An Adaptive Graph-Based K-Nearest Neighbor

Nils Murrugarra-Llerena and Alneu de Andrade Lopes

Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo – USP
Caixa Postal: 668 – CEP: 13560-970 – São Carlos – SP – Brasil
{nineil,alneu}@icmc.usp.br

Abstract. Ensemble techniques combine the predictions of a set of classifiers to obtain a resulting classifier that outperforms every one of them alone. The predictive quality of the ensembles is due to variability in the classifiers. Such variability has been demonstrated as a key factor in the success of ensemble techniques. In fact, known stable classifiers, such as those based on nearest neighbors, are not improved by the technique due to their stability with respect to resampling. In this paper, we propose an adaptive graph-based k-nearest neighbor algorithm that iteratively produces an adaptive k-nn network (k-nearest neighbor network), $G(V, E)$, where, V is the set of all training instances and E is a subset of all possible edges in the k-nn network from training data with a given $k = k_{max}$. In G, for each vertex, the number of edges depends on how difficult the vertex classification is, i.e., instead of using a single value of k for construction of G, the values of k range from 1 up to k_{max}. In this method, the k-nn network produced in iteration i, G_i, is given as a model to the learning algorithm in the following iteration $(i + 1)$. The learning algorithm uses the current network together with a graph-based classifier for building the next network, iteratively changing the distribution of the connections. Misclassified vertices in the iteration i have their degrees increased (up to reach k_{max}). The resulting classifier behaves like an adaptive k-nn since for different regions of the training data, a different value of k is set. Experimental evaluation shows that the proposed technique performed as well or significantly better than other state of the art methods tested, even when compared with ensembles of them.

Keywords: ensembles, boosting, graph-based learning, adaptive k-nn

1 Introduction

When people have to make a difficult decision, they usually prefer to take into account opinions of several experts than consider the opinion of just one assessor. In machine learning, a model induced by a learning algorithm can be regarded as an expert opinion (depending on the quality of the training data and if the algorithm is appropriate for the problem under study). An obvious approach to obtain more reliable decisions is to combine the output of different models.

That is the main idea behind ensembles technique [20]. An ensemble is a set of classifiers whose individual predictions are combined in some way (typically by voting) to classify new examples.

Ensembles Techniques are an active area of research in supervised learning with successful application in many fields, such as finance [13], bioinformatics [24], manufacturing [16], and image retrieval [14]. The main finding in those researches is that ensembles are generally more accurate than the individual classifiers that originated them [6].

A particularly interesting research field in voting classifiers is related to the use of ensembles of k-nearest neighbor (k-nn) classifiers [1]. For example, [10] constructed an ensemble of k-nn classifiers using cross-validated committees and [28] developed an approach to apply bagging to k-nn using Minkowsky distance. Additionally, [9] developed an approach using boosting and projections with k-nn classifiers. It was shown that k-nn classifier with bagging methods are not effective [3]. The reason is that k-nn is fairly stable with respect to resampling. Hence, ensemble techniques often fail in their attempt to improve the performance of k-nn classifiers [9].

To tackle with the above mentioned problem we propose a technique with a learning phase where an iterative strategy, similar to the AdaBoost [8], produces a model based on an adaptive k-nn network. There are two main differences from AdaBoost, (i) instead of using an ensemble of all classifiers induced, we use use only one model (network) induced from all networks produced during the training process, (ii) instead of changing the example distribution along the iterations we change the link distribution. The model induced is an adaptive k-nn network, $G(V, E)$, where, V is the set of all training instances and E is a subset of all possible edges from the k-nn network generated with the training data setting $k = k_{max}$. In k-nn networks, the parameter k is used to specify the number of neighboring vertices to connect. In the proposed adaptive k-nn network, G, for each vertex, the number of edges varies depending on how difficult the vertex classification is. Hence, instead of using a single value of k for constructing G, the value of k vary from 1 up to a given k_{max}. In this method, the k-nn network produced in iteration i is given as a model to the learning algorithm in the following iteration $(i + 1)$. The learning algorithm uses the current network together with a graph-based classifier (which considers only the links for classification purposes) for building the next network, iteratively changing the distribution of the connections. Misclassified vertices in a previous iteration have their degrees increased (up to k_{max}). In summary, our main goal is to develop a classification technique based on adaptive k-nn networks to improve the performance of k-nn classifiers.

