King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

A noise tolerant fine tuning algorithm for the Naïve Bayesian learning algorithm

Khalil El Hindi *

Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Received 20 November 2013; revised 13 February 2014; accepted 13 March 2014
Available online 19 May 2014

KEYWORDS

Machine learning;
Naive Bayesian learning;
Noise handling;
Overfitting;
Instance weighing

Abstract This work improves on the FTNB algorithm to make it more tolerant to noise. The FTNB algorithm augments the Naïve Bayesian (NB) learning algorithm with a fine-tuning stage in an attempt to find better estimations of the probability terms involved. The fine-tuning stage has proved to be effective in improving the classification accuracy of the NB; however, it makes the NB algorithm more sensitive to noise in a training set. This work presents several modifications of the fine tuning stage to make it more tolerant to noise. Our empirical results using 47 data sets indicate that the proposed methods greatly enhance the algorithm tolerance to noise. Furthermore, one of the proposed methods improved the performance of the fine tuning method on many noise-free data sets.

1. Introduction

The Naïve Bayesian learning algorithm is a simple machine learning algorithm that has proved to be comparable, in terms of its classification accuracy in many domains to many more complex algorithms, such as neural networks and decision trees (Langley and Sage, 1994). To classify a new instance, the algorithm uses the Bayesian rule for conditional probabilities to calculate the conditional probability of each class value and takes the class with the maximum probability as the predicted class. The algorithm uses the training data to estimate all the required probability values.

* Tel.: +966 583378881.
E-mail address: khindi@ksu.edu.sa.

Given a new instance of the form $< a_1, a_2, \cdots, a_n >$, the predicted class for this instance, $c_{predicted}$, is computed as

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

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, ..., n will take the values $a_1, a_2, \cdots, a_n$, respectively.
$p(a_1, a_2, \cdots, a_n|c)$ is the probability that attributes 1, 2, ..., n will take the values $a_1, a_2, \cdots, a_n$, 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 for all class values; therefore, formula 1 can be simplified as follows:

$$c_{predicted} = \underset{c \in C}{\operatorname{argmax}}\, p(a_1, a_2, \cdots, a_n|c) \cdot p(c) \qquad (2)$$

1319-1578 © 2014 King Saud University. Production and hosting by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jksuci.2014.03.008

To allow for computational tractability, the algorithm makes the Naïve assumption that all the attribute values are conditionally independent given the class value; therefore,

$$p(a_1, a_2, \cdots, a_n|c) = \prod_i p(a_i|c) \quad (3)$$

Thus, Eq. (2) can be rewritten as

$$c_{predicted} = \underset{c \in C}{\text{argmax}} p(c) \cdot \prod_i p(a_i|c) \quad (4)$$

The classification accuracy of NB degrades in domains where the independence assumption is violated (Friedman et al., 1997). The classification accuracy also degrades if a training set is too small to provide an accurate estimation of the required probability terms. Most methods for improving the classification accuracy of NB, (e.g., Friedman et al., 1997; Chickering, 1996; Zhang and Ling, 2001; Jiang et al., 2005; Palacios-Alonso et al., 2008), focus on the independence assumption problem. Some methods, however, were also proposed to address the problem of the lack of training data, (e.g., Jiang and Guo, 2005; Jiang and Zhang, 2005; El Hindi, 2014). The methods proposed by Jiang and Guo, 2005; Jiang and Zhang, 2005 clones some instances to increase the size of the training data while the method proposed by El Hindi, 2014 augments the NB with a second fine-tuning stage to determine better estimations of the needed probability terms.

Although the empirical results presented in El Hindi, 2014 indicate that the extra fine-tuning stage substantially increases the average classification accuracy in many domains, this work indicates that it degrades the classification accuracy in domains where the training data contain some noisy instances. This sensitivity to noise is an important issue because real life data are rarely free of noise. Several methods were developed for dealing with noisy instances. Some methods, (e.g., Muhlenbach et al., 2004; El Hindi and Al-Akhras, 2011; Sanchez et al., 2003; Jiang and Zhou, 2004; Koplowitz and Brown, 1981), attempt to identify and eliminate noisy instances. These methods are called noise filtering or data editing methods. We believe that eliminating the suspected noisy instances may be error prone because some noise-free instances may be eliminated. In this work, we do not eliminate noisy instances; we just assign them small weights to make their effect during the fine-tuning stage small. Of course, the proposed methods may incorrectly reduce the weight of a correct instance, but unless they reduce it to zero, they would still make use of that instance during training. This makes the proposed methods more robust than data editing methods.

The methods were tested using 47 benchmark data sets that were obtained for the UCI repository for machine learning (Blake, 1998). All the ordinal attributes were discretized using the method of Fayyad amd Irani (1993), as implemented in Witten and Frank (2005).

Section 2 reviews the fine-tuning algorithms and discusses the effect of noise on the classification accuracy of these algorithms. Section 3 proposes a method for dealing with noise. Section 4 presents our empirical results. Section 5 is the conclusion section.

2. Fine tuning the Naïve Bayesian (FTNB) algorithm

In an attempt to find better estimations for the probability terms used by the NB algorithm, the FTNB (fine-tuning NB) algorithm (El Hindi, 2014) augments the NB algorithm with a fine-tuning stage. In the first stage, the training set is used in the usual manner to estimate the probability terms required to build an NB classifier. In the second stage, the training set is used once again to fine tune these probability terms. In this stage, some probability values are modified in such a way that makes the algorithm more accurate in classifying the training instances. 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 actual class of the instance, c_{actual}, given the attribute values of the instance. Therefore, the FTNB algorithm increases the values of the probability terms involved in computing the probability of the actual class and decreases the probability terms involved in estimating the probability of the predicted class, $c_{predicted}$. Namely, the algorithm increases $p(a_i|c_{actual})$ and decreases $p(a_i|c_{predicted})$ for each attribute value a_i. This process is gradually performed using the formula

$$p_{t+1}(a_i|class) = p_t(a_i|class) + \delta_{t+1}(a_i, class) \quad (5)$$

where t is the cycle number, and δ_{t+1} is an update step. This process is repeated so long as the training classification accuracy (i.e., classification accuracy computed using the training data) continues to improve. Fig. 1 shows the details of the FTNB algorithm.The size of the update step, δ_i, must be proportional to the amount of error, which is computed as follows

$$error = |P(c_{actual}) - P(c_{predicted})| \quad (6)$$

$P(c_o)$ is calculated using the formula

$$P(c_o) = \frac{p(c_o|inst_{train})}{\sum_k^m p(c_k|inst_{train})} \quad (7)$$

where, $inst_{train}$ is a training instance (or vector) of the form $< a_1, a_2, \cdots, a_i, \cdots, a_n >$, and m is the number of classes (the number of class attribute values) and

$$p(c_k|inst_{train}) = \prod_i^n p(a_i|c_k) \cdot p(c_k) \quad (8)$$

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

To increase $p(a_i|c_{actual})$, the FTNB algorithm makes the size of the update step large for small probability values and small for large probability values. This task is required because small probability values are more likely to be responsible for the misclassification than large probability values. This task is performed using the formula

$$\delta_{t+1}(a_i, c_{actual}) = \eta \cdot (\alpha \cdot p(max_i|c_{actual}) - p(a_i|c_{actual})) \cdot error \quad (9)$$

where:

η is a constant, between 0 and 1, which determines the learning rate.
max_i is the value of the ith attribute, with the maximum probability given c_{actual}.

