# C-PRUNER: AN IMPROVED INSTANCE PRUNING ALGORITHM

KE-PING ZHAO[1], SHUI-GENG ZHOU [1, 2], JI-HONG GUAN[3], AO-YING ZHOU[1]

[1] Department of Computer Science & Engineering, Fudan University, Shanghai 200433, China
[2] State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China
[3] School of Computer Science, Wuhan University, Wuhan 430079, China
E-MAIL:{kpzhao, sgzhou, ayzhou}@fudan.edu.cn; zhousg@whu.edu.cn; jhguan@wtusm.edu.cn

**Abstract:**

**Instance-based learning faces the problem of deciding which instances could be discarded in order to save computation and storage costs. For large instance bases classifier suffers from large memory requirements and slow response. And present noisy instances may deteriorate the classification accuracy. This paper analyzes the strength and weakness of some of the existing algorithms for instance pruning, and propose an improved method C-Pruner. Experiments over real-world datasets verify C-Pruner's superior to the existing methods in classification accuracy.**

**Keywords:**

**Instance-based learning; *k*NN classification; Instance Pruning**

## 1 Introduction

The Nearest Neighbor Classifier is a simple supervised concept learning scheme which predicts the class of the unknown (i.e. unclassified) instance based on the existing training set, in which the instances' class are already labeled [1]. During classification, the nearest neighbor classifier finds out the nearest neighbor or the several nearest neighbors of the unlabeled instance and make a decision of its' class by observing the class(es) of its neighbor(s) found. In literature, nearest neighbor classification is also referred as *k*NN (k-Nearest Neighbor) classification, where *k* is the number of nearest neighbors used by the learner. The learning scheme based on *k*NN classification is termed as Instance-based learning (IBL) [2], Memory-based learning (MBL) [3], or Case-based reasoning (CBR) [4] respectively corresponding to different contexts.

In the early days of *k*NN research, because the size of the training instance set is relatively small, all the training instances could be stored in main memory and the classification efficiency is acceptable. However, as *k*NN is applied in more complex application domains, the set of training instances becomes larger, the requirements for storage of training instances and efficiency of classification are intensified. One reasonable solution is the technique of *Instance pruning* [5-20].

Instance pruning method tries to prune the original training set to get a smaller subset of it, which enables the classifier to get nearly similar (or better) classification accuracy to (or than), that of the original set. The algorithms of instance pruning seek to discard the *harmful* instances (e.g. *noisy* instances) and *superfluous* instances as to achieve a tradeoff between the reduction of the storage of the training set and the accuracy of classification. Instance pruning is studied in IBL area as well as the community of Case-based Reasoning as a task of Case-based Maintenance [5-7].

In this paper, we first analyze the strength and weakness of major existing instance pruning algorithms, and then present an improved algorithm, which is termed as ***C-Pruner*** because it ***combines*** major advantages of different existing instance-pruning algorithms. *C-Pruner* conducts instance pruning more carefully to avoid the deletion of some critical instances, consequently improves the classification accuracy. We show that our algorithm improve the performance of *k*NN classifier as expected through extensive experiments over real-world datasets.

The rest of this paper is organized as following. The next section examines the related work on instance pruning for classification problem, strength and weakness of major existed algorithms are analyzed. In section 3, we give the *C-pruner* algorithm in details. The results of the experiments together with the discussion of them will be presented in section 4. Finally we conclude the paper and highlight future work in section 5.

## 2 Related Work

Since the early days of nearest neighbor classification

Table 1.Comparisons of major instance pruning algorithms

| Algorithms | Application Field | Process | Type of Instances pruned | Instances altered | Accuracy of Classification |
|---|---|---|---|---|---|
| CNN [8] | NN | Incremental | Internal instances | No | Maintain |
| SNN [9] | NN | Decremental | Internal instances | No | Maintain |
| RNN [10] | NN | Decremental | Internal instances | No | Maintain |
| ENN [11] | NN | Decremental | Noisy instances Border instances | No | Improve |
| Tomek [12] | NN | Decremental | Noisy instances Border instances | No | Improve |
| Chang [13] | NN | Decremental | All | Yes | Maintain |
| RISE2.0 [14] | IBL | Decremental | All | Yes | Maintain |
| NGE [20] | NN | Decremental | All | Yes | Maintain |
| IB Family [2] | IBL | Incremental | Internal instances | No | Hybrid |
| Cameron-Jones [15] | IBL | Incremental | All | No | Hybrid |
| TIBL [16] | IBL | | Border instances | No | Maintain |
| Footprint Deletion [5] | CBR | Decremental | All | No | Depends on the amount of instance-pruning |
| DROP Family [17] | IBL | Decremental | All | No | Hybrid |
| Zhu [7] | CBR | Incremental | All | No | Maintain |
| ICF [18] | IBL | Decremental | Internal instances | No | Maintain |

