AI Communications 27 (2014) 133–141
DOI 10.3233/AIC-130588
IOS Press

Fine tuning the Naïve Bayesian learning algorithm

Khalil El Hindi
Computer Science Department, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia
E-mail: khindi@ksu.edu.sa

Abstract. This work augments the Naïve Bayesian learning algorithm with a second training phase in an attempt to improve its classification accuracy. This is achieved by finding more accurate estimations of the needed probability terms. This approach helps in dealing with the problem of the lack of training data. Unlike many previous approaches that deal with this problem, the proposed method is an eager method in the sense that it does most of the work during training and, therefore, it does not increase classification time. It consists of two phases. In the first phase, the algorithm builds a classical Naïve Bayesian classifier. The second phase is a fine tuning phase. In this phase each training instance is classified, if it is misclassified, the probability values involved are fine tuned in such a way that increases the chances of correctly classifying this instance in the next round. Our results show significant improvement in the classification accuracy of many benchmark data sets, compared to the classical Naïve Bayesian, and two other methods that improve on the Naïve Bayesian algorithm.

Keywords: Machine learning, Naïve Bayesian classification, improving estimated probabilities

1. Introduction

The Naïve Bayesian approach to learning uses the Bayesian rule for conditional probabilities, to classify new instances. The method calculates the probability of each class value, given the values of an instance's attributes. The class with the maximum probability is then taken as the predicted class of the new instance. The training set is used to estimate all needed probabilities. Given a new unclassified instance of the form $\langle a_1, a_2, \ldots, a_n \rangle$, the predicted class for this instance, $c_{predicted}$, is computed as

$$c_{predicted} = \operatorname*{argmax}_{c \in C} \frac{p(a_1, a_2, \ldots, a_n|c) \cdot p(c)}{p(a_1, a_2, \ldots, a_n)}, \tag{1}$$

where: C is a vector of all class attribute values. $p(c)$ is the probability of class c. $p(a_1, a_2, \ldots, a_n)$ is the probability that attributes $1, 2, \ldots, n$ will take the values $a_1, a_2, \ldots, a_n$, respectively. $p(a_1, a_2, \ldots, a_n|c)$ is the probability that attributes $1, 2, \ldots, n$ will take the values $a_1, a_2, \ldots, a_n$ given that the instance is of class c.

Given a certain instance (e.g., the instance to be classified), the probability $p(a_1, a_2, \ldots, a_n)$ is the same; therefore, formula (1) can be simplified as follows:

$$c_{predicted} = \operatorname*{argmax}_{c \in C} p(a_1, a_2, \ldots, a_n|c) \cdot p(c). \tag{2}$$

The approach, naively, assumes that attribute values are conditionally independent given the class value. In other words, it assumes that

$$p(a_1, a_2, \ldots, a_n|c) = \prod_{i=1}^{n} p(a_i|c). \tag{3}$$

Therefore, Eq. (2) can be rewritten as

$$c_{predicted} = \operatorname*{argmax}_{c \in C} p(c) \cdot \prod_{i=1}^{n} p(a_i|c). \tag{4}$$

The classification accuracy of the Naïve Bayesian was shown to be comparable, in many domains, to that of many more complex techniques, such as neural networks and decision trees [15]. However, its classification accuracy degrades in domains where the independence assumption is not satisfied [16]. Also, its accuracy is highly dependent on finding a good estimation

of needed probability terms; namely $p(a_i|c)$ and $p(c)$. This is hard to achieve in domains where little training data is available. This work addresses this problem by augmenting the Naïve Bayesian algorithm with a fine tuning phase. The aim of this phase is to fine tune the probability values, so that a Naïve Bayesian classifier achieves better classification accuracy.

In Section 2, we review related work. Section 3 presents the fine tuning method. In Section 4, we discuss the experimental work and analyze results. Section 5 is the conclusion section.

2. Related work

Most attempts to improve the NB algorithm can be categorized into two main groups. The first group focuses on the independence assumption problem, while the second group deals with the problem of the lack of training data. The optimal solution for the first problem is to build a Bayesian network [17]. However, this problem is NP-hard [3]. Therefore, other approximate methods have been developed. Some approaches attempt to make the problem more tractable by imposing a restriction on the structure of the network [4, 8,9,16,20]. Other approaches try to ease the independence assumption by using a feature selection method [13,15,16], expecting that the selected features will be truly conditionally independent. Other approaches try to ease the assumption by using a subset of instances to build the NB classifier. The subset can be chosen using a decision tree [14] or the kNN [7]. The expectation here is that the independence assumption is more likely to be satisfied by a small subset of training instances than by the entire set of training instances.

