Incremental Discretization for Naïve-Bayes Classifier

Jingli Lu, Ying Yang and Geoffrey I. Webb

Clayton School of Information Technology, Monash University
VIC 3800, Australia
{Jingli.Lu, Ying.Yang, Geoff.Webb}@infotech.monash.edu.au

Abstract. Naïve-Bayes classifiers (NB) support incremental learning. However, the lack of effective incremental discretization methods has been hindering NB's incremental learning in face of quantitative data. This problem is further compounded by the fact that quantitative data are everywhere, from temperature readings to share prices. In this paper, we present a novel incremental discretization method for NB, *incremental flexible frequency discretization* (IFFD). IFFD discretizes values of a quantitative attribute into a sequence of intervals of flexible sizes. It allows online insertion and splitting operation on intervals. Theoretical analysis and experimental test are conducted to compare IFFD with alternative methods. Empirical evidence suggests that IFFD is efficient and effective. NB coupled with IFFD achieves a rapport between high learning efficiency and high classification accuracy in the context of incremental learning.

1 Introduction

Naïve-Bayes classifiers (NB) are simple yet powerful [3, 4]. Its efficiency has witnessed its widespread deployment in real-world applications including medical diagnosis, fraud detection, email filtering and webpage prefetching. One key contributing factor to NB's efficiency is its capability of incremental learning from qualitative data [5, 6]. To accommodate a new training instance, NB only needs to update relevant entries in its probability table. This often has a much lower cost than non-incremental approaches that have to rebuild a new classifier from scratch in order to include new training data.

If learning involves quantitative data, NB often uses discretization to transform them into qualitative data. Briefly speaking, discretization groups sorted values of a quantitative attribute into intervals, treats each interval as a qualitative value and inputs them into NB. Ideally, discretization should also be incremental in order to be coupled with NB. When receiving a new training instance, incremental discretization is expected to be able to adjust intervals' boundaries and statistics, using only the current intervals and this new instance instead of re-accessing previous training data. Unfortunately, the majority of existing discretization methods are not oriented to incremental learning. To update discretized intervals with new instances, they need to add those new instances into previous training data, and then re-discretize on basis of the updated complete training data set. This is detrimental to NB's efficiency by inevitably slowing down its learning process. Hence there is a real and immediate need for appropriate incremental discretization methods for NB.

Some preliminary research has been contributed to exploring incremental discretization for NB. A representative is the method PiD proposed by Gama and Pinto [6]. PiD is based on a two layer histograms and is efficient in term of time and space complexity. However it can be sub-optimal in that the histograms are not exact and the splitting operation in the first layer possibly produces inexact counters.

This paper proposes a new effective approach, incremental flexible frequency discretization (IFFD). IFFD is based on fix frequency discretization (FFD) that has been demonstrated as a very efficient and effective discretization method for NB in the context of non-incremental learning [10, 11]. IFFD produces intervals with flexible sizes, stipulated by a lower bound and an upper bound. An interval is allowed to accept new values until its size reaches the upper bound. An interval whose size exceeds the upper bound is allowed to split if the resulting smaller intervals each have a size no smaller than the lower bound. Accordingly IFFD is able to incrementally adjust discretized intervals, effectively update associated statistics and efficiently synchronize with NB's incremental learning.

The remaining of this paper is organized as follows. Section 2 introduces naïve-Bayes learning and discretization. Section 3 explains the motivation and methodology of IFFD. Section 4 describes rival incremental methods from related work. Section 5 analyzes each alternative method's complexity in terms of learning time and space. Section 6 conducts experiments to verify IFFD's efficacy and efficiency. Section 7 gives concluding remarks.

2 Discretization for Naïve-Bayes Learning

2.1 Naïve-Bayes Classifier (NB)

Assume that an instance I is a vector of attribute values $<x_1, x_2, \ldots, x_n>$, each value being an observation of an attribute X_i ($i \in [1,n]$). Each instance can have a class label $c_i \in \{c_1, c_2, \cdots, c_k\}$, being a value of the class variable C. If an instance has a known class label, it is a training instance. If an instance has no known class label, it is a testing instance. The dataset of training instances is called the training dataset. The dataset of testing instances is called the testing dataset.

