2014-12

Adding diversity to rank examples in anytime nearest neighbor classification

International Conference on Machine Learning and Applications, 13th, 2014, Detroit.
http://www.producao.usp.br/handle/BDPI/48894

2014 13th International Conference on Machine Learning and Applications

Adding Diversity to Rank Examples in Anytime Nearest Neighbor Classification

Cristiano Inácio Lemes and Diego Furtado Silva and Gustavo E. A. P. A. Batista
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
{clemes, diegofsilva, gbatista}@icmc.usp.br

***Abstract*—In the last decade we have witnessed a huge increase of interest in data stream learning algorithms. A stream is an ordered sequence of data records. It is characterized by properties such as the potentially infinite and rapid flow of instances. However, a property that is common to various application domains and is frequently disregarded is the very high fluctuating data rates. In domains with fluctuating data rates, the events do not occur with a fixed frequency. This imposes an additional challenge for the classifiers since the next event can occur at any time after the previous one. Anytime classification provides a very convenient approach for fluctuating data rates. In summary, an anytime classifier can be interrupted at any time before its completion and still be able to provide an intermediate solution. The popular k-nearest neighbor (k-NN) classifier can be easily made anytime by introducing a ranking of the training examples. A classification is achieved by scanning the training examples according to this ranking. In this paper, we show how the current state-of-the-art k-NN anytime classifier can be made more accurate by introducing diversity in the training set ranking. Our results show that, with this simple modification, the performance of the anytime version of the k-NN algorithm is consistently improved for a large number of datasets.**

***Keywords*—Anytime Algorithm, Nearest Neighbor, Classification, Data Stream;**

I. Introduction

In the last decade we have witnessed a huge increase of interest in data stream learning algorithms. Data streams are ubiquitous in virtually every application domain, including finances, industry, robotics and data sensors, just to name a few.

A stream is an ordered sequence of data records. It is characterized by properties such as the potentially infinite and rapid flow of instances. However, a property that is common to various application domains and is frequently disregarded is the very high fluctuating data rates. In domains with fluctuating data rates, the events do not occur with a fixed frequency. This imposes an additional challenge for the classifiers since the next event can occur at any time after the previous one.

In those applications, the data rate is unknown and generated by external factors that are usually completely out of control of the learning algorithm. A typical example is in sensor mining, in which the external environment generates the events that are usually not under control of the sensors.

The high fluctuating data rates cause two main issues. The first one is how classical learning systems, which typically have a fixed classification time, can be able to classify all events, including those that occur with very brief time intervals. The second one is how to make use of the longer time intervals and avail of the additional time to improve classification inference and provide better classification performance.

The classical learning algorithms do not seem to be the answer in such a scenario. The data analyst may choose the most efficient classifier in terms of classification time, in order to minimize the number of events left unclassified due to short time intervals between events. However, this is not the answer for every application, since such classifier may not be induced by the learning algorithm with the most appropriate bias for the problem.

Anytime classification provides a very convenient approach for fluctuating data rates. In summary, an anytime classifier can be interrupted at any time before its completion and still be able to provide an intermediate solution. Thus, an anytime algorithm can provide answers to virtually all events. Additionally, an anytime algorithm is expected to find better solutions as it keeps running. Therefore, they can make a better use of the available time between events, using such a time to improve the output accuracy.

The popular k-nearest neighbor (k-NN) classifier can be easily made anytime by introducing a ranking of the training examples. A classification is achieved by scanning the training examples according to this ranking. At any time, the classifier can be interrupted and provide an intermediate classification according to the current class distribution.

In this paper, we show how the current state-of-the-art k-NN anytime classifier can be made more accurate by introducing diversity in the training set ranking. Our results show that, with this simple modification, the performance of the anytime version of the k-NN algorithm is systematically improved for a large number of datasets.

II. Background and Related Work

An anytime algorithm is an algorithm capable of providing a solution to a given problem at any moment after a short setup time. In other words, regardless of the time available for the execution of this algorithm, an approximate solution is provided as soon as it is requested after the setup time.

The setup time is necessary for the algorithm to have an initial approximate solution for the problem. For most implementations, this setup time can be extremely short. So that the algorithm can answer to virtually any request.