The second group, which has received less attention than the first group, deals with the problem of the lack of training data. If the training set is too small to reflect the underlying instance distribution, using it gives unreliable estimations of the probability terms used by the NB, which negatively affects the classification accuracy. Some approaches were developed to deal with this problem by cloning instances [10,12]. These methods are lazy methods in the sense that they build the NB classifier during classification. This leads to a relatively high classification time complexity [11]. In [10] a method called LNB (for Lazy Naïve Bayesian), is presented. The method clones some instances based on their dissimilarity to a new instance, while [12] uses a greedy search algorithm to determine the instances to clone. It is worth mentioning that although the cloning techniques [10] and [12] were developed for improving the ranking performance of NB, their classification performance significantly outperforms NB in the same way [11]. Therefore, we chose to compare our method with one of them, namely, with LNB [10], which is the one used in [11].

The method proposed in this work falls in the second group as it deals with the problem of the lack of training data. However, it is an eager method as it builds an NB classifier at training time and the classifier is therefore independent of the instance that will be classified. This also means that the classification time of new instances is the same as the classification time of NB. The method does not clone data, but it uses the training data, in a second stage, to fine tune the estimated probability values. We also compare the proposed method with ENB [13] which uses genetic algorithms to select a subset of the features hoping that the independence assumption would be satisfied by the selected features. In the next subsections, we review each of these methods in more details.

2.1. Lazy Naïve Bayesian (*LNB*)

As mentioned before, to deal with the problem of lack of data, LNB [10] expands a training set by cloning instances. It clones instances based on their dissimilarity to the instance that needs to be classified, then an NB classifier is built based on the expanded data set. This process takes place at the classification time, and thus it increases the classification time [11]. Dissimilarity of two instances x and y is measured using the distance function defined below[1]

$$s(x,y) = \sum_{i=1}^{n} \delta\big(a_i, a_i(y)\big), \tag{5}$$

where:

$$\delta\big(a_i(x), a_i(y)\big) = \begin{cases} 1, & a_i(x) = a_i(y), \\ 0, & a_i(x) \neq a_i(y). \end{cases} \tag{6}$$

LNB measures the distance between a new instance and every instance in the training set and expands the training set by adding $s(x,y)$ clones of the training instance to the training set. This takes place after prepro-

[1] Contrary to what distance metrics used in kNN do, the function actually measures the dissimilarity between instances, as it considers the distance between two nominal values 1 if they are equals and 0 otherwise.

cessing the training set by replacing missing values by the most common attribute value and discretizing ordinal attributes using the filters provided in Weka [19]. It is worth mentioning that although LNB performs better when missing values are replaced, the performance of the FTNB method proposed in this work slightly degrades as a result of this step. The experimental results reported in Section 4 of LNB were obtained after we replaced missing values by the most common attribute values, while the results of FTNB were obtained without doing this replacement.

2.2. Evolutionary NB (ENB)

ENB [13] uses a genetic algorithm to select a set of features that improves the classification accuracy of NB and then builds an NB classifier using the selected features. The method outperforms NB, as it tends to select a subset of the features that satisfy the conditional independence assumption.

3. A fine tuning stage

The method proposed in this work to improve the classification accuracy of the NB algorithm, consists of two learning stages. In the first stage, the training set is used in the usual way to estimate the probability terms needed by the NB algorithm. In the second stage, we use the training set once again in an attempt to fine tune these probability terms. If a training instance is mistakenly classified by the NB classifier, this means the predicted class, $c_{predicted}$, has a higher computed probability than the instance's actual class, c_{actual}, given the instance's attribute values. Therefore, we need to increase the value of the probability terms involved in computing the probability of the actual class and decrease those involved in computing the probability of the predicted class. In other words, we need to increase the probability terms $p(c_{actual})$ and $p(a_i|c_{actual})$ for each attribute value, a_i. We should, also, decrease the probability of the mistakenly predicted class, $c_{predicted}$, by decreasing the probability terms that contributed to this error, namely, the terms $p(c_{predicted})$ and $p(a_i|c_{predicted})$, for each attribute value a_i.