To classify an instance $I = \{x_1, x_2, \cdots, x_n\}$, NB estimates the probability of each class label given I, $P(C = c_i \mid I)$ using Formula (1, 2, 3,4). Formula (2) follows (1) because $P(I)$ is invariant across different class labels and can be canceled. Formula (4) follows (3) because of NB's attributes independent assumption. It then assigns the class with the highest probability to I. NB is called naïve because it assumes that attributes are conditionally independent of each other given the class label. Although its assumption is sometimes violated, NB is able to offer surprisingly good classification accuracy in addition to its very high learning efficiency, which makes NB popular with numerous real-world classification applications [2, 8].

$$P(C = c_i \mid I)$$

$$= \frac{P(C = c_i)P(I \mid C = c_i)}{P(I)} \quad (1)$$

$$\propto P(C = c_i)P(I \mid C = c_i) \quad (2)$$

$$= P(C = c_i)P(< x1, x2, , xn > \mid C = c_i) \quad (3)$$

$$= P(C = c_i)\prod_{j=1}^{n} P(X_j = x_j \mid C = c_i) \quad (4)$$

In naïve-Bayes classifier, the class type must be qualitative while the attribute type can be either qualitative or quantitative. When an attribute X_j is quantitative, it often has a large or even infinite number of values. As a result, the conditional probability that X_j takes a particular value x_j given the class label c_i covers very few instance if there is any at all. Hence it is not reliable to estimate $P(X_j=x_j|C=c_i)$ according to the observed instances. One common practice to solve the problem of quantitative data for NB is discretization.

2.2 Discretization

Discretization is a popular approach to transforming quantitative attributes into qualitative ones for NB. It groups sorted values of a quantitative attribute into a sequence of intervals, treats each interval as a qualitative value, and maps every quantitative value into a qualitative value according to which interval it belongs to. In the paper, the boundaries among intervals are sometimes referred to as *cut points*. The number of instances in an interval is referred to as *interval frequency*. The total number of intervals produced by discretization is referred to as *interval number*.

Incremental discretization aims at efficiently updating discretization intervals and associated statistics upon receiving each new training instance. Ideally, it does not require to access historical training instances to carry out the update. Instead it only needs the current intervals (with associated statistics) and the new instance.

3 Incremental Flexible Frequency Discretization

In this section, we propose a novel incremental discretization method, *incremental flexible frequency discretization* (IFFD). It is motivated by the pros and cons of fixed frequency discretization (FFD) in the context of naive-Bayes learning and incremental learning [10, 11].

3.1 Fixed Frequency Discretization (FFD)

FFD has been proposed as an effective and efficient discretization method for naïve-Bayes learning through bias and variance management. It has been found that large interval size tends to increase NB's classficiation bias while large interval number tends to increase NB's classification variance [12]. To discretize a quantitative attribute, FFD sets a sufficient interval frequency, m = 30 [11,13]. It then discretizes the ascendingly sorted values into intervals of frequency m. By introducing m, FFD aims to ensure that each interval has sufficient training instances for NB probability estimation, reducing classification variance error. On top of that, by not limiting the number of intervals formed, more intervals can be formed as the training data size increases, reducing classification bias error. Empirical evidence has demonstrated that FFD helps NB achieve lower classification error than alternative discretization methods do.

Although FFD is effective for naïve-Bayes learning, it is developed in the context of non-incremental learning. Every time when new training instances have arrived, FFD has to rebuild the discretization intervals from scratch. It is possible that even a single instance can push every boundary to (unnecessarily) move. For example, FFD discretizes the sorted values of a quantitative attribute into the following intervals. For simplicity, we assume m = 3:

{3.0, 4.0, 4.3}, {4.5, 5.1, 5.9}, {6.0, 6.1, 6.2}, {6.5, 6.7, 6.8}, {6.9, 7.1}

Suppose that a new instance has come with this attribute being value "5.2". According to the current cut points, the appropriate interval to accommodate "5.2" is {4.5, 5.1, 5.9}. Inserting "5.2" into {4.5, 5.1, 5.9} will make the interval frequency increase to 4, which is greater than FFD's specified threshold 3. Hence we need to move "5.9" out of the updated interval{4.5, 5.1, 5.2, 5.9} and insert it into the interval {6.0, 6.1, 6.2}, which produces another interval {5.9, 6.0, 6.1, 6.2} whose frequency is greater than 3. Following the same lines of reasoning, we have to move "6.2" into the next one and so on so forth until the last interval. As a result, the updated intervals are {3.0, 4.0, 4.3}, {4.5, 5.1, 5.2}, {5.9, 6.0, 6.1}, {6.2, 6.5, 6.7}, {6.8, 6.9, 7.1} and almost every cut point has been changed.

In this case, FFD has to rebuild the intervals and NB's conditional probability table from the second interval all the way to the last one. In the best situation, the new instance is inserted into the last interval and the computation cost can be non-trivial. However in the worst situation such as when the new instance is inserted into the first interval, FFD is extremely inefficient. The reason is that FFD specifies a fixed interval frequency. This observation motivates our new incremental discretization approach as follows.

3.2 Incremental Flexible Frequency Discretization (IFFD)

IFFD sets its *interval frequency* to be a range [*minBinsize*, *maxBinsize*) instead of a single value m. The two arguments, *minBinsize* and *maxBinsize*, are respectively the minimum and maximum frequency that IFFD allows intervals to assume. Whenever a new value arrives, IFFD first inserts it into the interval that the value falls into. IFFD then checks whether the updated interval's frequency reaches *maxBinsize*. If not, it accepts the change and update statistics accordingly. If yes, IFFD splits the overflowed interval into two intervals under the condition that any of the resulting intervals has its frequency no less than *minBinsize*. Otherwise, even if the interval overflows because of the insertion, IFFD does not split it, in order to prevent high classification variance [10,11]. In the current implementation of IFFD, *minBinsize* is set as 30, following FFD's lines of reasoning so as to minimize classification bias and variance; and *maxBinsize* is set as twice of *minBinsize*.