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 $p(max_i|c_{actual})$.

In contrast, to decrease $p(a_i|c_{predicted})$, the FTNB algorithm makes the size of the decrement large for large probability terms and small for small probability terms (see El Hindi

```
Algorithm FTNB(Training_instances)
phase 1
  Use Training_instances to build a Naïve Bayesian classifier
phase 2
let $t = 0$
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 $c_{predicted} <> c_{actual}$        //misclassified instance
      compute classification error for inst
      for each attribute value, $a_i$, of inst 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 $t = t + 1$
      endfor
    endif
  endfor
end while
```

Figure 1 The FTNB algorithm.

(2014) for more details). This task is achieved using the formula

$$\delta_{t+1}(a_i, c_{predicted}) = -\eta \cdot (\beta \cdot p(a_i|c_{predicted}) - p(min_i|c_{predicted})) \cdot error \quad (10)$$

where:

β is a constant that is greater or equal to 1.
min_i is the value of the ith attribute that has the minimum conditional probability, given $c_{predicted}$.

In all the experiments reported in this work, η was set to 0.005, while α and β were set to 2.

2.1. The effect of noise

Although the empirical results presented in El Hindi, 2014 indicate that the fine-tuning stage substantially improves the classification accuracy in many domains, it may have a negative effect on the classification accuracy if the training data contain some noisy instances. It is well known that noise is one of the main causes of the overfitting problem (Mitchell, 1997). Overfitting simply means that while the classification accuracy of a classifier measured on the training set may be high, a classifier may perform poorly on unseen instances. Because the FTNB algorithm continues to modify the values of the probability terms so long as the training classification accuracy continues to improve, this may produce a classifier that overfits a training set.

To study the effect of noise on the classification accuracy of the FTNB algorithm, some artificial noise was inserted in 47 benchmark data sets obtained from the UCI machine learning repository (Blake, 1998). Noise was inserted in training sets by re-placing some randomly chosen values of the class attribute values with other random class values. Noise was inserted only in the training set, leaving the test data set unchanged. Several sets of experiments were performed using different ratios of noise of 5%, 10%, 15% and 20%. Due to the random nature of the process, each experiment was repeated 5 times, performing a 10-fold cross validation each time. All the ordinal attributes were discretized using the discretization method of Fayyad amd Irani (1993), as implemented in Witten and Frank (2005).

Table 1 summarizes the results. Each figure in the table is the average of the 10-fold experiments repeated 5 times. The better results are highlighted in bold, and the significantly better results are highlighted in bold and underlined. A paired t-test with a confidence level of 95% was used to determine if each difference was statistically significant. The last two rows in the table present the number of data sets on which the methods achieved better accuracy and significantly better accuracy.

The table shows that at 0% noise, the FTNB algorithm outperforms the NB algorithm in terms of the average accuracy and in terms of the number of data sets on which it achieves better results. By 0% noise, we do not mean that the data sets are noise free because some of them are not; we simply mean that we did not deliberately insert any artificial noise in the data set.

At 0% noise, the average accuracy of the FTNB is 83.08%, while the average accuracy of NB is 81.11%. Additionally, the number of data sets on which FTNB achieves better results is

26, of which 18 are significantly better results at 95% confidence level while NB achieves better results on 15 data sets, of which only 3 results are significantly better results. However, the situation gradually changes in favor of the NB algorithm as we increase the noise ratio. At 5% noise, NB outperforms FTNB with respect to the average accuracy and the number of data sets on which each algorithm achieves significantly better results. Furthermore, the gap continues to widen between the two algorithms as we increase the noise ratio. The average classification accuracy of the FTNB algorithm decreased from 83.08% at 0% noise down to 70.92% at 20% noise, a decrease of 12.16%. In contrast, the average accuracy of NB decreased from 81.11% at 0% noise down to 79.43% at 20% noise, a decrease of only 1.68%.

At 20% noise ratio, the gap in the average accuracy between FTNB and NB becomes 8.51% in favor of NB. The number of data sets on which FTNB achieves better results, at 20% noise, is 7, with only 5 of them exhibiting significantly better results, while the number of data sets on which NB achieves better results is 38, 33 of which are significantly better results.

These results indicate that the NB algorithm is a noise-tolerant learning algorithm and that the FTNB algorithm sacrifices this good advantage of NB. The next section proposes 3 simple modifications to the weight update formula used by the FTNB algorithm to make it more noise tolerant.

3. Making FTNB more noise tolerant

To make the FTNB algorithm more tolerant of noise, this work proposes making the size of the update step proportional to our confidence that the misclassified instance is indeed not a noisy instance. If our confidence is low, so should be the size of the update step; however, if our confidence is high, the update step should be large. This task can be achieved by simply multiplying the update step by a confidence factor or an instance weight that reflects our confidence that the misclassified instance under consideration is not a noisy instance. Thus, the weight update equations become

$$\delta_{t+1}(a_i, c_{actual}) = \eta \cdot (\alpha \cdot p(max_i|c_{actual}) - p(a_i|c_{actual})) \cdot error * CF \tag{11}$$

and

$$\delta_{t+1}(a_i, c_{predicted}) = -\eta \cdot (\beta \cdot p(a_i|c_{predicted}) - p(min_i|c_{predicted})) \cdot error * \mathrm{CF} \tag{12}$$

where CF is the confidence factor.

This work uses three methods to compute the confidence factor (or instance weight): the first is based on conditional probabilities, the second is based on the neighboring instances of the misclassified instance, and the third combines the first two.

3.1. Probability-based confidence factor

This method is based on Bayes' conditional probability rule itself as used by the Naïve Bayesian algorithm. Given a misclassified instance, *inst*, of class, c, the conditional probability of c given the remaining attribute values of *inst* reflects our confidence that *inst* is not a noisy instance. Therefore, the CF is computed using the formula

$$CF = p(c|a_1, a_2, \cdots, a_n)$$

The probability of c given the remaining attribute values, a_1, a_2, $\cdots$, a_n, can be computed using Bayes' conditional probability rule, which is defined as

$$p(c|a_1, a_2, \cdots, a_n) = \frac{p(a_1, a_2, \cdots, a_n|c) \cdot p(c)}{p(a_1, a_2, \cdots, a_n)} \tag{13}$$

where:

$p(c)$ is the probability of class c.
$p(a_1, a_2, \cdots, a_n)$ is the probability that attributes 1, 2, ..., n will take the values $a_1, a_2, \cdots, a_n$, respectively.
$p(a_1, a_2, \cdots, a_n|c)$ is the probability that attributes 1, 2, ..., n will take the values $a_1, a_2, \cdots, a_n$, given that the instance is of class c.

