

# Feature Weighting Using Neural Networks

Xinchuan Zeng and Tony R. Martinez

Computer Science Department, Brigham Young University, Provo, Utah 84602

E-mail: zengx@axon.cs.byu.edu, martinez@cs.byu.edu

**Abstract**—In this work we propose a feature weighting method for classification tasks by extracting relevant information from a trained neural network. This method weights an attribute based on strengths (weights) of related links in the neural network, in which an important feature is typically connected to strong links and has more impact on the outputs. This method is applied to feature weighting for the nearest neighbor classifier and is tested on 15 real-world classification tasks. The results show that it can improve the nearest neighbor classifier on 14 of the 15 tested tasks, and also outperforms the neural network on 9 tasks.

## I. INTRODUCTION

The quality of features has a significant impact on the performance of a learning algorithm for classification tasks. The performance can be degraded if there are irrelevant or redundant features, which are often inevitable for tasks whose domains are not well understood.

The impact of features is particularly significant for the class of instance-based learning algorithms, such as the nearest neighbor algorithm [1], in which the distance function is very sensitive to the quality of features. How to improve the quality of features has been one of the critical issues concerned in the field of instance-based learning.

Various approaches have been proposed in the past to address this issue. One strategy is *feature selection* that applies a method to find irrelevant features and remove them from the feature set before constructing a classifier. For example, Cardic [2] applied a C4.5 decision tree to find relevant features by keeping only those features that appear in the decision tree. The set of the selected features is then fed into a nearest neighbor classifier. Experiment on a lexical acquisition task shows improvement using this method. John et. al. [3] applied a cross-validation method to filter irrelevant features before constructing ID3 and C4.5 decision trees. The experiment on 3 artificial datasets and 3 real-world datasets shows significant reduction in the size of constructed decision trees as well as slight improvement in accuracy.

The strategy used in this work is *feature weighting*, which seeks to estimate the relative importance of each feature (with respect to the classification task), and assign it a corresponding weight. When properly weighted, an important feature would receive a larger weight than less important or irrelevant features. Instead of making a binary decision on a feature's relevance (as applied by feature selection methods), feature weighting uses a continuous value and thus has a finer granularity in determining the relevance. It is more suitable for tasks in which some attributes are more relevant than others.

Salzberg [4] proposed a method to weight features in the system EACH (a variant of the nearest neighbor algorithm).

It is based on feedback of the performance in the training set. When a training instance is correctly classified using leave-one-out testing, the weights of matched features are incremented by a fixed amount while the weights of mismatched features are decremented by the same amount. The opposite action is taken for a misclassified instance. Salzberg [4] found that with a proper weight adjustment parameter, this method is able to achieve improvement on classification performance.

Another feedback-based feature weighting method was proposed by Aha [5] in the instance based learning algorithm IB4. In this approach, the weight adjustment parameter is not fixed but depends on the class distribution, which is more robust against skewed class distributions. Aha [5] reported improved results using IB4 compared to the nearest neighbor algorithm on tasks involving irrelevant features.

Ling et. al. [6] extended the work (a feature selection method) of Cardic [2] to use a C4.5 decision tree to weight features for the nearest neighbor classifier. The weight of a feature depends on its node position in the tree, as well as the number of training examples going through the node. The algorithm was tested on an artificial dataset with irrelevant features as well as on some real-world datasets. The results show an improvement in performance by using this feature weighting method over the feature selection method applied in [2]. Other feature weighting methods can be found in a review paper by Wettschereck et. al. [7].

In this work, we propose a feature weighting method that is based on a trained neural network. In this method, the importance of a feature is extracted from the strengths of related links in a trained neural network. The rationale behind this idea is that an important feature should have strong links along the nodes correlated to this feature, because of its influence on classification decisions. One advantage for using a neural network for feature weighting is its rich expressiveness in representing hypotheses (as shown by Hornik et.al. [8], a neural network has the capacity of representing virtually any function). This provides neural networks with the potential to better capture the relevance of features related to classification.

This method is tested on 15 real-world datasets drawn from UCI dataset repository. The results show its capability of improving the performance of the nearest neighbor classifier in 14 of the 15 tested datasets. It also performs better than the neural network in 9 datasets. The results also show that it is particularly effective in improving the nearest neighbor classifier when there are irrelevant attributes in a dataset.

## II. FEATURE WEIGHTING METHOD

The proposed feature weighting method is focused on the procedure to extract information about the relevance of features from a trained neural network. Given a training set  $S$ , the first step is to train a multi-layer neural network using the backpropagation algorithm. The learning rate is set to 0.2 and the momentum is set to 0.5. The other parameters of the neural network and training procedure are described as follows.