By assuming a more flexible interval frequency, IFFD is able to solve FFD's dilemma in incremental learning. Recall the example in Section 3.1. Assume *minBinsize* = 3 and hence *maxBinsize* = 6. When the new attribute value "5.2" comes, IFFD inserts it into the second interval {4.5, 5.1, 5.9}. That interval is hence changed into {4.5, 5.1, 5.2, 5.9} whose frequency (equal to 4) is still within [3, 6). So what we need do is only to modify NB's conditional probabilies related to the second interval. Assume another two new attribute values "5.4, 5.5" have come and are again inserted into the second interval. This time, the interval {4.5, 5.1, 5.2, 5.4, 5.5, 5.9} has a frequency as 6, reaching *maxBinSize*. Hence IFFD will split it into {4.5, 5.1, 5.2} and {5.4, 5.5, 5.9} whose frequencies are both within [3, 6). Then we only need to recalculate NB's conditional probabilities related to those two intervals. By this means, IFFD makes the update process local, affecting a minimum number of intervals and associated statistics. As a result, incremental discretization can be carried out very efficiently,

Table 1 shows the pseudo codes of the IFFD algorithm. For simplicity, we just consider one attribute value to update the discretization intervals and classifier and assume all attribute values are different. *cutPoints* is the set of cut points of discretization intervals. *counter* is the conditional probability table

of the classifier. *minBinsize* is minimum bin size. IFFD will update the *cutpoints* and *counter* according to new attribute value *V*. *classLabel* is the class label of *V*.

Table 1. Pseudo Codes of IFFD

```
Function: IFFD(cutPoints, counter, minBinsize, V, classLabel)
//If V is greater than the last cut point
if(V > cutPoints[size-2] ) //size is the interval number
// cutPoints counts from 0
{ insert V into interval[size-1];
   counter[size-1][classLabel]++;
   chaInt = size-1; //record changed interval
}
else
{ for(j = 0; j < size-1; j++)
    if(V =< cutPoints[j])
    { insert V into interval[j];
      intFre[j]++;
      counter[j][classLabel]++;  //update contingency table
      chaInt = j;  //record the interval which has been changed
      break;
    }
}
if(intFre[chaInt] > minBinsize*2)
{  get new cut point; //split interval[chaInt] into two c1 and c2
   insert the new cut point into cutPoints;
   calculate counter[c1] and counter[c2];//update contingency table
}
```

Please be noted that identical values are always kept in the same interval. For example, if the interval is {4.5, 5.1, 5.2, 5.2, 5.2, 5.6, 5.9}, IFFD will not split it into {4.5, 5.1, 5.2} and {5.2, 5.6, 5.9} even though its frequency has exceeds *maxBinsize* (=6). Nor will IFFD split it into {4.5, 5.1} and {5.2, 5.2, 5.2, 5.6, 5.9} or {4.5, 5.1, 5.2, 5.2, 5.2} and {5.6, 5.9}, because the smaller interval frequency is less than *minBinsize* (=3).

4 Rival Methods from Related Work

4.1 Move Boundary FFD (MFFD)

An intuitive way to relieve FFD's dilemma in incremental learning (Section 3.1) is to just move the interval boundaries instead of redoing discretizaiton. We name this method *move boundary FFD* (MFFD). For the same example as in Section 3.1, if MFFD is applied, we just calculate the change of every interval. The second interval {4.5, 5.1, 5.9} has been inserted into an attribute value "5.2" and delete an attribute value "5.9", then we just modify the conditional probability. Attention is only paid to the inserted and deleted values. Do like this until the last interval. NB coupled with MFFD has the same classification accuracy as NB coupled with FFD, but the former is more efficient than the latter.

Table 2 presents the pseudo codes of MFFD. For simplicity, we just consider one attribute value to update the discretization intervals and classifier and assume all attribute values are different. *cutpoints* is the set of cut points of discretization intervals. *counter* is the conditional probability table of the

classifier. MFFD will update the c*utpoints* and *counter* according to new attribute value *V*. *classLabel* is the class label of *V*.