Accurate estimates for the terms $p(a_i|c_{actual})$ and $p(a_i|c_{predicted})$ may not be obtainable using a small training set. The problem is less severe for the terms $p(c_{actual})$ and $p(c_{predicted})$. This is because estimating $p(a_i|c_{actual})$ and $p(a_i|c_{predicted})$ are based on fewer instances than estimating $p(c_{actual})$ and $p(c_{predicted})$. Therefore, it is natural to expect that the best improvement would come as a result of finding better estimates for the terms $p(a_i|c_{actual})$ and $p(a_i|c_{predicted})$ compared to the terms $p(c_{predicted})$ and $p(c_{actual})$. We will, therefore, focus on fine tuning $p(a_i|c_{actual})$ and $p(a_i|c_{predicted})$. We used similar methods to fine tune $p(c_{predicted})$ and $p(c_{actual})$, but our experimental results showed no or little improvement as a result of fine tuning these terms. Therefore, we will not take fine tuning these terms into account in this work.

The proposed method makes gradual changes (updates) to these terms by iteratively computing an update step size $\delta_{(t+1)}$ for each term and then adding it to the previous term's value using formula (7).

$$p_{t+1}(a_i|class) = p_t(a_i|class) + \delta_{t+1}(a_i, class), \tag{7}$$

where, t is the cycle number. This process is repeated so long as the training classification accuracy (i.e., classification accuracy computed using the training

Algorithm 1. FTNB(Training_instances)

Phase 1: Use Training_instances to estimate the values of each probability term used by the NB algorithm
Phase 2:
$t = 0$
while training classification accuracy improves do
  for each training instance, inst, do
    let c_{actual} **be the actual class of inst**
    let $c_{predicted} = classify(inst)$
    if $c_{predicted} <> c_{actual}$ **then**
      compute classification error
      for for each attribute value, a_i**, of inst do do**
        compute $\delta_{(t+1)}(a_i, c_{actual})$
        let $p_{(t+1)}(a_i|c_{actual}) = p_t(a_i|c_{actual}) + \delta_{(t+1)}(a_i, c_{actual})$
        compute $\delta_{(t+1)}(a_i, c_{predicted})$
        let $p_{(t+1)}(a_i|c_{predicted}) = p_t(a_i|c_{predicted}) - \delta_{(t+1)}(a_i, c_{predicted})$
        let $t = t + 1$
      end for
    end if
  end for
end while

data) keeps improving. Algorithm 1, which we call FTNB (Fine Tuning NB), shows the details.

3.1. Updating $p(a_i|c_{actual})$

Since, the training instance was incorrectly classified (i.e., c_{actual} got a probability that is too low), we need to increase the value of $p(a_i|c_{actual})$ for each attribute value a_i that occurred in the training instance. Clearly, the amount of update, δ_i, needs to be proportional to the amount of error. We calculate the error as follows

$$error = \big|P(c_{actual}) - P(c_{predicted})\big|. \tag{8}$$

We calculate $P(c_o)$ of a class c_o using the formula

$$p(c_o) = \frac{p(c_o|inst_{train})}{\Sigma_k^m p(c_k|inst_{train})}, \tag{9}$$

where $inst_{train}$ is a training instance (or vector) of the form $\langle a_1, a2, \ldots, a_n\rangle$, and m is the number of classes (the number of class attribute values) and $p(c_k|inst_{train})$ is the conditional probability of class c_k given the training instance, computed as

$$p(c_k|inst_{train}) = p(c_k) \cdot \prod_i^n p(a_i|c_k), \tag{10}$$

where, n is the number of attributes (excluding the class attribute) and a_i is the value of the ith attribute of $inst_{train}$.

Equation (9) ensures that the sum of the probability of all classes is 1. It also helps increase the computed probability values and thus the size of the update step.

The amount of update should also be large for small probability values and small for large probability values. This is because small probability values are more likely to be responsible for the misclassification than large probability values. We achieve this by making the update step proportional to the difference between $p(a_i|c_{actual})$ and the probability of the attribute value with the maximum probability given c_{actual}. This is achieved by making the size of the update proportional to

$$\alpha \cdot p(max_i|c_{actual}) - p(a_i|c_{actual}), \tag{11}$$

where, max_i is the value of the ith attribute with the maximum probability given c_{actual}. This formula ensures that the farther away $p(a_i|c_{actual})$ is from $p(max_i|c_{actual})$, the larger the update step is. α is a constant, greater than or equal to 1, which is used to control the size of the update step for the term $p(a_i|c_{actual})$ relative to its distance from $p(max_i|c_{actual})$. If it is set to 1, it makes the size of the update step for the $p(max_i|c_{actual})$ zero, and if it is set to a value greater than 1, this makes the size of the update step for $p(a_{max}|c_{actual})$ greater than zero.

