore, A customized Gaussian Naive Bayes classifier has been built as base of BMA. To ensure the experiment is complete reliable, I kept the customized Naive Bayes classifier available and have it joined in comparing group as well. It can get exactly same results with Naive Bayes of WEKA in any circumstance.

Data sets I used in this project is from UCI data sets which have kinds of classification questions. But, considering capability of linear classifiers, only those dataset which can be separated by linear classifier have been used. Additionally, since UCI data is in form of arff document, file reader function and storage class of WEKA API are also hired.

3.3 Continuous Feature Handling

In previous subsections, the basic idea of Bayesian model averaging for naive bayes classifier is introduced without considering continuous features which may significantly affect this algorithm. For handling continuous features, there are two main way that normally being used for naive bayes. The first one is focusing on Gaussian assumption which assume that the values in continuous feature are following gaussian distribution for each class. Another way is to discretize continuous value by supervised clustering.

The traditional Naive Bayes classifier with gaussian distribution assumption is called gaussian naive bayes. This approach focuses on assuming continuous features are normal distributed. Theoretically, according to the Central limit theorem, it is very natural to prove the rationality of assumption. However, as the theory implied, if the data set is very large, then it is good to apply this method. As contrast, if

\footnote{Actually, it is very common to see data mining introduction books which use WEKA as support theoretical practice such as "Data mining: Practical machine learning tools and techniques".}
the data set is not big enough, the accuracy of this method may be unsatisfiable. In this project, this method is applied for the Bayesian model averaging to handle numerical features.

Since pursuing the precision of probability of value in Gaussian distribution is costly, the probability of value is approximated by area of rectangular with density as height and a proper small value as width which is calculated based on \( \sigma \) as shown in equation. The expensive time consuming does not significantly impact Naive Bayes classifier, but, for BMA, the cost is over the roof. Note that the \( \prod_{i=1}^{N} \Pr(x_{ik}|y_i, f_k) \) requires \( 2 \times N \) times calculation.

\[
\Pr(x) = \int_{-\infty}^{x+\frac{1}{2}\delta} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \, dz - \int_{-\infty}^{x-\frac{1}{2}\delta} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \, dz \\
\approx \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \times \delta
\]

(32)

where \( \delta = \frac{\sqrt{\pi}}{N} \) and \( N \) is number of value added from training set.

Figure 4: Gaussian Distribution Assumption for Feature with Numerical Type

The missing value problem may cause gaussian BayesiaNaiven Model Averaging algorithm failure. To successful dealing with classification problem that contain missing value, the normally approach is to impute mean value of the gaussian distribution in the missing place. However, this will be time consuming. What the
3 IMPLEMENTATION

weka does is just discard that feature in training process and ignore any of missing value in testing period which means it does classification with incomplete feature set. To make our implementation get exactly same preprocessing methods to meet basic fairness requirement of comparison, I also used weka’s approach.[5]

The most popular approach to handle continuous value is discretisation which goal to separate continuous values into several discrete ranges. The advantage of this approach is that it performs very stable and normally provides better performance. There are several great way to implement this approach[7]. However, it also has some shortcomings such as increasing time complexity. The worst disadvantage for us is the feature discretised may not be always same with the increasing of dataset. That makes our performance comparison of algorithm very difficult to keep fairness, since the features in each algorithm may not same. The related implementation of this approach on BMA can be activated by using parameter $-D$. But in the following experiment, it is not activated.

3.4 Robustness and Missing Value Handling

Since size of training data is always limited, it is impossible to guarantee the classifier would never meet unseen feature values. To tolerate existence of new feature values, smoothing method has been applied in the algorithm to meet baseline robustness requirement. The smoothed estimator is following Dirichlet prior distribution as equation shown below. Normally, when $l$ is set to 1, then this approach is called as Laplace smoothing[M. Mitchell 2010]. Note that $l$ is also hyper-parameter, but in this work, we keep it to 1.

$$\Pr(X_i = x_{ij}|Y = y_k) = \frac{\#D\{X_i = x_{ij} \land Y = y_k\} + l}{\#D\{Y = y_k\} + lJ}$$ (33)

For numerical features, since we dropped missing value directly in training process, it is possible that one class may doesn’t contain any value at all. This means there exist feature $x_i$ in class $y$ that doesn’t follow normal distribution, since there is no mean and standard deviation. To handle this, a default baseline standard deviation has been signed. Note that, mean is always zero, since the normal distribution we used here is after standardized.

Another risk that may cause algorithm fail is missing class label of training dataset or class label is real. The goal of this algorithm is to deal with classification problem, so A rejection preprocessing function has been applied to check and reject regression problem as well as defective files.