978-1-4799-7415-3/14 $31.00 © 2014 IEEE
DOI 10.1109/ICMLA.2014.26

Ideally, the quality achieved by an anytime algorithm should be proportional to its runtime [1], [2]. In other words, the longer the time available to obtain an answer, the better the quality of the solution provided by the algorithm, as illustrated in Figure 1.

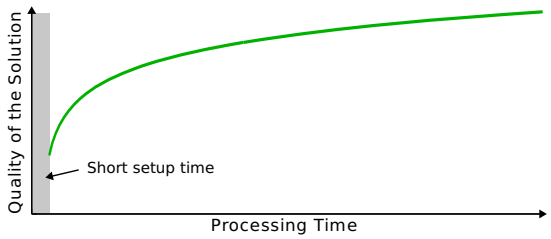

Figure 1. A generic illustration of the desired anytime algorithm performance. In gray, the setup time of the anytime algorithm, when it cannot be interrupted. The quality of the answer grows fast in the beginning of the execution and continue growing until the ending of it, with a smaller increasing rate

In summary, the desirable characteristics for anytime algorithms are [3]:

1) **Measurable quality:** The quality of an approximate solution can be evaluated precisely;
2) **Recognizable quality:** The quality can be measured at run time. For example, if we need the optimal solution to measure the quality of partial solution (such as in optimization problems), the quality cannot be recognized.
3) **Monotonicity:** The quality of the answer must be non-decreasing in a function of time;
4) **Diminishing returns:** The quality of the results improves in major steps in the first phases of the algorithm, diminishing its rate of increase over time;
5) **Consistency:** The quality of the result is correlated to computation time and deterministic for a given amount of time;
6) **Interruptability:** After a short setup time at the beginning of the execution, the algorithm can be stopped and provide an answer at any time;
7) **Preemptability:** After the algorithm be interrupted it can resume its execution at a low cost.

This kind of algorithm has been used in many artificial intelligence applications. One example is the multiple longest common subsequence (MLCS), used to deal with sequential data. Since it is a NP-hard problem, the time issue may be overcome by the use of anytime methods [4]. It also can be seen in the development of autonomous vehicles where the terrain is incompletely known. Then it needs a dynamic navigation algorithm that handle with a time constraint [5].

Due to the importance of this category of algorithms, several authors have proposed adaptations of traditional machine learning methods for anytime versions, such as boosting techniques [6], decision trees [7], support vector machine [8], nearest neighbor [9] and Bayesian networks [10].

In this paper we focus on the technique proposed in [9], a modification of the simple and well-known nearest neighbor algorithm. This technique, called Anytime Nearest Neighbor (ANN), computes a ranking of the training examples according to their contribution to the classification. In other words, the greater the contribution of a training example to the classification the better it is ranked.

After creating the ranking, the algorithm traverses this list measuring the distance from the example to be classified to each training example. When a solution is required, the class label of the new instance is associated with the class of the example with the smallest distance calculated so far.

Despite the good results achieved by the ANN algorithm, it also has some drawbacks. The first is that several instances of the same class may appear in sequence in the ranking. Thus, the method can calculate the distance from the example to be classified to training examples of the same class for a long period. Thus, the quality of the solution is not improved throughout this period.

To avoid this problem, Thuong and Anh [11] proposed to use SimpleRank in different bins, one for each class. Within these bins, the examples are sorted according to their rank value. Basically, the method modifies the SimpleRank doing a round robin regarding to the training example classes. In other words, the examples are analyzed in cycles, so that all classes are available in each cycle.

Another difficulty related to this method is the fact that the SimpleRank method typically produces many ties between the score value of two or more distinct examples. The method used by the authors to fix these ties was based on sorting the instances with the same rank value by their distance to their nearest neighbor of the same class.

In this paper, we propose a new technique to resolve the tie breaking of instances that obtained the same rank value. In the next section, we provide a more detailed description of ANN algorithms.

III. Anytime Nearest Neighbor

The ANN algorithm is composed of two main steps: ranking training examples and classification phase. In this section, we describe these steps in detail and the different methods proposed to generate the ranking of examples.

A. Classification Procedure

The ANN algorithm requires three input parameters: a training base, a list of sorted indexes of the training examples, which together represent what we call as the ranking of training examples, and an object with unknown class label. Its implementation is relatively simple and similar to traditional nearest neighbor algorithm, but with two distinct stages.