For a given training set  $S$ , it is randomly divided into two subsets: a training set  $S_1$  (2/3 of  $S$ ) and a holdout set  $S_2$  (1/3 of  $S$ ). The network is trained on  $S_1$  and its accuracy is estimated on  $S_2$ . The purpose of the holdout set is to estimate the accuracy on an independent dataset (instead of the training set itself) in order to avoid over-fitting the training data.

The network is constrained to have one hidden layer. The number of hidden nodes  $H$  is not pre-determined. Instead, it is automatically determined by monitoring its accuracy on  $S_2$ .  $H$  is initially set to 1. After training for a fixed  $H$  (the training procedure for a fixed  $H$  is explained in the next paragraph), the network accuracy on the holdout set  $S_2$  is recorded.  $H$  is then incremented by 1. This continues until finding the best  $H$  by the following criterion: if  $H$  gives a best result while  $H+1$  and  $H+2$  do not yield a better result, then  $H$  is considered to be the best number of hidden nodes. After determining the best  $H$ , the whole training set  $S$  (to better utilize available training data) is applied to train a network with the fixed  $H$  (starting from the saved initial weight settings for the best  $H$ ).

For a fixed number of hidden nodes  $H$ , the accuracy on  $S_2$  is monitored after every  $N$  ( $=10$ ) epochs. If after monitoring  $M$  ( $=20$ ) times (i.e.,  $N \times M = 200$  epochs), there is still no progress in training (no better accuracy on  $S_2$ ), then training is stopped for the fixed number of hidden nodes  $H$ .

After the training, the feature weighting is extracted from the trained network as follows. For an input node  $i$ , its feature weight is given by

$$W_i = \sum_{j=1}^H \sum_{k=1}^O |V_{i,j} \times V_{j,k}| \quad (1)$$

In the above equation,  $W_i$  is the feature weight for input node  $i$ ;  $V_{i,j}$  is the network weight (link strength) from input node  $i$  to hidden node  $j$ , and  $V_{j,k}$  is the weight from hidden node  $j$  to output node  $k$ ;  $H$  is the number of hidden nodes and  $O$  the number of output nodes. Each term in Eq. (1) represents one path from input node  $i$  to output node  $k$  through hidden node  $j$ . The summation covers all possible forward paths from input node  $i$  to all output nodes.

The rationale for using Eq. (1) to weight feature is that if a feature is important, it will have more influence on the output nodes by propagating forward through hidden nodes. Such influence is reflected in the strengths of links along all the related paths, and the purpose of Eq. (1) is to give a quantified overall measure on such influence.

There is a slight difference for extracting feature weighting for continuous and nominal features. For a continuous feature, there is only one corresponding input node, and its feature

weight is just the same as the one given by Eq. (1). For a nominal feature with  $T$  distinct values, there are  $T$  input nodes, with each one corresponding to a different value of the nominal feature. In this case, the feature weight is the average over the  $T$  input nodes:

$$W = \frac{1}{T} \sum_{t=1}^T W_t \quad (2)$$

After extracting weights for all features using Eq. (1) and Eq. (2), the weights are normalized such that their sum is equal to the number of features  $N$ . They are then applied for the nearest neighbor algorithm to calculate a weighted distance between any two data instances.

For two given data instances with feature vectors  $\mathbf{x} = (x_1, x_2, \dots, x_N)$  and  $\mathbf{y} = (y_1, y_2, \dots, y_N)$ , the weighted distance between them is given by

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^N W_i \times h(x_i, y_i) \quad (3)$$

In Eq. (3),  $h(x_i, y_i)$  is the difference between two feature values  $x_i$  and  $y_i$ . It is calculated differently for continuous and nominal features. For a continuous feature,

$$h(x_i, y_i) = |x_i - y_i| \quad (4)$$

For a nominal feature,

$$\begin{aligned} h(x_i, y_i) &= 1 && (x_i = y_i) \\ &= 0 && (x_i \neq y_i) \end{aligned} \quad (5)$$

## III. EXPERIMENT

This feature weighting method is evaluated on 15 real-world datasets drawn from UCI machine learning dataset repository [9]. Table 1 lists the properties of these datasets, including number of continuous attributes ( $C$ ), number of nominal attributes ( $N$ ), total number of attributes ( $Attr$ ), number of classes, and dataset size.