The following equation takes all of the above factors into consideration to determine the size of the update step

$$\begin{aligned}&\delta_{t+1}(a_i, c_{actual})\\ &\quad = \eta \cdot \big(\alpha \cdot p(max_i|c_{actual})\\ &\qquad\qquad - p(a_i|c_{actual})\big) \cdot error,\end{aligned} \tag{12}$$

where η is a constant, between 0 and 1, which determines the learning rate.

3.2. Updating $p(a_i|c_{predicted})$

As discussed before, the probability terms responsible for having $c_{predicted}$ get higher probability than c_{actual} need to be decreased. Again, the size of the decrement step needs to be proportional to the size of error. Also, since we blame probability terms with high values more than probability terms with small values, the size of the decrement should be proportional to the value of the term $p(a_i|c_{predicted})$. This is achieved by making the size of the update proportional to

$$\beta \cdot p(a_i|c_{predicted}) - p(min_i|c_{predicted}), \tag{13}$$

where, min_i is the value of the ith attribute with the minimum probability given $c_{predicted}$. Thus, the farther away $p(a_i|c_{predicted})$ is from $p(min_i|c_{predicted})$, the larger the update step is. β is a constant, that is greater or equal to 1, which is used to control the size of the update step for the term $p(a_i|c_{predicted})$ by making it relative to its distance from $p(min_i|c_{predicted})$. The larger the value of $beta$ is, the larger the update step. If it is set to 1, it makes the size of the update step for the $p(min_i|c_{predicted})$ zero, and if it is set to a value larger than 1, it makes the size of the update step for $p(min_i|c_{predicted})$ greater than zero.

The update step size is determined using the formula

$$\begin{aligned}&\delta_{t+1}(a_i, c_{predicted})\\ &\quad = -\eta \cdot \big(\beta \cdot p(a_i|c_{predicted})\\ &\qquad\qquad - p(min_i|c_{predicted})\big) \cdot error.\end{aligned} \tag{14}$$

4. Experimental results

The described fine tuning algorithm was tested on 47 data sets obtained from the UCI repository [2]. All ordinal attributes were discretized using Fayyad et al.'s [6] supervised discretization method as implemented in Weka [19]. Missing values were simply ignored by the FTNB and were replaced by the most common values for LNB and ENB as was described in Section 2. Ten-fold cross validation was used in all experiments.

Table 1 shows the result of comparing FTNB with NB, LNB, and ENB. The table shows the average classification accuracy over the 10 folds for each of these methods and for each training set. The last two rows of each column show, firstly, the number of data sets on which the corresponding method achieved better results, and secondly, the number of significantly better results. A paired t-test with a confidence level of 95% was used to determine if each difference was statistically significant. The better results are highlighted in bold in the table and the significantly better results are highlighted in bold and underlined.

The overall average of the NB, FTNB, LNB and ENB are 80.53%, 82.98%, 82.53% and 81.15%, re-