The first stage of this procedure provides an initial classification by checking one training example of each class. For this, the classifier analyzes the first examples of the ranking, performing the calculation of their distances to the example to be classified. It is important to ensure that the top positions of the ranking are occupied by examples belonging to all classes of the problem. This stage of the algorithm execution is called setup and the classifier cannot be interrupted during this period. However, this time is usually very short and does not impair the use of algorithm in data streams applications where the events occur in variable time. Thus, at the end of this short

setup period, the algorithm is able to provide a label estimate to the new example considering all classes of the problem.

After the initial setup, the algorithm is ready to be interrupted at any time. At each step of the second stage, the method measures the distance from the new example to the next training example in the ranking. The algorithm keeps a list of the k current nearest neighbors and updates this list, if necessary. This step is performed until the algorithm is interrupted or all examples in the ranking have been compared to the new object.

Thus, it is evident that a proper ranking of training examples is very important for the quick improvement of the solution quality at the beginning of the algorithm execution, a desirable property for any anytime algorithm.

B. Ranking Strategies

Based on the algorithm presented above, we can produce an anytime version of the nearest neighbor algorithm by proposing a new ranking method. There are several ways to accomplish this sorting.

The most straightforward way to generate the ranking of examples is randomly. For this purpose, we randomly select one example of each class as part of the initial setup phase. After this step, the remaining examples are taken in completely random order. This ranking method seeks to generate an uniform scattering of the examples in the input space.

This strategy, although simple, does not take into account the quality of the examples used for the construction of the ranking. Thus, the growing of the accuracy in the first steps of the algorithm execution may be slow.

The method SimpleRank [9] tries to overcome this limitation by sorting the training examples according to their contribution to classification.. The quality of each example is measured using the leave-one-out nearest neighbor classifier on the training data. For each left out example, the algorithm checks if its nearest neighbor is an *enemy* or an *associate*. The neighbor is considered enemy if its label is different from the label of the left out instance. Otherwise, it is considered an associate. The instances considered enemies are "punished" because they do not contribute to the correct classification of the left out example. In contrast, the associate instances are "rewarded" by the ranking method. SimpleRank uses Equation 1 to provide a score for each example.

$$rank(x_i) = \sum_{j \neq i} \begin{cases} 1, \text{ if } class(x_j) = class(x_i) \\ -2/(\#classes - 1), \text{ otherwise} \end{cases} \quad (1)$$

where x_j is a left out instance that has x_i as its nearest neighbor. Once these scores are calculated for all instances in the training set, sorting the instances according to their scores generates a rank.

Note that this criterion can generate several draws, since the number of times that an example is considered associate or enemy depends on the (limited) number of examples in which it is the nearest neighbor. In order to empirically confirm this observation, we performed a simple experiment with SimpleRank. In this experiment, we used a dataset in which the number of training examples was 17.118. The total number of different score values was only 14. In other words, the 17.118 examples ranked by SimpleRank were divided in only 14 groups, so that within each group all examples are tied with the same rank value. On average, each training example recieved the same rank value than other $1,221$ instances.

The tiebreaking criterion originally proposed to solve this problem was distance of each tied example to its nearest neighbor of the same class. However, this tiebreaking may group examples of very close regions in the space in close positions of the training ranking.

Another issue related to the strategy of the original tiebreaker is the fact that if a class have a distribution where the instances are more spread out in the input space than others, the algorithm privileges the most compact class.

Figure 2 exemplifies both issues. This figure graphically displays the first ten examples of the rank obtained by SimpleRank and the method proposed in this paper. Figure 2(*a*) shows that the examples chosen by the tiebreaking method are concentrated on a limited region, which does not happen with our method - Figure 2(*b*). When a class is more compact than the other, the original method prioritizes such a class - Figure 2(*c*) -, unlike the method proposed in this work - Figure 2(*d*).

