# An Adaptive Nearest Neighbor Classification Algorithm for Data Streams

Yan-Nei Law and Carlo Zaniolo

Computer Science Dept., UCLA, Los Angeles, CA 90095, USA
{ynlaw, zaniolo}@cs.ucla.edu

**Abstract.** In this paper, we propose an incremental classification algorithm which uses a multi-resolution data representation to find adaptive nearest neighbors of a test point. The algorithm achieves excellent performance by using small classifier ensembles where approximation error bounds are guaranteed for each ensemble size. The very low update cost of our incremental classifier makes it highly suitable for data stream applications. Tests performed on both synthetic and real-life data indicate that our new classifier outperforms existing algorithms for data streams in terms of accuracy and computational costs.

## 1 Introduction

A significant amount of recent research has focused on mining data streams for applications such as financial data analysis, network monitoring, security, sensor networks, and many others [3,8]. Algorithms for mining data streams have to address challenges not encountered in traditional mining of stored data: at the physical level, these include fast input rates and unending data sets, while, at the logical level, there is the need to cope with concept drift [18]. Therefore, classical classification algorithms must be replaced by, or modified into, incremental algorithms that are fast and light and gracefully adapt to changes in data statistics [17,18,5].

**Related Works.** Because of their good performance and intuitive appeal, decision tree classifiers and nearest neighborhood classifiers have been widely used in traditional data mining tasks [9]. For data streams, several decision tree classifiers have been proposed—either as single decision trees, or as ensembles of such trees. In particular, VFDT [7] and CVFDT [10] represent well-known algorithms for building single decision tree classifiers, respectively, on stationary, and time-changing data streams. These algorithms employ a criterion based on Hoeffding bounds to decide when a further level of the current decision tree should be created. While this approach assures interesting theoretical properties, the time required for updating the decision tree can be significant, and a large amount of samples is needed to build a classifier with reasonable accuracy. When the size of the training set is small, the performance of this approach can be unsatisfactory.

Another approach to data stream classification uses ensemble methods. These construct a set of classifiers by a base learner, and then combine the predictions

A. Jorge et al. (Eds.): PKDD 2005, LNAI 3721, pp. 108–120, 2005.

of these base models by voting techniques. Previous research works [17,18,5] have shown that ensembles can often outperform single classifiers and are also suitable for coping with concept drift. On the other hand, ensemble methods suffer from the drawback that they often fail to provide a simple model and understanding of the problem at hand [9].

In this paper, we focus on building nearest neighbor (NN) classifiers for data streams. This technique works well in traditional data mining applications, is supported by a strong intuitive appeal, and it rather simple to implement. However, the time spent for finding the exact NN can be expensive and, therefore, a significant amount of previous research has focused on this problem. A well-known method for accelerating the nearest neighbor lookup is to use k-d trees [4]. A k-d tree is a balanced binary tree that recursively splits a d-dimensional space into smaller subregions. However, the tree can become seriously unbalanced by massive new arrivals in the data stream, and thus lose the ability of expediting the search. Another approach to NN classifiers attempts to provide approximate answers with error bound guarantees. There are many novel algorithms [11,12,13,14] for finding approximate $K$-NN on stored data. However, to find the $(1 + \epsilon)$-approximate nearest neighbors, these algorithms must perform multiple scans of the data. Also, the update cost of the dynamic algorithms [11,13,14] depends on the size of the data set, since the entire data set is needed for the update process. Therefore, they are not suitable for mining data streams.

**Our ANNCAD Algorithm.** In this paper, we introduce an Adaptive NN Classification Algorithm for Data-streams. It is well-known that when data is non-uniform, it is difficult to predetermine K in the KNN classification [6,20]. So, instead of fixing a specific number of neighbors, as in the usual KNN algorithm, we adaptively expand the nearby area of a test point until a satisfactory classification is obtained. To save the computation time for finding adaptive NN, we first preassigning a class to every subregion (cell). To achieve this, we decompose the feature space of a training set and obtain a multi-resolution data representation. There are many decomposition techniques for multi-resolution data representations. The averaging technique used in this paper can be thought of Haar Wavelets Transformation [16]. Thus, information from different resolution levels can then be used for adaptively preassigning a class to every cell. Then we determine to which cell the test point belongs, in order to predict its class. Moreover, because of the compact support property inherited from wavelets, the time spent updating a classifier when a new tuple arrives is a small constant, and it is independent of the size of the data set. Unlike VDFT, which requires a large data set to decide whether to expand the tree by one more level, ANNCAD does not have this restriction.