Table 1
The results of comparing FTNB with NB, LNB and ENB

Data set	NB vs. FTNB		#FT Epochs	FTNB vs. LNB		FTNB vs. ENB	
	NB (%)	FTNB (%)		FTNB (%)	LNB (%)	FTNB (%)	ENB (%)
Anneal	96.99	**98.22**	3.9	**98.22**	97.66	**98.22**	97.44
Anneal.ORIG	79.06	**97.66**	3.7	**97.66**	93.32	**97.66**	93.54
Arrhythmia	54.2	**72.57**	3.9	72.57	**75.22**	**72.57**	54.2
Autos	60	**64.88**	7	64.88	**61.46**	**64.88**	58.54
Breast-cancer	**73.08**	67.83	2	67.83	**72.73**	67.83	**72.38**
Breast-w	**97.28**	95.99	2	95.99	**97.42**	95.99	**96.85**
Bridges-version1	59.81	**61.68**	3.3	**61.68**	60.75	**61.68**	59.81
Bridges-version2	57.94	**60.75**	2.5	**60.75**	59.81	**60.75**	53.27
Car	73.15	**84.72**	8.6	**84.72**	73.15	**84.72**	73.21
Colic	72.01	**80.16**	3.5	80.16	**80.98**	**80.16**	72.83
Colic.orig	70.65	**75.54**	6	**75.54**	72.01	**75.54**	72.28
Credit-a	**84.06**	82.17	3.1	82.17	**84.35**	82.17	**83.48**
Credit-g	**75.5**	72.8	2.1	72.8	**75.5**	72.8	**75.5**
Cylinder-bands	69.44	**71.85**	5.7	**71.85**	70.19	**71.85**	69.07
Dermatology	97.27	**97.54**	2.3	97.54	**98.36**	**97.54**	95.63
Diabetes	77.34	**77.47**	2.3	77.47	**77.73**	**77.47**	77.08
Flags	**60.31**	57.73	6	57.73	**58.76**	57.73	**59.28**
Haberman	**74.18**	70.26	2	70.26	**74.18**	70.26	**74.18**
Heart-c	**85.15**	84.16	2.2	84.16	84.16	**84.16**	83.17
Heart-h	**83.33**	80.27	2	80.27	**83.33**	80.27	**83.33**
Heart-statlog	**83.7**	82.59	2.9	82.59	**84.07**	**82.59**	82.22
Hepatitis	83.23	**87.1**	2.2	**87.1**	86.45	**87.1**	85.81
Hypothyroid	92.95	**99.26**	5.9	**99.26**	99.05	**99.26**	98.44
Ionosphere	90.88	**92.31**	4	**92.31**	91.74	**92.31**	90.6
Iris	95.33	**96**	2	**96**	95.33	**96**	94.67
Letter	74.11	**77.05**	8	**77.05**	77.04	**77.05**	74.34
Liver-disorders	54.78	**63.19**	2.1	**63.19**	54.78	**63.19**	54.78
Lung-cancer	81.25	81.25	2.4	81.25	81.25	81.25	81.25
Lymph	83.78	**85.81**	3.2	**85.81**	85.14	**85.81**	83.11
Mushroom	94.33	**99.68**	4.9	**99.68**	97.24	**99.68**	98.08
Nursery	81.37	**84.32**	3.1	**84.32**	82.05	**84.32**	81.37
Optdigits	92.12	**94.04**	11.6	**94.04**	92.51	**94.04**	91.87
Pendigits	87.92	**94.71**	10.1	**94.71**	91.03	**94.71**	88.25

Table 1
(*Continued*)

Data set	NB vs. FTNB		#FT Epochs	FTNB vs. LNB		FTNB vs. ENB	
	NB (%)	FTNB (%)		FTNB (%)	LNB (%)	FTNB (%)	ENB (%)
Segment	91.77	**93.9**	4.3	**93.9**	93.12	**93.9**	92.47
Sick	96.85	**96.98**	2.1	96.98	**97.08**	96.98	**97.08**
Solar-flare-1	91.64	**95.98**	4.3	**95.98**	93.5	**95.98**	94.12
Solar-flare-2	97	**98.87**	3.5	**98.87**	97.09	**98.87**	97.75
Sonar	**82.21**	78.37	2	78.37	**82.69**	78.37	**80.77**
Spambase	**89.35**	77.57	2	77.57	**89.5**	77.57	**89.52**
Splice	**94.92**	91.63	2	91.63	**95.45**	91.63	**95.02**
Sponge	51.32	**69.74**	3.4	**69.74**	60.53	69.74	69.74
Trains	70	70	2	70	70	**70**	60
Vehicle	63.36	**68.91**	4.9	**68.91**	65.37	**68.91**	64.54
Vote	89.43	**93.33**	6.2	**93.33**	90.8	**93.33**	91.95
Waveform-5000	80.64	**83.14**	2	**83.14**	81.18	**83.14**	81.12
Wine	98.88	98.88	2	98.88	98.88	98.88	98.88
Zoo	91.09	91.09	2	91.09	**95.05**	91.09	91.09
Average	80.53	82.98	3.81	82.98	82.53	82.98	81.15
#Better	12	31		28	16	32	11
#Sig Better	3	23		10	3	16	2