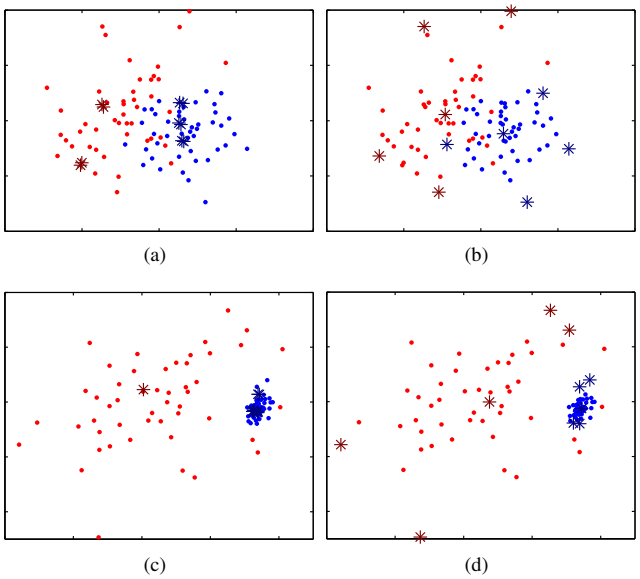

Figure 2. The first ten examples (marked with $*$) chosen according to SimpleRank - (*a*) and (*c*) - and according to the method proposed in this paper - (*b*) and (*d*) - for synthetic datasets with similar spatial distribution for both classes – *top* – and with a more compact class than the other – *bottom*

IV. SimpleRank with Diversity

The sorting and tie-breaking strategies of the SimpleRank method do not guarantee an uniform coverage of the input space. We believe that such an uniform coverage will be advantageous to the anytime nearest neighbor algorithm, since it will be more likely to find a nearest neighbor that is truly next to the unknown instance. However, it is still

important to account for the classification relevance of each training instance, as implemented in SimpleRank. Therefore, we introduce in this paper the SimpleRank method ranking using the DiversityTiebreak as the tie-breaking criterion.

Succinctly, our proposal augments the strategy for calculating the rank scores used by the SimpleRank method with a novel tie-breaking criterion. Our tie-breaking strategy provides a better coverage of the input space. Our approach is applied in two steps. The first one is the ranking of the training instance using a novel criterion, DiversityTiebreak, described in the Algorithm 1.

Algorithm 1: Diversity Tiebreak

```
   Input  : Training dataset
   Output: Sorted list of indexes
1  foreach Class c_i do
2      index_list_c_i =<>
3      k = centroid(c_i)
4      idx_add = index of the nearest neighbor of k that belongs to c_i
5      index_list_c_i = index_list_c_i+ < idx_add >
6      while |index_list_c_i| < |instances(c_i)| do
7          idx_add = identify the fartest instance from all instances in
           index_list_c_i;
8          index_list_c_i = index_list_c_i+ < idx_add >
9      end
10 end
11 diversity_list = merge index_list_c_i of each class c_i;
12 return diversity_list;
```

This algorithm defines different sorted lists for each class. The first element of each list is the nearest neighbor of the class centroid. In order to find such an example, line 3 returns the geometric center of the examples that belong to the current class c_i. Next, lines 4 and 5 find the index of the nearest neighbor of this centroid and add it in the first position of the current class' list, respectively.

The next instances to be added in the sorted list of the class are the most distant instance to all the object added in the list before. In other words, the instance with the largest sum of distances to the instances earlier added to sorted list should be the next value to be added in the list. This process repeats until all examples of the current class are added to the list.

The last step of the algorithm, in line 11, is the merge of the sorted lists of each class in a list containing the indexes to the whole training set. The most straightforward method to do this is to select one instance from each class list alternately and adding such an instance to the final ranking. But, this method may be inappropriate when applied to a problem with imbalanced classes. Thus, in our experiments we adopted an approach that considers the prior probability of each class. For instance, if a class a has N examples and a class b has $2N$, we take one example of the class a and two of the class b in each merge step.

After the creation of the diversity tiebreaking list, we calculate the rank value of SimpleRank. The instances that obtained the same rank value are tiebreaked considering its position obtained by the DiversityTiebreak ordering. Thus, the sorted index list used by the ANN algorithm considers the importance of the examples in the classification and a diversity of examples in case of ties.

V. Experiment and Results

We performed a comprehensive experimental evaluation to assess the effectiveness of our proposal. We compared the performance of the ANN using different methods of ranking: random, SimpleRank and SimpleRank with DiversityTiebreak.

A. Experimental Setup

In order to evaluate our proposal, we used 15 benchmark datasets, which can be found at the UCI Machine Learning Repository[1]. The datasets used in this work have different characteristics, summarized in Table I.

