Scalable Representative Instance Selection and Ranking

Xingquan Zhu and Xindong Wu

Department of Computer Science, University of Vermont, VT 05405

Abstract

Finding a small set of representative instances for large datasets can bring various benefits to data mining practitioners so they can (1) build a learner superior to the one constructed from the whole massive data; and (2) avoid working on the whole original dataset all the time. We propose in this paper a Scalable Representative Instance Selection And Ranking (SRISTAR pronounced 3STAR) mechanism, which carries two unique features: (1) it provides a representative instance ranking list, so that users can always select instances from the top to the bottom, based on the number of examples they prefer; and (2) it investigates the behaviors of the underlying examples for instance selection, and the selection procedure tries to optimize the expected future error. Given a dataset, we first cluster instances into small data cells, each of which consists of instances with similar behaviors. Then we progressively evaluate data cells and their combinations, and order them into a list such that the learners built from the top cells are more accurate.

1. Introduction

Having massive amounts of data does not necessarily mean that we have to use them all, even if all the data are collected from quality sources. Real-world applications often raise at least two concerns in this regard: (1) the efficiency of an algorithm on a large dataset could be unbearably low, which requires the algorithm to be conducted on a small set to reduce the computational complexity; and (2) the usefulness of some data in the dataset is questionable, even if the data are collected from trustworthy sources. Existing endeavors from data mining and machine learning have provided many solutions to resolve the first concern. E.g., using bagging [1], boosting [2], incremental learning [3], and sampling [4-5] to compromise the accuracy and efficiency in general for better performances. These methods are efficient from their own perspectives, but users still have to work on the whole original dataset all the time:

(1) **Extra data usage efforts**. Large datasets are often stored in a data repository or a warehouse, which imposes extra efforts on users to get familiar with these facilities. It often turns out to be infeasible without help from an expert.

(2) **Data accessibility and privacy**. When it comes to process the whole original data, privacy and accessibility issues become a major concern.

(3) **Flexibility**. Within the context of a massive data volume, it is difficult for users to try different mining mechanisms.

The above three issues are often solved in reality by sampling a small set of examples, which is manageable for general users. Then, preliminary mining results from this set are used to guide the subsequent procedures. On the other hand, to resolve the second real-world concern, existing research efforts, e.g., Instance Based Learning (IBL) [6], have suggested that it is very possible to refine the training set in such a way that a better learner can be constructed from the refined set.

The above observations motivate our work of selecting a representative instance subset S from a given data set D, where the instance selection procedure should carry three important features: (1) Algorithm complexity is reasonable for real-world usages; (2) The selected subset S has minimal information loss and the model built from S should be close to, if cannot outperform, the theory from D as much as possible; and (3) good scalability to fit different users' needs in instance selection. To consider the scalability of the instance selection, we adopt the concept of a ranking list for all instances in D, with $D=\{n_1, \ldots, n_N\}$ for a dataset with N instances, where n_i is an ordered instance. Whenever the user requests for a subset S of K instances, the instance selection procedure will directly select examples from the rank ordered list with $S=\{n_1,\ldots n_K\}$. There are several challenges regarding this procedure, if we consider the above three features as a whole.

1. What is an efficient ranking schedule to sort the instances based on their representation ability?
2. Given a request of K instances, how to guarantee that the top K instances selected from the ranking list can provide better performances than other selections?
3. When several distributed branches collaborate with each other to select representative instances, how to guarantee that the selected subset is globally optimal?

This paper addresses these three challenges in a novel manner. The motivation behind is simple. Assume a user wants to select K most representative instances from D. A possible solution, not necessarily the best one, motivated by the Monte Carlo sampling [7] works as follows. It continuously builds learners from a subset consisting of any K instances from D, and selects the one with the minimal error rate. This method, however, if implemented naively, would become hopelessly inefficient, because there are $\binom{K}{N} = \prod_{i=0}^{K-1}(N-i) \Big/ K$ combinations to form a K-instance subset. In addition, if the user changes the value of K, the algorithm must repeat to search for an optimal combination again. If we can merge similar instances into groups, then the representative instance selection can be regarded as a procedure of selecting the combination of the groups with the lowest expected future error. The above motivations have guided us to design 3STAR to solve the problem in an efficient way.