spectively. It is obvious, therefore, that in terms of the average classification accuracy, all methods have better classification accuracy than NB. However, FTNB achieves the best classification accuracy compared to LNB and ENB. Moreover, when we compare FTNB against LNB in terms of the number of data sets on which each method achieves better and significantly better results, FTNB emerges as a winner on 28 data sets of which 10 are significantly better results. It loses to LNB on 16 data sets of which it achieves significantly worse results on only 3 data sets. Comparing FTNB with ENB shows that it achieves better results on 32 data sets of which 16 are significantly better results. On the other hand, ENB achieves better results on 11 data sets of which only 2 are significantly better results. Compared to NB, FTNB achieves better results on 31 data sets of which 23 are significantly better results, while it achieves worse results on 12 data sets, only 3 of which are significantly lower results.

It is important to remember that contrary to LNB, FTNB is an eager method and performs the fining tuning stage at the learning phase and therefore it adds no extra overhead on the classification phase (or time). The fine turning process is an efficient process, as the number of epochs required to fine tune NB (shown in the fourth column of the table) is relatively small. On average the algorithm requires 3.81 fine tuning epochs. The results of Table 1 were achieved using $\eta = 0.01$ and $\alpha = \beta = 2$. These are the values that gave the best results.

The maximum average number of epochs (of the 10 folds) is 11.6 which was required to fine tune the Optidigits data set. The fact that the algorithm required a small number of fine tuning epochs shows that the algorithm is efficient and it indicates that the fine tuning process is very delicate. This is why the value of the learning rate that gave us the best results is relatively small (0.01). This is not surprising given the fact that the conditional probabilities NB uses are usually very small, which is mostly due to the lack of training data.

It is worth mentioning that a batch version of FTNB, in which no updates take place until the end of each epochs, was tested but did not give as good results as the online version discussed above.

In fact FTNB and each of LNB and ENB are not opposing methods and, therefore, FTNB, can be used in conjunction with LNB or ENB. With LNB it can be used at classification time to fine tune the probabilities estimated using the expanded data set. While with ENB, FTNB can be used to fine tune the classifier built using the features selected by ENB. Table 2, shows the results of comparing LNB with a combined method in which LNB followed by FTNB is used. We denote this method in the table by LNB&FTNB. The table also shows the results of comparing ENB and a combined method in which ENB followed by FTNB are used. We

Table 2
The results of comparing LNB and ENB with their fine tuned counterparts: LNB&FTNB and ENB&FTNB

Data set	LNB vs LNB&FTNB			ENB vs ENB&FTNB		
	LNB	LNB&FTNB	Epochs	ENB	ENB&FTNB	Epochs
Anneal	97.66	**98.22**	386.5	97.44	**97.77**	3.2
Anneal.ORIG	93.32	93.32	199.2	93.54	**98.66**	4.7
Arrhythmia	**75.22**	70.35	135.2	54.2	**71.02**	5.2
Autos	61.46	**61.95**	63.8	58.54	**63.41**	5
Breast-cancer	**72.73**	70.28	62.5	**72.38**	67.48	2.3
Breast-w	**97.42**	96.57	164	96.85	**97.28**	2.1
Bridges-version-1	60.75	**62.62**	25.7	**59.81**	58.88	2.7
Bridges-version-2	59.81	59.81	24.4	53.27	**54.21**	2.6
Car	73.15	**78.24**	1183.4	73.21	**79.51**	6.6
Colic	80.98	**83.15**	213.6	72.83	**80.98**	3.6
Colic.orig	72.01	**72.28**	185.8	72.28	**74.73**	5.1
Credit-a	**84.35**	81.59	161.4	**83.48**	83.33	2.6
Credit-g	**75.5**	74.5	210.4	**75.5**	75.2	2
Cylinder-bands	70.19	**70.56**	254.3	69.07	**71.85**	5.8
Dermatology	98.36	98.36	82.2	95.63	**96.99**	2.2
Diabetes	77.73	**78.52**	265.4	77.08	**78.52**	2.4
Flags	**58.76**	57.73	112.9	59.28	**59.79**	5.1
Haberman	**74.18**	73.86	73.3	**74.18**	71.57	2.2
Heart-c	**84.16**	83.83	72.1	83.17	**83.83**	2.3
Heart-h	83.33	**85.03**	98	**83.33**	76.87	2
Heart-statlog	**84.07**	83.7	64.4	82.22	**82.96**	2.5
Hepatitis	86.45	**87.1**	34.5	85.81	**86.45**	2.3
Hypothyroid	99.05	**99.28**	1894.8	98.44	**98.49**	4.4
Ionosphere	91.74	**93.16**	142.3	90.6	**90.88**	3.6
Iris	95.33	**96**	30	94.67	**97.33**	2
Letter	77.04	**79.11**	24,523	74.34	**77.79**	14.2
Liver-disorders	54.78	**63.19**	72.4	54.78	**63.19**	2.4
Lung-cancer	81.25	81.25	6.4	**81.25**	78.13	2
Lymph	85.14	85.14	36.4	83.11	**85.14**	2.3
Mushroom	97.24	**99.69**	6830.1	98.08	**99.51**	8.2
Nursery	82.05	**85.76**	4054.5	81.37	**84.83**	3
Optdigits	92.51	**94.02**	6823.7	91.87	**93.51**	10.3
Pendigits	91.03	**95.21**	23,813.2	88.25	**94.29**	13.9
Segment	93.12	**94.81**	1898.5	92.47	**94.59**	5.9
Sick	**97.08**	96.92	754.5	**97.08**	96.58	2.1
Solar-flare-1	93.5	**96.28**	142.9	94.12	**95.67**	4
Solar-flare-2	97.09	**98.78**	726.6	97.75	**99.06**	4.4
Sonar	**82.69**	79.81	46.1	80.77	**81.73**	2.5
Spambase	**89.5**	82.74	2043	**89.52**	80.24	2.7
Splice	**95.45**	93.67	1675.6	**95.02**	91.66	3.8
Sponge	60.53	**67.11**	30	69.74	**78.95**	2.3
Trains	70	70	2	60	**70**	2
Vehicle	65.37	**66.31**	372.5	64.54	**67.02**	5.1
Vote	90.8	**92.64**	291.4	91.95	**92.41**	4.5
Waveform-5000	81.18	**83.92**	1000.3	81.12	**82.54**	2.1