being proposed and applied in the 1960's [8], many schemes for instance pruning have been developed to save storage and computational costs of the classifiers. The objective of the algorithms for instance pruning are to find a subset S of the training set T to replace T during classification. These algorithms can be studied and classified from different perspectives. First, in the light of research area from which these algorithms are originated, we can see some of them are designed for NN classification, some are proposed by the community of Cased-based Reasoning, while the others are from IBL area. By considering the direction of instance pruning process, these algorithms can be divided into two categories: incremental and decremental. An incremental method begins with an empty S, and adds the instances satisfying some rules to S from the training set T, while a decremental one begins with $S = T$, then discards each instance from $S$ if it fulfils some pruning criteria. With respect to the types of instances pruned by the algorithms, some algorithms mainly prune noisy instances, some aim to prune internal instances or border instances, while some other prune all of them. And in terms of whether the original instances are altered during pruning, some algorithms create new instances to represent original instances, while most methods maintain the original instances untouched. At last, with respect to the effect of instance pruning, some can improve the classification accuracy, some just maintain the classification competence, while others may either improve or maintain classification competence, which depend on the concrete application fields, so these algorithms are referred as *hybrid* in this paper. In table 1, we show the major instance pruning algorithms and comparisons among them from different perspectives mentioned above.

Among these algorithms presented in table 1, *CNN*, *IB* family, *DROP* family, *ICF*, *Footprint Deletion* and *Zhu*'s are of the most influential in the research community. And *DROP* family and *ICF* are the-state-of-the-art of instance pruning algorithms. *CNN* is one of the first attempts to reduce the size of the training set. It starts by selecting randomly one instance from each class of *T* and put them in *S*, then checks every other instance of *T* whether it can be classified correctly with instances in *S*: if not, the instance is also put into *S*, otherwise, it is discarded. This process is repeated until all the instances of the original set *T* can be classified correctly by *S*. *Footprint Deletion* is mainly used in the field of case-based maintenance, and *Zhu*'s algorithm is developed to overcome the drawback of *Footprint Deletion*. However the advantage of *Zhu*'s algorithm have not been approved by enough experiments. *ICF* have adopted some concepts from *Footprint Deletion* and *DROP* family in description of the algorithm, while the rules for pruning instances are simpler than *Footprint Deletion* and *DROP* family. In what follows, we focus our discussion on *DROP* family and *IB* family.

*IB* family (*IB1*, *IB2*, *IB3*) was described by [3] in 1991 to reduce the training set without compromising classification accuracy. *IB1*, used as a baseline is actually identical to 1-NN algorithm. *IB2*, similar to *CNN*, it begins with an empty set $S$ and each instance of the training set $T$ is added to $S$ if it can not be correctly classified by $S$. And *IB3* was introduced to reduce the noise sensitivity of *IB2* by only reserving the *acceptable* instances of $T$. The essence of *IB* family algorithms is that they seek to discard those *superfluous* instances (corresponding to internal instances here), which can be correctly classified by *k*NN scheme with the remaining instances.

*DROP* family (*DROP1*, *DORP2* and *DROP3*) is decremental instance-pruning algorithms guided by the $k$ *nearest neighbors* set and *associates* set for each instance to reduce the size of training set. The associates of an instance $i$ are those instances who have $i$ as one of its $k$ nearest neighbors. *Drop* family starts with the original set and reduces each instance in an ordered way if at least as many of its associates can be correctly classified without it. The basic idea is to discard the *non-critical* instances, which are also mainly *internal instances.*

As we mentioned above, both the superfluous instances discarded by *IB* family and the non-critical instances removed by *DROP* family mainly correspond to internal instances. The reason that there are two different ways to prune the same kind of instances lies in the fact that these two algorithms conduct instance-pruning with respect to different criteria: IB family considers *superfluousness* (whether it can be correctly classified with other instances by *k*NN classification) of instance, while *DROP* family is based on *criticalness* (which it will influence other instances' correct classification in *k*NN scheme) of instances. Both the two series of algorithms have drawbacks, because there exist the cases that some superfluous instances are also critical instances which are not suitable to be discarded, and on the other hand, not all non-critical instances are superfluous ones, that is, some non-critical should not be discarded in order to maintain classification accuracy.