3.5 Pseudocode and Explanation

In this subsection, a thorough explanation of this Bayesian Model Averaging implementation is described by using pseudo-code as well as corresponding analysis. As most of classification algorithms, this method also consists of training step and testing step. Additionally, since this algorithm is implemented through Java, it is relatively easy to use purely Object-oriented programming style to present a good structure. Note that the pseudo-code provided in this work also in the perspective of OOP. Algorithm 1 shown below is training part of BMA.

Algorithm 1 BayesianModelAveragingForNB

Require: Global Variable: conditionalProbability and prioriProbability

1: function BUILDCLASSIFIER(trainingset)
2: \[
\begin{align*}
PD & \leftarrow \text{PREPROCESSING}(\text{trainingset}) \\
\text{for } i & \leftarrow 1, \text{PD}.\text{NumberOfClasses} + 1 \text{ do} \\
\text{for } j & \leftarrow 1, \text{PD}.\text{NumberOfAttributes} \text{ do} \\
\text{conditionalProbability}[i][j].\text{INITIALISE()} & \Rightarrow \Pr(x_i|y_j) \\
\text{end for} \\
\text{end for} \\
prioriProbability.\text{INITIALISE()} & \Rightarrow \Pr(y) \\
\text{for all } \text{inst} \in PD \text{ do} \\
\text{for } j & \leftarrow 1, \text{Inst}.\text{NumberOfAttributes} \text{ do} \\
\text{conditionalProbability}[j][\text{inst.class}].\text{AddValue(\text{inst.attr(j).value})} \\
\text{conditionalProbability}[j][\text{null}].\text{AddValue(\text{inst.attr(j).value})} \\
\text{end for} \\
prioriProbability.\text{AddValue()} \\
\text{end for} \\
\text{conditionalProbability}.\text{CALCULATEPROBABILITY()} \\
prioriProbability.\text{CALCULATEPROBABILITY()} \\
\text{LOGSUMPROCESSING(PD)} \\
\text{end function}
\end{align*}
\]

Note that in line 3, there are \( N + 1 \) number of classes space has been initialized, since one of them is used to store \( \Pr(x) \) that without any conditions. Correspondingly, in line 12, values are also added in to this storage without considering class information of that data.

The training procedure is quite similar with traditional Naive Bayes classifier except some additional functions and storages used to store extra information we need. Before explaining those extra functions, it is must clear that the global variables written in Algorithm 1 is not very global. It is actually private member of classifier class, from which our final model initialised. Additionally, the buildClassifier function in class BMA of NB classifier demonstrates training process.
LogSumProcessing() function shown in Algorithm 2 is a further processing of training part, which gathering information about weights of models that our Bayesian Model Averaging algorithm concerned mostly.

Algorithm 2 BayesianModelAveragingForNB

Require: Global Variable : conditionalProbability[] and prioriProbability

1: function LogSumProcessing(PD)
2:   SumOfCondProbInLog.INITIALIZE
3:   SumOfAveCondPorbinLog.INITIALIZE
4:   for i ← 1, PD.NUMBEROFATTRIBUTES do
5:     SumOfCondProbInLog[i] ← SUMLOGCONDPROB(i);
6:     SumOfAveCondProbInLog[i] ← SUMAVELOGCONDPROB(i);
7:   end for
8: end function
9: function SUMLOGCONDPROB(PD, Index) \[> \prod_{i=1}^{N}Pr(x_{ik}|y_i, f_k) & k = 1 \]
10:    Sum ← 0
11:   for i ← 1, PD.NUMBEROFINSTANCES do
12:      Temp ← conditionalProbability[Index][PD[i].class].value(PD[i])
13:      Sum ← Sum + log(Temp)
14:   end for
15:  return Sum
16: end function
17: function SUMAVELOGCONDPROB(PD, Index) \[> \prod_{i=1}^{N}Pr(x_{ik}|y_i, f_k) & k = 0 \]
18:    Sum ← 0
19:   for i ← 1, PD.NUMBEROFINSTANCES do
20:      Temp ← conditionalProbability[Index][null].value(PD[i])
21:      Sum ← Sum + log(Temp)
22:  end for
23:  return Sum
24: end function

In the implementation, Algorithm 2 is the final version that proved to be correct among all attempting. According to all of description so far, it seems that the training part of BMA for NB is quite increasing the running time. However, the theoretical time complexity is only $O(NF)$, where $N$ is the number of training samples and $F$ is number of features. Therefore, the time complexity for training process is linear for both $N$ and $F$. the time complexity of testing process is $O(CF)$ , where $C$ is number of potential classes in the classification problem.

The following pseudo-code is the testing function of classifier class. Since the priori probability $Pr(f_k = 0)$ equals to 1 and $Pr(f_k = 1)$ is effected by hyper pa-
3 IMPLEMENTATION

rameter, it doesn’t show in the testing part of algorithm explicitly. Note that the value of these two priori probability is re-ranked by discard common denominator. We only represent it by C in line 11.