Table I. Brief description of the benchmark datasets used in the experimental evaluation

Dataset	# of Attributes	# of Classes	# of Examples
EEG Eye State	15	2	14980
Image Segmentation	19	7	2310
Ionosphere	34	2	351
Letter Recognition	16	26	20000
MAGIC Gamma Telescope	11	2	19020
Mfeat - Factors	216	10	2000
Mfeat - Fourier	76	10	2000
Mfeat - Karhunen	64	10	2000
Mfeat - Morphological	6	10	2000
Mfeat - Zernike	47	10	2000
Optical Recognition of Digits	64	10	5620
Page Blocks Classification	10	5	5473
Pen-Based Recognition of Digits	16	10	10992
Spambase	57	2	4601
Waveform Generator (Version 2)	40	3	5000

We used the stratified 10-fold cross validation to estimate the classification performance of the methods in the selected datasets. We also linearly normalized each attribute in the range 0 to 1.This is a standard procedure in nearest neighbor classification in order to make the distance function invariant to the attribute scales.

Our evaluation procedure interrupts the ANN algorithm after the processing of each training instance following the ordering provided by the ranking methods. At each interruption, we take note of the performance in the test set assuming the ANN had time to process just the first j instances in the ranking. We vary j from the number of classes up to the total number of training examples.

The setup period is the time necessary to process as many instances as classes. That occurs because we guarantee that we have one instance of each class in the top of the ranking. This is the minimum amount of information provided to the algorithm so it can make an informed decision. We note that the setup time, during test processing, is very small and can be neglected in most practical situations, as the number of classes is usually quite small compared to the number of instances.

Such an evaluation procedure characterizes the performance of the methods with a set of performance scores for all possible interruption times. As we discuss in the next section, we can graphically compare these scores using a performance curve plot. We can also summarize such a performance curve in a single score calculating the area under this curve.

[1] https://archive.ics.uci.edu/ml/datasets.html

B. Ties in ANN using SimpleRank

The first step of our experimental evaluation is an experiment in which we calculate the number of ties obtained by the SimpleRank. Obviously, if the number of ties is insignificant, the proposal of a new tiebreaking criterion for the SimpleRank is useless.

Earlier, we anticipated a result obtained with a similar experiment on a specific dataset. In this section, we describe the results obtained in all the benchmark datasets used in this work.

Table II shows a summary of the number of ties for each dataset used in this study. The item "Number of distinct ranks" represents the average number of different rank value obtained within the training folds. The value of "Number of instances with the same rank" represents the average number of examples that obtained the same score with SimpleRank.

Table II. ANALYSIS OF NUMBER OF TIES OBTAINED BY SIMPLERANK

Dataset	Number of distinct ranks	Number of instances with the same rank
EEG Eye State	11.70	1152.30
Image Segmentation	11.90	174.71
Ionosphere	12.30	25.68
Letter Recognition	30.10	598.00
MAGIC Gamma Telescope	14.40	1188.80
Mfeat - Factors	15.20	118.42
Mfeat - Fourier	20.70	86.96
Mfeat - Karhunen	13.70	131.39
Mfeat - Morphological	12.30	146.34
Mfeat - Zernike	16.10	111.80
Optical Recognition of Digits	14.10	358.72
Page Blocks Classification	13.70	359.54
Pen-Based Recognition of Digits	12.10	817.58
Spambase	21.10	196.25
Waveform Generator (Version 2)	24.40	184.43

We can observe that even in problems with a few training examples, SimpleRank generated a significant amount of repeated rank values.

C. Evaluation

In this paper we use two main forms to assess our results. The first one is a plot of performance versus processing time. We use the ubiquitous accuracy as the main performance measure in our evaluation. As we want to characterize the performance of the methods for all possible interruption times, we chose to interrupt the ANN after the processing of each training instance, as discussed in the previous section.

This procedure generates a graph that evaluates the classification accuracy in terms of the number of training examples analyzed before an interruption. Ideally, this analysis should be done in terms of processing time. However, processing time is heavily dependent on code optimization and the hardware in the benchmarking, making it difficult to compare different methods. Therefore, it is a standard procedure in the literature to replace the processing time by other piece of information that allows a generic way to evaluate the performance of the anytime algorithm. In our evaluation, we used the number of processed example, similarly to [9].

