Instance-Based Prediction of Continuous Values

Tony Townsend-Weber and **Dennis Kibler**
Information and Computer Science
University of California, Irvine, 92715 U.S.A.
tweber@ics.uci.edu, kibler@ics.uci.edu

Abstract

Learning to predict a continuous value rather than a discrete class is an important problem in machine learning. Instance-based algorithms can be effectively used to solve this problem. Two methods of classification using weighted and unweighted averages, based on the k-nearest-neighbor algorithm, are presented and shown empirically to do equally well. Two methods for eliminating irrelevant attributes, 1-lookahead and sequential backward elimination, are presented and shown empirically to do equally well. In seven domains, choosing the best k for k-nearest-neighbor is shown to reduce the classification error by 1.3% over arbitrarily using $k = 3$, and eliminating irrelevant attributes reduces the error by 0.5%. Instance-based algorithms are compared to several other algorithms, including regression + instances, model trees + instances, neural networks + instances, regression trees, and regression rules. The instance-based approach is the best in a few domains, but overall is slightly less accurate than model trees + instances and neural networks + instances.

Introduction

Classification algorithms typically use discrete-valued attributes to predict discrete-valued classes. When values are continuous, they must be discretized before the algorithm can run. A good deal of information can be lost this way. Recent work in machine learning has focused on algorithms that use continuous-valued attributes without discretizing, and predicting continuous values. Weiss and Indurkhya (1993) build an ordered set of regression rules which can be thought of as a continuous-valued analog of decision lists. A new instance is classified by the first rule that it matches; each rule predicts the mean of the values it covers. Weiss and Indurkhya claim regression rules achieve higher accuracy than regression trees, such as built by CART (Breiman *et al*, 1984). Van de Merckt (1993) uses a decision tree with the class average at each leaf. The splitting metric is contrast-based and unsupervised, which chooses sets that are dense and far apart.

Quinlan's (1993) M5 builds a 'model tree', which is a decision tree with the standard attribute/value split at each node, plus a linear equation at each node. This linear equation is built with standard multivariate linear regression based on minimizing the sum of squares. Only the examples in the subtree formed by a node are used to construct the linear equation at that node. The splitting metric is based on the standard deviation, choosing sets with low variances; i.e. they are dense but not necessarily far apart. To predict a value, an example first traverses down the tree to a leaf node, where its value is given as the average of all the examples in that leaf. Then it travels back up the tree to the root in a "smoothing" process; at each internal node the predicted value is modified by the linear equation at that node. The new value at a node is the weighted average of the old value and the value predicted by the linear equation at that node, where the weight of the old value is the number of examples in the subtree formed by the node, and the weight of the linear equation's prediction is a fixed constant (mysteriously chosen to be 15).

In the same paper, Quinlan combines instance-based and model-based methods to predict continuous values. Instance-based algorithms explicitly choose some set of examples $\{P_1, P_2, \ldots, P_k\}$ to predict the class value of a new instance. Model-based methods form a generalization of examples in some language; they do not store the examples themselves. Quinlan uses three model-based methods in his experiments: model trees, equations built by multivariate linear regression, and artificial neural networks. Let the class value of an example P be $V(P)$ and the predicted value of P by a model be $M(P)$. To combine instance-based and model-based methods, Quinlan modifies the prediction of instance-based methods. If a given instance-based algorithm would use values $\{V(P_1), V(P_2), \ldots, V(P_k)\}$ to predict the value of a new instance U, now it uses the values

$$V(P_i) - (M(P_i) - M(U))$$

instead of $V(P_i)$ for $i = 1 \ldots k$. This takes into account the difference between an example instance P_i and the test instance U as predicted by the model. For exam-

ple, suppose we use 3-nearest-neighbor (3-NN) as our instance-based algorithm. The closest examples to new instance U are $\{P_1, P_2, P_3\}$. Using the 3-NN algorithm alone, we would predict the value of U to be

$$\frac{1}{3}\left(V(P_1) + V(P_2) + V(P_3)\right).$$

By combining it with a model-based method, we now predict the value of U to be

$$\frac{1}{3}\begin{pmatrix} V(P_1) - (M(P_1) - M(U)) + \\ V(P_2) - (M(P_2) - M(U)) + \\ V(P_3) - (M(P_3) - M(U)) \end{pmatrix}.$$