Algorithm 3 BayesianModelAveragingForNB

Require: Global Variable : \((SC)\)SumOfCondProbInLog[], \((P)\)prioriProbability (\((CP)\)conditionalProbability[]) and \((SAC)\)SumOfAveCondProbInLog[]

1: function CLASSIFYINSTANCE(instance) \(\uparrow\) Initial
2: \(\mathbb{I} \leftarrow \text{PREPROCESSING}(\text{instance})\)
3: SelectedClassIndex = 1
4: MaxProbability = 0
5: for \(i \leftarrow 1, \text{\text{NumberOfClasses}}\) do
6: \(\mathbb{I}.\text{classIndex} \leftarrow i\)
7: \(\text{Prob}[i] \leftarrow \log(\text{P.CLASS}(i))\)
8: for \(j \leftarrow 1, \text{\text{NumberOfAttributes}}\) do
9: \(A \leftarrow SC + \log(\text{CP}[j][i].\text{VALUEOF}(\mathbb{I}))\)
10: \(B \leftarrow SAC + \log(\text{AVERAGE}(\text{CP}[j][\mathbb{I}]))\)
11: \(\text{Prob}[i] = \text{Prob}[i] + \text{LOGSUM}(A/C,B)\) \(\uparrow\) C is Hyper-Parameter
12: end for
13: if \(i = 1\) then
14: \(\text{MaxProbability} \leftarrow \text{Prob}[i]\)
15: end if
16: if \(i > 1\) then
17: if \(\text{MaxProbability} < \text{Prob}[i]\) then
18: \(\text{MaxProbability} \leftarrow \text{Prob}[i]\)
19: \(\text{SelectedClassIndex} \leftarrow i\)
20: end if
21: end if
22: end for
23: return SelectedClassIndex
24: end function

The pseudo-code above is just for provide a clear description of this algorithm, which is without optimization. For obtaining faster running speed of testing process. We can explicitly compare \(SAC\) with \(SC\) and if \(SAC\) is larger then do nothing. Correspondingly, if \(SC\) is larger, then plus \(\log(\text{CP}[j][\mathbb{I}].\text{VALUEOF}(\mathbb{I}))\) on \(\text{Prob}[i]\). Therefore, the running time can be further reduced.
4 Experiment and Evaluation

4.1 Cross-Validation

Cross-validation, also named rotation estimation, is a method that separate dataset into several subsets to make some kinds of model analysis. Normally, cross-validation is not only used to get average error rate of classification but also is used to tune hyper-parameter of classifier to fit dataset.

There are several common types of cross-validation. The first one is called K-fold cross-validation, which randomly divide original sample set into subsets with equal size. One of those subsets then be chosen as data used to test model and rest of the subsets plays role of training data. This process repeats until every subset has been chosen to be testing data. The results of k times evaluation then be averaged to produce a single estimation. The second one called Repeated random sub-sampling validation. This method randomly splits the dataset into training data and testing data and build model on training set to predict label of testing data. Then, evaluate result. This process also repeat several times and the final evaluation is average of all results. However, the difference between this method with K-fold is that the proportion of the testing data is not dependent on the number of iterations. Cross-validation implementations are not limited in these two methods.

In this work, A nested random resampling cross-validation model has been used to do the evaluation task for two reasons. First of all, the 10-folds cross-validation is misleading when there are sparse data. Moreover, some task needs more than 10 times experiments without effecting proportion of training and testing data sets.

Figure 5: Random Resampling Cross Validation

2-fold cross-validation, leave-one-out cross-validation are commonly used in practice as well.
But 10-folds cross validation is still useful when dataset is large enough. Since the complete double level cross-validation would take very long time to do and doesn’t provide much better hyper-parameter tuning results than fixed fold tuning\(^7\), tuning hyper parameter is implemented on fixed training and validating dataset without crossing.

This method assumes the label of testing data set is unknown and it uses part of known dataset to adjust hyper-parameter, then apply it on whole training data set to predict label of testing data set. More detail about the cross-validation method can be find in the following algorithm.

**Algorithm 4** Fast Cross Validation

Require: Classifier, Dataset

1: function RANDOM CROSS-VALIDATION(Classifier, Dataset, numFold)

2: Initial Result[numFold]

3: for \(i \leftarrow 1\) to numFold do

4: Dataset.PERMUTATION(Seed) \(\triangleright Re – \text{Arrange}\)

5: Testingset \(\leftarrow\) RANDOMSELECTION(Dataset, 1/10)

6: Trainingset \(\leftarrow\) Dataset.RESTOFINSTANCES

7: Classifier.HYPERPARAMETERTUNING(Trainingset)

8: Classifier.TRAIN(Trainingset)

9: Result[i] \(\leftarrow\) Classifier.TEST(Testingset)

10: end for

11: ANALYSIS(Result)