2. Data Cell Construction

Because instance selection crucially depends on the underlying learner, we use Naïve Bayes due to its efficiency, simplicity and popularity in handling many real-world problems.

3.1 Naïve Bayes Classification

In classification learning, each instance is described by a vector of attribute values and a class label. A set of instances with their classes, the training data, is provided. The learner is asked to predict a test instance's class according to the evidence provided by the training data. We define:

- $X <A_1, A_2, .., A_k>$ as a vector of random variables denoting the observed attribute values (an instance).
- $x <a_1, a_2, \ldots, a_k>$ as a particular observed attribute value vector (a particular instance).
- $X=x$ as shorthand for $X_1=x_1 \wedge X_2=x_2 \wedge .. \wedge X_k=x_k$.
- Y as a random variable denoting the class of an instance.
- y as a particular class label, and y_x is the class label of x.

Suppose that $P(Y=y_j| x)$ denotes the probability that example x belongs to class y_j, the Bayes theorem can be used to optimally

The 18th International Conference on Pattern Recognition (ICPR'06)
0-7695-2521-0/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

predict the class label of a previously unseen example *x*, given a set of training examples in advance. With the Bayes theorem, the expected classification error can be minimized by choosing $\mathrm{argmax}_j\{P(Y=y_j\,|\,x)\}$. Given an example *x*, the Bayes theorem provides a method to compute $P(Y{=}y_j|\,x)$ with Eq. (1)

$$P(Y=y_j\,|\,x)=\frac{P(Y=y_j)\cdot P(X=x\,|\,Y=y_j)}{P(X=x)} \qquad (1)$$

Assuming that the attributes are independent given the class, $P(X{=}x|\ Y{=}y_j)$ can be decomposed into the product $P(x_1|y_j) \times P(x_2|\ y_j) \times \ldots \times P(x_a|\ y_j)$. Then the probability that an example belongs to class y_j is given by Eq. (2).

$$P(Y=y_j\,|\,x)=\frac{P(Y=y_j)\cdot\prod_a P(X=x_a\,|\,Y=y_j)}{P(X=x)} \qquad (2)$$

Which can be rewritten as Eq. (3).

$$P(Y=y_j\,|\,x)\propto\log(P(Y=y_j))+\sum_a\log(P(X=x_a\,|\,Y=y_j)) \qquad (3)$$

3.2 Instance Behavior Assessment

Let $P(y|\ x)$ be an unknown conditional distribution over the input, *x*, and output class, *y*, and $P(x)$ be the marginal "input" distribution. The learner is given a labeled training set, *E*, consisting of *IID* input/output pairs drawn from $P(x)P(y|\ x)$, and estimates a classification that given an input *x*, produces an estimated output distribution $\hat{P}_E(y\,|\,x)$. To measure the degree of our disappointment in any difference between the true distribution, $P(y|\ x)$, and the learner's prediction $\hat{P}_E(y\,|\,x)$, the log loss function in Eq. (4) is usually introduced. With the class label of *x*, we can produce $P(y_i|\ x)$ without bias, and then estimate the prediction loss for each instance *x*. However, existing studies often complain that the absolute class distribution values estimated by *NB* are likely poor. So we modify Eq. (4) slightly to consider the relative class distribution of each prediction, instead of relying on the absolute prediction values, as defined by Eq. (5).

$$L_x=-\sum_{j=1}^{|Y|}P(y_j\,|\,x)\log(\hat{P}_E(y_j\,|\,x)) \qquad (4)$$

$$\hat{L}_x=-\sum_{j=1}^{|Y|}\hat{P}_E(y_j\,|\,x)\log(\hat{P}_E(y_j\,|\,x)) \qquad (5)$$