Table 2. The Pseudo Codes of MFFD

```
Function: MFFD(cutPoints, counter, V, classLabel)
curVal=V; curClasslabel= classLabel;
for(j = 0; j < size-1; j++)   //size is the interval number
{ if(curVal =< cutpoints[j])
  { // interval[j] is the jth interval of the attribute
    insert curVal into interval[j];
    //fre is the specified interval frequency
    // V[j][fre-1] is the last value in interval[j]
    remove V[j][fre-1] from interval[j];
    cutPoints[j]= V[j][fre-2];  //modify cut points
    counter[j][curClasslabel]++;  //update contingency table
    counter[j][ V[j][fre-1].class]--;
    curVal = V[j][fre-1];
    curClasslabel = V[j][fre-1].class;
  }
}
If(fre[size-1] < split threshold)
{ insert curVal into interval[size-1];
   counter[size-1][curClasslabel]++;
}
else
{ split interval[size-1];
   calculate counter[size-1] and counter[size];
   size = size+1;
}
```

4.2 Partition Incremental Discretization (PiD)

PiD is a two layer histograms incremental discretization method [6]. The first layer based on equal-width or equal-frequency determines the candidate cut points according to observed values. At this layer, the interval number is significantly greater that the final interval number. For example, the final interval number is 40, probably the interval number in the first layer is 200. For incremental learning, it inserts the incremental data into the appropriate intervals. To any interval whose frequency is greater than the specified threshold, it will be split. Because in this layer, it does not store the historical data, the splitting result is inaccurate. It just splits an interval into two uniformly. The second layer merges the intervals gained at the first layer. In the second layer, PiD can construct the final discretization interval by any different strategies. Namely, PiD discretizes quantitative attributes twice. At first, it uses a loose interval number to discretize; and then merges intervals if necessary. The main advantage of PiD is low time and space complexity, but during the splitting operation in the first layer, it possibly produces inexact counters.

4.3 Kernel Density Estimation (KDE)

A counterpart of discretization is probability density estimation to handle quantitative attributes for NB. It models each quantitative attribute by some continuous probability distribution. Probability

density estimation methods can manipulate quantitative attributes for naïve-Bayes incremental learning. A representative method is *kernel density estimation* (KDE) [7].

KDE is a non-parametric approach that does not assume the underlying distribution to take any particular form. Instead it estimates from sample values. This circumvents unsafe assumptions and achieves better accuracy because of real world diversity. For KDE, it calculates the conditional class probability as:

$$P(X_j = x_j \mid C = c_i) = \frac{1}{n_i} \sum_k \int_l^h f(x_j, \mu_k, \sigma_c) dx_j \quad (\mathbf{5})$$

where n_i is the number of training instances with class label c_i. For every quantitative attribute of testing instance, KDE has to perform probability calculation n_i times to get $P(X_j=x_j|C=c_i)$. If the instance number is large, it has a potential computational problem.

5 Time and Space Complexity Comparison

In this section, we analyse the time and space complexity incurred by accommodating a new training instance. It includes updating the discretized intervals as well as updating required probabilities for NB.

5.1 Time Complexity

In the following, n is abbreviation of instance number; k is the attribute number; C is the number of class label, specified *Interval Frequency* is abbreviated by *IntF*, *IntN* represents *Interval Number*, then *IntN=n/IntF*.

5.1.1 Train Time Complexity on a New Instance

Train Time Complexity of MFFD

Assume the probability of the new attribute value inserting into every interval is equal. $IntN - i + 1$ is the number of intervals which has to be changed, where i is the appropriate interval for the new instance. Inserting an instance into the interval while deleting another one from the interval has a constant cost in time complexity *O(1)*. So for every incremental attribute value, the training time complexity is presents in equation (6). This complexity repeating for k attribute is *O(k)*, so resulting in the totally complexity is *O(n)*O(k)=O(nk)*.

$$\frac{\sum_{i=1}^{IntN} (IntN - i + 1) * \mathrm{O}(1)}{IntN} = \frac{\frac{IntN(IntN+1)}{2} * \mathrm{O}(1)}{IntN} = \frac{IntN+1}{2} * \mathrm{O}(1) = \frac{\frac{n}{IntF}+1}{2} * \mathrm{O}(1) = O(n). \quad (\mathbf{6})$$

Train Time Complexity of PiD

The time complexity of PiD depends on the discretization methods selected in each layer. In our experiments, we select equal-width and PD for the two layers separately (the reason that we select them is explained in 4.2.1). Here we just analyze time complexity in this situation.

In the first layer, when the interval frequency of a specified interval is greater than a user defined threshold (a percentage of the total instance number), the interval will be split. The more interval number is defined in the first layer, the less probability some interval will be split. In the first layer, the interval frequency is a large number, so the time for splitting operation can be ignored. The input of the second layer is the intervals and associated statistics of first layer. If the interval gained in the first layer is m, then the time complexity of PiD is $O(mk)$.

Train Time Complexity of IFFD

Assume the probability of the new attribute value inserting into some interval is equal. *Max* is the maximum interval frequency; *Min* is the minimum interval frequency.