## 3 The C-Pruner

Our algorithm for instance pruning, *C-Pruner*, considers both *superfluousness* and *criticalness*, thus make our approach more careful when selecting instances to discard than the existing algorithms. In this section, we introduce some concepts underlying our algorithm. Then we propose the rule applied to identify which instances can be deleted, as well as the rule to identify noisy instances which should be discarded before the process of instance pruning.

For convenience, firstly we introduce some notations that will be used repeatedly in what follows. A training set $T$ consists of a set of instances $\{p_1, p_2, \ldots, p_n\}$. If $p$ is an instance belonging to $T$, *kNN(p)* is denoted as the set of k nearest neighbors of $p$. An instance $p$ also has a nearest *enemy,* which is the nearest instance with different class from its own.

**Definition 1.** For an instance $p$ in $T$, the $k$ nearest neighbors of $p$ make up its *k-reachability* set, denoted as *k-reachability*($p$), which is defined formally as:

$$k\text{-}reachability(p) = \{p_i \mid p_i \in T \text{ and } p_i \in k\text{NN}(p)\}$$

**Definition 2.** For an instance $p$ in $T$, those instances with similar class label to that of $p$, and have $p$ as one of their nearest neighbors are called the *k-coverage* set of $p$, denoted as *k-coverage*($p$). In a formal way:

$$k\text{-}coverage(p) = \{\, p_i \mid p_i \in T,\, p_i \in class(p) \text{ and } p \in k\text{-}reachability(p_i)\}.$$

In fact, *k-reachablity*($p$) is identical to *k*NN($p$). In our definition, *k*-coverage($p$) only contains only those instances which belong to the same class as $p$, which is different from the definition by [17-18]. The reason will be presented later. Based on Definition 1 and 2, we introduce the definition of three kinds of instances, *i.e. superfluous instances*, *Critical instances* and *Noisy instances*.

**Definition 3.** For instance $p$ in $T$, if $p$ can be classified correctly by *k-reachability*($p$), than we say p is *implied* by *k-reachability*($p$), and $p$ is a *superfluous* instance in $T$.

**Definition 4.** For instance $p$ in $T$, $p$ is a *critical* instance, if the following condition holds:

At least one instance $p_i$ in *k-coverage*($p$) is not implied by *k- reachability* ($p_i$), or

After $p$ is deleted, at least one instance $p_i$ in *k-coverage*($p$) is not implied by *k-reachability*($p_i$).

Now, let us review our definition of *k-coverage*($p$), which only includes instances having the same class as $p$ has. This is based on the following observation. The objective of instance pruning is to reduce the size of training set, and at the same time to try to avoid the deterioration of classification accuracy caused by instance removal. However, the deletion of an instance $p$ only impacts classification decision of instances with similar class of $p$'s, while benefits classification decision of the instances with different class label from $p$'s. So we only include those instances with similar class to that of $p$ when considering whether $p$ is *critical.*

**Definition 5.** For instance $p$ in $T$, if $p$ is not a superfluous instance and $|k\text{-}reachability(p)| > |k\text{-}coverage(p)|$, then $p$ is a *noisy* instance.

Here, we define noisy instances as those that are incorrectly classified by its neighbors. The reason is like this: a noisy instance is far away from the instances of its own class, i.e., its neighbors are instances of enemy classes, thus the result class inferred from its *k-reachability* is prone to being different from its *genuine* class. On the other hand, its size of *k-coverage* is usually smaller than that of its *k-reachability*. Certainly, the definition of noisy instance is not very strict, for according to definition 5 some border instances between different classes may be considered as noisy instances. However, Definition 5 describes the characteristics of most noisy instances. The definition adopted by most existing algorithms considers only the first condition, which make them less strict.