Table 2
(*Continued*)

Data set	LNB vs LNB&FTNB			ENB vs ENB&FTNB		
	LNB	LNB&FTNB	Epochs	ENB	ENB&FTNB	Epochs
Wine	98.88	98.88	35.6	98.88	**99.44**	2
Zoo	95.05	95.05	20.2	**91.09**	90.1	2
Average	82.53	83.2	1730.62	81.15	82.86	3.96
#Better	13	26		11	36	
#Sig Better	3	13		1	15	

denote this method by ENB&FTNB. The table shows that using FTNB with each of LNB and ENB gives better results than using either of them alone.

Clearly, FTNB improves on the classification accuracy of LNB and ENB in terms of the classification average and the number of data sets on which the combined methods achieve better and significantly better results. The average classification accuracy of LNB is 82.53%, while the average classification accuracy of LNB&FTNB is 83.20%, an increase of 0.67%. Moreover, the combined method achieves better results on 26 data sets of which 13 are significantly better results. On the other hand it achieves worse results on 13 data sets of which only 3 are significantly worse results. This clearly shows that using FTNB to fine tune LNB leads to better classification accuracy than using LNB alone.

However, this came at a cost in terms of a dramatic increase in the number of fine tuning epochs. The average number of the fine tuning epochs is 1730.62, which is far larger than the average number of the fine tuning epochs required by FTNB to fine tune NB. This increase is partly because we had to decrease the learning rate from 0.01 (the learning rate we used with FTNB) to 0.001 with LNB&FTNB in order to get better result. Since, LNB and the following fine tuning process are done at classification time this overhead on the classification time may not be acceptable in all application domains.

Similarly, ENB&FTNB achieved better results than ENB alone, which shows that FTNB is also effective in fine tuning classifiers built using the selected features. The average classification accuracy of ENB and ENB&FTNB are 81.15% and 82.86%, respectively, an increase of 1.71%. Moreover, ENB&FTNB gave better results than ENB on 36 data sets 15 of them are significantly better results, while it gave worse results on 11 data sets, only 1 of them is significantly worse.

5. Conclusion

This work augmented the Naïve Bayesian algorithm with a fine tuning phase in an attempt to improve the classification accuracy in domains where insufficient training data is available to reflect the underlying distribution. This phase updates the probability values involved in misclassifying a training instance in such a way that makes it more likely that this training instance will be correctly classified in the future. Our empirical comparison using 47 data sets shows that the fine tuning phase improved the average classification accuracy of NB on 31 data sets of which 23 were statistically significant improvements (at 95% confidence level), while, the Naïve Bayesian algorithm achieved better results on 12 data sets, 3 of which were significantly better results.