Quinlan uses this 3-nearest-neighbor algorithm as his instance-based method, which predicts the class value to be the (unweighted) average of the classes of the three most similar instances. In this paper we show the utility of choosing the best k for the k-nearest-neighbor (k-NN) algorithm, and compare a weighted-average method of classification to the unweighted-average method. Quinlan also uses all attributes to compute similarity. We show that eliminating irrelevant attributes improves the classification accuracy.

Algorithms

Using instance-based learning, we describe two methods of predicting real values (weighted and unweighted) and two methods of eliminating irrelevant attributes (1-lookahead and sequential backward elimination).

Instance-based learning relies on keeping all previously seen instances. When attempting to classify a new instance, the k-NN algorithm picks the k instances that are most similar to the new instance, and uses those k instances to predict the class of the new one. The similarity of two instances X and Y is given by the formula

$$\text{Similarity}\ (X, Y) = \sum_{i=1}^{d} 1 - \mid X_i - Y_i \mid$$

where d is the number of attributes of each instance, and X_i is the normalized value of the ith attribute of X. "Normalized" means divided by the range of the attribute, so that $X_i \in [0, 1]$ for every attribute i and instance X. This causes all attributes to have an equal weight in the similarity. Similarity between two instances varies between 0 and d, with a similarity of d indicating the two instances are identical. If attribute i is symbolic, then we define $\mid X_i - Y_i \mid$ in the above equation to be 0 if the values are the same and 1 if the values are different.

When the k most similar instances are chosen, two methods can be used to predict the class of the test instance. The *unweighted method* predicts the class value to be the average of the class values of the k known instances. The *weighted method* is a weighted average of the k known class values where the weight of an instance X is the similarity between X and the test instance. The intuition is that if a known instance is very similar to the test instance, its class will be more similar to the unknown class than the class of another known instance that is farther away. For 2-NN in two dimensions, the weighted method will give the interpolated point of two instances on each side of the test instance. The weighted method is due to Kibler, Aha, and Albert (1989). This paper extends their work by comparing the weighted method with the unweighted method and examining the effect of choosing the best k and eliminating irrelevant attributes.

The k-NN algorithm can be used simply by choosing k and determining whether to use the weighted method or the unweighted method of prediction. However, by preprocessing the training data to choose the best k for the domain and eliminating irrelevant attributes, we can increase the accuracy of the algorithm.

To determine the best k, the training data is broken down into two sets, a subtraining set and a subtest set. The subtest set is classified using the subtraining set with many different values of k. The best k is the one that has the lowest error. We use leave-one-out testing to choose the best k since this gives the largest subtraining sets. The subtest set is a single instance, and the subtraining set is all the other instances.

Every instance in the training set is classified using many values of k, and the best k is found by taking the lowest average error over all n instances in the training set. We test k from 1 to d, where d is the number of attributes of each instance. This upper value of d has nice theoretical properties (described below) and is a good balance between testing too few and too many values of k. The total time to choose the best k in this fashion is $O(n^2 d)$. The derivation of this complexity follows. Computing the similarities between the one subtest instance and $n-1$ subtraining instances takes $O(nd)$ time. Then we choose the d most similar instances and sort them based on similarity. This can be done in $O(n + d \log n)$ using d iterations of heapsort, but we use an easier method based on insertion sort that takes at most $O(nd)$ time. The errors for all d k-NN can be calculated in $O(d)$ time, since the error with $k = i$ can be computed from the error with $k = i - 1$ in constant time. This whole process is performed n times, and the best k is found in $O(d)$ time. The total time, then, is $O(n^2 d)$. If we tested k from 1 to n instead of from 1 to d, it would raise the complexity to $O(n^2 d + n^2 \log n)$. Testing k from 1 to x, where x is any value less than d, still has a complexity of $O(n^2 d)$. So the upper value of d is the maximum value that will not change the complexity (barring addition of constants).