In the paper, we use grid-based approach for classification. The main characteristic of this approach is the fast processing time and small memory usage, which is independent of the number of data points. It only depends on the number of cells of each dimension in the discretized space, which is easy to adjust in order to fulfill system constraints. Therefore, this approach has been widely employed in clustering problem. Some examples of novel clustering algorithms

are *STING* [19], *CLIQUE* [1] and *WaveCluster* [15]. However, there is not much work using this approach for classification.

**Paper Organization.** In this paper, we present our algorithm ANNCAD and discuss its properties in §2. In §3, we compare ANNCAD with some existing algorithms. The results suggest that ANNCAD will outperform existing algorithms. Finally, conclusions and suggestions for future work will be given in §4.

## 2 ANNCAD

In this section, we introduce our proposed algorithm ANNCAD, which includes four main stages: (1) Quantization of the Feature Space; (2) Building classifiers; (3) Finding predictive label for a test point by adaptively finding its neighboring cells; (4) Updating classifiers for newly arriving tuples. This algorithm only read each data tuple at most once, and only requires a small constant time to process it. We then discuss its properties and complexity.

### 2.1 Notation

We are given a set of $d$-dimensional data $D$ with attributes $X_1, X_2, ..., X_d$. For each $i = 1, ..., d$, the domain of $X_i$ is bounded and totally ordered, and ranges over the interval $[L_i, H_i)$. Thus, $X = [L_1, H_1) \times ... \times [L_d, H_d)$ is the feature space containing our data set $D$.

**Definition 1.** *A discretized feature space is obtained by dividing the domain of each dimension into g open intervals of equal length. The discretized feature space so produced consists of $g^d$ disjoint rectangular blocks, of size $\Delta x_i = (H_i - L_i)/g$ in their $i^{th}$ dimension.*

Let $B_{i_1,...,i_d}$ denote the block:

$$[L_1 + (i_1 - 1)\Delta x_1, L_1 + i_1\Delta x_1) \times ... \times [L_d + (i_d - 1)\Delta x_d, L_d + i_d\Delta x_d).$$

Alternatively, we denote $B_{i_1,...,i_d}$ by $B_{\mathbf{i}}$, with $\mathbf{i} = (i_1, ..., i_d)$ the unique identifier for the block. Then, two blocks $B_{\mathbf{k}}$ and $B_{\mathbf{h}}$, $\mathbf{k} \neq \mathbf{h}$, are said to be *adjacent* if $|k_i - h_i| \leq 1$, for each $i = 1, ..., d$. In this case, $B_{\mathbf{k}}$ is said to be a neighbor of $B_{\mathbf{h}}$. $Ctr_{B_{\mathbf{i}}}$ denotes the center of block $B_{\mathbf{i}}$, computed as the average of its vertices:

$$Ctr_{B_{\mathbf{i}}} = (L_1 + (i_1 - 1/2)\Delta x_1, ..., L_d + (i_d - 1/2)\Delta x_d).$$

**Definition 2.** *Let $x$ be a point and $B_{\mathbf{i}}$ be a block in the same feature space. The distance between $x$ and $B_{\mathbf{i}}$ is defined as the distance between $x$ and $Ctr_{B_{\mathbf{i}}}$.*

Note that the distance in Def. 2 can be any kind of distance. In the following, we use Euclidean distance to be the distance between a point and a block.

### 2.2 Quantization of the Feature Space

The first step of ANNCAD is to partition the feature space into a discretized space with $g^d$ blocks as in Def. 1. It is advisable to choose different sizes of grid according to system resource constraints and desirable fineness of a classifier. For each nonempty block, we count the number of training points contained in it for each class. Now we get the distribution of the data entities in each class. To decide whether we need to start with a finer resolution feature space, we then count the number of training points that do not belong to the majority class of its block as a measure of the training error. We then calculate the coarser representations of the data by averaging the $2^d$ corresponding blocks in the next finer level. We illustrate the above process by Example 1.

*Example 1.* A set of 100 two-class training points in the 2-D unit square is shown in Fig. 1(a). There are two classes for this data set, where a circle (resp. triangle) represents a training point of class I (resp. II). First we separate the training points of each class, discretize them using a $4 \times 4$ grid and count the number of training points for each block to get the data distribution of each class (see Fig. 1(b)). Moreover, Fig. 1(c)-(d) show the coarser representations of the data.