To simplify the calculations and make the application of the rule computationally feasible, this work will, following the Naïve Bayesian algorithm, make the naïve assumption that all attribute values are conditionally independent given the class value. In other words, this work assumes that

$$p(a_1, a_2, \cdots, a_n|c) = \prod_i p(a_i|c) \tag{14}$$

3.2. Neighboring-instance based confidence factor

The second method that is used to measure the confidence factor is based on the k most similar instances of the misclassified instance. The intuition is simple if a small number of these instances have the same class as the instance, in this case, the instance is most likely a noisy instance, and the value of the confidence factor should be small. However, if this number is large, then the value of the confidence factor should be large. In other words, the larger this number is, the more confident we are that the instance is actually not a noisy instance. Therefore, the confidence factor is computed using the formula

$$CF = \frac{\text{number of similar instances of the same class}}{k} \tag{15}$$

where k is a constant representing the number of neighbors. In all the empirical experiments reported in this work, k was set to 5.

To measure the similarity between instances, this work uses the DISDM function (El Hindi, 2013), defined as follows

$$dist(\boldsymbol{x}, \boldsymbol{y}) = \sqrt[2]{\sum_{a=1}^{m} DISCDM_a^2(x_a, y_a)} \tag{16}$$

Where

- $\boldsymbol{x}$ and $\boldsymbol{y}$ are two vectors; typically one vector is a training instance and the other is a vector that needs to be classified.
- x_a and y_a are the values of attribute a in the vectors x and y, respectively.
- m is the number of attributes.

DISCDM is defined as follows

$$\boldsymbol{DISCDM}(\boldsymbol{val_{new}}, \boldsymbol{val_{train}}) = 1 - p(val_{new}|class_{train}) \tag{17}$$

where $\boldsymbol{val_{new}}$ and $\boldsymbol{val_{train}}$ are the attribute value of the misclassified instance and training instance against which we measure the distance, respectively, and $class_{train}$ is the class of the training instance.

Table 1 The classification accuracy of the NB and FTNB at different noise ratios.

	0% noise		5% noise		10% noise		15% noise		20% noise	
	NB	FTNB	NB	FTNB	NB	FTNB	NB	FTNB	NB	FTNB
Anneal	97.00	**97.55**	**96.88**	89.96	**96.42**	81.67	**96.68**	76.21	**96.41**	72.99
Anneal. ORIG	79.07	**97.10**	85.15	**88.91**	**86.53**	82.48	**87.33**	78.44	**87.40**	71.83
Arrhythmia	54.22	**72.79**	54.22	**70.75**	54.22	**70.26**	54.22	**67.95**	54.22	**62.12**
Autos	59.95	**62.93**	59.48	**62.28**	58.68	**60.61**	**58.11**	57.71	**58.39**	56.20
Breast-cancer	**73.13**	67.83	**72.43**	66.93	**71.99**	65.46	**69.96**	63.23	**68.96**	59.91
Breast-w	**97.28**	96.14	**97.48**	96.86	**97.43**	94.74	**97.34**	89.93	**97.34**	76.43
Bridges_version1	**59.00**	**60.75**	60.32	**60.90**	**60.95**	59.67	**59.37**	58.84	**60.10**	58.89
Bridges_version2	57.64	**58.88**	54.86	**57.13**	55.46	**56.34**	54.27	**55.56**	52.59	**53.17**
Car	73.14	**80.96**	73.55	**77.62**	74.21	**74.39**	**72.86**	68.78	**74.19**	67.91
Colic	72.02	**80.71**	70.65	**79.03**	69.84	**76.47**	68.81	**74.64**	68.69	**71.49**
Colic.orig	70.65	**75.54**	69.77	**70.74**	68.14	**70.53**	66.19	**67.27**	**66.96**	63.55
Credit-a	**84.06**	82.03	**84.00**	81.91	**83.94**	80.29	**84.14**	78.41	**84.20**	74.52
Credit-g	75.50	74.10	**75.08**	72.20	**73.98**	70.26	**73.62**	68.36	**73.88**	67.48
Cylinder-bands	69.44	**71.30**	69.04	**69.19**	**68.48**	66.93	**69.07**	65.56	**67.85**	63.93
Dermatology	97.27	97.27	**97.45**	97.07	**97.66**	96.02	**97.00**	94.29	**96.90**	91.20
Diabetes	**77.34**	77.08	76.87	**77.24**	**76.22**	74.58	**76.20**	73.17	**76.17**	71.04
Flags	**60.13**	55.15	**59.72**	56.40	**58.92**	54.64	**59.85**	53.72	**58.41**	52.91
Haberman	**74.19**	70.92	**73.69**	70.43	**73.69**	67.70	**73.68**	64.54	**72.24**	62.23
Heart-c	**85.20**	83.83	**84.79**	83.46	**84.66**	83.33	**84.26**	81.80	**84.07**	77.20
Heart-h	**83.21**	79.93	**83.29**	80.75	**83.09**	81.87	**83.50**	83.28	**82.61**	77.09
Heart-statlog	**83.70**	83.33	**83.63**	83.33	**83.19**	83.04	**83.63**	82.89	**83.11**	81.41
Hepatitis	83.00	**87.74**	83.39	**86.10**	**89.79**	84.66	80.67	**82.24**	78.25	**81.09**
Hypothyroid	92.95	**99.26**	**90.97**	88.93	82.35	**84.67**	**88.67**	71.02	**87.35**	67.41
Ionosphere	90.88	**92.31**	**90.37**	89.68	**89.97**	87.92	**90.26**	82.67	**90.14**	75.93
Iris	95.33	95.33	95.60	**95.87**	**95.87**	95.47	**95.60**	95.47	**95.87**	95.47
Letter	74.11	**77.16**	73.44	**75.93**	73.02	**74.14**	**72.50**	71.89	**72.09**	68.20
Liver-disorders	54.84	**63.19**	53.10	**63.22**	55.36	**63.22**	58.38	**63.22**	57.73	**63.22**
Lung-cancer	**80.00**	78.13	78.12	**80.78**	**75.11**	73.64	**73.94**	70.78	**72.10**	69.09
Lymph	83.81	**85.81**	84.47	**85.13**	**84.32**	84.04	**83.37**	80.67	**81.63**	77.81
Mushroom	94.33	**99.61**	**92.16**	74.31	**91.48**	69.41	**91.16**	74.48	**91.00**	75.79
Nursery	81.37	**85.00**	**81.56**	64.48	**81.42**	50.02	**81.64**	40.97	**81.70**	35.42
Optdigits	92.12	**93.93**	**91.61**	90.72	**91.46**	87.72	**91.12**	85.09	**90.86**	78.54
Pendigits	87.92	**94.65**	87.17	**89.85**	**86.60**	85.32	**86.08**	83.90	**85.58**	77.60
Segment	91.77	**93.85**	90.53	**92.08**	89.63	**90.88**	**89.13**	88.33	**88.16**	86.15
Sick	**96.85**	96.79	93.83	**93.89**	90.98	**92.31**	88.41	**89.52**	85.75	**86.22**
Solar-flare_1	91.62	**95.98**	**91.74**	89.83	**92.54**	83.82	**92.80**	78.82	**91.38**	69.88
Solar-flare_2	97.00	**98.87**	**96.32**	87.19	**97.05**	77.82	**97.43**	69.74	**97.19**	64.09
Sonar	**82.36**	79.33	82.72	**83.09**	**82.91**	81.67	**82.70**	79.42	**84.33**	74.84
Spambase	**89.35**	80.87	**89.18**	76.06	**89.38**	59.12	**89.51**	58.43	**89.28**	53.10
Splice	**94.92**	93.01	**93.72**	80.93	**92.54**	75.97	**91.38**	74.55	**90.53**	74.06
Trains	70.00	70.00	70.00	70.00	70.00	70.00	62.00	62.00	62.00	62.00
Vehicle	63.36	**67.85**	62.76	**65.98**	62.76	**65.39**	62.64	**64.16**	62.48	**64.18**
Vote	89.43	**93.10**	89.47	**92.22**	**89.89**	88.90	**89.57**	84.85	**89.43**	81.13
Waveform-5000	80.64	**83.94**	80.02	**82.36**	**79.73**	77.00	**79.40**	69.14	**79.06**	62.90
Wine	98.86	98.88	**98.10**	97.54	**97.99**	97.55	**97.44**	96.55	**97.66**	97.10
Zoo	91.00	91.09	90.41	**90.61**	87.81	87.81	86.41	86.41	**87.61**	87.81
Average	**81.11**	**83.08**	**80.72**	**80.00**	**80.38**	**76.95**	**79.83**	**74.11**	**79.43**	**70.92**
#Sig better	**3**	**19**	**18**	**15**	**25**	**9**	**30**	**5**	**33**	**5**
Better	**15**	**26**	**21**	**24**	**32**	**12**	**36**	**8**	**38**	**7**