A superfluous instance is *superfluous* because it can be implied by other instances, so as a training instance, it can be replaced by other instances. And a critical instance is *critical* for it is essential to correctly classifying some instances. If it is discarded, there are some of its neighbors cannot be correctly classified by their *k-reachability* set. For *k*NN classification, a superfluous instance can obviously be discarded, however a critical instance must remained in the training set in order to preserve the classification accuracy. So an instance is both superfluous and critical, it cannot be deleted. If an instance can be deleted safely, it must be one of the following two cases: 1) it is noisy; 2) it is superfluous, but not critical.

Based on the discussion above, we give the following rule to guide instance pruning.

**Rule 1.** Instance pruning rule

For an instance $p$ in training set $T$, if it can be pruned, it must satisfy one of the following two conditions:

It is a noisy instance;

It a superfluous instance, but not a critical one.

In rule 1, the first condition is for removing noisy instances; the second one involves two sub-conditions: 'is superfluous instance' is the necessary condition, and 'not a critical one' ensures that the deletion of the instance does not deteriorate classification accuracy.

The order of instances removal is also very crucial in instance-pruning process, for the deletion of an instance will affect the decision on whether other instances can be discarded. Intuitively, for instances fulfilling rule 1, the internal instances should be removed first, so that the border instances can be kept as much as possibly. Otherwise, it may cause the domino effect of border instances removal so that a lot of border instances are removed, while most reserved instances are internal instances. This is not what we expect, because the pruning border instances may lead to loosing classification competence. Generally, internal instances have some common characteristics compared to border instances, e.g. they have more homogeneous instances in their neighbors; they are closer to the centers of their classes and farther from their nearest enemy instances.

We present a heuristic rule for deciding the order in which instances are pruned. Let H-*k*NN($p$) be the number of the instances of its class in *k*NN($p$), and D-NE($p$) be the distance of $p$ to its nearest enemy. Note that the following rule is only applied to the removal of non-noisy instances. The removal of noisy instances is not restricted by this rule.

**Rule 2.** Rule for deciding the order of instances removal.

For two prunable instances $p_i$ and $p_j$ in training set $T$,

If H-*k*NN($p_i$) > H-*k*NN($p_j$), $p_j$ should be removed before $p_j$;

If H-*k*NN($p_i$) = H-*k*NN($p_j$) and D-NE($p_i$) > D-NE($p_j$), $p_j$ should be removed before $p_j$;

If H-*k*NN($p_i$) = H-*k*NN($p_j$) and D-NE($p_i$) = D-NE($p_j$), the order of removal is random decided.

Following, we present the algorithm *C-Pruner* in Figure 1.

```
C-Pruner(T, S)
//T is the original training set, S is the pruned one.
S = T;
For all $p \in S$ do
  Compute k-rechability(p) and k-coverage(p);
For all $p \in S$ do
  If p is a noisy instance
    Remove p from S;
    For all $p_i \in$ k-coverage(p)
      Remove p from k-rechability($p_i$);
      Update k-rechability($p_i$);
    For all pi $\in$ k-rechability(p)
      Remove p from k-coverage($p_i$);
Sort the removal order of instances in S according to rule 2;
For all $p \in S$
  If p satisfies rule 1
    Remove p from S;
    For all $p_i \in$ k-coverage(p)
      Remove p from k-rechability($p_i$);
      Update k-rechability($p_i$);
Return S;
```

***Figure 1*** The C-Pruner Algorithm

C-Pruner first performs noisy instances filtering, then removes instances satisfying rule 1 in an ordered way. Lines 2-3 compute the *k-reachability* and *k-coverage* for each instance in $S$. A noise instance filtering pass is performed by lines 3-11. When a noisy instance $p$ is removed, those instances having $p$ as one of their nearest