12: end function

13: function HYPERPARAMETERTUNING(Sampleset)

14: Initial Result[ValueOfHyperparameter.size] \(\triangleright Initial\)

15: for \(h \leftarrow 1\) to ValueOfHyperparameter.size do

16: Classifier.SETTING(h) \(\triangleright Setting\ \text{Parameter}\)

17: Folds[] \(\leftarrow\) Sampleset.SPLITTOSETS

18: Validingset \(\leftarrow\) Folds[1]

19: Trainingset \(\leftarrow\) Sampleset.RESTOFINSTANCES

20: Classifier.TRAIN(Trainingset)

21: Result[i] \(\leftarrow\) Classifier.TEST(Testingset)

22: end for

23: OptimalParameter \(\leftarrow\) GETBESTHYPERPARAMETER(Result)

24: Classifier.SETTING(OptimalParameter)

25: end function

\(^7\text{use one fix fold as validating dataset and others as training dataset to tune hyper-parameter without crossing}\)
4.2 UCI Dataset

Since the Bayesian Model Averaging for Naive Bayes algorithm we derived is expected to be capable for general usage, kinds of classification data sets that from different research fields are required to test the algorithm. Furthermore, to analyze the conditions that effect the performance of BMA, some feature of datasets is also preferred such as features number, data size missing value as well as how many target classes involved, etc.

<table>
<thead>
<tr>
<th>FileName</th>
<th>Features</th>
<th>Data Size</th>
<th>Classes</th>
<th>Ratio</th>
<th>Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>anneal.arff</td>
<td>38</td>
<td>798</td>
<td>6</td>
<td>4.76%</td>
<td>Yes</td>
</tr>
<tr>
<td>autos.arff</td>
<td>26</td>
<td>205</td>
<td>7</td>
<td>12.68%</td>
<td>Yes</td>
</tr>
<tr>
<td>vote.arff</td>
<td>16</td>
<td>435</td>
<td>2</td>
<td>3.68%</td>
<td>Yes</td>
</tr>
<tr>
<td>sick.arff</td>
<td>30</td>
<td>3772</td>
<td>2</td>
<td>0.80%</td>
<td>Yes</td>
</tr>
<tr>
<td>crx.arff</td>
<td>16</td>
<td>690</td>
<td>2</td>
<td>2.32%</td>
<td>No</td>
</tr>
<tr>
<td>mushroom</td>
<td>22</td>
<td>8124</td>
<td>2</td>
<td>0.27%</td>
<td>Yes</td>
</tr>
<tr>
<td>heart-statlog.arff</td>
<td>13</td>
<td>270</td>
<td>2</td>
<td>4.81%</td>
<td>No</td>
</tr>
<tr>
<td>segment.arff</td>
<td>19</td>
<td>2310</td>
<td>7</td>
<td>0.82%</td>
<td>No</td>
</tr>
<tr>
<td>labor.arff</td>
<td>16</td>
<td>57</td>
<td>2</td>
<td>28.07%</td>
<td>No</td>
</tr>
<tr>
<td>vowel.arff</td>
<td>14</td>
<td>990</td>
<td>11</td>
<td>1.41%</td>
<td>No</td>
</tr>
<tr>
<td>audiology.arff</td>
<td>69</td>
<td>226</td>
<td>23</td>
<td>30.53%</td>
<td>Yes</td>
</tr>
<tr>
<td>iris.arff</td>
<td>5</td>
<td>150</td>
<td>3</td>
<td>3.33%</td>
<td>No</td>
</tr>
<tr>
<td>zoo.arff</td>
<td>18</td>
<td>101</td>
<td>7</td>
<td>17.82%</td>
<td>No</td>
</tr>
<tr>
<td>lymph.arff</td>
<td>19</td>
<td>148</td>
<td>4</td>
<td>12.84%</td>
<td>No</td>
</tr>
<tr>
<td>soybean.arff</td>
<td>35</td>
<td>683</td>
<td>19</td>
<td>5.12%</td>
<td>Yes</td>
</tr>
<tr>
<td>balance-scale.arff</td>
<td>4</td>
<td>625</td>
<td>3</td>
<td>0.64%</td>
<td>No</td>
</tr>
<tr>
<td>glass.arff</td>
<td>10</td>
<td>214</td>
<td>7</td>
<td>4.67%</td>
<td>No</td>
</tr>
<tr>
<td>letter.arff</td>
<td>16</td>
<td>20000</td>
<td>26</td>
<td>0.08%</td>
<td>No</td>
</tr>
<tr>
<td>hepatitis.arff</td>
<td>19</td>
<td>155</td>
<td>2</td>
<td>12.25%</td>
<td>No</td>
</tr>
<tr>
<td>haberman.arff</td>
<td>4</td>
<td>306</td>
<td>2</td>
<td>1.31%</td>
<td>No</td>
</tr>
</tbody>
</table>

In this project, Datasets are abstracted from UCI datasets package. The above table shows some datasets used in the experiments, which can be handled by linear classifiers with only one single target label. To discover some kinds of potential pattern that can effect the performance of those algorithms including the Bayesian Model Averaging implemented in this project, My experiment works on about 20 classification problem with more than one label which represent that data is belong to more than one classes.
datasets which include medical records, selling records and some classification problem of sociology.

