Learning Decision Trees for Loss Minimization in Multi-Class Problems

Technical Report 99-30-03.

Department of Computer Science, Oregon State University.

Dragos D. Margineantu
Department of Computer Science, Oregon State University
Corvallis, OR 97331-3202, U.S.A.

Thomas G. Dietterich
Institut D'Investigació en Intel·ligéncia Artificial, Consejo
Superior de Investigaciones Científicas, Campus de la UAB
08193-Bellaterra, Barcelona, Spain

Abstract

Many machine learning applications require classifiers that minimize an asymmetric loss function rather than the raw misclassification rate. We study methods for modifying C4.5 to incorporate arbitrary loss matrices. One way to incorporate loss information into C4.5 is to manipulate the weights assigned to the examples from different classes. For 2-class problems, this works for any loss matrix, but for $k > 2$ classes, it is not sufficient. Nonetheless, we ask what is the set of class weights that best approximates an arbitrary $k \times k$ loss matrix, and we test and compare several methods: a wrapper method and some simple heuristics. The best method is a wrapper method that directly optimizes the loss using a hold-out data set. We define complexity measure for loss matrices and show that this measure can predict when more efficient methods will suffice and when the wrapper method must be applied.

Keywords: *supervised learning, decision trees, loss, misclassification costs, function minimization.*

Email address of authors: {margindr,tgd}@cs.orst.edu

1 Introduction

For most of the history of machine learning research, a central goal has been to develop supervised learning algorithms that minimize the misclassification error rate on unseen examples. Great progress has been made toward this goal. Indeed, recent algorithms are so good that current research papers must employ very sophisticated statistical tests to measure the improvements in accuracy produced by various algorithms.

While academic research has continued to improve the misclassification error rate, applications in business, medicine, and science have shown that real problems require more subtle measures of performance (see, for example, (Fawcett & Provost, 1997; Kubat, Holte, & Matwin, 1997; Provost, Fawcett, & Kohavi, 1998)). In particular, one important problem is that in these applications, different kinds of errors have different costs. For example, in medical applications, the cost of a false positive diagnosis is usually the cost of putting the patient through an unnecessary medical treatment. But the consequences of a false negative diagnosis can be fatal. In text-to-speech systems, misclassification errors between different consonants are usually much more important than misclassifications between vowels. The users of machine learning programs need to be able to make tradeoffs between different kinds of errors.

The relative importance of different kinds of errors can be represented by a loss matrix. If there are k classes, the loss matrix is a $k \times k$ matrix, L, of non-negative values. The rows correspond to the class predicted by the learning algorithm, and the columns correspond to the correct class. The value in $L(i, j)$ gives the "loss" of predicting class i when the true class is j. The diagonal elements, $L(i, i)$, are always zero. For example, Table 1 shows a 4-class loss matrix. The cell in column 4 of row 3 gives the loss (5.5) of predicting class 3 when the true class is 4. The standard 0/1 loss function (misclassification rate) can be represented by a matrix all of whose off-diagonal elements are equal to 1.

Several papers in the literature have presented approaches for minimizing the misclassification costs in supervised learning algorithms. (Pazzani, Merz, Murphy, Ali, Hume, & Brunk, 1994) present algorithms for learning and avoiding overfitting in decision lists for minimizing the loss. (Bradford, Kunz, Kohavi, Brunk, & Brodley, 1998) experiment with different methods for pruning decision trees to minimize loss. (Kubat & Matwin, 1997; Kubat, Holte, & Matwin, 1998) also address the issue of minimizing misclassification costs.

For neural network algorithms (Haykin, 1994), we can minimize the loss on the training set by incorporating the loss matrix directly into the back-propagated error signal. However, it isn't clear how to combine such a general loss function with existing regularization methods (e.g., weight decay and other complexity penalties).

In this paper, we study how to incorporate general loss matrices into decision-tree algorithms (Breiman, Friedman, Olshen, & Stone, 1984; Quinlan, 1993). These algorithms pose particular challenges, because they do not directly minimize a loss function. Instead, a splitting heuristic—such as information gain or the GINI index—is employed. These heuristics have the dual role of finding splits that minimize loss and also finding splits that "make progress" toward good subsequent splits. The rules need to detect "progress" in order to overcome the greedy nearsightedness of the top-down splitting process (Dietterich, Kearns, & Mansour, 1996). This is why the training set loss alone does not give a good splitting heuristic.