Preliminary results on sixty-six data sets (22 original datasets from UCI-repository [25] plus 44 versions of them with noise addition) comparing our proposal with other well-known classifiers (k-nn, Naive Bayes [12], C4.5 [21] and SVM with SMO [19]) show that the proposed algorithm achieves the top rank using the Demšar's method [4] for comparing classifiers. In addition, we compare

the propose algorithm with the adaptive k-nn ($adaNN$) technique proposed by [23].

The remainder of the paper is organized as follows. Section 2 describes some concepts on neighborhood-based networks and propositional boosting. Section 3 describes the proposal algorithm (Graph-Based Adaptive k-nn). Experiments are presented in Section 4. Then, Section 5 presents discussion of experiments, and the last section shows conclusions and future works.

2 Related Work

Two approaches related with adaptive k-nn can be identified in the literature. The first estimates best values of k for a neighborhood or regions, and the second estimates a value of k for each new instance. Locally adaptive nearest neighbor algorithms [5] are representatives of the first approach. These algorithms select a value k for certain neighborhoods considering three strategies: $localk - nn_{unrestricted}$, $localk - nn_{class}$, and $localk - nn_{cluster}$. In the $localk - nn_{unrestricted}$ algorithm, for each training example it stores a list of values k that correctly classify that example under leave-one-out cross validation. To classify a new instance x, M nearest neighbors of this query are computed and is determined a k ($k_{M,x}$) which correctly classifies most of these M neighbors. Then, the instance x is classified with the class of the majority of its $k_{M,x}$ neighbors. In the $localk - nn_{class}$, it determines a k for each output class which classify correctly most instances of each class. Lets consider an example of 2 classes, c_1 and c_2. An instance x will be assigned to class c_1, if the percentage of the k_1 nearest neighbors of x that belong to class c_1 is larger than the percentage of the k_2 nearest neighbors of x that belong to class c_2. On the contrary, x is assigned to class c_2. Finally, in $localk - nn_{cluster}$, it defines clusters in the training data. Then, for each cluster a k value is defined. Each new example is classified using the value of k assigned for its cluster.

In another work, [27] uses the idea of confidence measure to partition the feature space into a set of prototype regions. For each region is assigned a value of k to classify new instances. Furthermore, [17] considers concepts of Laplacian eigenmaps, kernel mixtures and non-linear feature extraction to improve large-margin k-nn classification. It employs a distance function to transform the feature space. In this new feature space, instances of the same class are grouped together and its distances are decreased, while the distances for instances of different classes are increased. [26] finds k nearest neighbors in a small hyperball. To estimate the hyperball, it starts with a radius r_{min} which increases to an effective radius $r_{efective}$ to classify new instances.

Among representatives of the second approach there are the following techniques. The algorithm ($adaNN$) [23] finds a suitable value of k for each test example. First, for each training instance it estimates and stores its optimal k. Then, to classify a new instance x, (1) it finds the nearest neighbor of x (nn); (2) determines the optimal k (ok) for nn and (3) classifies x based on its ok nearest neighbors. Another approach considers the nearest neighbors of an instance

as an item of evidence [30]. It uses a dempster-shaffer theory and basic belief assignment defined as a function of distance/weight between the instance and its nearest neighbors. Using this information, it estimates a value of k for each training instance. The classification phase is done as in [23].