Table 2 Comparisons of accuracy and storage among K-NN, IB3, DROP3 and C-Pruner over 15 datasets

| **Datasets** | **KNN** | | **IB3** | | **DROP3** | | **C-Pruner** | |
|---|---|---|---|---|---|---|---|---|
| | ***Acc. (%)*** | ***Stor. (%)*** | ***Acc. (%)*** | ***Stor. (%)*** | ***Acc. (%)*** | ***Stor. (%)*** | ***Acc.(%)*** | ***Stor. (%)*** |
| Anneal | 93.11 | 100 | 91.23 | 10.07 | 93.99 | 8.63 | 94.12 | 11.79 |
| Australian | 84.78 | 100 | 85.65. | 4.98 | 84.20 | 5.94 | 85.07 | 10.90 |
| Breast Cancer (WI) | 96.28 | 100 | 97.00 | 3.39 | 96.14 | 3.61 | 95.86 | 4.42 |
| Bridge | 44.00 | 100 | 44.00 | 10.05 | 44.00 | 4.28 | 44.00 | 8.06 |
| Glass | 73.83 | 100 | 62.19 | 33.18 | 64.55 | 23.68 | 68.74 | 31.62 |
| Heart (Long Beach) | 74.50 | 100 | 70.00 | 4.89 | 74.50 | 1.11 | 74.50 | 1.72 |
| Heart (Swiss) | 93.46 | 100 | 93.26 | 3.70 | 93.46 | 1.81 | 93.46 | 1.81 |
| Image Segmentation | 93.10 | 100 | 91.43 | 15.87 | 92.62 | 11.11 | 94.05 | 12.67 |
| LED Creator | 73.40 | 100 | 70.90 | 22.87 | 71.40 | 11.87 | 72.80 | 38.48 |
| Liver (Bupa) | 65.57 | 100 | 57.93 | 10.66 | 60.56 | 25.16 | 65.50 | 45.70 |
| Sonar | 87.55 | 100 | 71.24 | 12.98 | 78.00 | 26.82 | 79.90 | 35.79 |
| Soybean (large) | 88.59 | 100 | 85.65 | 30.15 | 84.65 | 25.12 | 86.60 | 27.87 |
| Vehicle | 71.76 | 100 | 66.09 | 28.41 | 65.38 | 23.09 | 67.62 | 34.13 |
| Vowel | 96.57 | 100 | 88.43 | 36.89 | 89.56 | 44.82 | 91.07 | 46.53 |
| Zoo | 94.44 | 100 | 93.33 | 30.99 | 88.89 | 20.86 | 88.89 | 24.20 |
| **Average** | **82.06** | **100** | **77.89** | **17.27** | **78.79** | **15.86** | **80.15** | **22.37** |

neighbors (i.e. the instances in *k*-coverage(*p*)) remove *p* from their *k-reachability* sets, and the instances in *p*'s k-reachability also remove p from their k-coverage set. Line 12 sorts the instances remained in S with respect to rule 2. Lines 13-18 check each instance in S to decide whether it can be removed according to rule 1. In this pass, if a instance p is removed, we only update the k-reachability of those instances in k-coverage(p).

## 4 Experimental Results

In order to verify the expected benefits of C-Pruner, we carry out experiments over 15 datasets from the Machine Learning Database Repository at the University of California, Irvine [19]. We implement three other typical instance pruning algorithm IB3, DROP3 and KNN for comparison with *C-Pruner*. *k*NN is the basic *k* nearest neighbor algorithm that uses all instances in the original training set during classification. In this section, we focus on the comparison between *C-Pruner* and *DROP3*, for *DROP3* is pruning algorithm of state of the arts. The comparison between other methods can be found in [17].

We adopted 10-fold cross-validation in the experiments. Each dataset is divided into 10 equal parts. For every algorithm included, 9 of these parts makes up the training set *T*, which is pruned and returned as *S*. Instances of the remaining part are then classified by using instances in *S*. This process will be repeated for 10 times with different combinations of the ten parts, and the average accuracy of the 10 trails is reported, as well as the average percentage of instances remained in *S*. Table 2 shows the results of accuracy and storage after instances pruning for each algorithm over the 15 datasets. And the average accuracy and storage after pruning are listed in the last row in bold.

Several observations can be made from the results. The basic *K*NN algorithm has the highest average classification accuracy, however it stores all the training instances. As expected, our approach *C-Pruner* achieves considerably high average classification accuracy, which is about 2% inferior to the *k*NN method. Moreover, *C-Pruner* outperforms *DROP3* in average classification accuracy by nearly 1.4%, and wins *IB3* by more than 2%. In fact, of all the 15 datasets tested, *C-Pruner* obtains a higher accuracy than *DROP3* in 12 datasets.