This work uses the DISCDM function because it has proved to be more tolerant of noise than other more known distance functions (see El Hindi (2013) for more details).

3.3. A combined method for computing the confidence factor

Note that the probability-based method for computing the confidence factor is based on global information in the sense that it uses the entire training set in computing the conditional probability and thus the confidence factor, while the second method is based on local information provided by the k neighboring instances. The third method for computing the confidence factor makes use of the two types of information by combining the first and second method into one method that takes their product according to the formula

$$CF = p(c|a_1, a_2, \cdots, a_n) * \frac{\text{number of similar instances of the same class}}{k} \tag{18}$$

4. Empirical results

This section discusses the results of the empirical experiments obtained using the methods presented in the previous section. The section compares the different methods with the original NB algorithm. The tables present the classification accuracy of the NB algorithm and each one of the different methods for calculating the confidence factor at different noise ratios: 0%, 5%, 10%, 15%, and 20%. Each pair of columns presents the results of NB compared with the modified FTNB at a certain noise ratio.

4.1. The results of the probability-based method

Table 2 summarizes the results obtained using the NB algorithm and the FTNB algorithm modified to take into account a confidence factor computed using the probability-based method presented in Section 3.1. The modified FTNB algorithm is called PFTNB. For the 0% noise ratio, Tables 1 and 2 indicate that PFTNB achieves smaller average classification accuracy than does FTNB. PFTNB achieves 82.54% average classification accuracy, while FTNB achieves 83.08% average classification accuracy.

Also, for the 0% noise ratio, there is a decrease in the number of data sets on which PFTNB achieves significantly better results than NB compared with the number for FTNB. FTNB achieves significantly better results than NB on 19 data sets (see Table 1), while PFTNB achieves significantly better results than NB on 14 data sets (see Table 2). However, PFTNB achieves significantly worse results than NB on 2 data sets, while FTNB achieves significantly worse results than NB on 3 data sets.

For the 5% noise case, the average classification accuracy of PFTNB is higher than the average classification accuracy of both NB and FTNB. The average classification accuracies of PFTNB, NB, and FTNB for the 5% noise case are 81.80%, 80.76% and 80%, respectively. Moreover, PFTNB achieves for the 5% noise ratio significantly better results than NB on 20 data sets and significantly worse results on 9 data sets, while FTNB achieves significantly better results than NB for the 5% noise ratio on 15 data sets and significantly worse results on 18 data sets (see Table 1).

Furthermore, PFTNB continued to achieve better results than NB as we increased the noise ratio to 10%. However, for the 15% noise ratio, the situation turned in favor of NB and continued to do so at the 20% noise ratio. However, for the 20% noise ratio, the drop in the average classification accuracy of PFTNB was less severe than the drop of the average classification accuracy of FTNB. For the 20% noise ratio, the average classification accuracy gap between the FTNB (without a confidence factor) and NB was 8.51% in favor of NB (see Table 1), while at the same noise ratio, the gap between PFTNB and NB is only 1.49% in favor of NB. Additionally, for the 20% noise ratio, PFTNB still achieves significantly better results than NB on 15 data sets and significantly worse results on 17 data sets, while FTNB achieves for the 20% noise ratio significantly better results than NB on 5 data sets and significantly worse results on 33 data sets (see Table 1).

All of this clearly shows that the probability-based confidence factor improves the ability of the FTNB method in dealing with noise. Our experiments indicate that FTNB slightly outperforms PFTNB only on noise-free data, while PFTNB outperforms FTNB when the data sets contain 5%, 10%, 15%, or 20% of noise. Additionally, PFTNB outperforms NB for the 0%, 5% and 10% noise ratios, while NB outperforms PFTNB for the 15% and 20% noise ratios.

4.2. The results of the neighborhood-based method

A similar set of experiments was conducted to test the effectiveness of the neighborhood method (NFTNB), as presented in Section 3.2. Table 3 summarizes the results. The table shows that this method outperforms FTNB, even for the 0% noise ratio, with respect to the average classification accuracy. The average classification accuracy of the NFTNB method is 83.55%, while the average classification accuracy of FTNB is 83.08%, and the average classification accuracy of NB is 81.11%. Although both methods, FTNB and NFTNB, achieve for the 0% noise ratio significantly better results than NB on 19 data sets, NFTNB achieves significantly worse results than NB on only 2 data sets, while FTNB achieves significantly worse results than NB on 3 data sets.

NFTNB continues to achieve better results than NB for the 5% and 10% noise ratios; however, for the 15% and 20% noise ratios, the situation changes in favor of the NB. For the 20% noise ratio, the average classification accuracies of NB and NFTNB are 79.49% and 77.92%, respectively. Also for the 20% noise ratio, NFTNB achieves significantly better results than NB on 11 data sets and significantly worse results on 16 data sets.

Compared with PFTNB, NFTNB achieves better results for the 0%, 5%, and 10% noise ratios and almost equal results for the 15% noise ratio. However, the situation changes for the 20% case in favor of PFTNB. At this noise ratio, each of PFTNB and NFTNB achieves an average classification accuracy of 78.12% and 77.92%, respectively. Furthermore, for the 20% noise ratio, PFTNB achieves significantly better results than NB on 15 data sets, while NFTNB achieves significantly better results than NB on only 11 data sets.

4.3. The results of the combined method

Table 4 summarizes the results of a set of similar experiments conducted using the combined method (CFTNB) that combines the probability-based and the neighborhood-based methods into one method, as discussed in Section 3.3.

For the 0% noise ratio, the CFTNB method exhibits significantly better results than the NB method on 16 data sets and worse results on 2 data sets (see Table 4), while the neighborhood-based method (NFTNB) gave significantly better results than the NB method on 19 data sets and significantly worse results on 2 data sets (see Table 3). Additionally, the NFTNB method achieves an average classification accuracy for the 0% noise ratio of 83.55%, which is better than the 82.78% average classification accuracy of CFTNB.