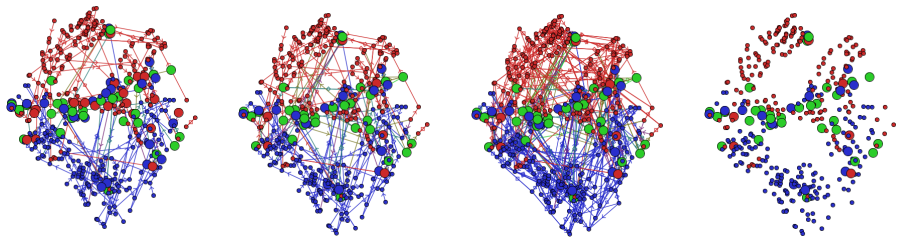
3 An Adaptive Graph-Based K-Nearest Neighbor Classifier

In this section, we present an adaptive k-nn algorithm ($A-knn$) based on an adaptive k-nn network and in a graph-based boosting algorithm. Its main purpose is to improve the $k-nn$ algorithm by mapping the propositional boosting technique [8] into a graph-based one.

A network is a structure $G(V, E)$, where V is a nonempty set of objects called vertices and A is a set of unordered pairs of V, called edges. Such formalism in general is applied to networked data. Nonetheless, propositional data can be transformed to enable the use of the network formalism. One way to handle an attribute-value table by using the network formalism is to construct a network based on similarity relation between vertices using the training data. The mostly studied approaches for building networks from propositional data are based on neighborhood relationships, the basic ones is k-nn networks [29]. In k-nn networks, the parameter k is used to specify the number of neighboring vertices to connect. Any vertex v_i connects with its k nearest neighbors using a directed edge.

The rationale behind the proposed adaptive k-nn network is to change the link distribution of the k-nn network according to how difficult is the classification of the vertices . The degree of a vertex (its number of links) is modified in a way similar to what is done with examples distribution by the AdaBoost algorithm in the propositional scenario. The degree of hard vertex increases and of easy one decreases. Hence, in our approach, the final model corresponds to a k-nn capable of using different values of k for different regions of the data space. For example, Fig. 1 illustrates the changes in the connection distribution during 3 iterations of a k-nn network (Fig. 1(a), 1(b), and 1(c)) generated from the well known dataset *balance* [25]. The networks are plotted using a version of the multidimensional projection tool *PEx* [18], for graph visualization. Such dataset has 3 classes, two of them easy classes (represented by red and blue circles) and a hard one (represented by green circles). The circle size represents the degree of a vertex. Fig. 1(d) does not show the edges in order to keep the figure as clean as possible. This figure shows how the hard examples (green circles intertwined between the other classes) become larger (more links) along iterations.

The A-Knn algorithm consists in two phases: the learning and classification phases. The learning phase, as presented in Fig. 2 and Algorithm 1, can be summarized in the following steps: (i) building a k-nn network $G'(V, E)$ from the training data, setting the value of k to k_{max}. Actually, this network provides the set of all possible edges E which can used in the final network $G(V, E_{final})$, $E_{final} \in E$. After that, (ii) using V and E, iteratively build k-nn networks (k



(a) (b) (c) (d)

Fig. 1. Progress of link distribution in the dataset Balance. (a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Final Iteration.

raging from 1 to k_{max}, with increments of two). In each iteration, all examples are classified using the current network and a graph-based classifier based on weights ($wvrn$) [15] (using the same weight for all edges). The misclassified examples have their number of connections increased, changing the link distribution of the current network. When a misclassification occurs, the idea is keep trying greater values of k until the instance is correctly classified or k_{max} is reached.