When a new attribute value inserts into the appropriate interval, the probability that the interval does not split is $\frac{Max - Min}{Max - Min + 1}$. In this situation, the operation is just to insert the new instance. Inserting an instance into the interval has a constant cost in time complexity *O(1)*. The probability that the interval splits is $\frac{1}{Max - Min + 1}$. In this situation, the operation is to recalculate the conditional probability table of the two new intervals and change the cut points. For a single attribute, if the data structure of cutPoints is array, the time complexity is presented in equation (7), $\frac{IntN}{2}$ means the number of cut points have to move, when insert a new cut point into the *cutPoints*. And if tree or list structure is selected, the time complexity is demonstrated as equation (8). This complexity repeating for *k* attribute is *O(k)*, so resulting in the totally complexity for array structure is *O(n)*O(k)=O(nk)* and for tree structure is *O(1)*O(k)=O(k)*. In our experiment, we select array structure to store *cutPoints*, because our select Weka as the platform, in Weka, *cutPoints* is stored in an array.

$$\frac{(Max - Min) * \mathrm{O}(1)}{Max - Min + 1} + \frac{1}{Max - Min + 1} * (Min + 1 + \frac{IntN}{2}) = O(n). \tag{7}$$

$$\frac{(Max - Min) * \mathrm{O}(1)}{Max - Min + 1} + \frac{Min + 1}{Max - Min + 1} = \frac{2Min + 1}{Min + 1} \approx 2 = O(1). \tag{8}$$

Train Time Complexity of KDE
At training time, KDE just store the attribute values, so its time complexity is $O(k)$.

5.1.2 Test Time Complexity on a New Instance

Test Time Complexity of MFFD, IFFD and PiD
For every class label, the classifiers which manipulate quantitative attributes by discretization methods can get the conditional probability from the conditional probability table directly, so testing time complexity on the new instance is $O(Ck)$.

Test Time Complexity of KDE
At testing time, from equation (5) we can see, for every class label c_i and every quantitative attributes, KDE must evaluate *f* for every observed different attribute value whose class label is in class c_i. So the testing time complexity of KDE is $O(nk)$.

5.2 Space Complexity

5.2.1 Space Complexity of MFFD, IFFD & KDE on a New Instance

MFFD, IFFD and KDE have to store the historical quantitative attributes, so their space complexity is $O(nk)$.

MFFD has to change the cut points and modify the conditional probability table, so historical quantitative attributes are necessary.

For IFFD, when the interval frequency of some interval exceeds the threshold, the interval has to be split. Historical quantitative data is necessary to splitting operation. So IFFD must store the historical quantitative attribute values for every instance. But for every new instance, the modified interval is just one: split it or insert a point into it, namely the adjustment is local. So we can store the historical data in external storage. When change is necessary, we copy it from external storage to memory. With the development of hardware, storage is not a big problem.

KDE must store every different quantitative attribute value for every class label. To classify an instance, KDE has to access every attribute value to calculate the conditional class probability. So it is necessary to store the attributes values in the memory. However memory store is more expensive than external storage. If for every class label there are many duplicate quantitative attribute values, KDE has a lower space then MFFD and IFFD; otherwise their storage space are equal.

5.2.2 Space Complexity of PiD on a New Instance

Splitting operation in PiD is to split an interval uniformly. PiD does not need to store historical quantitative attribute values. It just stores the interval information which gained at the first layer. So its space complexity is $O(m)$, where m is the number of interval in the first layer. Compared with other methods, PiD has the lowest space complexity.

The time and space complexity are summarized in Table 3.

Table 3. Algorithmic complexity. n is abbreviation of instance number; k is the attribute number; C is the number of class label; m is the number of interval number in the first layer for PiD

Method		MFFD	IFFD	PiD	KDE
Time Complexity	Trainning	$O(nk)$	$O(nk)$ (Array) $O(k)$ (Tree)	$O(mk)$	$O(k)$
	Testing	$O(Ck)$	$O(Ck)$	$O(Ck)$	$O(nk)$
Space Complexity		$O(nk)$	$O(nk)$	$O(mk)$	$O(nk)$

6 Experimental Evaluation

In this section, we compare the incremental learning performance of NB when coupled with IFFD, PiD, MFFD and KDE respectively to handle quantitative attributes.

6.1 Data

The experiments use a large suite of 30 benchmark datasets from the UCI machine learning repository [1]. For the purpose of incremental learning, the chosen datasets each have more than 500 instances. Table 4 describes the statistics of each dataset.

Table 4. Experimental Datasets. For each dataset, *Size* is the number of instances, *Qa* is the number of quantitative attributes, *Ql* is the number of qualitative attributes and *C* is the number of classes.