However, CFTNB achieves better results than all the other methods with respect to the number of data sets on which the methods achieve significantly better results than NB at all noise ratios. This is true despite the fact that for the 5% noise ratio, CFTNB and NFTNB achieve significantly better results on 22 data sets because CFNTB achieves significantly worse

Table 2 The results of the probability-based method compared to the NB method at different noise ratios.

Data set	0% noise		5% noise		10% noise		15% noise		20% noise	
	NB	PFTNB	NB	PFTNB	NB	PFTNB	NB	PFTNB	NB	PFTNB
Anneal	**97.28**	96.29	**97.48**	96.51	**97.40**	96.03	**97.34**	95.77	**97.34**	93.60
Anneal. ORIG	96.99	**97.11**	96.71	**97.20**	96.95	**97.20**	**96.42**	95.77	**96.70**	94.15
Arrhythmia	79.07	**91.54**	85.02	**91.12**	87.33	**90.07**	88.13	**88.64**	**87.08**	84.77
Autos	54.22	**65.73**	54.22	**65.91**	54.22	**65.42**	54.22	**66.66**	54.22	**66.34**
Breast-cancer	**59.96**	59.44	**59.21**	59.08	**58.39**	58.36	**58.01**	57.89	**57.62**	57.30
Breast-w	**73.12**	72.38	**72.00**	71.20	**70.60**	68.81	**70.66**	69.09	**69.47**	65.18
Bridges_version1	59.07	59.07	60.68	**60.88**	**58.58**	58.20	60.70	**60.71**	**59.59**	59.40
Bridges_version2	57.66	**59.57**	57.22	**58.31**	53.21	**53.75**	54.57	**54.75**	52.26	**53.54**
Car	73.14	**76.61**	73.36	**77.24**	74.24	**77.17**	74.11	**77.09**	74.20	**76.27**
Colic	72.02	**83.70**	70.60	**82.23**	69.68	**80.61**	68.92	**78.37**	68.43	**77.07**
Colic.orig	70.65	**73.61**	69.82	**72.64**	68.31	**71.50**	66.68	**68.14**	66.41	**68.19**
Credit-a	**84.06**	81.59	**83.77**	80.96	**83.77**	81.04	**84.06**	80.81	**83.91**	78.00
Credit-g	75.50	75.50	**75.12**	75.10	**74.52**	74.16	**74.08**	72.58	**73.62**	70.94
Cylinder-bands	69.44	**71.11**	68.81	**70.07**	68.33	**69.48**	**67.15**	65.26	**68.56**	64.26
Dermatology	97.27	97.27	**97.55**	97.39	97.55	97.55	**97.22**	97.01	**96.95**	96.73
Diabetes	77.34	**77.60**	77.18	**77.60**	76.77	**77.71**	**76.20**	76.09	**75.41**	74.73
Flags	**60.15**	57.59	**59.74**	59.26	**58.79**	58.63	**59.32**	58.62	**59.78**	58.86
Haberman	74.19	74.19	73.42	**73.49**	**72.57**	71.27	**74.52**	72.61	**71.35**	69.44
Heart-c	**85.20**	84.20	**84.73**	84.26	**84.53**	84.26	84.13	**84.19**	**84.59**	84.57
Heart-h	**83.22**	81.86	**83.43**	81.80	**83.43**	82.01	**83.21**	82.76	83.63	**82.90**
Heart-statlog	83.70	83.70	83.11	**83.26**	**83.26**	82.67	83.19	**83.41**	**83.26**	82.30
Hepatitis	83.02	**85.64**	82.10	**85.95**	80.06	**85.95**	80.83	**85.01**	79.04	**84.50**
Hypothyroid	92.95	**96.95**	90.85	**97.54**	89.74	**95.02**	88.79	**92.34**	87.62	**90.59**
Ionosphere	90.88	**91.15**	90.37	**90.76**	90.76	90.76	**90.60**	88.71	**90.48**	85.81
Iris	95.33	95.33	**95.33**	95.20	96.40	**96.53**	95.47	95.47	95.60	95.60
Letter	74.11	**74.75**	73.51	**74.80**	72.93	**75.29**	72.55	**75.54**	72.12	**75.24**
Liver-disorders	54.84	**63.22**	54.32	**63.22**	57.15	**63.10**	55.20	**63.22**	56.20	**63.22**
Lung-cancer	**80.11**	76.80	**78.79**	78.12	**78.79**	77.61	**73.94**	73.28	69.39	**71.44**
Lymph	83.81	**85.81**	84.60	84.60	83.81	**84.07**	83.13	**84.20**	82.30	**83.22**
Mushroom	94.33	**99.50**	**92.13**	78.72	**91.44**	65.77	**91.17**	64.84	**91.10**	63.98
Nursery	81.37	**83.50**	81.45	**83.50**	81.46	**81.54**	**81.75**	79.19	**81.69**	77.52
Optdigits	92.12	**93.20**	91.62	**92.28**	91.36	**91.87**	91.20	**91.76**	90.93	**91.14**
Pendigits	87.92	**93.17**	87.14	**91.86**	86.63	**91.10**	86.03	**90.22**	85.55	**89.46**
Segment	91.77	**93.20**	90.55	**91.38**	89.71	**91.06**	89.11	**90.39**	87.92	**90.13**
Sick	**96.85**	95.04	93.81	**96.77**	90.94	**95.79**	88.90	**93.54**	87.36	**91.09**
Solar-flare_1	91.62	**96.28**	91.99	**96.47**	93.61	**95.41**	**92.57**	91.78	**91.92**	86.74
Solar-flare_2	97.00	**98.97**	96.75	**98.69**	97.05	**98.12**	**96.75**	96.21	**97.17**	93.28
Sonar	**82.34**	79.96	**82.73**	80.49	**83.95**	81.16	**83.74**	79.64	**84.63**	78.64
Spambase	**89.35**	82.62	**89.28**	82.74	**89.39**	70.08	**89.44**	63.88	**89.46**	60.43
Splice	**94.92**	92.92	**93.84**	88.29	**92.62**	77.98	**91.47**	74.14	**90.04**	71.99
Trains	70.00	70.00	70.00	70.00	70.00	70.00	68.00	68.00	70.00	70.00
Vehicle	63.36	**64.30**	62.83	**63.59**	62.93	**64.21**	63.00	**64.52**	63.14	**64.80**
Vote	89.43	**90.33**	89.70	**90.69**	89.71	**90.75**	89.56	**90.19**	**89.43**	88.91
Waveform-5000	80.64	**84.44**	80.12	**84.38**	79.74	**83.89**	79.44	**82.49**	79.08	**81.96**
Wine	98.86	98.86	**98.20**	97.87	**98.21**	97.88	**98.00**	97.44	**97.66**	97.32
Zoo	91.01	91.01	**89.81**	89.61	**88.01**	87.81	88.21	88.21	**88.01**	87.81
Average	81.11	**82.54**	**80.76**	**81.80**	**80.41**	**80.41**	**80.04**	**79.36**	**79.61**	**78.12**
#Sig better	**2**	**14**	**9**	**20**	**12**	**19**	**15**	**14**	**17**	**15**
Better	**13**	**26**	**19**	**27**	**21**	**25**	**24**	**21**	**28**	**17**

results than NB on 5 data sets, while NFTNB achieves significantly worse results than NB on 8 data sets.