We observe that, by varying k from 0 to k_{max} (as illustrated in Fig. 3), three cases arise: (a) a vertex classification can be changed from the state 0 (misclassified or unclassified) to 1 (correctly classified, when $k = k_a$) and later go back to state 0 (when $k = k_c$); (b) classification is kept in the state 1, after a $k = k_b$; or (c) kept in state 0. Hence, in cases (a) and (b) we may select any value of k that leads to state 1 for every vertex in the final network, and for (c) does not matter the value of k. This is, the appropriate values of k for the vertices in $G(V, E_{final})$, in the experiments described in the next section, are computed by $(k_a + k_c)/2$, $(k_b + k_{max})/2$, k_{max} for each case, respectively.

In the classification phase, as summarized in Algorithm 2, using the final network, firstly, we find the nearest neighbor (v_j) of the test instance; secondly, considering the number of edges of the nearest neighbor, k_j, we classify the test instance looking at its k_j nearest neighbors. Notice the class of v_j is not taken into consideration. We are interested in apply the new connections produced in the learning phase, i.e., when an instance was difficult to classify its distribution of connections was changed in the hope the new links would convey new information.

4 Results

In this section we present classification results achieved by A-Knn on real data sets. We also compare the A-Knn with well-known classifiers. Specifically, this section presents three evaluation settings, the first compares the proposed method with stand-alone versions of the selected classifiers, the second with ensemble versions of those classifiers, and the third with the *adaNN* algorithm (with $k = 51$).

Input :
- *data* // data set used for train the classifier.
- $classifier$ // base classifier.
- *kmax* // maximum number of neighbors per instance.

Output :

- $classifier\_f$ // final graph-based classifier.

Algorithm:
G' ←Create a k-nn network using $k = kmax$
$matInd$ ← Create a network with $k = 1$
for $i \leftarrow 1$ *to* $size(data)$ **do**
  $D_i \leftarrow 1$ // degree of vertex i
  $states_i \leftarrow 0$
end
$nit \leftarrow kmax/2$
// number of iterations.
for $i \leftarrow 1$ *to* nit **do**
  $classifier \leftarrow$ train_classifier($classifier$,*data*)
  $(d, states) \leftarrow$ update_link_distribution($classifier$,*data*,*D*,*states*) // as explained in Section 3
  $matInd \leftarrow$ Create_Graph(*data*,*d*,*pG*)
  if *matInd doesn't change* **then**
    Break
  end
end
$classifier\_f \leftarrow$ CreateFinalClassifier($classifier$,*states*) // according to Fig. 3
return $classifier\_f$

Algorithm 1: Training Phase.

Input :
- *test_instance* // instance to classify.
- $classifier$ // classifier generated on training process.

Output :

- *class* // class predicted.

Algorithm:
nn ← Find the nearest Neighbor of *test_instance*
k ← Get the number of connections of *nn*
class ← Classify *test_instance* using *k* neighbors
return *class*

Algorithm 2: Classification Phase.

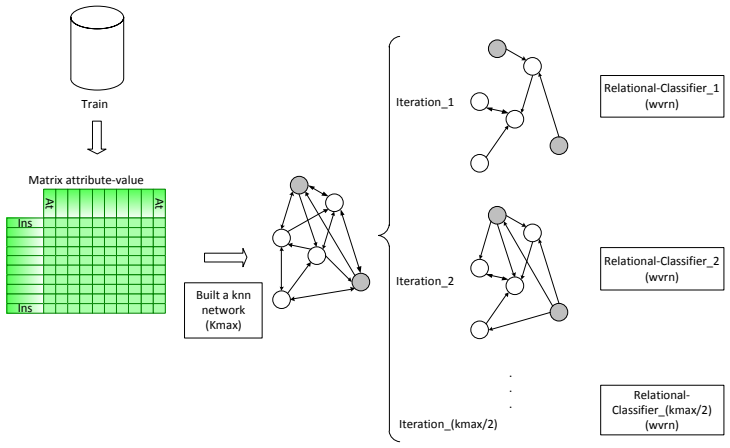


Fig. 2. Learning Phase. Gray and white circles represent different classes in the examples.