With the entropy based loss function in Eq. (5), we can quantitatively assess the loss of a single instance *x* from an underlying classifier, but it is inadequate to evaluate the instance behavior because the loss function does not take the class label of instance *x* into consideration, and is therefore not able to differentiate the loss of a correct or incorrect classification. Then we modify Eq. (5) by taking the class label into consideration to measure instance behaviors, as defined by Eq. (6)

$$Bh_x=\begin{cases}+\infty & \mathit{if}\ \{y_j=y_x\wedge\hat{L}_x=0\,|\,j=\arg\max_l P(y_l\,|\,x)\}\\ {}^{1}\!/\!{}_{\hat{L}_x} & \mathit{if}\ \{y_j=y_x\wedge\hat{L}_x\neq 0\,|\,j=\arg\max_l P(y_l\,|\,x)\}\\ -\,{}^{1}\!/\!{}_{\hat{L}_x} & \mathit{if}\ \{y_j\neq y_x\wedge\hat{L}_x\neq 0\,|\,j=\arg\max_l P(y_l\,|\,x)\}\\ -\infty & \mathit{if}\ \{y_j\neq y_x\wedge\hat{L}_x=0\,|\,j=\arg\max_l P(y_l\,|\,x)\}\end{cases} \qquad (6)$$

3.3 Instance Behavior Study and DC Construction

Eq. (6) has provided a quantitative measure to study each separate instance. Although well motivated, two pieces of evidence are needed before we can use this measure to guide the instance selection: (1) whether Eq. (6) can indeed characterize instances with similar behaviors; and (2) whether instances with similar behaviors can contribute similarly to build a learner. We refer to an empirical study in this section to resolve these two concerns. For this purpose, we design a toy problem as shown in Fig. 1. It is a two class problem with a discriminant plane at *y*=*x*. Instances above and below y=x are positive and negative examples respectively (denoted by rectangle and triangle boxes in Fig. 1). Examples on line y=x are equally separated into two classes. Although this problem can be easily solved by Linear Discriminant Analysis [8], it turns out to be a hard task for learners like NB or C4.5, especially when the number of training examples is inadequate. Our first objective is to study whether Eq. (6) can indeed characterize instances with similar behaviors. For this purpose, we adopt a cross-validation oriented approach to group all instances in *D* into *n* Data Cells (DC), based on their behavior values, as described in Algorithm 1.

In this experiment, we construct 10 cells for the toy dataset, with all cells and their members showing in Fig. 2 (we intentionally make each cell reserve the same class distribution as the original dataset *D*, so each cell has exactly five positive and five negative examples). In Fig. 2, from Cell$_0$ to Cell$_9$, the members in each cell should have lower and lower behavior values, because all instances are ranked by their behavior values in descending order. Among all instances in *D*, the farther the instance from y=x, the easier the instance can be classified. The most difficult instances are those around the *y*=*x* line. As we can see from Fig. 2, the proposed measure does cluster well instances with similar behaviors, with DC_0 to DC_9 containing examples more and more difficult to classify.

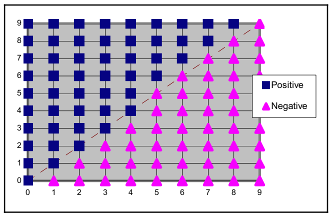
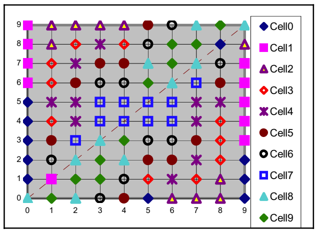


