Discrete quasi-gradient features weighting algorithm

Norbert Jankowski

Department of Informatics, Nicholas Copernicus University
ul. Grudziadzka 5, 87-100 Toruń, Poland, http://www.phys.uni.torun.pl/~norbert

Abstract. A new method of feature weighting, useful also for feature extraction has been described. It is quite efficient and gives quite accurate results. Weighting algorithm may be used with any kind of learning algorithm. The weighting algorithm with k-nearest neighbors model was used to estimate the best feature base for a given distance measure. Results obtained with this algorithm clearly show its superior performance in several benchmark tests.

1 Introduction

It is well known [1,2] that initial feature set of data sets/databases used for classification or approximation is not the *optimal information source* and the feature analysis may be very helpful in further processing of data. Several methods which use different strategy of feature extraction and weighting were already presented in books and articles [3,4,1,5]. Some of them measure amount of information belonging to a given attribute (or a subset[1]) using information theory or statistics. Other methods use learning models to observe accuracy changes what help to estimate weights in the next phase of feature weighting (or extraction) process [6,7,4]. Such type of feature weighting may be used with several learning models types. Algorithm presented in this article belongs to the second type of feature weighting/extracting algorithms, and will be presented in conjunction with k-nearest neighbors model [8], although algorithm itself may be used with any learning model.

2 Feature weighting algorithm

General idea of a Discrete quasi-gradient algorithm is based on *looking for optimal* vector of weight changes in given state of weighting procedure. The process is repeated as long as any improvement of accuracy may be done. In parallel some control parameters change to stimulate quasi-gradient directions of weights changes. Another goal of this algorithm was to create stable feature weighting algorithm. It means that new (weighted) set of features should never significantly decrease accuracy of the final model. Of course it is impossible to expect any progres if original feature set is optimal. Next,

[1] But if subset become bigger the complexity grow exponentially.

repeating the whole procedure of weighting a few distinguishable solutions may be found. This is not an error, it is nature of some databases. Such information may be important in propagation of this information (separate weighted sets of features) to other models and it may help to obtain different final models.

The main loop of the algorithm consists of cross-validation (CV) as a learning technic. This means that a whole data set is divided in n equal (if possible) parts (commonly called folds) $\mathcal{S}_i,\ i = 1, \ldots, n$. In each CV–iteration procedure $FindWeights$ is called. $FindWeights$ use two sets $\hat{\mathcal{S}}_i$ ($\hat{\mathcal{S}}_i = \bigcup_{k=1,\ldots,n,\ k\neq i} \mathcal{S}_k$) as a learning set, and $\mathcal{S}_i$ as a validation set. As a result procedure $FindWeights$ returns vector of feature weights $\mathbf{w}_i$. Lets define:

$$\mathbf{w}_i = FindWeights(\hat{\mathcal{S}}_i, \mathcal{S}_i) \tag{1}$$

Each weight of $\mathbf{w}_i$ is in $[0, 1]$ interval. Details of procedure $FindWeights$ will be presented later. Vectors $\mathbf{w}_i$ consist of weights for each feature $k = 1, \ldots, d$, where d is the dimension of data set $\mathcal{S}$. Now the sum of weights vectors is calculated and the final weights vector is obtained through normalization:

$$\tilde{\mathbf{w}} = \mathbf{w} / \max_{k=1,\ldots,d} w_k \qquad \mathbf{w} = \textstyle\sum_{i=1}^{n} \tilde{\mathbf{w}}_i \tag{2}$$

where $\mathbf{w} = [w_1, \ldots, w_d]$. The above part of CV learning is really simple: if all CV iterations are completed, the final model is estimated as a *consequence* of intermediate models. Now the $FindWeights$ as the heart of whole algorithm will be described.