In the experiments, the Euclidean distance was adopted to compute similarity measure and we set $k_{max} = 51$ in the A-Knn algorithm. Preliminary results showed that in average the prediction performance does not significantly increases with k_{max} greater than this value.

The classifiers employed in the two first settings were: K-Nearest Neighbor (k-nn), with values of k from $\{1, 3, 5, 7, 9, 15, 20, 30, 40\}$, decision tree C4.5 [21], Naive Bayes [12], and Support Vector Machine (SVM) using the Sequential Minimum Optimization (SMO) algorithm [19]. We used the implementations available in WEKA [11] and their default parameters for each classification algorithm.

The tests were carried out on twenty-two data sets from the UCI-repository [25] presented in Table 1. The results, for each algorithm, were obtained through an average of a ten runs of 10-fold stratified cross-validation process. Furthermore, from each one of these twenty-two data sets were produced two new data sets with different levels of noise, by randomly changing the classes at 5% and 10 % of the training data, respectivelly. The adition of noise was stratified according to each class frequency in each dataset. The number of data sets and algorithms total 66 and 14, respectivelly. To simplify understanging, we sumarize the results in a boxplot visualization [2] (Fig. 4(a) and Fig. 4(b)) and in Table 2. Table 2 compares the proposal $A - knn$ with the other algorithms. It was used the Bonferroni-Dunn test from [4] to determine statistical difference between any pair of classifiers. It was used a confidence level at p =0.05 and $\triangle$ represents a significant difference and $-$ represents no significant difference between the pair of algorithms.

In order to analyze the results, we employed the Demšar's method [4] to compare multiple classifiers over multiple data sets. This method assigns a rank to each algorithm on each data set according to its classification accuracy. The

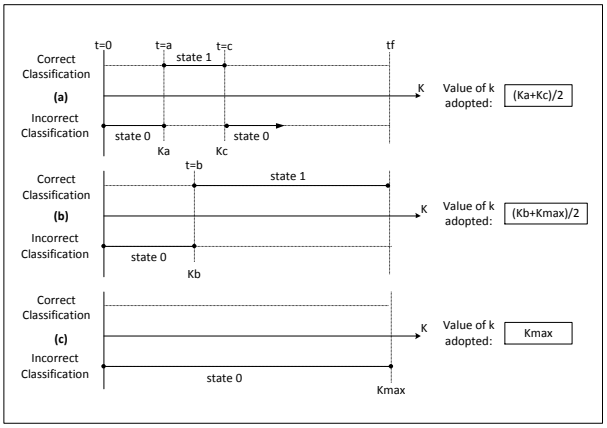
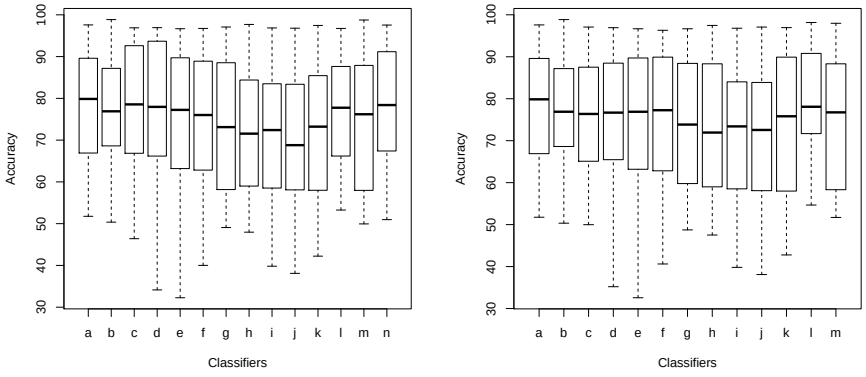


Fig. 3. Possible values of k adopted in the final classifier for a given vertex.