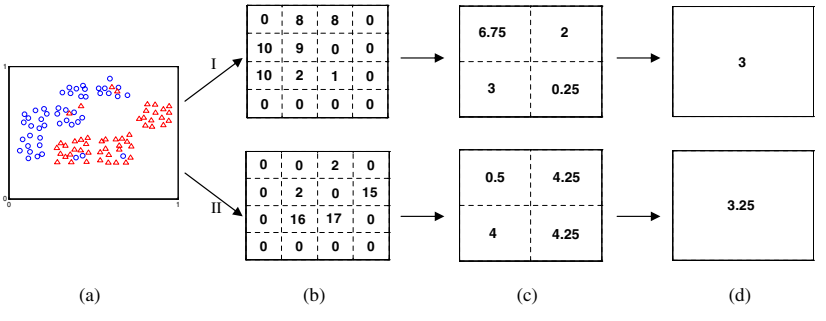


**Fig. 1.** Multi-resolution representation of a two-class data set

Due to the problem of the curse of dimensionality, the storage amount is exponential in the number of dimensions. To deal with this, we store the nonempty blocks in the leaf nodes of a $B^+$-tree using their z-values [21] as keys. Thus the required storage space is much smaller and is bounded by $O(\min(N, g^d))$ where $N$ is the number of training samples. For instance, in Fig. 1, we only need to store information for at most 8 blocks even though there are 100 training points in the $4 \times 4$ blocks feature space. To reduce space usage, we may only store the data array of the finest level and calculate the coarser levels on the fly when building a classifier. On the other hand, to reduce time complexity, we may precalculate and store the coarser levels. In the following discussion, we assume that the system stores the data representation of each level.

### 2.3 Building a Classifier and Classifying Test Points

The main idea of ANNCAD is to use a multi-resolution data representation for classification. Notice that the neighborhood relation strongly depends on the quantization process. This will be addressed in next subsection by building several classifier ensembles using different grids obtained by subgrid displacements. Observe that in general, the finer level the block can be classified, the shorter distance between this block and the training set. Therefore, to build a classifier and classify a test point (see Algorithms 1 and 2), we start with the finest resolution for searching nearest neighbors and progressively consider the coarser resolutions, in order to find nearest neighbors adaptively.

We first construct a single classifier as a starting point (see Algorithm 1). We start with setting every block to have a default tag $U$ (Non-visited). In the finest level, we classify any nonempty block with its majority class label. We then classify any nonempty block of every lower level as follows: We label the block by its majority class label if the majority class label has more points than the second majority class by a threshold percentage. If not, we use a specific tag $M$ (Mixed) to label it.

**Algorithm 1.** BuildClassifier($\{\mathbf{x}, y\}|\mathbf{x}$ is a vector of attributes, $y$ is a class label.)

**Quantize** the feature space containing $\{\mathbf{x}\}$
**Label** majority class for each nonempty block in the finest level
**For** each level $i = \log(g)$ downto 1
  **For** each nonempty block $B$
    **If** |majority $c_a$| − |2nd majority $c_b$| > threshold %, label class $c_a$
    **else** label tag $M$
**Return** Classifier

**Algorithm 2.** TestClass(test point: $\mathbf{t}$)

**For** each level $i = \log(g) + 1$ downto 1
  **If** label of $B^i(\mathbf{t})$ <> $U$  /*$B^i(\mathbf{t})$ is nonempty */
    **If** label of $B^i(\mathbf{t})$ <> $M$, class of $\mathbf{t}$ = class of $B^i(\mathbf{t})$
    **else** class of $\mathbf{t}$ = class of NN of $B^{i+1}(\mathbf{t})$ /*$B^{i+1}(\mathbf{t})$ contains $\mathbf{t}$ in level $i + 1$*/
    **Break**
**Return** class label for $\mathbf{t}$, $B^i(\mathbf{t})$

*Example 2.* We build a classifier for the data set of Example 1 and set the threshold value to be 80%. Fig. 2(a), (b) and (c) show the class label of each nonempty block in the finest, intermediate and coarsest resolution respectively.