4.3 Control Group

The control group of experiments consists of SVM, Logistic Regression and Naive-Bayes classifiers. Naive Bayes classifier is the most representative generative linear classification method and normally used as baseline classifier. Logistic Regression represents classical discriminative linear classifier. And SVM is well known for its great performance but relative lower speed. In this work, the BMA for NB classifier is expected to get better performance than Naive Bayes classifier and still maintain linear time complexity and, of course, is also expected faster than SVM.

The SVM and Logistic Regression is abstracted from Liblinear API package. The equation is regularized error function, which is used by both SVM and Logistic Regression to optimize accuracy but avoid overfitting problem. Note that the specific error functions of those two classifiers are different.

\[
\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{l} \xi (\mathbf{w}; \mathbf{x}_i; y_i) \tag{34}
\]

Normally, as a tradition, regularization parameter is denoted as $\lambda$. But in Liblinear package, the $\lambda$ is replaced by its reciprocal named $C$. Therefore, $C$ is regarded as penalty factor or cost factor. With increasing of $C$, the classifiers are more sensitive to new dataset and even overfitting. However, A small value of $C$ may not fit data, and bias of prediction with true value can be very big.

Another important parameter of Liblinear package is $B$, which refers to bias. Note that this bias is a constant which represent one additional dimension, but not error bias. The following equation corresponds to separating hyperplane.

\[
y (\mathbf{x}, \mathbf{w}) = \sum_{k=1}^{K} w_k \phi_k (\mathbf{x}) + B \tag{35}
\]

where $\phi_k$ is base function, $\mathbf{w}$ is parameter vector and $B$ is bias.

For Naive Bayes classifier, it is component of WEKA API and works without Hyper-Parameter Tuning. However, it is still has some useful setting options. The most important one is $- D$, which active supervised discretizing pre-processing.

\[\text{Liblinear: A library of Large Linear Classification is released by Machine Learning Group of National Taiwan University. This package is widely used in machine learning practice, and compatible with WEKA, R and other machine learning softwares}\]
4.4 Experiment Results

In this sub section, a performance comparison of BMA for NB with other classifier is made. Since parameter tuning of SVM and LR is very time consuming, only 36 parameter combination are provided for automatic tuning as shown bellow.

\[
C \in \{0.01, 0.1, 1, 10, 20, 100\} \quad (36)
\]
\[
B \in \{0.01, 0.1, 1, 10, 20, 100\} \quad (37)
\]

The following table gives 95% confidence interval of true error, in which outstanding classifier is bolded. Since the modification limitation of classifier from WEKA, my implement of Bayesian Model Averaging is established on a customized naive bayes classifier. To guarantee there is no doubt on performance bias of customised classifier with that of WEKA, both of their error rates are included in this table and following tables.