The two methods for eliminating irrelevant attributes are also based on classification accuracy. Again, we do leave-one-out testing on the training set. The *1-lookahead* method of eliminating irrelevant attributes is as follows. One attribute at a time is re-

Loop until error can not be lowered any further:

- For each attribute i
  - Remove attribute i
  - Classify all training instances using leave-one-out testing. Compute the average error.
  - Reinstate attribute i.
- Permanently remove the attribute whose previous removal produced the lowest error, provided that this error is lower than the current error.

Figure 1: The 1-lookahead method of eliminating irrelevant attributes.

For each attribute i (in random order)

- Remove attribute i
- Classify all training instances using leave-one-out testing. Compute the average error.
- If the error is lower than the current error, permanently remove attribute i. Otherwise reinstate it.

Figure 2: The SBE method of eliminating irrelevant attributes.

moved, then all instances are classified and the average error computed, then the attribute is reinstated. The attribute that reduced the error the most is then thrown out permanently. The cycle repeats until the error cannot be improved by eliminating any attribute. The pseudocode is given in Figure 1. The time complexity is $O(n^2d^3)$. The best k is chosen at each leave-one-out testing phase, since it does not change the complexity.

In the *sequential backward elimination (SBE)* (Kittler, 1986) method of eliminating attributes, we first randomize the order of the attributes. We make one pass over the attributes, eliminating an attribute if it reduces the error. Standard SBE has a parameter that specifies the maximum number of attributes that will be thrown out. We set this parameter to be $d - 1$, where d is the number of attributes. Figure 2 gives the pseudocode. The time complexity is $O(n^2d^2)$, a factor of d less than the 1-lookahead method. The 1-lookahead method corresponds to steepest-ascent hill-climbing and SBE corresponds to greedy hill-climbing. In other domains such as *N queens* (Minton *et al* 1990) and *3-SAT* (Selman and Kautz, 1993), greedy hill-climbing has been found to do as well as steepest-ascent. The same pattern is repeated in our results.

Domain	Cases	Continuous attributes	Discrete attributes	Range
housing	506	13	0	45
cpu	209	6	2	1144
auto-price	159	2	7	29938
auto-mpg	392	7	0	37.6
servo	167	2	2	6.97
iris	150	3	0	2.4
heart	297	6	6	48
sinormal	150	7	0	0.794

Figure 3: Basic features of the eight domains.

Results

We ran five versions of k-NN on eight domains. The versions include the four combinations of (weighted, unweighted) and (1-lookahead, SBE) k-NN and a control—unweighted 3-NN using all attributes. The domains include seven natural domains and one artificial domain. We chose these domains so that we could compare our results with those of Quinlan (1993) and Weiss and Indurkhya (1993). The natural domains are housing, cpu, auto-price, auto-mpg, servo, iris, and heart, all from the UCI repository. Figure 3 gives the number of examples in each domain, as well as the number of continuous and discrete attributes, and the range of the class value. The iris domain contains four continuous attributes and one class label. We eliminate the class labels and predict the fourth attribute, as Weiss and Indurkhya did. Similarly, class labels in the heart domain are eliminated and we predict the person's age. The artificial domain is *sinormal*, from Weiss and Indurkhya, which is 150 randomly chosen instances of the function

$$\text{sinormal}(x_1, x_2, \ldots, x_7) = 0.2 \sin x_1 + \text{normal}(x_2)$$

where *normal* is the standard normal curve with mean of 0 and standard error of 1, $x_1 \in [0, 2\pi]$, $x_2 \in [-5, 5]$, and $x_i \in [0, 1]$ for $i = 3, 4, 5, 6, 7$. $x_3, \ldots, x_7$ are noise attributes. We used stratified sampling and ten-fold cross validation, as the other authors did. We wished to answer the following questions:

1. Which method of classifying (weighted or unweighted) is best?
2. Which method of eliminating irrelevant attributes (1-lookahead or SBE) is best?
3. How well does k-NN, with choosing the best k and eliminating irrelevant attributes, fare against other classification algorithms?
4. What is the effect of choosing the best k?
5. What is the effect of eliminating irrelevant attributes?