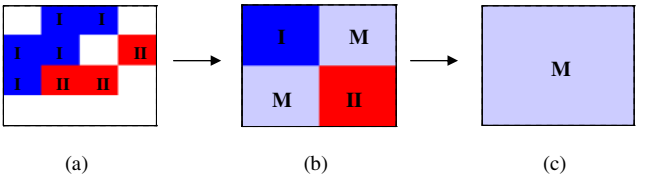


**Fig. 2.** Hierarchical structure of classifiers

For each level $i$, a test point $\mathbf{t}$ belongs to a unique block $B^i(\mathbf{t})$. We search from the finest to the coarsest level until reaching a nonempty block $B^i(\mathbf{t})$. If the label of $B^i(\mathbf{t})$ is one of the classes, we label the test point by this class. Otherwise, if $B^i(\mathbf{t})$ has tag $M$, we find the nearest neighbor block of $B^{i+1}(\mathbf{t})$ where $B^{i+1}(\mathbf{t})$ is a block containing $\mathbf{t}$ in level $i+1$. To reduce the time spent, we only consider the neighbors of $B^{i+1}(\mathbf{t})$ which belong to $B^i(\mathbf{t})$ in level $i$. It is very easy to access these neighbors as they are also neighbors of $B^{i+1}(\mathbf{t})$ in the $B^+$-tree with their z-values as keys. Note that $B^{i+1}(\mathbf{t})$ must be empty, otherwise we should classify it at level $i+1$. But some of the neighbors of $B^{i+1}(\mathbf{t})$ must be nonempty as $B^i(\mathbf{t})$ is nonempty. We simply calculate the distance between test point $\mathbf{t}$ and each neighbor of $B^{i+1}(\mathbf{t})$ and label $\mathbf{t}$ by the class of NN.

*Example 3.* We use the classifier built in Example 2 to classify a test point $\mathbf{t} = (0.6, 0.7)$. Starting with the finest level, we found that the first nonempty block containing $\mathbf{t}$ is $[0.5, 1) \times [0.5, 1)$ (see Fig. 3(b)). Since it has tag $M$, we calculate the distance between $\mathbf{t}$ and each nonempty neighboring block in the next finer level ($[0.75, 1) \times [0.5, 0.75), [0.5, 0.75) \times [0.75, 1)$). Finally, we get the nearest neighboring block $[0.75, 1) \times [0.5, 0.75)$ and label $\mathbf{t}$ to be class I (see Fig.

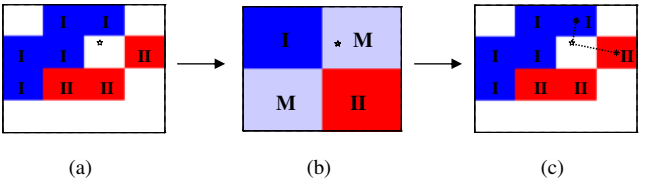


**Fig. 3.** Hierarchical classifier access

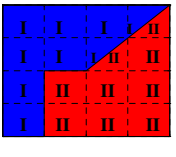


**Fig. 4.** The combined classifier

3(c)). When we combine the multi-resolution classifier of each level, we get a classifier for the whole feature space (see Fig. 4).

### 2.4 Incremental Updates of Classifiers

The main requirement of a data stream classification algorithm is that it is able to update classifiers incrementally and effectively when a new tuple arrives. Moreover, updated classifier should be adapt to concept drift behaviors. In this subsection, we present incremental update process of ANNCAD for a stationary data, without re-scanning the data and discuss an exponential forgetting technique to adapt to concept drifts.

Because of the compact support property, arrival of a new tuple only affects the blocks of the classifier in each level containing this tuple. Therefore, we only need to update the data array of these blocks and their classes if necessary. During the update process, the system may run out of memory as the number of nonempty blocks may increase. To deal with this, we may simply remove the finest data array, multiple the entries of the remaining coarser data arrays by $2^d$, and update the quantity $g$. A detailed description of updating classifiers can be found in Algorithm 3. This solution can effectively meet the memory constraint.

**Algorithm 3.** UpdateClassifier(new tuple: **t**)

**For** each level $i = \log(g) + 1$ downto 1
  **Add** $\delta_t / 2^{d \times (\log(g)+1-i)}$ to data array $\Phi^i$
    /*$\delta_t$ is a matrix with value 1 in the corr. entry of $t$ and 0 elsewhere.*/
  **If** $i$ is the finest level, label $B^i(\mathbf{t})$ with the majority class
  **else if** |majority $c_a$| − |2nd majority $c_b$| > threshold %, label $B^i(\mathbf{t})$ by $c_a$
    **else** label $B^i(\mathbf{t})$ by tag $M$
**If** memory runs out,
  **Remove** the data array of level $\log(g) + 1$
  **For** each level $i = \log(g)$ downto 1, $\Phi^i = 2^d \cdot \Phi^i$
  **Label** each nonempty block of the classifier in level $\log(g)$ by its majority class
  **Set** $g = g/2$
**Return** updated classifier