The fine tuning method was also compared with LNB which clones training data at classification time to deal with the problem of lack of training data. FTNB proved to be more effective than LNB in this regard. Also this improvement in classification accuracy did not increase classification time. Moreover, FTNB proved to be effective in fine tuning LNB, but this came at a huge cost in terms of the high number of fine tuning epochs.

We also compared FTNB with ENB which is a method that uses genetic algorithms to select a subset of the features and use them to build an NB classifier. Again, FTNB proved to be not only more effective than ENB in improving the classification accuracy of NB, but also proved to be effective in fine tuning ENB classifiers.

We expect that similar fine tuning methods can improve the classification accuracy of instance based learners [1], that use distance measures that involve conditional probability terms (e.g., [5,18]) similar to those used by NB. This is a subject for future research.

Acknowledgements

This work was supported by the Research Center of College of Computer and Information Sciences, King Saud University. The author is grateful for this support.

References

[1] D.W. Aha, D. Kibler and M.K. Albert, Instance-based learning algorithms, *Machine Learning* **6**(1) (1991), 37–66.

[2] C. Blake and C. Merz, UCI repository of machine learning databases [online], Department of Information and Computer Science, University of California, Irvine, CA, 1998, available at: http://www.ics.uci.edu/˜mlearn/MLRepository.html.

[3] D.M. Chickering, Learning Bayesian networks is np-complete, in: *Learning from Data: Artificial Intelligence and Statistics V*, Springer-Verlag, 1996, pp. 121–130.

[4] C. Chow and C. Liu, Approximating discrete probability distributions with dependence trees, *IEEE Trans. Inf. Theor.* **14**(3) 1968.

[5] K. El Hindi, Specific-class distance measures for nominal attributes, *AI Communications* (2013), to appear.

[6] U. Fayyad and K. Irani, Multi-interval discretization of continuous values attributes for classification learning, in: *Proceedings of 13th International Joint Conference on Artificial Intelligence*, 1993, pp. 1022–1027.

[7] E. Frank, M. Hall and B. Pfahringer, Locally weighted naive Bayes, in: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2003.

[8] N. Friedman, D. Geiger and M. Goldszmidt, Bayesian network classifiers, *Machine Learning* **29**(2,3) (1997), 131–163.

[9] L. Jiang, Z. Cai, D. Wang and H. Zhang, Improving tree augmented naive Bayes for class probability estimation, *Knowledge-Based Systems* **26** (2012), 239–245.

[10] L. Jiang and Y. Guo, Learning lazy naive Bayesian classifiers for ranking, in: *17th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 05*, 2005.

[11] L. Jiang, D. Wang, Z. Cai and X. Yan, Survey of improving naive Bayes for classification, in: *Advanced Data Mining and Applications*, Springer, 2007, pp. 134–145.

[12] L. Jiang and H. Zhang, Learning instance greedily cloning naive Bayes for ranking, in: *Fifth IEEE International Conference on Data Mining*, IEEE, 2005.

[13] L. Jiang, H. Zhang, Z. Cai and J. Su, Evolutional naive Bayes, in: *Proceedings of the 1st International Symposium on Intelligent Computation and Its Applications*, ISICA, China University of Geosciences Press, 2005, pp. 344–350.

[14] R. Kohavi, Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid, in: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1996, pp. 202–207.

[15] P. Langley and S. Sage, Induction of selective Bayesian classifiers, in: *The Tenth Conference on Uncertainty in Artificial Intelligence*, 1994.

[16] M.A. Palacios-Alonso, C.A. Brizuela and L.E. Sucar, Evolutionary learning of dynamic naive Bayesian classifiers, *Journal of Automated Reasoning* **45**(1) (2010), 21–37.

[17] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[18] D.R. Wilson and T.R. Martinez, Improved heterogeneous distance functions, 1997, arXiv preprint available at: cs/9701101.

[19] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2005.

[20] H. Zhang and C.X. Ling, An improved learning algorithm for augmented naive Bayes, in: *Bayes, Pacific-Asia Conference on Knowledge Discovery and Data Mining*, LNCS, Springer Verlag, 2001, pp. 581–586.