Figure 4 gives the results of the five versions on the eight domains. Statistical significance was checked

Domain	Control: unweighted 3-NN	weighted, SBE k-NN	weighted, 1-lookahead k-NN	unweighted, 1-lookahead k-NN	unweighted, SBE k-NN
housing	2.63	**2.37***	2.45*	2.45*	2.37*
cpu	35.4	26.8*	**25.9***	26.0*	27.0*
auto-price	**1689**	1793	1752	1719	1773
auto-mpg	**1.90**	2.00	2.01	1.97	2.02**
servo	0.99	**0.42***	0.44*	0.44*	0.42*
iris	**0.147**	0.152	0.147	0.147	0.152
heart	7.51	6.49*	6.47*	**6.47***	6.49*
sinormal	0.12	0.045*	**0.035***	0.035*	0.045*
Average	8.5%	5.9%	5.9%	5.9%	5.9%

* Statistically better than the control, at the 1% significance level.
** Statistically worse than the control, at the 1% significance level.

Figure 4: Average, absolute error on unseen cases. The 'Average' column represents the average over all domains of the absolute error as a fraction of the range of the class value. Boldface numbers are the lowest in their row. Statistical significance between the control and the four k-NN variations are indicated. There is no statistical difference between any of the four k-NN variations in any natural domain (the first seven), at the 1% significance level. The weighted variations in the *sinormal* domain are significantly better than the unweighted variations, at the 1% level.

by the one-tailed t-test, since all algorithms were performed on exactly the same partitions of data. In five domains, choosing the best k and eliminating irrelevant attributes was significantly better than the control 3-NN. Only one algorithm in one domain performed statistically worse than the control (unweighted, SBE k-NN in the auto-mpg domain). The weighted and unweighted classification methods do equally well; there is no statistical difference between them in any natural domain at the 1% significance level. Even at the 5% level, only one comparison in one natural domain (out of 28 comparisons total for all natural domains) was significant: weighted 1-lookahead k-NN was statistically better than unweighted 1-lookahead k-NN in the housing domain. We hypothesize that the methods do equally well in the natural domains because the concepts cannot be modeled well as continuous functions. Also, the best values of k are small, between 2 and 4, and thus the difference between the predicted values using the two methods will be small, especially when the instances are close together.

In the artificial domain *sinormal*, the weighted variations did statistically better than the unweighted variations, although by a tiny amount. From Figure 4 it appears that the weighted and unweighted methods did equally well, and the significant difference is between the 1-lookahead and SBE attribute elimination methods. However, exactly the opposite is true, which is only shown by statistical tests. The difference between the weighted and unweighted methods is significant, though tiny, and the difference between the SBE and 1-lookahead methods is not significant, though large. The *sinormal* function is smooth, and the weighted method predicts a value which is close to the interpolated value, which is better than an unweighted average. Therefore it makes sense that the weighted method consistently did better than the unweighted method in this domain.

The SBE and 1-lookahead attribute selection methods also do equally well. No difference was statistically significant, even at the 5% level. We hypothesize that this is due to the origin of the databases for the domains. These databases are usually processed before they reach the UCI repository. Most attributes are significant to some extent, and the attributes that are not significant are obviously irrelevant. Often these attributes will be descriptions of an example, such as the name of a person in a medical database. These obviously irrelevant attributes will be eliminated by either the SBE or 1-lookahead methods. The extra lookahead is not needed. Therefore, an algorithm designer should use SBE, since it is faster and simpler.