For the 10%, 15%, and 20% noise ratios, CFTNB outperforms PFTNB and NFTNB with respect to the average classification accuracy and number of data sets on which the methods achieve significantly better results than NB. In fact, CFTNB is the only method that outperforms NB for the 15% and 20% noise ratios (see Table 4). CFTNB outperforms NB even for the 20% noise ratio with respect to the average classification accuracy (79.84% compared to 79.59%). Also for the 20% noise ratio, CFTNB achieves significantly better results than NB on 18 data sets and significantly worse results on only 8 data sets. This result clearly indicates that the combined method makes the FTNB method more noise tolerant than each of its constituent methods individually.

4.4. Complexity concerns

In this section, we discuss the effect of the proposed methods on the execution time of the algorithm. There are two issues

Table 3 The results of the neighborhood-based method compared to the NB method at different noise ratios.

Data set	0% noise		5% noise		10% noise		15% noise		20% noise	
	NB	NFTNB	NB	NFTNB	NB	NFTNB	NB	NFTNB	NB	NFTNB
Anneal	97.00	**97.33**	**96.97**	96.73	**96.66**	93.74	**96.46**	89.38	**96.26**	85.76
Anneal. ORIG	79.07	**97.22**	85.08	**95.15**	86.59	**92.96**	88.91	**90.98**	86.71	**87.62**
Arrhythmia	54.22	**73.90**	54.22	**73.55**	54.22	**72.62**	54.22	**71.91**	54.22	**70.76**
Autos	59.95	**60.93**	59.66	**60.72**	59.01	**60.32**	55.52	**57.48**	58.02	**58.39**
Breast-cancer	**73.13**	70.32	**71.99**	70.22	**71.36**	69.88	**70.37**	65.79	**69.67**	65.11
Breast-w	**97.28**	97.14	**97.34**	96.97	**97.43**	97.08	**97.40**	97.00	**97.28**	96.34
Bridges_version1	**59.00**	58.00	**58.79**	58.59	**60.52**	59.95	**60.60**	60.43	**60.03**	59.80
Bridges_version2	57.64	**59.55**	**56.52**	57.08	54.20	**54.38**	**53.99**	52.71	51.05	50.51
Car	73.14	**80.43**	73.69	**80.11**	74.36	**78.42**	74.05	**76.11**	74.73	**76.50**
Colic	72.02	**83.69**	70.60	**81.84**	69.57	**81.42**	70.06	**79.79**	67.23	**77.78**
Colic.orig	70.65	**74.43**	68.90	**71.72**	66.63	**68.94**	67.28	**68.29**	65.49	**66.52**
Credit-a	**84.06**	82.75	**84.12**	82.78	**84.26**	82.35	**84.14**	81.83	**83.94**	79.94
Credit-g	75.50	75.50	**75.20**	74.60	**74.76**	73.78	**74.16**	71.88	**73.26**	70.40
Cylinder-bands	69.44	**70.74**	69.96	**70.89**	69.93	**70.96**	**68.15**	68.04	67.63	**68.30**
Dermatology	97.27	97.27	97.61	**97.72**	97.55	**97.66**	**97.17**	96.96	97.28	97.28
Diabetes	77.34	**78.00**	76.77	**78.20**	76.69	**77.26**	76.43	**76.87**	**76.43**	74.92
Flags	**60.13**	59.13	**59.93**	58.33	**58.40**	58.02	**59.03**	56.78	**57.45**	56.19
Haberman	**74.19**	73.87	**73.81**	72.77	**72.64**	72.25	**72.71**	70.36	**72.56**	71.07
Heart-c	**85.20**	84.87	**84.53**	83.80	**84.79**	84.13	**84.46**	84.19	**84.13**	83.60
Heart-h	**83.21**	82.90	**83.08**	82.44	**83.15**	82.50	**83.08**	82.83	**83.49**	83.17
Heart-statlog	**83.70**	83.33	83.78	**84.07**	**83.33**	82.59	**83.26**	82.96	**83.63**	83.26
Hepatitis	83.00	**88.29**	83.14	**87.89**	82.36	**85.14**	80.03	**85.66**	78.06	**82.25**
Hypothyroid	92.95	**98.94**	90.92	**96.10**	89.90	**91.29**	**89.02**	87.28	**87.77**	84.10
Ionosphere	90.88	**91.72**	90.37	**91.39**	90.20	**91.27**	90.20	**91.00**	**90.37**	88.94
Iris	95.33	95.33	95.47	95.47	**95.47**	95.20	**95.87**	95.60	96.27	96.27
Letter	74.11	**78.28**	73.49	**78.12**	72.90	**77.82**	72.56	**77.29**	72.08	**76.54**
Liver-disorders	54.84	**63.22**	54.62	**63.22**	57.27	**63.22**	54.58	**63.22**	58.38	**63.22**
Lung-cancer	**80.00**	76.67	**78.79**	79.45	74.81	**76.14**	**75.11**	72.46	**70.27**	69.60
Lymph	83.81	**85.81**	**85.01**	84.86	83.12	**83.25**	**84.05**	83.91	**80.98**	80.68
Mushroom	94.33	**99.57**	**92.17**	91.82	**91.39**	76.31	**91.31**	69.74	**91.09**	66.27
Nursery	81.37	**84.78**	81.44	**83.30**	**81.59**	79.10	**81.72**	76.04	**81.67**	70.42
Optdigits	92.12	**94.02**	91.69	**93.15**	91.34	**92.50**	91.06	**91.58**	**90.94**	90.90
Pendigits	87.92	**94.71**	87.22	**93.35**	86.63	**92.41**	86.04	**91.77**	85.57	**90.45**
Segment	91.77	**94.03**	90.49	**93.05**	89.70	**92.46**	89.16	**91.88**	88.22	**91.15**
Sick	96.85	**96.98**	93.60	**96.47**	90.53	**95.22**	88.87	**93.94**	87.62	**92.09**
Solar-flare_1	91.62	**96.28**	92.99	**95.84**	**93.30**	92.62	**93.44**	86.56	**92.69**	83.43
Solar-flare_2	97.00	**98.78**	96.72	**98.22**	**97.24**	95.57	**97.47**	90.41	**97.11**	83.64
Sonar	**82.36**	80.90	**82.82**	82.24	**83.86**	82.48	**83.86**	81.97	**83.94**	80.44
Spambase	**89.35**	86.25	89.16	**90.97**	**89.26**	84.22	**89.49**	77.29	**89.54**	69.04
Splice	**94.92**	93.89	**93.66**	87.73	**92.46**	77.99	**91.43**	73.29	**90.31**	71.61
Trains	70.00	70.00	70.00	70.00	70.00	70.00	52.00	52.00	68.00	68.00
Vehicle	63.36	**67.02**	63.19	**67.11**	62.88	**65.86**	62.69	**66.29**	62.29	**65.70**
Vote	89.43	**91.48**	89.52	**92.18**	89.33	**90.89**	89.61	**90.19**	**89.29**	88.58
Waveform-5000	80.64	**85.08**	80.19	**85.15**	79.73	**84.80**	79.32	**83.83**	79.07	**82.94**
Wine	98.86	98.86	**98.42**	98.31	**98.20**	98.09	**98.78**	98.55	**97.55**	97.44
Zoo	91.00	91.00	90.21	**90.41**	88.41	88.41	**88.41**	88.21	87.01	**87.21**
Average	**81.11**	**83.55**	**80.82**	**82.91**	**80.39**	**81.16**	**79.73**	**79.18**	**79.49**	**77.92**
#Sig better	**2**	**19**	**8**	**22**	**14**	**19**	**16**	**15**	**16**	**11**
Better	**13**	**27**	**15**	**29**	**20**	**24**	**27**	**18**	**27**	**17**

of concern here. The first issue is related to the effort required to compute the instance weights, and the second issue is the effect that the proposed methods will have on the number of cycles (or iterations) the fine-tuning algorithm takes before it converges.