We also provide a single numeric score that represents the general performance of each method, as a second form of evaluation. Such a score typically provides less information than the curves; however, it is a very convenient to summarize the results over several datasets. For this score, we use the area under the anytime performance curve. Formally, this area is defined by the Equation 2.

$$AUC_{anytime} = \frac{1}{n - \#classes} \sum_{i=\#classes}^{n} acc_i \qquad (2)$$

where acc_i is the accuracy obtained when the algorithm interrupted at the i-th example and n is the number of training examples. Due to the setup stage, the initial value of i is the number of classes.

D. Classification Results

Table III shows the areas under the curves obtained using the ANN with the three ranking methods for the 15 datasets considered in this work.

Table III. AREA UNDER THE PERFORMANCE CURVE OF THE ANN ALGORITHM USING THREE DIFFERENT RANKING METHODS

Dataset	Random	SimpleRank	SimpleRank with DiversityTiebreak
EEG Eye State	0.7986	0.7992	**0.8028**
Image Segmentation	0.9388	0.9417	**0.9441**
Ionosphere	0.8444	0.8721	**0.8821**
Letter Recognition	0.9106	0.9083	**0.9173**
MAGIC Gamma Telescope	0.7972	**0.8177**	0.8173
Mfeat - Factors	0.9399	0.9414	**0.9472**
Mfeat - Fourier	0.7684	0.7834	**0.7883**
Mfeat - Karhunen	0.9288	0.9297	**0.9382**
Mfeat - Morphological	0.6749	**0.6842**	0.6824
Mfeat - Zernike	0.7737	0.7857	**0.7920**
Optical Recognition of Digits	0.9767	0.9773	**0.9806**
Page Blocks Classification	0.9480	0.9518	**0.9538**
Pen-Based Recognition of Digits	0.9857	0.9852	**0.9872**
Spambase	0.8791	0.8810	**0.8825**
Waveform Generator (Version 2)	0.7178	**0.7585**	0.7556
Wins	0	3	12

The results indicate the effectiveness of the proposed method. When analyzing the area under the performance curve of different versions of the ANN algorithm, the SimpleRank with DiversityTiebreak obtained the best results in 12 of the 15 datasets used.

To ensure the effectiveness of the proposed method, we realized a hypothesis test. We chose the Friedman with and the Li's post-hoc test and 95% as confidence level, as suggested in [12]. The test rejected the null hypothesis that SimpleRank with DiversityTiebreak has similar performance with the opponent methods.

For sake of space, we do not show the curves for all datasets. However, we created a website that contains detailed results of our experiments[2]. In this paper, we show some plots that summarizes the results. Figure 3 shows the results obtained by the proposed method applied on the Mfeat - Karhunen dataset.

In this figure, we can see that the curves are according to the expected results for an anytime algorithm. More specifically, the accuracy gain at the beginning of the execution is quite remarkable, especially for SimpleRank with diversity. We note also that SimpleRank with DiversityTiebreak provided

[2]http://sites.labic.icmc.usp.br/clemes/diversityrank

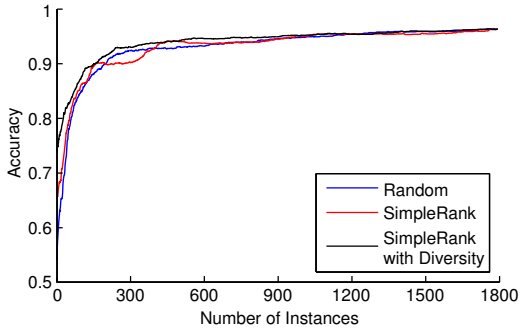

Figure 3. Results obtained for the Mfeat - Karhunen dataset

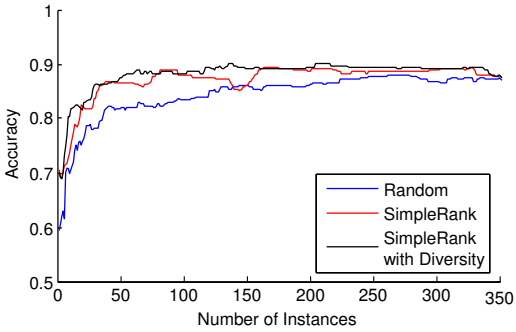

Figure 5. Results obtained for the Ionosphere dataset

more consistent accuracy results, with less variability than SimpleRank.