Our research has focused on the C4.5 decision tree learner (Quinlan, 1993). We are particularly concerned with multi-class problems, where the number of classes, k, is greater than two. The next section describes several candidate methods for incorporating general loss matrices. Section 3 introduces a measure of the complexity of a loss matrix that we call the *cost irregularity*. This measure is useful for predicting the performance of the different candidate methods. Finally, Section 4 describes the results of our experiments.

Table 1: Example of loss matrix for a four-class problem.

Predicted	Correct Class			
Class	1	2	3	4
1	0.0	3.2	1.0	2.7
2	1.0	0.0	3.0	0.5
3	4.5	2.2	0.0	5.5
4	1.0	0.1	7.1	0.0

2 Growing Decision Trees for Loss Minimization

There are two distinct steps in learning a decision tree: growing the tree and pruning it to avoid overfitting the data. Quinlan's C4.5 decision tree learner (Quinlan, 1993) uses a set of weights associated with each of the examples. The weights were originally introduced as part of the mechanism for handling missing values in the data. However, more recently, Quinlan and others have used them to implement boosting algorithms (Quinlan, 1996).

The weight of an example can be viewed as a measure of the importance of classifying the example correctly (or equivalently, as the cost of misclassifying it). To incorporate classification costs, we can consider manipulating these weights. For example, consider the general two-class loss matrix shown in Table 2. We can place a weight of $cost_1$ on every training example in class 1 and a weight of $cost_2$ every example in class 2.

Table 2: General loss matrix for two-class learning problems.

Predicted	Correct Class	
Class	1	2
1	0	$cost_2$
2	$cost_1$	0

The values of the weights of the examples are used during both the growing and the pruning phases of C4.5. In preliminary experiments, we have found that this method works very well for incorporating asymmetric loss functions for 2-class problems.

However, in a general $k \times k$ loss matrix, there are $k(k-1)$ off-diagonal weights, whereas there are only k possible class weights that can be manipulated in C4.5. (In fact, since it is only the *relative* values of the weights that matter, only $k-1$ of the input weights to C4.5 are independent, and only $k(k-1)-1$ of the weights in an arbitrary loss matrix are independent.) Hence, this simple technique of weighting the inputs does not directly extend to problems with more than 2 classes.

Nonetheless, adjusting the input weights is a very attractive technique for incorporating loss information into C4.5, so it is worth considering whether there might be a good way of choosing a set of input weights $\mathbf{w} = (w_1, \ldots, w_k)$ for C4.5 to "best implement" a general loss matrix L. Of course this will not be the ideal method, but it may work well in practice.

We have developed five different methods for choosing the input weight vector $\mathbf{w}$. These methods can be divided into three different class: wrapper methods, class-distribution methods, and loss-matrix methods. We describe each of these in turn.

2.1 Wrapper Methods

The most expensive approach is a "wrapper" method, which treats the decision-tree learning algorithm as a subroutine as follows. Suppose we randomly partition our training data into a sub-training set and a validation set. When the learning algorithm is applied to the subtraining set with input weight vector $\mathbf{w}$, the loss of the resulting decision tree can be measured using the validation set and the loss matrix L. Hence, we can use a general-purpose optimization algorithm to generate different $\mathbf{w}$'s and try to find the $\mathbf{w}$ that minimizes the loss on the validation set.

We chose Powell's method (Press, Teukolsky, Vetterling, & Flannery, 1992), which is a gradient-free, quadratically-convergent algorithm. The method works by performing a sequence of one-dimensional optimizations. The search directions are chosen to try to make them orthogonal, so that maximum progress is made toward the minimum.

We used Powell's method to develop two different wrapper methods: *Powell10* and *Powell20*. *Powell10* uses a randomly-chosen 10% of the training data for its validation set, whilc *Powell20* uses 20%.

The overall procedure is the following: we run Powell's algorithm using the sub-training set and the validation set to determine the values of the weights $\mathbf{w}$. Then, we build the final decision tree using $\mathbf{w}$ on the whole training set.

2.2 Class Distribution-based Methods

The second group of methods is based on making some measurements of the frequencies of the different classes (or different errors) on the training set and then computing a good input vector $\mathbf{w}$ from this information.

Most prediction errors committed by learning algorithms tend to result from predicting items that belong to less frequent classes as being in more frequent classes. This observation suggests the following two methods.

We call the first method, *ClassFreqCount*. In this method, we compute the class frequencies in the training data, such that n_i is the number of training examples belonging to class i. Then, we set the input weights to C4.5 so that $n_i \times w_i$ is constant for all classes $1 \leq i \leq k$. This gives higher weight to classes that are less frequent.