Procedure $FindWeights$**.** First, the initial values are assigned to weights: $w_k = 1, \quad k = 1, \ldots, d$. In this case the algorithm starts from full feature set. It may be useful sometimes to start from $w_k = 0$, what mean that algorithm starts from an empty feature set and tries to add feature by feature — this strategy may be more efficient, computationally, if working with highly dimensional data sets. The main loop (one of three) is repeated as long as any progress in the maximization of accuracy on validation set may be done. The contents of that loop is called *main phase*. The goal of the *main phase* is to observe changes, determining which features may help to improve the accuracy by changing current weights by Δ, and next to apply positive/progresive changes and simultaneously coordinating the change of Δ. In the inner loop for each feature two quantities v_i^+ and v_i^- are computed:

$$v_i^+ = validate(\mathbf{w}^+, i) \qquad v_i^- = validate(\mathbf{w}^-, i) \tag{3}$$

where $\mathbf{w}^\pm = [w_1, \ldots, w_{i-1}, w_i \pm \Delta, w_{i+1}, \ldots, w_d]$. $validate(\mathbf{w})$ is a function which compute the accuracy on validation set $\mathcal{S}$ (see eq. 1) for the learning model $\mathcal{M}$ which is trained on a data set $\hat{\mathcal{S}}$. To obtain results presented in section 3 the k-nearest neighbor model was used as $\mathcal{M}$.

Vectors $\mathbf{v}^+$ and $\mathbf{v}^-$ contain information about weight changes by step Δ that may help to improve results. If any $v_i^\pm$ indicate that given change may

help the new set of weights is defined according to:

$$w'_i = w_i - \theta\Delta \quad \text{if} \quad v_i^- > v \tag{4}$$

$$w'_i = w_i + \theta\Delta \quad \text{if} \quad v_i^+ > v \wedge v_i^- <= v \tag{5}$$

where θ defines the speed (I like $\theta = 1$ – really fast!), and $v = validate(\mathbf{w})$.

The above equation says that all features which may help if they are changed, are indeed changed. If for new weights $v_{new} = validate(\mathbf{w}')$ is smaller than $v = validate(\mathbf{w})$ the procedure goes back to weights $\mathbf{w}$, else $\mathbf{w}'$ become the new weight vector.

As long as the current parameter Δ leads to a better accuracy, weight changes are made and the inner loop is continued. Usually the starting Δ is equal to 1. The step Δ is decreased if no progress in maximization of accuracy on validation set is done: $\Delta = \Delta/2$. After that if Δ is still bigger (or equal to) than Δ_{min}, the second loop is repeated again. If Δ is smaller than Δ_{min} (for example 0.01) loop is broken and weights are normalized:

$$\tilde{\mathbf{w}} = \mathbf{w} / \max_{k=1,\ldots,d} w_k \tag{6}$$

The *main phase* loop is repeated (with initial $\Delta = 1$) as long as the weighting algorithm gives with better accuracy on a validation test. Since the main phase loop is repeated the algorithm is able to jump out from local minima keeping the best solution found so far. Without that loop the performance of the algorithm was average or even worst.

Outline of this algorithm may be found at my WWW page.

3 Results

The performance on several databases was checked to test the stability of the algorithm and to test whether feature weighting helps or may destroy the generalization ability. All test were performed on databases from Machine Learning Database Repository at the UCI [9] using 10-fold CV test, except that data sets which have original testing sets. All results were averaged from 10 runs. Standardization of most database was performed before the learning phase. Each table with results have four parts. First presents results which were obtained with feature weighting algorithm for kNN, second without feature weighting, and third show default accuracy – percentage of most majority class. Fourth shows results obtained by other models. This part contains only the best models for a given database[2]. Note that k-weight [10] and GIBL [7] are algorithms which estimate feature influence. It is common that some models have really good accuracy for one benchmark giving much worst results for other data sets. In tables with results the notation is used:

[2] If you want to browse more results see WWW pages of my LAB: http://www.phys.uni.torun.pl/kmk/projects/datasets.html.