(a) (b)

Fig. 4. *Fig 4(a).* Comparative results between the proposed A-Knn (a) and k-nn (k=1(b), 3(c), 5(d), 7(e), 9(f), 15(g), 20(h), 30(i) and 40(j)), Naive Bayes (k), j48 (l), svm (m) e adaNN (n) through twenty-two data sets, each of them with two levels of noise. and *Fig 4(b).* Comparative results between the proposed A-Knn (a) and the boosting versions of k-nn (k=1(b), 3(c), 5(d), 7(e), 9(f), 15(g), 20(h), 30(i) and 40(j)), Naive Bayes (k), j48 (l), svm (m) through twenty-two data sets, each of them with two levels of noise.

best algorithm gets rank of 1, the second rank 2, and so on. In case of ties, averaged ranks are assigned. Then, the average accuracy for each algorithm was computed which is used in the Friedman test.

Table 1. Domains Specifications.

Domain	# Cases	# Attributes	# Classes
balance (ba)	625	4	3
blood (bl)	748	4	2
breast_tissue (bt)	106	9	6
ecoli (ec)	336	7	8
glass (ga)	214	9	6
Hayes-Roth (hr)	132	5	3
heart-statlog (hs)	270	13	2
ionosphere (io)	351	34	2
iris (ir)	150	4	3
libras (li)	360	90	15
pima_diabetes (pd)	768	8	2
segment (se)	2310	19	7
sonar (so)	208	60	2
statlog (st)	94	18	4
teaching (te)	151	5	3
vehicle (ve)	846	18	4
vowel (vo)	990	11	11
wine (wi)	178	13	3
wine-q-red (wqr)	1599	11	6
w-breast-cancer (wbc)	699	9	2
yeast (ye)	1484	8	10
zoo (zo)	101	16	7

Friedman test uses the F-distribution to calculate a critical value in order to reject the null-hypothesis under a defined confidence level at p <0.05. Consider N the number of data sets and K the number of tested algorithms. In the first study case (comparing the proposed method with k-nn), $N = 66$, $K = 10$, thus, the critical value for our experiment is $CV \approx 1.8959$. Any result with the Friedman test higher than this critical value rejects the null-hypothesis. In our experiment, the value of the Friedman test is $F_f = 7.4733$, so the null-hypothesis is rejected, thus the studied algorithms are not equivalent.

Following with a post-hoc test, we compare the performance of the proposed algorithm with the others. For that, we use the Bonferroni-Dunn test as mentioned in [4]. This test consists of comparing a control algorithm (in this case the proposed algorithm). Additionally, it uses a Critical Difference measure (CD) with a given significance level. Any difference between two pair of algorithms higher than CD shows that they are significantly different. In this case, we use a significance level of $\alpha = 0.05$, thus the critical difference is $CD \approx 1.4615$. Finally, Table 2 presents the comparative study with the proposed algorithm.

Table 2. Comparison of algorithms using the Bonferroni-Dunn test.

	alone versions												
k-nn	k=1	k=3	k=5	k=7	k=9	k=15	k=20	k=30	k=40	NB	C4.5	SVM	adaNN
proposal	-	△	-	-	-	-	-	△	△	△	-	-	-

	Boosting versions								
k-nn	k=1	k=3	k=5	k=7	k=9	k=15	k=20	k=30	k=40
proposal	-	△	△	△	△	-	-	△	△

The second experiment compares our algorithm with Naive Bayes, C4.5 and SVM. We calculated the values of $N = 66$, $K = 4$, $CV \approx 2.6509$, $F_f = 4.1648$ and $CD \approx 0.538$. Hence, the algorithms are not equivalent. In addition, Table 2 presents the significant difference comparing them with our method.