Fortunately, computing the probability-based weights does not increase the computational cost of the algorithm because all the required probabilities are computed anyway to construct the Naïve Bayesian classifier. In contrast, computing the neighborhood-based weights requires O(n^2) effort because finding the *k* nearest neighbors for each instance requires O(n) effort.

Regarding the cost in terms of the number of training cycles the algorithm executes before it converges, Table 5 presents the average number of training cycles we obtained using the 47 data sets. The first observation is that the average number of cycles in all cases is relatively small, which indicates how delicate the fine-tuning process is.

The table also indicates that the FTNB without confidence factor requires on average a lower number of iterations and

Table 4 The results of the combined method compared to the NB method at different noise ratios.

Data set	0% noise		5% noise		10% noise		15% noise		20% noise	
	NB	CFTNB	NB	CFTNB	NB	CFTNB	NB	CFTNB	NB	CFTNB
Anneal	97.00	**97.33**	96.97	**97.53**	96.79	**97.51**	96.77	**97.15**	96.68	**96.71**
Anneal. ORIG	79.07	**88.42**	84.99	**90.31**	86.53	**89.80**	88.42	**90.42**	88.36	**89.74**
Arrhythmia	54.22	**65.73**	54.22	**65.24**	54.22	**65.60**	54.22	**65.24**	54.22	**65.41**
Autos	59.95	59.95	**59.96**	59.37	58.62	**58.80**	57.22	**58.19**	**58.21**	58.10
Breast-cancer	**73.13**	72.07	**72.14**	71.71	71.16	**71.64**	70.38	**70.71**	**69.16**	68.11
Breast-w	**97.28**	97.14	**97.34**	96.97	**97.45**	97.14	**97.48**	97.03	**97.28**	96.91
Bridges_version1	59.00	59.00	60.88	60.88	**60.56**	60.38	59.26	59.26	60.81	60.81
Bridges_version2	57.64	**59.55**	56.16	**56.55**	54.20	**54.40**	**51.67**	51.47	**51.05**	50.87
Car	73.14	**76.32**	73.29	**76.16**	73.55	**76.02**	73.39	**76.03**	74.24	**76.03**
Colic	72.02	**82.88**	70.71	**82.77**	68.87	**81.36**	68.87	**81.42**	68.81	**80.01**
Colic.orig	70.65	**73.63**	70.91	**73.95**	69.22	**71.23**	65.59	**69.33**	63.51	**67.81**
Credit-a	**84.06**	82.46	**84.09**	82.38	**84.67**	82.96	**84.35**	82.41	**84.46**	82.35
Credit-g	75.50	**76.00**	74.86	**75.06**	**75.04**	74.86	**74.26**	74.12	**73.54**	73.08
Cylinder-bands	69.44	**70.19**	69.11	**69.78**	68.63	**70.33**	**69.04**	68.89	**68.89**	68.48
Dermatology	97.27	97.27	**97.55**	97.50	97.50	**97.55**	97.61	97.61	97.11	**97.17**
Diabetes	77.34	**77.86**	76.92	**77.84**	76.45	**78.07**	76.77	**77.84**	75.73	**77.29**
Flags	**60.13**	59.63	**59.11**	58.71	**59.22**	58.93	**58.63**	58.32	**59.02**	58.41
Haberman	74.19	74.19	73.42	**73.75**	72.63	**73.09**	**72.90**	72.26	**71.65**	71.21
Heart-c	**85.20**	84.87	**84.60**	84.20	**84.99**	84.66	84.66	**84.80**	84.21	**84.27**
Heart-h	**83.21**	82.18	**82.81**	82.34	**83.43**	82.34	**83.43**	82.89	**83.09**	82.28
Heart-statlog	83.70	83.70	83.04	**83.11**	**83.19**	82.96	83.70	**83.78**	**82.74**	81.56
Hepatitis	83.00	**88.29**	82.50	**85.94**	81.72	**86.58**	79.71	**85.28**	79.92	**84.88**
Hypothyroid	92.95	**97.24**	90.86	**97.17**	89.87	**97.54**	88.76	**97.15**	87.61	**95.74**
Ionosphere	90.88	**91.44**	90.43	**91.38**	90.03	**91.05**	**90.26**	90.20	90.54	**90.88**
Iris	95.33	95.33	95.47	**95.33**	95.60	**95.47**	**95.73**	95.60	95.60	95.60
Letter	74.11	**75.49**	73.44	**75.01**	72.98	**74.79**	72.56	**74.66**	72.13	**74.27**
Liver-disorders	54.84	**63.22**	54.32	**62.19**	55.74	**61.16**	54.99	**62.19**	55.14	**60.22**
Lung-cancer	**80.00**	76.67	78.12	**78.79**	**75.47**	74.15	71.29	71.29	71.74	**72.41**
Lymph	83.81	**85.81**	84.60	**84.87**	83.53	**83.80**	**83.65**	83.38	82.99	**83.52**
Mushroom	94.33	**99.51**	92.09	**95.59**	**91.41**	84.76	**91.31**	74.85	**91.10**	66.58
Nursery	81.37	**83.48**	81.55	**83.47**	81.51	**82.66**	81.51	**81.85**	81.77	**82.14**
Optdigits	92.12	**92.81**	91.59	**92.50**	91.34	**92.19**	91.10	**91.87**	90.91	**91.68**
Pendigits	87.92	**93.06**	87.15	**92.55**	86.54	**92.27**	86.01	**91.85**	85.60	**91.46**
Segment	91.77	**92.42**	90.42	**91.08**	89.71	**90.72**	89.05	**90.40**	88.36	**89.99**
Sick	**96.85**	95.73	93.38	**96.36**	91.33	**96.47**	89.26	**96.08**	87.23	**93.87**
Solar-flare_1	91.62	**95.97**	92.87	**96.16**	92.05	**95.53**	90.75	**93.24**	**92.18**	91.38
Solar-flare_2	97.00	**98.68**	96.47	**98.87**	97.35	**98.82**	97.56	**98.44**	96.79	**97.79**
Sonar	**82.36**	80.88	**82.63**	81.64	**83.09**	82.10	**82.60**	81.41	**84.14**	80.10
Spambase	**89.35**	87.09	**89.28**	88.47	89.56	**89.96**	**89.57**	82.94	**89.42**	76.82
Splice	**94.92**	93.32	**93.76**	92.03	**92.74**	88.08	**91.44**	78.41	**90.14**	73.64
Trains	70.00	70.00	70.00	70.00	70.00	70.00	64.00	64.00	70.00	70.00
Vehicle	63.36	**65.61**	63.19	**65.30**	62.98	**64.12**	62.74	**64.68**	62.62	**64.39**
Vote	89.43	**90.33**	89.66	**89.84**	89.61	**89.79**	89.57	**89.83**	89.43	**90.24**
Waveform-5000	80.64	**85.24**	80.07	**85.25**	79.70	**85.05**	79.31	**84.99**	79.08	**84.98**
Wine	98.86	98.86	98.99	98.99	**98.77**	98.65	**98.10**	97.77	**97.33**	96.99
Zoo	91.00	91.00	89.41	89.41	88.41	88.41	**88.01**	87.81	**86.61**	86.41
Average	**81.11**	**82.78**	**80.77**	**82.48**	**80.39**	**81.82**	**79.73**	**80.64**	**79.59**	**79.84**
#Sig better	**2**	**16**	**5**	**22**	**6**	**22**	**6**	**20**	**8**	**18**
Better	**11**	**26**	**12**	**30**	**14**	**30**	**17**	**25**	**18**	**25**