`kNN [CVx] [M]. k` defines the number of neighbors. CVx if exists means that weighting algorithm was used, and x defines the number of folds in CV. If M occurs non-Euclidian metric was used, such as the Manhattan, Minkovsky or Canberra[3]. For some data set significant improvement of accuracy on test sets is observed, and for others data sets the accuracy is not (essentially) changed while for others the classification model decreases its accuracy.

Thyroid disease is one of the best known benchmark data sets, as well one really nontrivial for classification. The training set has 3772 cases, and testing set has 3428 cases. There are 3 classes, and the most frequent has 92.71% of all cases. Number of attributes is 21 (15 binary, 6 continuous). The accuracy on test set of plain 1NN (kNN with 1 neighbor) is really poor, just 93.14% (see tab. 1). For 3NN with Manhattan distance the accuracy is 94.4%, but using weighted features 98.36%. The best results were obtained for 3NN with the Canberra metric – 98.56%. This very good results were obtained only because irrelevant features were removed – see figure 1.

Flags classification problem have 193 vectors, 28 attributes, and each vector may be assigned to one of 8 classes. Feature weighting for this data set was very efficient (see tab. 2). For 1NN CV3 with Manhattan metric accuracy in test was 62.04% and without weighting 50.91%. For 3NN with weighting 61.51% and 48.07% without. The best non-weighted kNN have accuracy 52.56% and this is nearly 10% less than for the best plain kNN.

Glass data contains 214 vectors, each vector has 9 attributes and may be classified to one of the 6 classes. For this benchmark weighting looks very important, and the improvement may be really large (see tab. 3). The best results were obtained for 1NN CV3 with Manhattan metric giving 80.81% accuracy, and the plain version finished with just 73.50%. For 3 neighbors (3NN CV3 Manh) weighted version has 78.04%, and non-weighted 72.06%.

The number of Δ changes change from 25 to 40 for glass data set and from 25 to 50 for flags data set (it is around 1 minute on Pentium 4).

For the following data sets results of feature weighting were not significant (around 0.3-2%): australian credit, appendicitis, cleveland heart, wine.

4 Conclusions

Discrete quasi-gradient weighting algorithm presented in this paper is really efficient and results have good accuracy on several tests. Sometimes the improvement was around 10% of accuracy. This algorithm may be used as feature extraction tool. The starting feature set may be full or empty, and because of that it may be useful for huge dimensional data sets. Also alternative weighting sets may be obtained, what may be very important in further analysis. Using this feature weighting algorithm the kNN model was placed at the top of ranking of the best methods for presented data sets. Sometimes the

[3] Canberra metric: $canb(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|/|x_i + y_i|$.

Model	test	train
3NN CV3 Camberra	98.56	99.37
3NN CV2 Manh	98.36	99.28
1NN CV2 Manh	98.26	100.00
3NN CV2	98.28	99.24
1NN CV2	98.26	100.00
2NN CV2	98.36	99.28
3NN Camberra	96.30	98.49
3NN Manh	94.40	96.58
1NN Manh	93.76	100.00
1NN	93.14	100.00
default	92.71	-
C-MLP2LN+ASA [11]	99.36	99.90
CART [12]	99.36	99.80
PVM [12]	99.36	99.80
SSV [13]	99.36	99.76
IncNet [14]	99.24	99.68
MLP a,b opt. [11]	99.36	99.90
MLP2LN [11]	99.00	99.70
Cascade correlation [15]	98.50	100.00
BP+genetic [15]	98.40	99.40
Quickprop [15]	98.30	99.60
kNN+weights [10]	98.22	98.89
RProp [15]	98.00	99.60
kNN+weights [10]	97.96	98.89...

Table 1. Accuracy for thyroid data set.

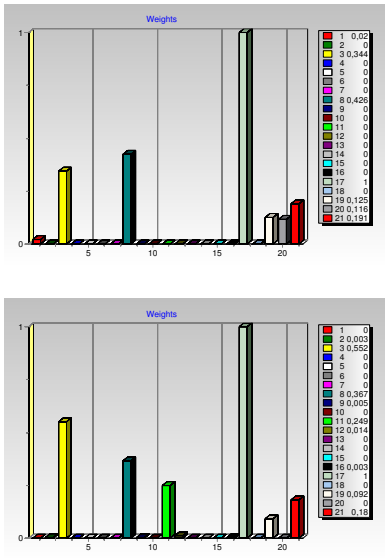

Fig. 1. Feature weights for thyroid data set.