Fig. 1: A two-class toy problem



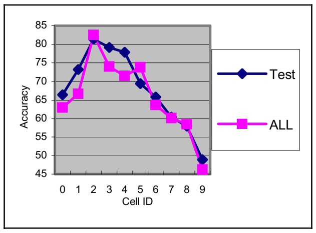


Fig. 2: Constructed data cells **Fig. 3:** Cell performances

However, having instances with similar behaviors in cells does not necessarily tell us which cell is better in constructing a learner? Shall we select instances from DC_0 or from DC_9, if a user requests for a subset of 10 instances? As there is no clear answer at this stage, we build a learner DT_i from each cell DC_i, and then evaluate its performance. We repeat the same procedure for all learners DT_0, …, DT_9, and provide their accuracies in Fig. 3. Where "Test" and "ALL" mean the accuracy of the learner tested on a testing set (randomly selected 10 examples) and all the data (100 examples) respectively. As shown in Fig. 3, the overall trend of the accuracy is that cells in the middle are better than those at both ends. A simple analysis can explain why. Intuitively, instances in DC_0 comply well with the existing model (the model built from the whole dataset). But complying with the data model well does not necessarily contribute significantly to build the current learner. It is often the case that models built from these quality data are over simplified and cannot generalize well for other instances [5]. On the other hand, instances in DC_9 are highly untrustworthy and confusing for the existing model. It consists of two types of instances: noisy instances and truly confusing examples. Since no noise exists in the toy problem, all examples in DC_9 belong to the later case. It is worth noting that uncertainty sampling based approaches actually believe that instances like those in DC_9 are more important than others in instance selection, but it turns out to be not the case for even this toy problem. Therefore, selecting instances from confusing examples will result in a low quality learner, and it is still not an option for representative instance selection. Cross-validation is an effective tool to determine which cell is better than others, and this procedure discards any prior knowledge in instance selection and directly refers to instances' behaviors for answers.

```
Procedure: Data_Cell_Construction()
Input: original dataset $D$; Output: $n$ data cells $DC_i$ of $D$.
Parameter: $p$ (# of subsets for cross-validation, typically 10); $n$ (# of data cells)
    Form $p$ disjoint equal-size subsets $E_i$, where $\cup_i E_i = D$.
    For $i$=1, …, $p$
        Form $E_y \leftarrow D \setminus E_i$
        Induce a Classifier $H_y$ based on examples in $E_y$.
        For every $x \in E_i$
            Calculate its behavior value $Bh_x$ with Eq. (6)
    Rank all instances in $D$ based on their $Bh_x$ values in descending order.
    For $i$=1, .., $n$
        Construct $DC_i$ by selecting instances continuously from the top to the bottom of the list, with each cell having the same instance numbers and class distributions.
    Return
```

Algorithm 1: Data cell construction

3.4 Progressive Estimation of Error Reduction

To enhance the scalability of the algorithm for general situations, we propose a progressive estimation of error reduction to progressively reorder data cells and optimize the expected further error, as shown in Algorithm 2. Denote $\tilde{E}_{D^*}(D)$ by the error rate of a learner built from dataset D^* and was tested on dataset D. Progressive error reduction is a greedy search mechanism thus aims to select a data cell, DC^*, such that when the instances of the cell are put into the existing dataset, the learner trained on the resulting set (D^*+DC^*) has a lower error rate than any others.

```
Procedure: Progressive_Estimation_of_Error_Reduction ()
Input: Original data cells ($DC_0, DC_1, \ldots, DC_{n-1}$);
Output: $L$ Ranking list of data cells.
Parameter: $n$ (# of data cells)
    $D = DC_1 \cup DC_2 \cup .. \cup DC_{n-1}$;   $D^* \leftarrow \varnothing$;   $L \leftarrow \varnothing$
    For $i$=1, .., $n$
        $D^* \leftarrow D^* \cup DC_k \mid \{k = \text{argmin}_l \{ \tilde{E}_{D^*+DC_l}(D),\ DC_l \not\subset D^* \}\}$
        $L \leftarrow L \cup k$
    Return $L$
```

Algorithm 2: Progressive Estimation of Error Reduction

$$\forall (DC_i) \quad \tilde{E}_{D^*+DC^*}(D) < \tilde{E}_{D^*+DC_i}(D) \tag{7}$$

The algorithm details are shown in Algorithm 2. It first selects a cell with the minimal expected error on D. Then we put instances to D^*, and progressively assess other cells one at a time. Each time a data cell DC^* is selected, it should guarantee that the learner built from the aggregation of the previous selected data D^* and DC^*, e.g., D^*+DC^*, can minimize the error rate of the learner on D.

The proposed progressive error reduction is actually a suboptimal mechanism, because it does not test all possible combinations to select the one with the minimal prediction error. The reason of taking this approach is twofold (1) **Efficiency**, testing all possible combinations of data cells is expensive; and (2) **Scalability**, the optimal solution does not accommodate well the scalability of instance selection. Changing the query requires the whole system to repeat again.

In Fig. 4, we show the first two selected cells (which are Cell_2 and Cell_1 in Fig. 2) with Algorithm 2, where items with different colors (and shapes) mean the members of different cells (the blue circles denote the members of the first selected cell, and the pink squares mean the members of the second selected cell). For comparison, the optimal solution of a representative instance set with 20 instances is shown in Fig. 5. Understanding the average classification accuracy (from the NB perspective) of the learner built from this cell is easy. Since the priori probability of all classes is equal, at each data point (i, j) we sum up the training examples horizontally and vertically (based on the discriminant function in Eq. (3)). If the number of positive examples is more than negative examples, the learner will classify the instance at (i, j) as a positive instance, and vice versa. In the case that the numbers of positive and negative examples are the same, the learner will have to do a random guess. The average accuracy from the learner built from Fig. 4 is about 84% which is very close to the learner from the optimal approach in Fig. 5 (88%), and far better than a random approach, which is about 70.8%.