**Exponential Forgetting.** If the concept of the data changes over time, a very common technique called exponential forgetting may be used to assign less weight to the old data to adapt to more recent trend. To achieve this, we multiply an exponential forgetting factor $\lambda$ to the data array, where $0 \leq \lambda \leq 1$. For each level $i$, after each time interval $t$, we update the data array $\Phi^i$ to be:

$$\Phi^i|_{(n+1)t} \leftarrow \lambda \Phi^i|_{n \cdot t}$$

where $\Phi^i|_{n \cdot t}$ is the data array at time $n \cdot t$. Indeed, if there is no concept change, the result of classifier will not be affected. If there is a concept drift, the classifier

can adapt to the change quickly since the weight of the old data is exponentially decreased. In practice, an exponential forgetting technique is easier to implement than a sliding window because we need extra memory buffer to store the data of the most current window for implementing the sliding window.

### 2.5 Building Several Classifiers Using Different Grids

As mentioned above, the neighborhood relation strongly depends on the quantization process. For instance, consider the case that there is a training point $u$ which is close to the test point $v$ but they are located in different blocks. Then the information on $u$ may not affect the classification of $v$.

To overcome the problem of initial quantization process, we build several classifier ensembles starting with different quantization space. In general, to build $n^d$ different classifiers, each time we shift $\frac{1}{n}$ of the unit length of feature space for a set of selected dimensions. Fig. 5 shows a reference grid and its 3 different shifted grids for a feature space with $4 \times 4$ blocks. For a given test point $\mathbf{t}$, we use these $n^d$ classifiers to get $n^d$ class labels and selected blocks $B^i(\mathbf{t})$ of $\mathbf{t}$ in each level $i$, starting from the finest one. We then choose the majority class label. If there is tie, we calculate the distance between each selected block $B^i(\mathbf{t})$ with majority class label and $\mathbf{t}$ to find the closest one. Algorithm 4 shows this classifying process using $n^d$ classifiers.

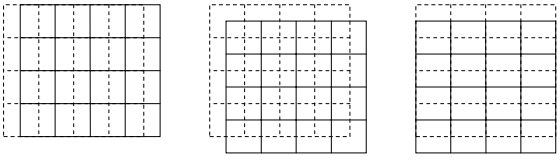

**Fig. 5.** An example of 4 different grids for building 4 classifiers

**Algorithm 4.** Test$n^d$Class(objects: $\mathbf{t}$)

**For** each level $i = \log(g) + 1$ downto 1
  **Get** the label of $\mathbf{t}$ for each classifier
  **If** there is a label <> U, choose the majority label
    **If** there is a tie, label $\mathbf{t}$ by class of $B^i(\mathbf{t})$ with closest center to $\mathbf{t}$
    **Break**
**Return** class label for $\mathbf{t}$

The following theorem shows that the approximation error of finding nearest neighbors decreases as the number of classifier ensembles increases.

**Theorem 1.** *For d attributes, let x be the test point and Y be the set of training points which are in blocks containing x of those $n^d$ classifiers. Then, for every training point $z \notin Y$, dist(x, y) < $(1 + \frac{1}{n-1})$*dist(x, z) for every $y \in Y$.*

*Proof.* For simplicity, we consider the case when $d = 1$. This proof works for any $d$. For $d = 1$, we build $n$ classifiers, where each classifier $i$ use the grid that is shifted $\frac{i}{n}$ unit length from the original grid. Let $\epsilon$ be the length of a block. Consider a test point $x$, $x$ belongs to an interval $I_k$ for classifier k. Note that $[x - \frac{n-1}{n}\epsilon, x + \frac{n-1}{n}\epsilon] \subset \bigcup I_k \subset [x - \epsilon, x + \epsilon]$. Hence, the distance between $x$ and its nearest neighbor that we found must be less than $\epsilon$. Meanwhile, the points that we do not consider should be at least $\frac{n-1}{n}\epsilon$ far away from $x$. If $z \notin Y$, $\frac{dist(x,y)}{dist(x,z)} < \frac{\epsilon}{(n-1)\epsilon/n} = (1 + \frac{1}{n-1})$ for every $y \in Y$.