Eliminating irrelevant attributes and choosing the best k produces lower errors than the control 3-NN in five of eight domains. In the cases where the control seems to do better, the difference is significant for only one algorithm in one domain. It is surprising that this happened in even one case, however. Our explanation goes as follows. Both 1-lookahead and SBE methods will throw out an attribute even when the improvement in error is tiny and not statistically significant. This may result in overfitting the training data, and thus a higher error when classifying unseen instances. In the auto-mpg domain, the best k was usually three or four, and no single attribute was always dropped. This indicates that all attributes are significant, and the averaged 3-NN is the best that can be done. This fits our hypothesis. A future improvement on the al-

	housing	cpu	auto-price	auto-mpg	servo	Average
Control: unweighted 3-NN	2.63	35.4	1689	**1.90**	0.99	6.8%
Quinlan's unweighted 3-NN	2.90	34.0	1689	2.72	0.52	6.0%
regression + instances	2.45	30.0	1430	2.37	0.48	5.2%
model trees + instances	2.32	28.1	**1386**	2.18	0.30	4.5%
neural nets + instances	**2.23**	29.0	1677	2.06	**0.29**	4.5%
weighted, SBE k-NN	2.45	**26.0**	1719	1.97	0.44	5.0%

Figure 5: Average, absolute error on unseen cases. Boldface numbers are the lowest in their column. The 'Average' column represents the average over all domains of the absolute error as a fraction of the range of the class value.

	sinormal	iris	heart
weighted, 1-lookahead k-NN	0.035	0.147	6.47
regression trees (CART)	0.075	0.142	6.05
regression rules	0.052	0.142	6.05

Figure 6: Average, absolute error on unseen cases.

gorithm will be to only eliminate an attribute when it is statistically significant to do so.

Figure 5 compares the weighted SBE k-NN with the results of three systems from Quinlan (1993): regression + instances, model trees + instances, and neural networks + instances. First, note the differences between our averaged 3-NN control and Quinlan's averaged 3-NN. These occur because of different partitions of the data, and indicate the difficulty of directly comparing results. No statistical tests could be run since we do not have the same partitions of data. In the cpu domain k-NN clearly does best of all approaches. In the auto-mpg domain our control 3-NN does best, but since it is substantially different from Quinlan's 3-NN, no conclusions can be drawn. In the other domains, our k-NN does worse than model trees + instances and neural nets + instances. The 'average' column gives a rough indication of the algorithms: neural nets + instances and model trees + instances have the lowest errors, then our k-NN, then regression + instances, and finally the control 3-NN algorithms. Note that the difference between our control 3-NN and Quinlan's is 0.8%, which is due to different partitions of the data. This indicates that the error in the 'average' column is also this much. If we scaled our k-NN by this 0.8% so that both of our 3-NN systems agreed, our k-NN would achieve a 4.2% error rate, which makes it the most accurate algorithm! Thus although neural nets + instances and model trees + instances seem to do better than k-NN, these results could well be reversed by re-running all algorithms on the same data partitions and doing a fairer comparison.

Figure 6 presents a comparison of k-NN with regression trees and regression rules from Weiss and Indurkhya (1993). The regression trees were built by CART. k-NN does much better in the artificial domain *sinormal*, and slightly worse in the natural domains iris and heart. In the *sinormal* domain, both SBE and 1-lookahead methods correctly eliminated all five noise attributes. Regression trees and regression rules do not eliminate irrelevant attributes; this is the most likely reason that k-NN does so much better. In the iris and heart domains, regression rules and regression trees do equally well, and have lower errors than k-NN. It is unknown if the differences are statistically significant.

Figure 7 shows the effects of choosing the best k and eliminating irrelevant attributes in the seven natural domains. The first row is the percentage difference between the worst-k classifier and the best-k classifier, averaged over all leave-one-out training phases of each of the ten folds of cross validation, and over all four k-NN variations (weighted/unweighted and SBE/1-lookahead). All attributes were used to classify, so there would be no effect of SBE vs. 1-lookahead on choosing the best k. The average across the seven domains is a 2.3% decrease in error. Since many researchers (including Quinlan) arbitrarily choose $k = 3$ for k-NN, we wished to measure what benefit choosing the best k had over this policy. The second row shows the difference between the 3-NN classifier and the best k classifier, again averaged over all runs, folds, and k-NN variations, and using all attributes. The average across the seven domains is a 1.3% decrease in error. The third row shows the decrease in error that comes from eliminating irrelevant attributes, averaged over all runs, folds, and variations. The average is a 0.5% decrease in error. We hypothesize that this improvement is so slight because, again, most attributes were significant in these domains, given that they are usually hand-tuned before reaching the UCI repository. Figure 8 presents the best k found in each domain, averaged over all training runs, folds, and variations. Values of two, three, and four are most common. An average of 9.5 in the heart domain is an unusual case. We conclude that choosing the best k and eliminating irrelevant attributes is a good idea when accuracy is the main priority. If simple, quick classification is needed with reasonable accuracy, 3-NN is a good choice.