The second method, called *EvalCount*, is based on first growing a decision tree to minimize the traditional 0/1 loss (raw misclassification rate). This is accomplished by subdividing the training data into a sub-training set and a validation set, growing the tree on the sub-training set, and then measuring its losses (according to loss matrix L) on the validation set. We then set w_i to be the sum of the losses of the examples from class i that were misclassified, plus 1 (to avoid having weights with a value of zero).

As with *Powell10* and *Powell20*, we implemented two versions of *EvalCount*: *EvalCount10* and *EvalCount20*, that use 10% and 20% of the data (respectively) for the validation set.

2.3 Loss-based Methods

The third group of methods for computing $\mathbf{w}$ calculates them directly from the loss matrix without either invoking C4.5 or measuring class frequencies in the training data. We call such methods *loss-based methods.* The big advantage of these methods is that they are extremely efficient! We propose two loss-based methods for calculating the weights.

The first method is called *MaxClassCost*. Each weight is computed as the maximum of the corresponding column:

$$w_j = \max_{1 \leq i \leq k} L(i,j)$$

The intuition is that the maximum value within a column is the worst-case cost of misclassifying an example whose true class corresponds to that column.

The second method is called *AvgClassCost*. Each weight is computed as the mean of the off-diagonal elements in the corresponding column:

$$w_j = \frac{\sum_{i=1, i \neq j}^{k} L_k(i,j)}{(k-1)}.$$

The intuition here is that the average of the non-zero values within a column approximates that average cost of misclassifying an example whose true class corresponds to that column.

3 *Cost-irregularity*: A measure for the structure of the loss matrix

In designing and analyzing our experiments, we wanted some measure of the "difficulty" or "complexity" of a loss matrix. The following measure is based on the idea that if it always costs more to misclassify examples of class j than class i, then the "costs" of the classes are monotonically related. Hence, our simple approach of using C4.5's input weights is more likely to work.

To be more precise, consider the following binary relation , (j_1, j_2) over pairs of classes j_1 and j_2:

Definition 1 *Two classes j_1 and j_2 are cost-transitive (we write , (j_1, j_2)) iff either*

$$\forall i_1, i_2 \textit{ if } L(i_1, j_1) \geq L(i_1, j_2) \textit{ then } L(i_2, j_1) \geq L(i_2, j_2)$$

or

$$\forall i_1, i_2 \textit{ if } L(i_1, j_1) \leq L(i_1, j_2) \textit{ then } L(i_2, j_1) \leq L(i_2, j_2).$$

Note that , is a symmetric relation: if , (i, j) is true, then , (j, i) is also true.

We can use cost-transitivity to define a measure of complexity for loss matrices.

Definition 2 (Cost Irregularity) *For $k \geq 4$, we define the cost-irregularity of a loss matrix as the number of pairs of classes that are not cost-transitive.*

Our intuition is that if all pairs of classes are cost-transitive, then a simple loss-based method, such as *MaxClassCost*, will work well for finding the weight vector $\mathbf{w}$. But if many pairs of classes are not cost-transitive, then a fixed weight vector will not represent the loss matrix very well.

4 Experimental Tests

4.1 Experiment Design

Section 2 defined seven different techniques for choosing a weight vector $\mathbf{w}$ for input to C4.5. To evaluate and compare these methods, we chose eight multi-class data sets from the UC Irvine Repository (Merz & Murphy, 1996)

Table 3: Data sets studied in this paper

Name	Number of Classes	Dataset Size
Splice	3	3190
Waveform	3	4500
Iris	3	150
Lymphography	4	148
Vehicle	4	846
Annealing	5	898
Hypo	5	3772
Glass	6	214

and designed a series of loss matrices (with different levels of cost irregularity). We then evaluated each method on each data set with each loss matrix via 10-fold cross-validation and analyzed the results.

Table 3 lists the data sets and the number of classes.

For the three-class problems ($k = 3$) we employed the loss-matrices shown in Table 4. For the problems with $k > 3$, we started with a loss-matrix whose entries are powers of two, generated as follows:

$$L(i,j) = \begin{cases} 2^j, & \text{when } j \neq i \\ 0, & \text{otherwise.} \end{cases}$$

This matrix has cost-irregularity zero.

Starting with this matrix, we generated four additional matrices with different levels of cost-irregularity by permuting elements among the rows. Table 5 summarizes the cost-irregularities of the generated loss matrices. Each matrix has been given a name (L2, L3, etc.).

We also tested each method on the standard 0/1 loss matrix.