The above theorem shows that the classification result using one classifier does not have any guarantee about the quality of the nearest neighbors that it found because the ratio of approximation error will tend to infinity. When $n$ is large enough, the set of training points selected by those classifier ensembles are exactly the set of training points with distance $\epsilon$ from the test point. To achieve an approximation error bound guarantee, theoretically we need an exponential number of classifiers. However, in practice, we only use two classifiers to get a good result. Indeed, experiments in §3 show that few classifiers can obtain a significant improvement at the beginning. After this stage, the performance will become steady even though we keep increasing the number of classifiers.

### 2.6 Properties of ANNCAD

As ANNCAD is a combination of multi-resolution and adaptive nearest neighbors techniques, it inherits both their properties and their advantages.

- *Compact support:* The locality property allows a fast update. As a new tuple arrival only affects the class of the block containing it in each level, the incremental update process only costs a constant time (number of levels).
- *Insensitivity to noise:* We may set a threshold value for classifying decisions to remove noise.
- *Multi-resolution:* This algorithm makes it easy to build multi-resolution classifiers. Users can specify the number of levels to efficiently control the fineness of the classifier. Moreover, one may optimize the system resource constraints and easy to adjust on the fly when the system runs out of memory.
- *Low complexity:* Let $g$, $N$ and $d$ be the number of blocks of each dimension, training points and attributes respectively. The time spent on building a classifier is $O(\min(N, g^d))$ with constant factor $\log(g)$. For the time spent on classifying a test point, the worst case complexity is $O(\log_2(g) + 2^d)$ where the first part is for classifying a test point using classifiers and the second part is for finding its nearest neighbor which is optional. Also, the time spent for updating classifiers when a new tuple arrives is $\log_2(g) + 1$. Comparing with the time spent in VFDT, our method is more attractive.

## 3 Performance Evaluation

In this section, we first study the effects on parameters for ANNCAD by using two synthetic data sets. We then compare ANNCAD with VFDT and CVFDT

on three real-life data sets. To illustrate the approximation power of ANNCAD, we include the results of *Exact ANN*, which computes ANN exactly, as controls. *Exact ANN*: For each test point $t$, we search the area within 0.5 block side length distance. If the area is nonempty, we classify $t$ as the majority label of all these points in this area. Otherwise, we expand the searching area by doubling the radius until we get a class for $t$. Note that the time and space complexities of *Exact ANN* are very expensive making it impractical to use.

### 3.1 Synthetic Data Sets

The aim of this experiment is to study the effect on the initial resolution for ANNCAD. In this synthetic data set, we consider a 3-D unit cube. We randomly pick 3k training points and assign those points which are inside a sphere with center (0.5, 0.5, 0.5) and radius 0.5 to be class 0, and class 1 otherwise. This data set is effective to test the performance of a classifier as it has a curve-like decision boundary. We then randomly draw 1k test points and run ANNCAD starting with different initial resolution and 100% threshold value. In Fig. 6(a), the result shows that a finer initial resolution gets a better result. This can be explained by the fact that we can capture a curve-like decision boundary if we start with a finer resolution. On the other hand, as discussed in last section, the time spent for building a classifier increases linearly for different resolutions. In general, we should choose a resolution according to system resource constraints.

The aim of this experiment is to study the effect on number of classifier ensembles for ANNCAD. As in the previous experiment, we randomly pick 1k training examples and assign them labels. We then randomly draw 1k test points and test them based on the voting result of these classifiers. We set $16 \times 16 \times 16$ blocks for the finest level and 100% threshold value. In Fig. 6(b), the result shows that having more classifiers will get a better result in the beginning. The performance improvement becomes steady even though we keep increasing the number of classifiers. It is because there is no further information given when we increase the number of classifiers. In this experiment, we only use 2 or 3 classifiers to obtain a competitive result with the *Exact ANN* (90.4%).