	housing	cpu	auto-price	auto-mpg	servo	iris	heart	Average
worst-k-NN - best-k-NN	1.3	1.1	1.5	1.0	7.9	0.7	3.5	2.3
3-NN - best-k-NN	0.0	0.4	0.1	0.1	7.5	0.1	1.1	1.3
all attr. - best attr.	0.8	0.5	0.6	0.1	0.7	0.2	0.7	0.5

Figure 7: Effect of choosing the best k and eliminating irrelevant attributes. All results are percentages of the range of the class value.

housing	cpu	auto-price	auto-mpg	servo	iris	heart	sinormal	Average
3.1	2.1	2.9	3.8	2.5	3.6	9.5	3.3	3.8

Figure 8: Average best k

Conclusions

Instance-based algorithms can be used effectively to predict continuous values. Choosing the best k and eliminating irrelevant attributes generally improves the accuracy of the k-NN algorithm. In seven natural domains, there is no difference between the weighted and unweighted methods of classification, or between the 1-lookahead and SBE methods of removing attributes. In five domains, k-NN was the best in one and reasonable in the others, when compared to model trees + instances, neural networks + instances, and regression + instances (Quinlan 1993). The k-NN algorithms had lower errors than regression + instances, but higher errors than model trees + instances and neural networks + instances. However, the results are inconclusive because of a difference in identical control experiments resulting from different partitions of data. The k-NN algorithms had lower errors than regression trees and regression rules (Weiss and Indurkhya 1993) in one artificial domain because of the facility of eliminating irrelevant attributes. In two natural domains, regression trees and regression rules were more accurate, but it is unknown if the differences are statistically significant. In seven natural domains, choosing the best k lowered the error an average of 1.3% over arbitrarily using $k = 3$, and eliminating irrelevant attributes lowered the error an average of 0.5%.

Future work will include investigation of why choosing the best k and eliminating attributes sometimes results in a higher error. Eliminating an attribute only when the error reduction is statistically significant may solve the problem. Weighing attributes instead of throwing them out may be an even better approach, since most attributes in the domains we studied contained some predictive information. We have developed but not implemented a continuous-valued analog to Kira and Rendell's (1992) RELIEF system of weighing attributes.

Acknowledgments

Our sincere thanks goes to Doo Song for his help with the statistical verification of the algorithms and to Mike Pazzani for several insightful comments on the paper.

References

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984), *Classification and regression trees*, Belmont, California: Wadsworth International Group.

Kibler, D., D. W. Aha, and M. K. Albert (1989), Instance-based prediction of real-valued attributes, *Comput. Intell. 5*, 51–57.

Kira, K. and L. A. Rendell (1992), The feature selection problem: traditional methods and a new approach. *Proceedings AAAI 92*, 129–134.

Kittler, J. (1986), Feature selection and extraction. In Young & Fu (Eds.), *Handbook of pattern recognition and image processing.* New York: Academic Press.

Minton, S., M. D. Johnston, A. B. Philips, and P. Laird (1990): Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *Proceedings AAAI 90*, 17–24.

Quinlan, J. R. (1993), Combining instance-based and model-based learning, *Machine Learning Conference 1993*, 236–243.

Selman, B. and H. Kautz (1993). An empirical study of greedy local search for satisfiability testing. *Proceedings NCAI-93*, 46–51.

Van de Merckt, T. (1993), Decision trees in numerical attribute spaces, *Proceedings IJCAI 93*, 1016–1021.

Weiss, S. M. and N. Indurkhya (1993), Rule-based regression, *Proceedings IJCAI 93*, 1072–1078.