We also carried out experiments comparing our algorithm with ensemble (boosting) of the classifiers. We compare with k-nn versions (k=1, 3, 5, 7, 9, 15, 20, 30 and 40). First, we calculated the values of $N = 66$, $K = 10$, $CV \approx 1.8959$, $F_f = 6.1228$ and $CD \approx 1.4615$. So, the studied algorithms are not equivalent. Table 2 presents the significant difference comparing with our method. Also, in the fourth experiment we compare with Naive Bayes, C4.5 and SVM. First of all, we calculated the values of $N = 66$, $K = 4$, $CV \approx 2.6509$, $F_f = 1.3962$ and $CD \approx 0.538$. So, the studied algorithms are equivalent (there is no significant difference between these methods).

Finally, Table 2 presents results comparing the proposed $A - knn$ with the algorithm $adaNN$. It was used the Wilcoxon test from [4] to determine statistical difference between the two classifiers. In this experiment, it was used a confidence level of $\alpha = 0.05$ and the algorithms did not present statistical difference.

5 Discussion

The relational boosting proposed performs as well or significantly better than the other methods tested, even when compared with ensembles of them.

As posed by Schapire [22], "the final classifier produced by AdaBoost when used, for instance, with a decision-tree base learning algorithm, can be extremely complex and difficult to comprehend. With greater care, a more human-understandable final classifier can be obtained using boosting.". On the other hand, when visualizing the final network of our graph-based boosting we clearly realize the hard examples, as one can see in Fig. 1(d).

The performance of boosting is clearly dependent on the data and the base learner and can fail to perform well given insufficient data, overly complex base classifiers or base classifiers that are too weak as stated by Schapire [22]. Therefore, we see the need to test our method with other base classifiers.

It is interesting to note, as stated by Dietterich, that boosting seems to be especially susceptible to noise [7]. That is, when the number of outliers or noise is very large, the focus on the hard examples can become detrimental to the performance of AdaBoost. Our experiment, however, shows a good performance of the proposed approach on noisy data.

6 Conclusions

In this paper we present a graph-based adaptive k-nn algorithm which incorporates a relational boosting method in its learning phase. The experimental results indicate that the proposed algorithm improves the k-nn classification accuracy and performed as well or significantly better than the other methods tested,

even with ensembles of them. In addition, it shows equivalent results to other adaptive-knn approach ($adaNN$). Furthermore, the visualization of outputs of the graph-based boosting technique sheds lights on data understanding and hard examples detection.

Further work includes adapting and testing the proposed algorithm for networked data. In this case, the parameter k_{max} is not necessary, because we can consider all available edges in the network. Additionally, other relational base classifiers will be tested.

7 Acknowledgments

This research was supported by the Brazilian agency CNPq.

References

1. Bauer, E., Kohavi, R.: An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. In: Machine Learning, vol. 36, pp. 105–139. (1999).
2. Benjamini, Y. : Opening the Box of a Boxplot. In: Proceedings of the American Statistician, pp. 257–262. (1988).
3. Breiman, L. : Bagging Predictors. In: Machine Learning, pp. 123–140. (1996).
4. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. In: Journal of Machine Learning Research. vol. 7, pp. 1–30. JMLR.org. (2006).
5. Dietterich, T. and Wettschereck, D. : Locally Adaptive Nearest Neighbor Algorithms. In: Advances in Neural Information Processing Systems 6, pp. 184–191. Morgan Kaufmann (1994).
6. Dietterich, T. G.: Ensemble Methods in Machine Learning. In: Proceedings of the First International Workshop on Multiple Classifier Systems, pp. 1–15. Springer-Verlag, London, UK (2000).
7. Dietterich, T. G.: An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. In: Machine Learning, pp. 139–157. Kluwer Academic Publishers, Hingham, MA, USA (2000).
8. Freund, Y., Schapire, R. E. : Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning, pp. 148-156, Morgan Kaufmann Publishers Inc, Bari, Italy (1996).
9. García-Pedrajas, N., Ortiz-Boyer, D.: Boosting k-nearest neighbor classifier by means of input space projection. In: Expert Systems with Applications, vol. 36,pp. 10570–10582. Pergamon Press, Inc., Tarrytown, NY, USA (2009).
10. Grabowski, S.: Voting over multiple k-nn classifier. In: Proceedings of the international conference on modern problems of radio engineering, telecommunications and computer science, pp. 223–225. (2002).
11. Hall, M., Frank E., Holmes, G., Pfahringer, B., Reutemann P., Witten,I. H.: The WEKA data mining software: an update. In: SIGKDD Explor. Newsl. , vol. 11, pp. 10–18. ACM (2009).
12. John, G. H., Langley, P. : Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, pp. 338-345, Morgan Kaufmann Publishers Inc, San Mateo (1995).