Model	test	train
1NN CV3 Manh	62.04	100.0
3NN CV3 Manh	61.51	76.07
7NN CV3 Manh	58.30	68.28
27NN CV3 Manh	56.19	59.78
27NN Manh	52.56	56.53
7NN Manh	52.03	63.57
1NN Manh	50.91	100.00
17NN Manh	50.25	57.97
3NN Manh	48.07	72.53
27NN	46.76	51.75
1NN	48,52	100,00
default	31.09	-
CART [16]	62.40	-
IB1 [16]	56.60	-
C 4.5 [16]	56.20	-
GIBL [7]	55.11	-
IBL [7]	48.29	- ...

Table 2. Accuracy for flags data set.

Model	test	train
1NN CV3 Manh	80.81	100.00
3NN CV3 Manh	78.04	87.64
3NN CV3 Mink–0.7	77.97	88.78
3NN CV2 Manh	77.49	87.53
3NN CV2	75.00	85.11
1NN Manh	73.50	100.00
3NN Manh	72.06	84.98
3NN	70.85	83.63
1NN	70.45	100.00
default	35.51	-
GIBL [7]	78.55	-
Bayes [16]	71.80	-
CART	71.40	-
IB1 [16]	71.10	-
ID3 [16]	69.10	-
IBL [7]	70.52	-

Table 3. Accuracy for glass data set.

accuracy on testing and training parts were very similar, which means that solution found are close to optimum for a given model. It is important that algorithm may be used not only for kNN, but for several machine learning models, including artificial neural networks.

References

1. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
2. R. O. Duda, P. E. Hart, and D. G. Stork. *Patter Classification and Scene Analysis*. Wiley, 2nd ed. edition, 1997.
3. M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.
4. M. Fernández and C. Hernández. How to select the inputs for a multilayer feedforward by using the training set. In *5th International Work Conference on Artificial an Natural Neural Networks*, pages 477–486, Alicante, Spain, 1999.
5. H. Almuallim and T. G. Dietterich. Efficient algorithms for identifying relevant features. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, pages 38–45, Vancouver, 1992. Morgan Kaufmann.
6. D. R. Wilson and T. R. Martinez. Instance-based learning with genetically derived attribute weights. In *International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, pages 11–14, 1996.
7. D. R. Wilson. *Advances in Instance-Based Learning Algorithms*. PhD thesis, Department of Computer Science Brigham Young University, 1997.
8. T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13(1):21–27, jan 1967.
9. C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
10. W. Duch and K. Grudziński. Search and global minimization in similarity-based methods. In *International Joint Conference on Neural Networks*, page 742, Washington, 1999.
11. W. Duch, R. Adamczak, and K. Grabczewski. Extraction of logical rules from backpropagation networks. *Neural Processing Letters*, 7:1–9, 1998.
12. S.M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets and machine learning classification methods. In J.W. Shavlik and T.G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kauffman, 1990.
13. K. Grabczewski and Włodzisław Duch. The separability of split value criterion. In L. Rutkowski and R. Tadeusiewicz, editors, *Neural Networks and Soft Computing*, pages 202–208, Zakopane, Poland, June 2000.
14. N. Jankowski. *Ontogenic neural networks and their applications to classification of medical data*. PhD thesis, Department of Computer Methods, Nicholas Copernicus University, Toruń, Poland, 1999.
15. W. Schiffman, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. In *Proceedings of ESANN'93*, pages 97–104, Brussels, 1993.
16. Frederick Zarndt. A comprehensive case study: An examination of machine learning and connectionist algorithms. Master's thesis, Department of Computer Science Brigham Young University, 1995.