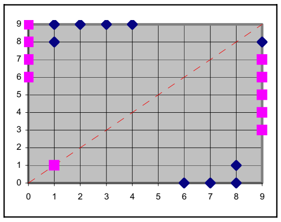


Fig. 4: Members of the first two selected cells

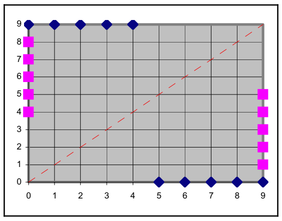


Fig. 5: Members of the optimal solution with 20 examples

The final result (ranking list) of the progressive estimation of error reduction on the toy problem is: Cell_2, Cell_1, Cell_4, Cell_5, Cell_6, Cell_0, Cell_9, Cell_7, Cell_3, and Cell_8.

5. Experimental Evaluations

The majority of our experiments use NB, and for comparison purposes we also use C4.5 [9] decision trees. For each experiment, we perform 10 times 10-fold cross-validation. We evaluate our approach on 30 benchmark datasets from the UCI data repository [10-11]. In Table 1, ALL(C4.5) and ALL(NB) mean the accuracy of the model trained from the whole dataset with C4.5 and NB respectively. For each entry of 3STAR, if 3STAR is better than both Random and ALL(NB), we tag "‡" right after the value, and if STAR is better than Random but inferior to ALL(NB), we place a tag "†" instead. On the other hand, for each entry of Random, if its performance is better than both 3STAR and ALL(NB), we tag "‡", and if its performance is better than 3STAR but inferior to ALL(NB), we place a tag "†".

The results in Table 1 indicate that 3STAR remarkably outperforms a random approach for representative instance selection. When the size of the user requested subset is small, Random is simply incomparable to 3STAR. 3STAR also has remarkable absolute improvement values. It is possible that 3STAR can beat Random with a 20% or more improvement. For many datasets, if the user requested subset size is 5%, the model built from Random is significantly worse than the learner of the whole dataset. Increasing the sampling ratio likely always improves Random's performance. This brings two negative impacts to those which heavily rely on the random selection: (1) it misleads users to draw incorrect conclusions, simply because the sampled set is not representative enough to build a trustworthy model; and (2) it incorrectly leads users to believe that the more instances they use, the better their learner can be.

In addition to the fact that 3STAR can outperform Random, it can even bring a better learner in comparison with the theory built from the whole dataset. Obviously, having massive amounts of data does not necessarily mean the algorithms have to use them all. Many real-world possibilities can result in imperfect data, which leads the learner to draw biased decisions. 3STAR provides a practical solution to help users assess the data behaviors and select representative instances to build a superior learner. This can be conducted through many ways, and the most obvious approach is to continuously increase the number of representative instances until the decrease of the model accuracy has been observed. Detailed discussions in this regard are beyond the coverage of this paper.

Table 1: Representative instance selection results (30 benchmark datasets)