13. Leigh, W. and Purvis, R. and Ragusa, J. M.: Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural networks, and genetic algorithm: a case study in romantic decision support. In: Decision Support Systems, vol. 32,pp. 361–184. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (2002).
14. Lin, H., Kao, Y., Yang, F., Wang, P.: Content-based image retrieval trained by AdaBoost for mobile applications. In: International Journal of Pattern Recognition and Artificial Intelligence, vol. 20, pp. 525–541. (2006).
15. Macskassy, S. A. and Provost, F. : A simple relational classifier. In Proceedings of the Multi-Relational Data MiningWorkshop (MRDM) at the Ninth ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 64–76 (2003).
16. Maimon, O., Rokach, L.: Ensemble of Decision Trees for Mining Manufacturing Data Sets. In: Machine Engineering, vol. 4, pp. 32–57. (2004).
17. Min, R. : Adaptive KNN classification based on Laplacian Eigenmaps and kernel mixtures. Technical report, University of Toronto. (2008).
18. Paulovich, F. V. , Oliveira, M. C. F. a and Minghim, R. : The Projection Explorer: A flexible tool for projection-based multidimensional visualization. In: Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing - SIBGRAPI, pp. 27–36. IEEE CS Press, Belo Horizonte, Brazil (2007).
19. Platt, J. C.:Fast training of support vector machines using sequential minimal optimization. In: Advances in kernel methods. pp. 185–208. MIT Press, Cambridge, MA, USA (1999).
20. Polikar, R: Ensemble based systems in decision making. In: IEEE Circuits and Systems Magazine, vol. 6, pp. 21-45. IEEE Press (2006).
21. Quinlan, J. R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (1993).
22. Schapire, R. E. : The Boosting Approach to Machine Learning: An Overview. `http://www.cs.princeton.edu/~schapire/uncompress-papers.cgi/msri.ps` (2002).
23. Sun, S. and Huang, R. : An adaptive k-nearest neighbor algorithm. In: Fuzzy Systems and Knowledge Discovery (FSKD), pp. 91–94.(2010).
24. Tan A. C., Gilbert D., Deville Y.: Multi-class Protein Fold Classification using a New Ensemble Machine Learning Approach. In: Genomic Informatics, vol. 14, pp. 206–217. Japan (2003).
25. UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml`.
26. Yu, X. and Xiaogao, Y. and Zhou, D. : A Depth-first Adaptive KNN Searching Algorithm. In: Intelligent Control and Automation, pp. 10318–10322.(2006).
27. Wang, J. and Neskovic, P. and Cooper, L. : Neighborhood size selection in the k-nearest-neighbor rule using statistical confidence. In: Pattern Recognition, pp. 417–423.Elsevier Science Inc. New York, NY, USA (2006).
28. Zhou, Z. H., Yu, Y. : Adapt bagging to nearest neighbor classifier. In: Journal of Computer Science and Technology, vol. 20, pp. 48–54. (2005).
29. Zhu, X. : Semi-supervised learning literature review. Technical report, University of Wisconsin-Madison (2005).
30. Zouhal, L. and Denoeux, T. : An Adaptive K-NN Rule Based on Dempster-Shafer Theory. In: Proceedings of the 6th International Conference on Computer Analysis of Images and Patterns, pp. 310–317. Springer-Verlag, London, UK (1995).