ID	Dataset	Size	Qa	Ql	C	ID	Dataset	Size	Qa	Ql	C
1	cylinder-bands	540	20	19	2	**16**	Abalone	4177	8	0	3
2	balance-scale	625	4	0	3	**17**	spambase	4601	57	0	2
3	credit-a	690	6	9	2	**18**	waveform-5000	5000	40	0	3
4	breast-w	699	9	0	2	**19**	page-blocks	5473	10	0	5
5	diabetes	768	8	0	2	**20**	optdigits	5620	48	0	10
6	vehicle	846	18	0	4	**21**	satellite	6435	36	0	6
7	anneal	898	6	32	6	**22**	Musk2	6598	166	0	2
8	vowel	990	10	3	11	**23**	pioneer	9150	30	6	57
9	German	1000	7	13	2	**24**	Thyroid	9169	7	22	20
10	cmc	1473	2	7	3	**25**	ae	9961	12	0	9
11	yeast	1484	7	1	10	**26**	pendigits	10992	16	0	10
12	volcanoes	1520	3	0	4	**27**	Sign	12546	8	0	3
13	mfeat-zernike	2000	47	0	10	**28**	letter	20000	16	0	26
14	segment	2310	19	0	7	**29**	Adult	48842	6	8	2
15	hypothyroid	3772	7	23	4	**30**	Shuttle	58000	9	0	7

6.2 Design

For each instance, we randomly shuffle the instances and use the first 200 instances to initialize an NB classifier. The remaining instances come one after the other. Each instance is to be classified by the current NB first. Its true class label is then made known to the classifier which takes it as a new training instance. Accordingly, the discretized intervals are updated and so is the classifier. Then the next instance comes and the same procedure runs again, and so on so forth until the last instance is classified. We call this complete process a *trial*. We conduct five trails and average their classification error rates.

For IFFD, *minBinSize* is 30 while *maxBinsize* is 60. For PiD, the first layer is equal-width discretization and the interval number is 200 [5]. In the second layer, we choose to proportional discretization [9], which has been demonstrated efficient and work well [9].

Statistically a win/draw/lose record is calculated when we compare IFFD against each alternative method. The record represents the number of data sets in which IFFD respectively beats, tie with or loses to the rival method. A one-tailed binomial sign test will be applied to the record. If its result is less than the critical level of 0.05, the wins against losses are statistically significant, supporting the claim that IFFD has a systematic (instead of by chance) advantage over the rival method.

6.2.1 Comparing at Ten Observation points

Along the time line, 10 observed classification error rates are recorded when 10%, 20%, 30%,…, 100% of instances have been classified respectively. At every observation point, we calculate the win/draw/lose records on classification error rate when comparing IFFD against alternative methods. Table 5 lists the records as well as their sign test results.

Table 5. Classification error win/draw/lose records on 10 observation points

Method		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
IFFD & PiD	Win	20	19	19	20	22	21	20	21	18	19
	Draw	1	0	0	0	0	0	1	1	2	1
	Lose	9	11	11	10	8	9	9	8	10	10
	Sign test	0.031	0.1	0.1	0.049	0.008	0.021	0.031	0.012	0.092	0.068
IFFD & MFFD	Win	14	14	15	17	17	16	17	18	17	17
	Draw	1	2	0	1	0	0	0	0	0	2
	Lose	15	14	15	12	13	14	13	12	13	11
	Sign test	0.644	0.575	0.572	0.229	0.292	0.428	0.292	0.181	0.292	0.172
IFFD & KDE	Win	17	15	16	18	17	19	19	19	19	19
	Draw	0	0	0	0	0	0	0	0	0	0
	Lose	13	15	14	12	13	11	11	11	11	11
	Sign test	0.292	0.572	0.428	0.181	0.292	0.1	0.1	0.1	0.1	0.1

At every observation points, we also record the arithmetic mean of each method's classification error rate averaged on 30 datasets, as in figure 1.

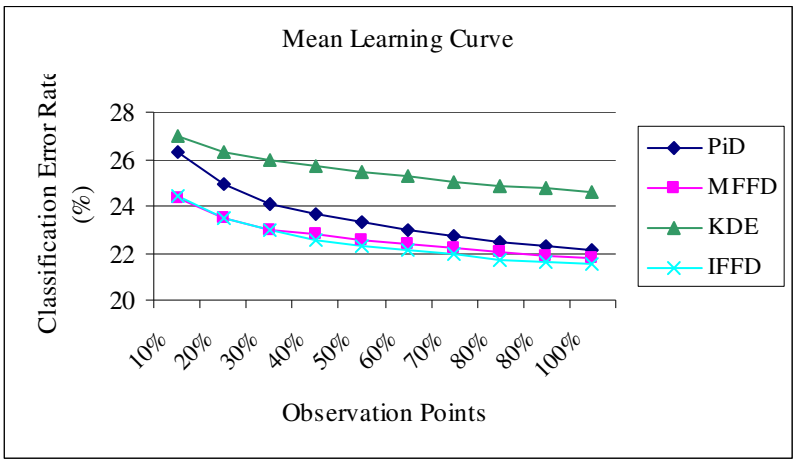

Fig. 1. Incremental Learning Curve. Comparing the classification error rate of naïve-Bayes classifiers which use the 4 methods to deal with quantitative attributes respectively at the 10 observation points, we can see, the error rate of IFFD is marginally lower than that of MFFD's for the whole learning curve, the separation between IFFD and PiD becomes smaller and smaller with instances increasing. IFFD has substantially lower error rate than KDE.