Dataset	ALL (C4.5)	ALL (NB)	5%		10%		20%		30%		50%		70%		90%	
			3STAR	Random	3STAR	Random	3STAR	Random	3STAR	Random	3STAR	Random	3STAR	Random	3STAR	Random
Abalone	21.17	24.68	25.12 ‡	18.56	25.18 ‡	22.82	25.73 ‡	23.76	25.61 ‡	24.03	25.58 ‡	24.5	25.72 ‡	24.84	25.43 ‡	24.96
Anneal	94.16	91.91	90.29 †	88.07	91.82 †	88.69	93.67 ‡	90.8	94.21 ‡	91.45	93.85 ‡	91.94	93.02 ‡	92.09	92.11	92.18 ‡
Auto-mpg	75.98	65.98	62.99 †	59.43	65.53 †	61.25	65.29 †	63.97	64.98 †	62.85	65.06 †	64.13	64.16 †	62.68	62.6 †	62.21
Balance	69.37	91.69	87.52 †	64.61	83.57 †	63.11	83.53 †	72.65	87.34 †	83.02	90.11 †	87.08	90.7 †	89.24	91.16 †	90.91
Breastc	66.54	71.62	72.16 ‡	64.66	72.41 ‡	62.67	71.81 ‡	66.62	70.69 †	68.94	71.27 †	70.08	71.16	71.59 †	72.1 ‡	71.82
CMC	49.89	46.67	45.47 †	42.91	45.12 †	44.17	45.1	45.11 †	45.42	45.49 †	45.40	45.89 †	46.15 †	45.91	46.46 †	46.32
Connect	79.41	80.26	76.44 †	70.78	78.51 †	70.95	79.29 †	75.31	79.82 †	78.39	80.1 †	79.57	80.21 †	79.87	80.32 ‡	80.25
Credit	82.85	84.94	85.79 ‡	70.31	86.32 ‡	79.42	86.08 ‡	82.62	85.92 ‡	83.37	85.14 ‡	84.54	85.44 ‡	84.96	85.43 ‡	85.15
Ecoli	82.59	80.15	59.02 †	53.23	59.37 †	54.72	66.32 †	62.09	71.93 †	66.79	75.41 †	73.55	77.31 †	75.81	78.54 †	78.46
German	69.52	69.99	70.72 ‡	69.84	71.32 ‡	70.24	70.95 ‡	69.99	71.23 ‡	70.13	71.17 ‡	70.07	70.88 ‡	69.97	70.28 ‡	69.99
Glass	66.29	51.15	43.59 †	34.45	48.52 †	36.92	50.83 †	41.87	52.21 ‡	43.71	52.26 ‡	46.35	53.59 ‡	48.29	51.82 ‡	50.06
Kdd99	99.61	95.58	98.25 ‡	93.04	98.29 †	93.63	97.85 ‡	93.69	98.3 ‡	96.77	98.56 ‡	95.58	96.51 ‡	95.57	95.81	96.02 ‡
Krvskp	99.42	87.88	85.05 †	59.57	86.81 †	75.72	88.15 ‡	86.47	91.72 ‡	86.81	93.68 ‡	87.73	93.69 ‡	87.75	90.97 ‡	87.9
Led24	100	100	83.76 †	77.42	99.94 †	98.68	100	100	100	100	100	100	100	100	100	100
Liver	65.71	61.89	59.38 †	57.42	62.21 †	58.03	64.89 ‡	61.16	64.66 ‡	60.13	65.74 ‡	61.31	64.77 ‡	60.38	65.01 ‡	62.43
Monks1	96.14	74.98	72.97 †	53.34	74.03 †	64.22	78.16 ‡	66.65	78.94 ‡	71.03	79.55 ‡	72.63	77.57 ‡	74.14	74.61	74.98 †
Monks2	47.65	65.95	61.14 †	52.48	62.24 †	57.52	62.19 †	59.55	61.48 †	57.85	61.48 †	61.31	62.17 †	61.81	64.59 †	63.83
Mushroom	100	99.68	95.46 †	92.89	98.3 †	93.47	99.89 ‡	96.6	99.95 ‡	99.55	99.94 ‡	99.62	99.95 ‡	99.68	99.89 ‡	99.68
Nursery	98.72	91.32	88.11 †	78.51	89.43 †	82.7	91.22 †	87.45	91.04	90.1†	91.61 ‡	90.08	91.63 ‡	90.15	91.29 †	90.21
Pima	72.83	68.06	69.26 ‡	65.09	68.63 ‡	65.25	68.61 ‡	65.26	68.53 ‡	65.07	66.78 †	65.06	67.85 †	66.69	68.03	68.1 ‡
Sick	98.74	95.02	93.28 †	90.1	94.68 †	92.45	95.65 ‡	94.41	95.93 ‡	94.77	95.93 ‡	94.88	96.1 ‡	94.98	95.47 ‡	95.04
Sonar	72.82	73.62	70.75 †	55.48	71.65 †	58.29	72.31 †	62.41	72.26 †	66.52	73.84 ‡	71.95	75.28 ‡	73.99	74.14 ‡	74.09
Soybean	91.72	94.38	78.25 †	70.37	82.32 †	76.79	84.45 †	82.27	87.45 †	84.52	91.05 †	89.11	92.77 †	92.45	93.19	93.78 †
Tictactoe	85.97	70.67	66.06 †	60.42	69.29 †	61.93	71.65 †	68.08	71.54 ‡	70.17	70.79 ‡	70.66	69.23	70.4 †	70.32	70.44 †
Tumor	37.94	45.16	31.08 †	26.31	38.56 †	34.45	41.11 †	36.32	40.75 †	39.66	41.88 †	40.77	43.16	43.23 †	43.88	44.57 †
Wbreastc	93.97	96.67	80.71 †	74.14	86.76 †	82.34	93.97 †	90.31	95.6 †	92.91	96.02 †	95.95	96.13	96.37 †	96.34 †	96.27
WDBC	93.31	88.51	90.01 ‡	86.0	90.59 ‡	87.68	90.77 ‡	88.28	90.52 ‡	87.94	90.14 ‡	88.27	89.35 ‡	88.32	88.6 ‡	88.51
Wine	92.71	94.71	83.74 †	76.01	88.32 †	80.69	91.74 †	86.9	93.32 †	90.21	94.14 †	91.77	95.24 †	93.84	95.02 ‡	94.17
Vote	95.5	90.32	94.42 ‡	87.4	94.94 ‡	88.49	95.15 ‡	90.32	95.1 ‡	90.08	93.92 ‡	90.13	92.63 ‡	90.64	91.93 ‡	90.39
Zoo	92.83	95.88	93.41 †	89.17	94.28 †	89.49	94.59 †	90.41	94.38 †	91.17	94.75 †	92.25	96.03 ‡	94.05	96.9 ‡	95.83
‡	N/A	N/A	8/30	0/30	7/30	0/30	14/30	0/30	15/30	0/30	16/30	0/30	16/30	0/30	15/30	3/30
† + ‡	N/A	N/A	30/30	0/30	30/30	0/30	28/30	1/30	27/30	2/30	28/30	1/30	25/30	4/30	22/30	7/30