As shown in table 2, the storage requirement of *C-Pruner* is a little higher than *IB3* and *DROP3*. This is not out of our expectation because *C-Pruner* performs instance pruning in a more careful way in order to prevent from deterioration of classification accuracy. We argue that higher classification accuracy is worthy of a little more storage cost.

## 5 Conclusions and Future Work

This paper presents an improved instance-pruning algorithm *C-Pruner*, which is based on the formal definition of superfluous, critical and noisy instances. *C-Pruner* method deletes superfluous instances in a more careful way than the DROP family and IB family in order to achieve higher classification accuracy. It aims to delete the internal instances and retain the border instances. In practice, C-Pruner achieves this by discarding the superfluous but not critical instances in an ordered way after a noise-filtering pass. Experiments over 15 datasets show C-Pruner achieved higher classification accuracy than *DROP3* and *IB3* at a little storage cost as we expected.

The future work includes 1) using more datasets from different application domains to evaluate C-Pruner; 2) Combing C-pruner with prototyping techniques to build classifiers of higher performance; 3) Using domain knowledge and structure features of the datasets to enhance C-Pruner's performance.

**References**

[1] Dasarathy B V. Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. Los Alamitos, CA: IEEE Computer Society Press. 1991.

[2] Aha D., Kibler D., and Albert M., Instance-Based Learning Algorithms. Machine Learning, 6:37-66, 1991

[3] Stanfill C, Waltz D. Toward memory-based reasoning. Communications of the *ACM*, 1986, 29(12): 1213-1228.

[4] Kolodner J L. Case-based Reasoning. San Mateo, Calif.: Morgan Kaufmann, 1993.

[5] Smyth B, McKenna E. Competence Guided Instance Selection for Case-Based Reasoning. Netherlands: Kluwer Academic Publishers. 2000: 1-18.

[6] Leake D B, Wilson D C. Remembering Why to Remember: Performance-Guided Case-Base Maintenance. Proceedings of the 5th European Workshop on Case-Based Reasoning. Berlin: Springer-Verlag. 2000: 161-172.

[7] Zhu Jun, Yang Qiang. Remember to add: competence-preserving case addition policies for case base maintenance. Dean T (Ed.): Proceedings of International Joint Conference in Artificial Intelligence 1999 (IJCAI-99), San Francisco: Morgan Kaufmann, 1999: 234-241.

[8] Hart P E. The Condensed Nearest Neighbor Rule. IEEE Transactions on Information Theory, 1968, 14: 515-516.

[9] Ritter G L, Woodruff H B, Lowry S R, *et al.* An Algorithm for a Selective Nearest Neighbor Decision Rule. IEEE Transactions on Information Theory, 1975, 21(6): 665-669.

[10] Gates G W. The Reduced Nearest Neighbor Rule. IEEE Transactions on Information Theory, 1972, IT-18(3): 431-433.

[11] Wilson, Dennis L. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics*, 1972, 2(3): 408-421.

[12] Tomek I. An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*. 1976, 6(6): 448-452.

[13] Chang Chin-Liang. Finding Prototypes for Nearest Neighbor Classifiers. IEEE Transactions on Computers, 1974, 23(11): 1179-1184.

[14] Domingos P. Rule Induction and Instance-Based Learning: A Unified Approach. The 1995 International Joint Conference on Artificial Intelligence (IJCAI-95), San Francisco: Morgan Kaufmann. 1995: 1226-1232

[15] Cameron-Jones R M. Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence, 1995: 99-106.

[16] Zhang J. Selecting Typical Instances in Instance-Based Learning. Sleeman D, Edwards P (Eds.): Proceedings of the Ninth International Conference on Machine Learning (ICML'92), San Francisco: Morgan Kaufmann, 1992: 470-479.

[17] Wilson D R, Martinez T R. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 2000, 38(3): 257-286.

[18] Brighton H, Mellish C S. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*. 2002, 6(2): 153-172.

[19] Merz C. J., and Murphy P. M., UCI Repository of Machine Learning Databases. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.us/~mlearn/MlRepository.html

[20] Salzberg, S. A Nearest Hyperrectangle Learning Method. *Machine Learning*, 1991, vol. 6, 277-309.