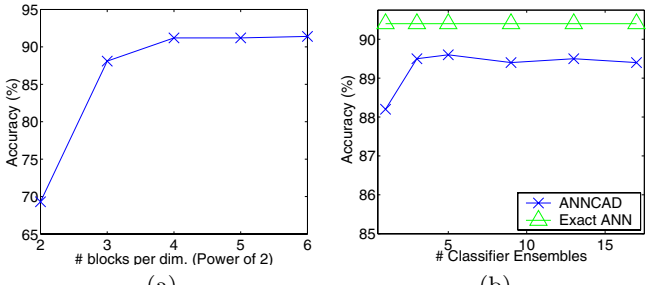


**Fig. 6.** Effect on initial resolutions and number of classifiers

### 3.2 Real Life Data Sets

The aim of this set of experiments is to compare the performance of ANNCAD with that of VFDT and CFVDT on stationary and time-changing real-life data sets respectively. We first used a letter recognition data set from the UCI machine learning repository web site [2]. The objective is to identify a black-and-white pixel displays as one of the 26 English alphabet. In this data set, each entity is a pixel display for an English alphabet and has 16 numerical attributes to describe its pixel displays. The detail description of this data set is provided in [2]. In this experiment, we use 15k tuples for training set with 5% noise added and 5k for test set. We obtain noisy data by randomly assigning a class label for 5% training examples. For ANNCAD, we set $g$ for the initial grid to be 16 units and build two classifiers. Moreover, since VFDT needs a very large training set to get a fair result, we rescan the data sets up to 500 times for VFDT. So the data set becomes 7,500,000 tuples. In Fig. 7(a), the performance of ANNCAD dominates that of VFDT. Moreover, ANNCAD only needs one scan to achieve this result, which shows that ANNCAD even works well for a small training set.

The second real life data set we used is the Forest Cover Type data set which is another data set from [2]. The objective is to predict forest cover type (7 types). For each observation, there are 54 variables. Neural network (backpropagation) was employed to classify this data set and got 70% accuracy, which is the highest one recorded in [2]. In our experiment, we used all the 10 quantitative variables. There are 12k examples for training set and 90k examples for testing set. For ANNCAD, we scaled each attribute to the range $[0, 1)$. We set $g$ for the initial grid to be 32 units and build two classifiers. As the above experiment, we rescan the training set up to 120 times for VFDT, until its performance becomes steady. In Fig. 7(b), the performance of ANNCAD dominates that of VFDT. These two experiments show that ANNCAD works well in different kinds of data sets.

We further tested ANNCAD in the case when there are concept drifts in data set. The data we used was extracted from the census bureau database [2]. Each observation represents a record of an adult and has 14 attributes including age, race etc. The prediction task is to determine whether a person makes over 50K a year. Concept drift is simulated by grouping records with same race (Amer-Indian-Eskimo(AIE), Asian-Pac-Islander(API), Black(B), Other(O), White(W)). The distribution of training tuples of each race is shown in Fig. 7(c). Since the models for different races of people should be different, concept drifts are introduced when $n = 311, 1350, 4474, 4746$. In this experiment, we used the 6 continuous attributes. We used 7800 examples for learning and tested the classifiers for every 300 examples. For ANNCAD, we build two classifiers and set $\lambda$ to be 0.98 and $g$ for the initial grid to be 64 units. We scaled the attribute values as mentioned in the previous experiment. The results are shown in Fig. 7(c). The curves show that ANNCAD keeps improving in each region. Also, as mentioned in §2.6, computations required for ANNCAD are much lower than CVFDT.

Moreover, notice that ANNCAD works almost as well as *Exact ANN* on these three data sets, which demonstrates its excellent approximation ability.