<table>
<thead>
<tr>
<th>Problems</th>
<th>BMA</th>
<th>CNB</th>
<th>NB</th>
<th>LR</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>anneal</td>
<td>7.87±2.00</td>
<td>7.87±2.00</td>
<td>7.87±2.00</td>
<td><strong>6.74±1.86</strong></td>
<td>17.98±2.86</td>
</tr>
<tr>
<td>* autos</td>
<td>30.00±7.13</td>
<td>40.00±7.62</td>
<td>40.00±7.62</td>
<td><strong>25.00±6.74</strong></td>
<td>55.00±7.74</td>
</tr>
<tr>
<td>* vote</td>
<td>4.65±2.25</td>
<td>11.63±3.42</td>
<td>11.63±3.42</td>
<td>4.65±2.25</td>
<td>4.65±2.25</td>
</tr>
<tr>
<td>* sick</td>
<td>6.37±0.89</td>
<td>7.69±0.97</td>
<td>7.69±0.97</td>
<td><strong>2.65±0.58</strong></td>
<td>4.51±0.75</td>
</tr>
<tr>
<td>crx</td>
<td>23.19±3.58</td>
<td>23.19±3.58</td>
<td>23.19±3.58</td>
<td><strong>18.84±3.32</strong></td>
<td>21.74±3.50</td>
</tr>
<tr>
<td>* mushroom</td>
<td><strong>1.35±0.29</strong></td>
<td>3.82±0.47</td>
<td>3.82±0.47</td>
<td>2.96±0.42</td>
<td>4.06±0.49</td>
</tr>
<tr>
<td>heartstat</td>
<td>7.41±3.55</td>
<td>7.41±3.55</td>
<td>7.41±3.55</td>
<td>18.52±5.27</td>
<td>11.11±4.26</td>
</tr>
<tr>
<td>segment</td>
<td>23.81±1.97</td>
<td>23.81±1.97</td>
<td>23.81±1.97</td>
<td><strong>10.82±1.44</strong></td>
<td>14.72±1.64</td>
</tr>
<tr>
<td>labor</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
</tr>
<tr>
<td>* vowel</td>
<td><strong>33.33±3.34</strong></td>
<td>42.42±3.50</td>
<td>42.42±3.50</td>
<td>44.44±3.52</td>
<td>47.47±3.54</td>
</tr>
<tr>
<td>* audiology</td>
<td>22.73±6.21</td>
<td>27.27±6.60</td>
<td>27.27±6.60</td>
<td><strong>9.09±4.26</strong></td>
<td>13.64±5.09</td>
</tr>
<tr>
<td>* iris</td>
<td>6.67±4.54</td>
<td>13.33±6.18</td>
<td>13.33±6.18</td>
<td>6.67±4.54</td>
<td>6.67±4.54</td>
</tr>
<tr>
<td>zoo</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
</tr>
<tr>
<td>soybean</td>
<td>11.76±2.75</td>
<td>11.76±2.75</td>
<td>11.76±2.75</td>
<td><strong>10.29±2.59</strong></td>
<td>11.76±2.75</td>
</tr>
<tr>
<td>glass</td>
<td>52.38±7.61</td>
<td>52.38±7.61</td>
<td>52.38±7.61</td>
<td><strong>33.33±7.18</strong></td>
<td>47.62±7.61</td>
</tr>
<tr>
<td>* letter</td>
<td><strong>34.05±0.75</strong></td>
<td>35.25±0.75</td>
<td>35.25±0.75</td>
<td>STUCK</td>
<td>STUCK</td>
</tr>
<tr>
<td>hepatitis</td>
<td>26.67±7.91</td>
<td>26.67±7.91</td>
<td>26.67±7.91</td>
<td>60.00±8.77</td>
<td>46.67±8.93</td>
</tr>
<tr>
<td>haberman</td>
<td>33.33±6.00</td>
<td>33.33±6.00</td>
<td>33.33±6.00</td>
<td>36.67±6.14</td>
<td>33.33±6.00</td>
</tr>
</tbody>
</table>

The classification problem letter consists of 20,000 data, which is a challenge for Logistic Regression and SVM to finish classification in relatively short time. In
this work, the threshold of running time is 10 minutes. They don’t finish running in the limited time and are labeled to be stuck. According to the designer of the Liblinear package, this is caused by large assignment of bias $B$. However, small $B$ is not sufficient to get good prediction accuracy. It seems to be a time complexity and accuracy dilemma.

According to table 3, Bayesian Model Averaging for Naive Bayes classifier gets either better performance than Naive Bayes or gets equivalent performance with its base. It is because of the automatic feature selection mechanism of BMA for NB as shown in Figure 6.

![Figure 6: Example of Automatic Model Weighting and Feature Selection](image)

Since feature selection doesn’t guarantee getting better performance and can even decrease prediction accuracy\textsuperscript{10}, it is good to know that for which kinds of clas-

\textsuperscript{10}For some classification problem, the feature set has already thoroughly chose and collected. Therefore, feature extraction maybe more useful than selection.
sification problem feature selection is good and then decide whether or not applying feature selection.

In natural language processing field, it is very important to do entity recognition, feature selection as well as dimension reduction, since there are so many useless or similar features which can decrease accuracy. Therefore, text classification problems are good to test the affection of feature selection. In this work, a binary classification problem which is abstracted from 20 newsgroup dataset has been used to test the affection of feature number on prediction accuracy as shown in Figure 7.

![Figure 7: Accuracy based on number of Features](image)

This plot is worked on 1000 data which selected from two categories of news. it is an exactly balanced dataset. The 1000 features are selected from 20,000 words with high frequency of occurrence after applied stop list checking. Note that this plot is averaged on results from multiply runs with feature resampling.

After around 150 features selected, the BMA, SVM and Logistic Regression show significantly better performance than Naive Bayes classifier. Furthermore, the accuracy of BMA, SVM and Logistic Regression are also stabilized earlier. It shows that BMA has better noise resistant property.
To show whether training dataset size can significantly impact the accuracy of news classification task, the related experiment has made. This plot is generated when classifiers perform on news group dataset with 2000 data and 500 selected features. The testing dataset which has 500 data is fix in each run. Note that this plot is averaged on results from multiply runs with data resampling.

![News Group](image)

Figure 8: Accuracy based on size of Dataset

The experiment shows that the prediction accuracy is stable after enough training data is used and further training is no significant affection.