Although the performance difference is not very large, it is consistent during entire curve. In general, SimpleRank with DiversityTiebreak presented results with less variability than the other methods. For instance, there are a few moments that SimpleRank is outperformed by the Random approach. It rarely happens with our proposal.

Figure 4 presents the results for the Optical Recognition of Digits. In this dataset the performance differences are very small. This is due to the fact that Random already achieved very good performance results, leaving very little space for performance improvement. Note how the curve is very steep even for the first ranked examples. Even tough, SimpleRank with DiversityTiebreak achieved consistently better results than the other two methods;

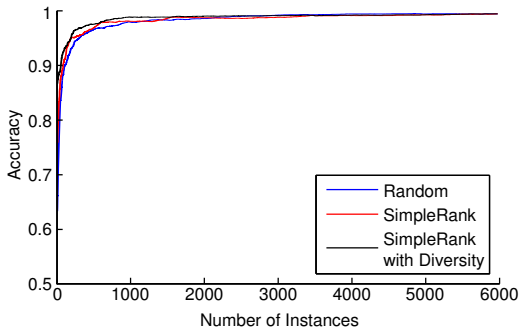

Figure 4. Results obtained for the Optical Recognition of Digits dataset

Figure 5 shows the results for the Ionosphere dataset. This a smaller dataset in which the performance differences are more noticeable. Once again, SimpleRank with DiversityTiebreak presented better results than the competing methods for the entire range of number of training instances.

VI. Conclusion

In this paper, we presented a new tiebreaking method to be used as part of the Anytime Nearest Neighbor algorithm. Such a method considers the diversity in the space between examples of the same class as tiebreaking criterion.

Our method better covers the space of training examples, since it prioritizes distant examples to construct the ranking. Furthermore, the procedure for merging the lists by diversity of different classes considering the class distributions avoids long sequences of examples from a single class.

We showed that the proposed method is able to consistently improve the performance of the SimpleRank algorithm. Considering the 15 benchmark datasets used to evaluate our proposal, our method outperformed the SimpleRank in 12 of them. In fact, the hypothesis test demonstrated the superiority of the proposed method.

As future work, we intend to apply the proposed method to a larger number of application domains. We believe the proposed method can be particularly fruitful in applications in sensor data.

References

[1] J. Grass and S. Zilberstein, "Anytime algorithm development tools," *ACM SIGART Bulletin*, vol. 7, no. 2, pp. 20–27, 1996.

[2] S. Zilberstein and S. Russell, "Approximate reasoning using anytime algorithms," *Imprecise and Approximate Computation*, pp. 43–62, 1995.

[3] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI magazine*, vol. 17, no. 3, p. 73, 1996.

[4] Y. Shang, G. Chen, and Y. Xu, "A space-bounded anytime algorithm for the multiple longest common subsequence problem," *IEEE Transactions on Knowledge and Data Engineering*, p. 1, 2014.

[5] W. Yue, J. Franco, Q. Han, and W. Cao, "Improved anytime d* algorithm," in *IAENG Transactions on Engineering Technologies*. Springer, 2014, pp. 383–396.

[6] K. Myers, M. Kearns, S. Singh, and M. A. Walker, "A boosting approach to topic spotting on subdialogues," *Family Life*, vol. 27, no. 3, p. 1, 2000.

[7] S. Esmeir and S. Markovitch, "Interruptible anytime algorithms for iterative improvement of decision trees," in *International Workshop on Utility-Based Data Mining*, 2005, pp. 78–85.

[8] D. DeCoste, "Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry," in *ICML*, 2002, pp. 99–106.

[9] K. Ueno, X. Xi, E. Keogh, and D.-J. Lee, "Anytime classification using the nearest neighbor algorithm with applications to stream mining," in *ICDM*, 2006, pp. 623–632.

[10] T. Seidl, I. Assent, P. Kranen, R. Krieger, and J. Herrmann, "Indexing density models for incremental learning and anytime classification on data streams," in *International Conference on Extending Database Technology: Advances in Database Technology*, 2009, pp. 311–322.

[11] N. C. Thuong and D. T. Anh, "Comparing three lower bounding methods for dtw in time series classification," in *Symposium on Information and Communication Technology*, 2012, pp. 200–206.

[12] B. Trawiński, M. Smetek, Z. Telec, and T. Lasota, "Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms," *International Journal of Applied Mathematics and Computer Science*, vol. 22, no. 4, pp. 867–881, 2012.