A stratified 10-fold cross-validation [10] is applied to estimate the error rate. Each dataset is randomly partitioned into 10 equally sized folds, with stratification for each class (i.e. instances with same class are evenly distributed across the 10 folds). In each run, nine of the ten folds are used as the training set while the other fold is the test set. This repeats 10 times, with a different fold as the test set each time. The reported error rate for a dataset is the average over the 10 runs.

Three algorithms are evaluated: the baseline nearest neighbor algorithm with constant feature weights (**1-NN**), a neural network trained with the backpropagation (**BP**), and the nearest neighbor algorithm with weighted features by neural network (**BP-1-NN**) as explained in the last section. The results on the 15 datasets are shown in Table 2.

The results show that the nearest neighbor algorithm using the proposed weighting method (**BP-1-NN**) performs better than the baseline nearest neighbor algorithm (**1-NN**) in 14 of the 15 tested datasets (except *Flag*), showing a strong

TABLE I  
PROPERTIES OF 15 DATASETS

<i>Data Set</i>	C	N	Attr	Class	Size
Chess	0	36	36	2	3196
Flag	3	25	28	8	194
Glass	9	0	9	7	214
Horse	12	14	26	3	366
Hypthoid	7	18	25	2	3163
Iono	34	0	34	2	351
Iris	4	0	4	3	150
Letter	16	0	16	26	20000
Lymph	0	18	18	4	148
PageBlk	10	0	10	5	5473
Promot	0	57	57	2	106
SatImge	36	0	36	6	6435
Segment	19	0	19	7	2310
Vowel	10	0	10	11	990
Water	38	0	38	13	527

TABLE II  
ERROR RATES (%) OF THE THREE ALGORITHMS

<i>Data Set</i>	1-NN	BP	BP-1-NN
Chess	9.5	1.2	1.1
Flag	39.7	35.4	40.1
Glass	29.9	39.7	29.0
Horse	33.9	35.5	31.4
Hypthoid	2.8	2.2	2.1
Iono	13.1	8.6	11.4
Iris	4.7	2.7	4.0
Letter	4.0	21.2	3.5
Lymph	18.1	20.8	16.1
PageBlk	4.1	3.6	3.7
Promot	19.9	10.5	12.2
SatImge	9.7	12.1	9.5
Segment	3.1	4.8	2.3
Vowel	1.4	23.5	0.8
Water	27.5	24.1	26.2

capability for the neural network to capture feature weighting information.

In 9 datasets (*Chess*, *Glass*, *Horse*, *Hypothoid*, *Letter*, *Lymph*, *SatImage*, *Segment*, *Vowel*), the nearest neighbor algorithm with feature weighting (**BP-1-NN**) not only improves the baseline nearest neighbor algorithm (**1-NN**), but also outperforms the neural network itself (**BP**). In 7 datasets (*Glass*, *Horse*, *Letter*, *Lymph*, *SatImage*, *Segment*, *Vowel*) among these 9 datasets, the baseline nearest neighbor algorithm (**1-NN**) performs better than the neural network (**BP**). Yet **1-NN** can be further improved by using the feature weighting information (**BP-1-NN**) provided by the neural network. In the other 2 datasets (*Chess*, *Hypothoid*) of the 9 datasets, the

neural network (**BP**) performs better than the baseline nearest neighbor algorithm (**1-NN**). However, feature weighting (**BP-1-NN**) by a neural network is capable of not only improving the nearest neighbor algorithm (**1-NN**) but also surpassing the neural network (**BP**).

Note that for the datasets *Letter* and *Vowel*, the nearest neighbor algorithm is significantly better than the neural network (4.0% vs. 21.2% for *Letter*, and 1.4% vs. 23.5% for *Vowel*). Such superior performance on these two datasets by the nearest neighbor algorithm is quite interesting. The following analysis on characteristics of the two datasets is helpful in gaining insights on this phenomenon. The task for *Vowel* is to classifying 11 vowels based on speech data collected from 15 different persons, where each person speaks each vowel 6 times. Thus within each vowel (class), there could exist different clusters due to the distinct acoustic characteristic of each individual speaker. While a neural network is likely to try to form decision boundary for the whole class, it is much easier for a nearest neighbor classifier to form a flexible boundary for each cluster. For example, for a given test speech sample, the nearest neighbor classifier would likely find the most similar instance to be one of the 6 vowel utterances by the same speaker. For the task *Letter* there is a similar characteristic: there are 20 different fonts in each of 26 classes (letters), and thus they are very likely to form font-based intra-class clusters.