In general, the classification error rate decreases gradually while more training instances are available. The error rate of IFFD is marginally lower than that of MFFD's for the whole learning curve. There is a larger gap between IFFD and PiD at the beginning, which shrinks with time going on. IFFD has substantially lower error rate than KDE and its leading position remains through the whole learning period. The learning curve of PiD and KDE have small gaps at the beginning which enlarges later.

Specifically, to compare IFFD against PiD, IFFD is statistically more accurate than PiD at the 0.05 critical level when the training data size is medium (from the column 40% to the column 80%). On the pther hand, IFFD is not significantly better than PiD when the training data size is extremely small or large. We suggest the reason that PiD employs proportional discretization at its second layer, which controls the interval frequency better than IFFD's interval [30,60) does.

For discretization, large interval frequency tends to produce low variance but high bias while large interval number tends to produce low bias but high variance. Proportional discretization attains equal bias and variance reduction by setting both interval frequency and interval number to be square root of the number of training instances, a strategy that has been demonstrated to react sensibly to varying training data size [9]. Figure 2 shows the ideal interval frequency's changing while training instances increase from 1 to 5000. From figure 3, we can see that when instances are fewer then 900, the ideal interval frequency should be less then 30, and when instances are more than 3600, the ideal interval frequency should be greater than 60. However, the current implementation of IFFD only allows the interval frequency to vary in the interval [30, 60). Hence for small datasets, IFFD's interval frequency can be too big; whereas for large datasets, IFFD's interval frequency can be too small. This explains why IFFD's performance is not significantly better then PiD's at the beginning and at the very end of the incremental learning curve. Our understanding of this issue also leads to an interesting future research issue, that is, how to make IFFD's flexible frequency range change according to different training data size.

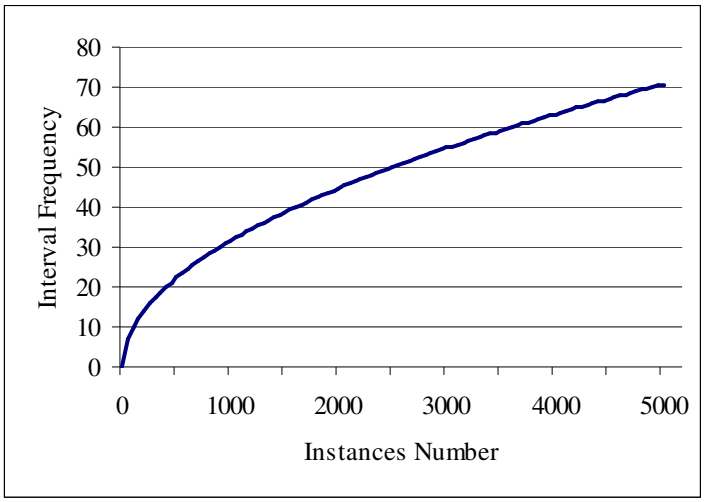

Fig. 2. Different sizes of training data require different ideal interval frequencies. Proportional discretization answers this call by setting both interval number and interval frequency to be the square root of the number of training instances. With instance number increasing, the interval frequency and number increase accordingly. When the instances number is less than 900, the ideal interval frequency should be less then 30 and when the instance number is greater than 3600, the ideal interval frequency should be greater then 60.

To compare IFFD against MFFD, according to Table 5, the difference between classification error rate of IFFD and that of MFFD's is not significant. When there are a small number of training instances, MFFD is better than IFFD. When more training instances are available, IFFD becomes better than MFFD. We suggest the reason is that the interval frequency of MFFD is 30 and is smaller than the interval frequency [30, 60) of IFFD. According to the interval frequency analysis in Fig 1, 30 is more suitable for small datasets.

To compare IFFD against KDE, according to Table 5, the difference between classification error rate of KDE and that of IFFD's is not significant. However, for some datasets, IFFD is dramatic better than KDE, as to be demonstrated in Section 6.2.2.

Table 6. Classification error win/draw/lose records on 30 datasets

Method	Win	Draw	Lose	Sign Test
IFFD & PiD	20	0	10	0.049
IFFD & MFFD	16	0	14	0.428
IFFD & KDE	19	0	11	0.1

6.2.2 Comparing on Every Dataset

For every dataset, if the classification error rate of a rival method is less than that of IFFD's at more than half of the 10 observation points, we deem that the rival method is better than IFFD for this dataset, and vice versa. The resutling win/draw/lose records across the 30 datasets are listed in Table 6. Accordingly, IFFD is significant better than PiD at the 0.05 critical level. Although not statistically significant, IFFD wins more often than not when compared with MFFD or KDE.