The following is time consuming table which records running time of classifiers for all of those classification problems. Note that this table generated without taking hyper parameter tuning time into account. In practice, hyper parameter tuning occupies much more time then classifier training procedure itself. In this experiment, 36 setting options are used for SVM and Logistic Regression, but they still can’t guarantee to provide relative reliable results for all of those classification problems. As contrast, even though no setting options are used to optimize performance of BMA, it is sufficient to make sure the performance is not worse than baseline classifier.
Table 4 Time Consuming Table measured in µs

<table>
<thead>
<tr>
<th>Problems</th>
<th>BMA</th>
<th>CNB</th>
<th>NB</th>
<th>LR</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>anneal</td>
<td>709956</td>
<td>169878</td>
<td>151395</td>
<td>581915</td>
<td>2059308</td>
</tr>
<tr>
<td>autos</td>
<td>106283</td>
<td>46622</td>
<td>66247</td>
<td>177798</td>
<td>1287179</td>
</tr>
<tr>
<td>vote</td>
<td>11200</td>
<td>5836</td>
<td>7404</td>
<td>21812</td>
<td>33452</td>
</tr>
<tr>
<td>sick</td>
<td>323999</td>
<td>94433</td>
<td>82246</td>
<td>426820</td>
<td>2487667</td>
</tr>
<tr>
<td>crx</td>
<td>76626</td>
<td>24516</td>
<td>14912</td>
<td>64160</td>
<td>541535</td>
</tr>
<tr>
<td>mushroom</td>
<td>271375</td>
<td>81395</td>
<td>88018</td>
<td>1061161</td>
<td>5759504</td>
</tr>
<tr>
<td>heart-statlog</td>
<td>36333</td>
<td>14071</td>
<td>7421</td>
<td>21932</td>
<td>216576</td>
</tr>
<tr>
<td>segment</td>
<td>588319</td>
<td>211444</td>
<td>156669</td>
<td>2611166</td>
<td>6703656</td>
</tr>
<tr>
<td>labor</td>
<td>3726</td>
<td>1763</td>
<td>2119</td>
<td>4973</td>
<td>26152</td>
</tr>
<tr>
<td>vowel</td>
<td>139689</td>
<td>48997</td>
<td>46199</td>
<td>579517</td>
<td>5329444</td>
</tr>
<tr>
<td>audiology</td>
<td>95783</td>
<td>13767</td>
<td>42654</td>
<td>170953</td>
<td>264345</td>
</tr>
<tr>
<td>iris</td>
<td>4970</td>
<td>2167</td>
<td>2508</td>
<td>10140</td>
<td>119829</td>
</tr>
<tr>
<td>zoo</td>
<td>6982</td>
<td>2806</td>
<td>3524</td>
<td>31512</td>
<td>219978</td>
</tr>
<tr>
<td>lymph</td>
<td>12320</td>
<td>5566</td>
<td>4678</td>
<td>21867</td>
<td>173912</td>
</tr>
<tr>
<td>soybean</td>
<td>115032</td>
<td>17828</td>
<td>53516</td>
<td>572924</td>
<td>824965</td>
</tr>
<tr>
<td>balance-scale</td>
<td>25865</td>
<td>6602</td>
<td>5630</td>
<td>40782</td>
<td>530705</td>
</tr>
<tr>
<td>glass</td>
<td>22901</td>
<td>9315</td>
<td>8075</td>
<td>50113</td>
<td>620718</td>
</tr>
<tr>
<td>hepatitis</td>
<td>8205</td>
<td>3313</td>
<td>3644</td>
<td>15664</td>
<td>127486</td>
</tr>
<tr>
<td>haberman</td>
<td>7474</td>
<td>3370</td>
<td>2321</td>
<td>11180</td>
<td>153196</td>
</tr>
</tbody>
</table>

The combination of evidences above shows that the implementation of BMA based on Naive Bayes can provide better prediction for some specific classification problem with relatively less time consuming such as vote, vowel and mushroom. And even if it can’t give better performance, it can still guarantee the accuracy is equal to baseline algorithm. through comparing the true error confidence interval table with experiment dataset table, we believe the accuracy improvement of BMA with respect to Naive Bayes classifier is not related to number of features, number of dataset, number of classes as well as missing values. The improvement is only related to how many of features in that classification problem is trivial or just noise.

As previous discussed, the hyper parameter of this implementation of BMA for NB is C, which is further expressed as shown in equation 38.

\[
C = H^{N+1}
\]  

(38)

where N is training data size and H is tuned parameter.

In this experiment,

\[
H \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}
\]  

(39)
The following plots shows the affection of tuning hyper parameter.