6. Conclusions

In this paper, we have proposed 3STAR for scalable representative instance selection and ranking. We have shown that a carefully selected representative instance set can often result in a learner which is superior to the model from the whole dataset, and it is almost always better than a random selection. 3STAR is based on an optimal instance selection theory - Monte Carlo sampling, but with substantial complexity reduction. The novel features that distinguish 3STAR from others are twofold: (1) 3STAR does not base on any prior knowledge to select representative instances, and the selection procedure is based on the contribution of instances with similar behaviors, which makes 3STAR adapt well to the uniqueness of different datasets; and (2) 3STAR adopts progressive estimation of error reduction to directly optimize the expected future error for representative instance selection, and seamlessly incorporates the scalability and the utility maximization of instance selection for better performances. It can be easily demonstrated that this framework is general enough to incorporate any learning theory for representative instance selection, as long as the learner can estimate the class distribution of an instance.

References

[1] Breiman L., (1996), Bagging predictors, *Machine Learning*, 24(2):123-140.

[2] Schapire R., (1990), The strength of weak learnability, *Machine Learning*, 5(2):197-227.

[3] Utogoff P., (1989), Incremental induction of decision trees, *Machine Learning*, 4:161-186.

[4] Provost F., Jensen D., & Oates T., (1999), Efficient progressive sampling, *in Proc. of SIGKDD*, pp. 23-32.

[5] Lewis D. & Catlett J., (1994), Heterogeneous uncertainty sampling for supervised learning, *in Proc. of ICML*, 148-156.

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

[7] Malvin H. Kalos, Paula A. Whitlock, (1987), *Monte Carlo Methods, Volume 1, Basics*, John Wiley & Sons, Inc.

[8] McLachlan G., (1992), *Discriminant analysis and statistical pattern recognition*, NewYork: John Wiley and Sons.

[9] Quinlan, J.R. (1993), *C4.5: Programs for machine learning*, Morgan Kaufmann, San Mateo, CA.

[10] Blake, C.L. and Merz, C.J. (1998), *UCI Repository of Machine Learning Databases.*

[11] Hettich, S. and Bay, S. D. (1999). *The UCI KDD Archive*