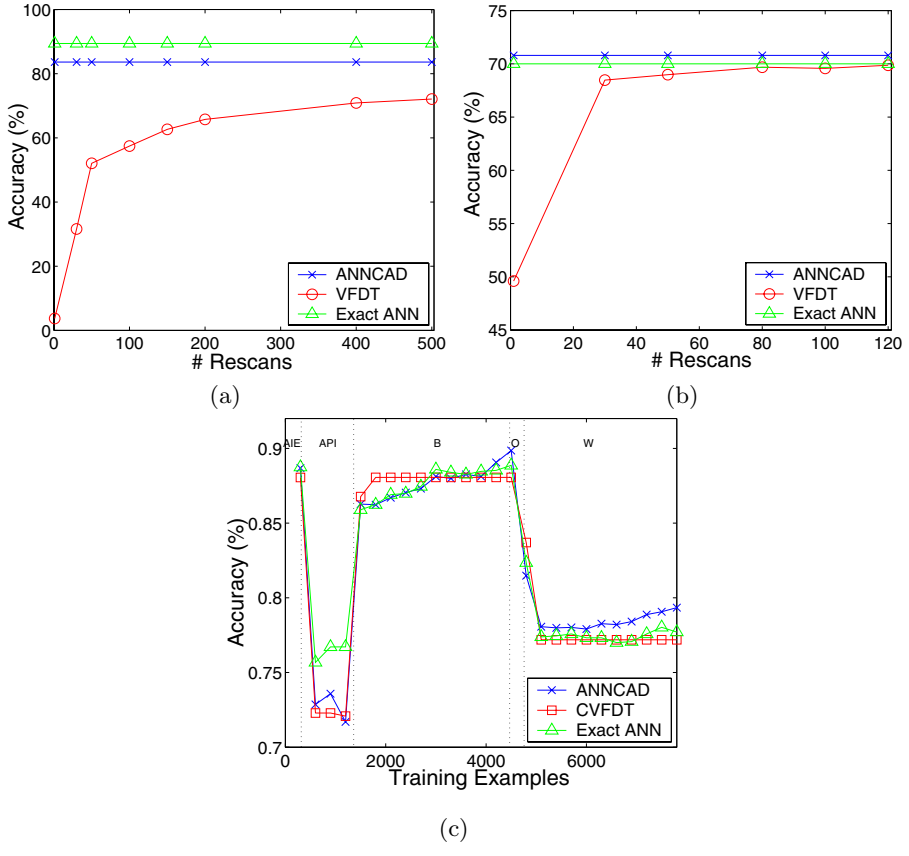


**Fig. 7.** Three real-life data sets:(a) Letter Recognition (b) Forest Covertype (c) Census

## 4 Conclusion and Future Work

In this paper, we proposed an incremental classification algorithm ANNCAD using a multi-resolution data representation to find adaptive nearest neighbors of a test point. ANNCAD is very suitable for mining data streams as its update speed is very fast. Also, the accuracy compares favorably with existing algorithms for mining data streams. ANNCAD adapts to concept drift effectively by the exponential forgetting approach. However, the very detection of sudden concept drift is of interest in many applications. The ANNCAD framework can also be extended to detect concept drift–e.g. changes in class label of blocks is a good indicator of possible concept drift. This represents a topic for our future research.

**Acknowledgement.** This research was supported in part by NSF Grant No. 0326214.

## References


1. R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *SIGMOD 1998: 94–105.*
2. C. L. Blake and C. J. Merz. UCI Repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
3. B. Babcock, S. Babu, R. Motawani and J. Widom. Models and issues in data stream systems. *PODS 2002: 1–16.*
4. J. Bentley. Multidimensional binary search trees used for associative searching. *Communication of the ACM 18(9): 509–517 (1975).*
5. F. Chu and C. Zaniolo. Fast and light boosting for adaptive mining of data streams. *PAKDD 2004: 282–292.*
6. C. Domeniconi, J. Peng and D. Gunopulos, Locally adaptive metric nearest-neighbor classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence 24(9): 1281–1285 (2002).*
7. P. Domingos and G. Hulten. Mining high-speed data streams. *KDD 2000: 71-80.*
8. L. Golab and M. Özsu. Issues in data stream management. *ACM SIGMOD 32(2): 5–14 (2003).*
9. J. Han and M. Kamber. *Data Mining – Concepts and Techniques (2000).* Morgan Kaufmann Publishers.
10. G. Hulten, L. Spence and P. Domingos. Mining time-changing data streams. *KDD 2001: 97–106.*
11. P. Indyk, R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *STOC 1998: 604–613.*
12. P. Indyk. Dimensionality reduction techniques for proximity problems. *ACM-SIAM symposium on Discrete algorithms 2000: 371–378.*
13. P. Indyk. High-dimensional computational geometry. *Dept. of Comput. Sci., Stanford Univ., 2001.*
14. E. Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM J. Comput. 30(2): 457–474 (2000).*
15. G. Sheikholeslami, S. Chatterjee and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. *VLDB 1998: 428–439.*
16. G. Strang and T. Nguyen. *Wavelets and Filter Banks (1996).* Wellesley-Cambridge Press.
17. W. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. *SIGKDD 2001: 377–382.*
18. H. Wang, W. Fan, P. Yu and J. Han. Mining concept-drifting data streams using ensemble classifiers. *SIGKDD 2003: 226–235.*
19. W. Wang, J. Yang and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining *VLDB 1997: 186–195.*
20. D. Wettschereck and T. Dietterich. Locally Adaptive Nearest Neighbor Algorithms. *Advances in Neural Information Processing Systems 6: 184–191 (1994).*
21. C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian and R. Zicari. *Advanced Database Systems (1997).* Morgan Kaufmann Press.