With the increasing of the Hyper-parameter value, the accuracy of prediction gradually grows at first period. But after passing the peak, it declines normally very fast. It is caused by over punishment of model with higher dimensions. The left hand side which shown as straight line means there is no affection that brought by hyper parameter yet. And, the right hand looks like gradient descent and each of the descent corresponding the prediction accuracy provided by models with slightly less features.

Figure 9: Example of Hyper Parameter Tuning Affection 1
However, even though there exist peak or platform for some classification problems, it is not guaranteed the hyper parameter with the best prediction accuracy can be selected by automatic hyper parameter tuning process in cross validation, because the validating dataset may not be able to represent the property of testing dataset as well as training dataset.

And, it is also worth to mention that the above cases is not always true, since the essence of this algorithm is based automatic feature selection. Therefore, if a classification problem whose features all make contributions, tuning hyper parameter to punish models with relative more features can only get worse result. Thus, there is no peak at all as shown below.

![Hyper Parameter Tuning Affection 2](image)

**Figure 10: Example of Hyper Parameter Tuning Affection 2**
5 Conclusion

This work is to implement a well-known model combination technology Bayesian Model Averaging into the classic and robust classifier Naive Bayes. The motivation is that, even though Bayesian model averaging is a commonly used approach to combine models and do automatic feature selection\textsuperscript{11} there are not much researches implementing it on Naive Bayes. In this work, we derive a new BMA for Naive Bayes classifier algorithm with new automatic feature selection mechanism. The time complexity of the algorithm is mathematically proved to be linear with respect to both number of features and number of data. Then, the algorithm is implemented through Java program and evaluated through several aspects.

The experiment results are quite near what we expected: the performance is better than original Naive Bayes algorithm and its time complexity is similar with Naive Bayes as well. The challenge is to improve this method to get better performance which is competitive with other linear classification models. According to the experiment results, it is clear that the implementation is very competitive on several cases such as text classification. For some classification problem like vote.arff or sick.arff, performance of BMA for Naive Bayes classifier is much better than traditional Naive Bayes classifier, and In other cases, however, the performances is just same with Naive Bayes classifier. Furthermore, this algorithm is faster than SVM and Logistic Regression and, for some cases, it also provides better accuracy than them.

The possible usage of this classifier is for those classification problems which include lots of useless features or noise features with relatively less dependence with class label. Text classification problem in this experiment is proved to be a potential field for which the classifier has its advantage, since automatic feature selection mechanism of the BMA on NB can play a role.

Hyper parameter tuning is very critical for this classifier. It controls the cost of hiring features for a model. However, because C is real number, its tuning range is very large, therefore, tuning every possible value of the hyper parameter C is costly and almost impossible. To roughly adjust this parameter, exponential adjusting point is used, which is $H^{N+1}$, where N is number of training data and $H \in 0.5, 0.6..1.5$. The advantage of this method is it can significantly reduce the parameter tuning time. As contrast, it is possible to ’miss’ some crucial value which may increase the performance accuracy.

\textsuperscript{11}some researchers are more likely to believe BMA is a feature selection tool rather than that of model combination, since it tend to give ”the best” model too much weights(almost 20times bigger, even if there is only 1% difference in accuracy for Logistic Regression.)
The most difficult period in my work is the time that I struggle to adjust hyperparameters automatically. The current solution is just a compromise between accuracy and time consuming. This compromise clearly limited the usage of this classifiers, since there is no guarantee to say this classifier can beat Naive Bayes in most circumstance with relatively higher computational cost. So how to find value of hyper parameters more efficiently will become the further work.
A Appendix

A.1 Notations

A.1.1 General Notations

1. BMA: Bayesian Model Averaging
2. NB: Naive Bayes
3. SVM: Support Vector Machine
4. LR: Logistic Regression
5. OOP: Object-oriented Programming

A.1.2 Notation In Equations

1. $y$: class label
2. $\vec{x}$: descriptive attributes of a data sample, it is a vector
3. $x_i$: descriptive attributes of data sample $i$ in training data set
4. $x_{ik}$: descriptive attribute $k$ of data sample $i$ in data set
5. $M$: model sets
6. $m$: a single model
7. $m_k$: the number $k$ model in model set
8. $m_{ki}$: the number $i$ feature in model $k$
9. $F$: model sets which is notated by feature selection perspective.
10. $\vec{f}$ feature set vector
11. $f_k$: number $k$ feature in explanation features. The value rank is in 0 and 1.
12. $d_i$: number $i$ data sample in data set
Bibliography

DOUGHERTY, J., KOHAVI, R., AND SAHAMI, M. 1995. Supervised and unsupervised discretization of continuous features. (p. 18)


NG, A. Cs229 lecture notes part IV: Generalized linear models.

NG, A. Cs229 lecture notes part V: Support Vector Machines.


YUAN, Z. AND YANG, Y. 2004. Combining linear regression models: when and how?

Zhang, H. 2004. The optimality of naive bayes. A A 1, 2, 3. (p. 8)