6.2.3 Comparing Running Time

This section compares the running time of the four rival methods. Figure 3 demonstrates each method's running time averaged on the 30 datasets. From the fastest to slowest is PiD, IFFD, KDE and MFFD. It is consistent with our theoretical analysis in Section 5. PiD is the fastest algorithm. Although IFFD and MFFD have the same time complexity, for IFFD, it just modify one or two intervals and update the cutPoints, while for MFFD, on average it has to modify IntN /2 intervals and associated statistics, where IntN is the interval number.

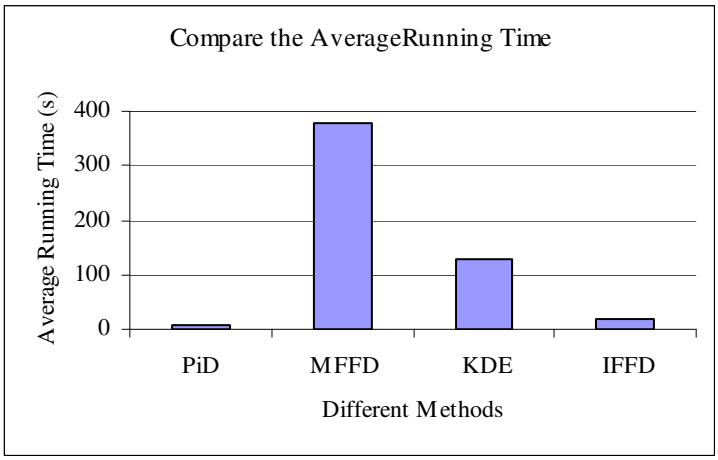

Fig. 3. NB's running time averaged on 30 datasets when coupled with PiD IFFD, KDE and MFFD respectively. PiD and IFFD are more efficient than KDE and MFFD.

7. Conclusion

In this paper, we have argued that most existing discretization methods do not suit incremental learning of naïve-Bayes classifiers (NB). This is sub-optimal because NB is extensively deployed for real-world applications which often involve quantitative data. Accordingly, we have proposed a novel incremental discretization method *incremental flexible frequency discretization* (IFFD). IFFD inherits from fxed frequency discretization the strength of minimizing classification bias and variance fir NB. Meanwhile, it adopts a more flexible strategy to handle to interval size so as to efficiently update discretized intervals upon receiving each new training instance. A comprehensive, theoretical and empirical study has been conducted to compare IFFD with representative alternative approaches. Observations suggest NB coupled with IFFD can achieve higher classification efficiency than those with MFFD and KDE, while achieve higher classification accuracy than those with PiD and KDE. Hence IFFD is a promising discretization approach for NB in practice where people want a rapport between learning accuracy and efficiency.

Acknowledgement

This research was supported by Australian Research Council grant DP0556279. We wish to thank João Gama and Carlos Pinto for providing the source code of PiD method.

Reference

[1] C.L.Blake, & C.J.Merz,(1998). UCI repository of machine learning databases [http://www.ics.uci.edu/»mlearn/mlrepository.html].

[2] Peter Clark & Tim Niblett. (1989). The CN2 induction algorithm, Machine Learning 3(4), 261-283

[3] Bojan CESTNIK. (1990). Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence* (1990), pp. 147–149. (pp. 3, 23)

[4] Richard O. Duda & Peter E. Hart. (1973). Pattern classification and scene analysis. New York: John Wiley and Sons.

[5] João Gama & Gladys Castillo (2002): Adaptive Bayes. Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence: 765-774

[6] João Gama & Carlos Pinto .(2005) Discretization from Data Streams: Applications to Histograms and Data Mining Second International Workshop on Knowledge Discovery from data Streams

[7] George H. John and Pat Langley (1995). *Estimating continuous distributions in Bayesian classifiers*. In Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, pages 338--345.

[8] Pat Langley, Wayne Iba &Kevin Thompson. (1992). An analysis of Bayesian classifiers. Proceedings of the Tenth National Conference on Artificial Intelligence (pp. 223-228). San Jose, CA: AAAI Press.

[9] Ying Yang & Geoffrey I. Webb (2001). Proportional kinterval discretization for naive-Bayes classifiers. 12th European Conference on Machine Learning (ECML'01) (pp. 564–575). Springer.

[10] Ying Yang and Geoffrey I. Webb (2003). Discretization For Naive-Bayes Learning: Managing Discretization Bias And Variance. Technical Report 2003/131, School of Computer Science and Software Engineering, Monash University.

[11] Ying Yang. Discretization for Naïve-Bayes Learning. PhD thesis, school of Computer Science and Software Engineering of Monash University.

[12]Ying Yang & Geoff Webb (2003), On why discretization works for naïve-Bayes classifiers. In Proceedings of the 16th Australian Joint Conference on Artificial Intelligence (AI)

[13] Neil A. Weiss. (2002). *Introductory Statistics, Sixth Edition*. Greg Tobin. (p. 98)