that this number decreases as the noise ratio increases. This behavior is understandable because noise has the greatest effect in this case and the classification accuracy quickly deteriorates, thus causing the loop to terminate. In contrast, taking the instance weights into account increases the average number of training cycles. This increase is a result of an instance weight being a number between 0 and 1, which makes the update step size smaller and causes the fine-tuning process to be even more delicate. Of course, the more noise we have, the more instances with smaller weights we have, and thus, more small updates are used. This process explains why the average number of training cycles tends to increase as we increase the noise ratio. The fact that the number of training cycles increases when we increase the noise ratio and take the weights of the instances into account indicates that the classification accuracy continues to improve in spite of the noisy instances. This continued improvement is a good sign, indicating that the proposed methods are successful in limiting the effect of noise.

Table 5 The average number of cycles the algorithms take to converge.

	0% noise	5% noise	10% noise	15% noise	20% noise
FTNB	3.82	3.56	3.42	3.39	3.33
PFTNB	4.4	4.32	4.65	4.7	4.53
NFTNB	4.41	3.81	4.23	4.33	4.59
CFTNB	4.42	4.59	4.98	5.94	5.81

5. Conclusions

The results of this work confirmed that although the fine-tuning method (FTNB) improves the classification accuracy of NB, it makes the algorithm more sensitive to noise and thus sacrifices one of the most important advantages of the NB learning algorithm, namely its tolerance of noise (Nettleton et al., 2010). To address this problem, this work proposed that the update step should not only be proportional to the size of the error but also be proportional to our confidence that the misclassified instance under consideration is not a noisy instance.

Three different methods were proposed to measure our confidence that a training instance is not a noisy instance. The first method is probability based, the second method is neighborhood based (based on the neighboring instances), and the third method combines the first two into one method. The three methods were demonstrated to improve the noise tolerance of the FTNB method. However, using the combined method to calculate the confidence factor was found to be more effective than each of its constituent methods alone. Using this method, CFTNB outperforms NB even at a 20% noise ratio.

Furthermore, the neighborhood-based method proved to be effective at improving the classification accuracy of the FTNB method in many domains, even on noise-free data sets. Investigating more methods for calculating the confidence factor and the effect of incorporating such confidence factors in other machine learning algorithms are topics for future work. Applying the methods proposed in this work for the intrusion detection problem and comparing it with the Naive Bayesian classifier (Altwaijry and Algarny, 2011) can also be an interesting topic of future research.

Acknowledgments

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

Altwaijry, H., Algarny, S., 2011. Bayesian based intrusion detection system. J. King Saud. Univ. Comput. Inf. Sci. 24, 1–6.

Blake, C.A.M.C., 1998. UCI Repository of Machine Learning Databases [online]. University of California, Department of Information and Computer Science, Irvine, CA. Available from: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Chickering, D.M., 1996. "Learning Bayesian Networks is NP-Complete", in Learning from Data: Artificial Intelligence and Statistics. Springer, pp. 121–130.

El Hindi, K.M., 2014. Fine tuning the Naive Bayesian learning algorithm. J. AI Commun. 27, 133–141.

El Hindi, K., 2013. Specific-class value distance measures for nominal attributes. J. AI Commun. 26, 261–279.

El Hindi, K., Al-Akhras, M., 2011. Smoothing decision bound-aries to avoid overfitting in neural network training. Neural Network World 21, 311–325.

Fayyad, U. Irani, K., 1993. Multi-interval discritization of continuous values attributes for classification learning. In: Proceedings of 13th International Joint Conference on Artificial Intelligence.

Friedman, Geiger, Goldszmidt, 1997. Bayesian Network Classifiers. Machine Learning, vol. 29, pp. 131–163.

Jiang, L., Guo, Y., 2005. Learning lazy Naive Bayesian classifiers for ranking. In: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence ICTAI 2005, Los Alamitos.

Jiang, L., Zhang, H., 2005. Learning instance greedily cloning Naive Bayes for ranking. In: Proceedings of the 5th IEEE International Conference on Data Mining, ICDM, Los Alamitos.

Jiang, Y., Zhou, Z.-H., 2004. "Editing Training Data for kNN Classifiers with Neural Network Ensemble", in Advances in Neural Networks. Springer, pp. 356–361.

Jiang, L., Zhang, H., Cai, Z., Su, J., 2005. Evolutional Naive Bayes. In: The 1st International Symposium on Intelligent Computation and its Applications.

Koplowitz, J., Brown, T., 1981. On the relation of performance to editing in nearest neighbor rules. Pattern Recogn. 13 (3), 251–255.

Langley, P., Sage, S., 1994. Induction of selective Bayesian classifiers. In: The Tenth Conference on Uncertainty in Artificial Intelligence.

Mitchell, T.M., 1997. Machine Learning. McGraw-Hill Science.

Muhlenbach, F., Lallicch, S., Zighed, D., 2004. Identifying and Handling Mislabelled Instances. J. Intell. Inf. Syst. 22, 89–109.

Nettleton, D., Orriols-Puig, A., Fornells, A., 2010. A study of the effect of different types of noise on the precision of supervised learning techniques. Artif. Intell. Rev. 33, 275–306.

Palacios-Alonso, M.A., Brizuela, C.A., Enrique Sucar, L., 2008. Evolutionary learning of dynamic Naive. Autom. Reasoning 45, 21–37.

Sanchez, J.S., Barandela, R., Marques, A.I., Alejo, R., Badenas, J., 2003. Analysis of new techniques to obtain quality training sets. Pattern Recogn. Lett. 24, 1015–1022.

Witten, Ian, Frank, Eibe, 2005. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann.

Zhang, H., Ling, C., 2001. An improved learning algorithm for augmented Naive Bayes. In: Bayes, Pacific-Asia Conference on Knowledge Discovery and Data Mining, LNCS.