Also note that for the datasets *Letter* and *Vowel*, the performance of the nearest neighbor algorithm can be further improved (from 4.0% to 3.5% for *Letter*, and from 1.4% to 0.8% for *Vowel*) by extracting the weighting information from a neural network. Although the neural network itself is not most suitable on these domains, it can still capture important information on the relevance of features to help the nearest neighbor algorithm. We observe in the experiment that for the dataset *Vowel*, the magnitude of the highest weighted feature by the neural network is about 5 times that of the lowest one, while for *Letter* it is about 3 times. This implies that not all of the original features are equally relevant to the classification task, and in this case weighted features are preferred over constant ones.

The results in Table 2 also show that in 6 datasets (*Flag*, *Iono*, *Iris*, *PageBlks*, *Promot*, *Water*), **BP** outperforms both **1-NN** and **BP-1-NN**. In 5 of these 6 datasets (except *Flag*), **BP-1-NN** outperforms **1-NN**, showing that the weighting information provided by a neural network is still helpful to improve the nearest neighbor algorithm even though not capable of outperforming the neural network itself. In the dataset *Flag*, there are 28 attributes and 8 classes but only 194 instances. The sparseness of training data is likely an contributing factor for the neural network’s inaccuracy in estimating feature weighting.

To study how the three algorithms respond to irrelevant attributes, we artificially add a certain number of (continuous) irrelevant attributes to the dataset *Vowel*. The experimental results are shown in Table 3. We can see that the nearest neighbor algorithm without weighting is very sensitive to irrelevant attributes, while the same algorithm with weighting

is more robust against irrelevant attributes. We observe in the experiment that the magnitude of the weight of an irrelevant feature estimated by the neural network is typically about one tenth that of a relevant feature, and thus its impact is significantly reduced. The results in the last two columns also show that the slow degradation with irrelevant attributes is closely correlated to the similar degradation displayed by neural networks.

TABLE III  
ERROR RATE (%) VS. IRRELEVANT ATTRIBUTES

<i>Irrelv</i>	1-NN	BP	BP-1-NN
0	1.4	23.5	0.8
1	7.6	23.6	1.2
2	24.1	24.9	2.1
3	31.4	25.7	2.9
4	42.4	25.8	3.2
5	49.3	28.5	4.8
10	67.1	34.4	8.5
15	76.4	35.3	18.5
20	79.0	40.8	21.4

#### IV. SUMMARY

In this work, we propose a neural network based feature weighting approach to improve instance based learning algorithms. After training a neural network, the weight of a feature is extracted from the network based on link strengths among related nodes. The main motive of using this approach is to take advantage of neural network's rich capacity to represent hypotheses, as well as its ability to capture underlying regularities inside datasets by forming internal representations.

The performance of this approach is tested on 15 real-world datasets drawn from the UCI dataset repository. The results show that the proposed weighting scheme not only improves the nearest neighbor algorithm in 14 of the 15 tested datasets, but also outperforms the neural network classifier itself in 9 datasets.

This approach is particularly useful for domains that are suitable for instance based learning algorithms and when their features are not of equal relevance, as demonstrated in the example datasets *Letter* and *Vowel*. The results also show that the proposed approach is robust against irrelevant attributes.

In the future we plan to explore different ways to extract feature weighting information from neural networks. One direction will be to study the effect of weighting schemes that differ for different layers in a neural network. Another direction is to improve the current feature weighting method by identifying and reducing potential redundancy amongst links in a neural network to achieve more accurate weighting information.

#### REFERENCES

[1] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[2] C. Cardic, "Using decision trees to improve case-based learning," *Proceedings of 1993 International Conference on Machine Learning*, pp. 25–32, 1993.

[3] G. H. John, R. Kohavi and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," *Proceeding of Eleventh International Conference on Machine Learning*, pp. 121–129, 1994.

[4] S. L. Salzberg, "A nearest hyper-rectangle learning method," *Machine Learning*, vol. 6, pp. 251–276, 1991.

[5] D. W. Aha, "Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms," *International Journal of Man-Machine Studies*, vol. 36, pp. 267–287, 1992.

[6] C. X. Ling, J. J. Parry and H. Wang, "Setting attribute weights for nearest neighbor learning algorithms using C4.5," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 11(3), pp. 405–415, 1997.

[7] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, pp. 273–314, 1997.

[8] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximator," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[9] C. J. Merz and P. M. Murphy, UCI repository of machine learning databases, <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 1996.

[10] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.