4.2 Results

The performance of *Powell20*, *MaxCost*, *ClassFreq* and *EvalCount20* are compared for each dataset in the plots in Figures 1 and 2. The performance of the *Powell10*, *EvalCount10*, and *AvgClassCost* methods is not shown because it

Table 4: Loss matrices used in our experiments for the three-class problems.

$$\begin{bmatrix} 0.0 & 1.0 & 1.0 \\ 1.0 & 0.0 & 1.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 2.0 & 3.0 \\ 1.0 & 0.0 & 3.0 \\ 1.0 & 2.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 1.0 & 1.0 \\ 2.0 & 0.0 & 2.0 \\ 3.0 & 3.0 & 0.0 \end{bmatrix}$$

$$\begin{bmatrix} 0.0 & 2.0 & 4.0 \\ 1.0 & 0.0 & 4.0 \\ 1.0 & 2.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 1.0 & 1.0 \\ 2.0 & 0.0 & 2.0 \\ 4.0 & 4.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 2.0 & 3.0 \\ 4.0 & 0.0 & 1.0 \\ 1.0 & 2.0 & 0.0 \end{bmatrix}$$

$$\begin{bmatrix} 0.0 & 2.0 & 4.0 \\ 8.0 & 0.0 & 1.0 \\ 1.0 & 2.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 4.0 & 2.0 \\ 1.0 & 0.0 & 2.0 \\ 2.0 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 2.0 & 4.0 \\ 1.0 & 0.0 & 4.0 \\ 8.0 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 1.0 & 2.0 \\ 1.0 & 0.0 & 4.0 \\ 8.0 & 4.0 & 0.0 \end{bmatrix}$$

Table 5: Cost-irregularity values of loss matrices used in the experiments in this paper (based on the number of classes)

Number of			Loss	Matrix	
Classes	L2	L3	L4	L5	L6
4	0	1	2	2	3
5	0	1	2	3	4
6	0	1	3	4	5

was generally worse than *Powell20* and *EvalCount20*, and we wanted to avoid cluttering the graphs.

In the Splice, Hypo, Annealing, and Vehicle data sets, *Powell20* is clearly superior to the other methods for all loss matrices tried. In all of the other domains, this method is typically the best or second-best method for all loss matrices. In some cases, *MaxCost* or *ClassFreq* will out-perform *Powell20*, but the differences are usually small. Hence, the first conclusion that we can draw is that *Powell20* performs very well.

Unfortunately, *Powell20* is also very expensive to execute, so this raises the question of whether some of the cheaper methods will work well in certain situations. Unfortunately, there are examples of domains and loss matrices where each of the other methods performs very badly. For example, *ClassFreq* performs very badly on Splice with L7 and in Annealing with L6. *MaxCost* performs badly in Lymphography with L6 and Glass with L6. *Eval20* performs badly on Iris with L7 and Hypo with L4 and L6. This suggests that a reasonable strategy might be to use hold-out methods to evaluate each of these techniques and choose the one that gives the best results. If the best technique can be identified reliably, then that will give good results in all domains except Splice and for all loss matrices. And it might still be faster than executing *Powell20*.

In addition to measuring the performance of the different methods for choosing the weight vector, these experiments also show that the class-irregularity measure is a reasonable, but imperfect, predictor of the degree of difficulty of a loss matrix. In most domains, the performance of *MaxCost* degrades as the class-irregularity increases, although its behavior is not monotonic. When class-irregularity is zero, all of the methods give similar results (except that the *ClassFreq* method is sometimes worse than the others). This suggests another reasonable strategy: If class-irregularity is zero, use the *MaxCost* method.

Table 6 summarizes the results of applying the different techniques to standard 0/1 loss matrices. Each row lists the methods that gave the best results. In two domains, Hypo and Glass, uniform weighting gave worse results than some of the other methods. But in two other domains, Lymphography and Annealing, uniform weighting was better than all of the others. This shows, not surprisingly, that Uniform weighting is generally the best approach. However, it is interesting that Powell's method could sometimes manipulate the weight vector to obtain better 0/1 loss than uniform weights provide. This probably reflects the fact that decision tree algorithms are heuristic and do not guarantee to find the optimal decision tree. By manipulating the input weights, Powell's method is able to "fool" C4.5 into finding a better decision tree.

5 Conclusions

The paper introduced five methods for determining the class weights for constructing decision trees for loss minimization on multiple-class problems. From the experiments, we conclude that we can improve the performance of the learned decision trees for multi-class problems by manipulating C4.5's input weights, no matter how complex the associated loss matrix is. Furthermore, a wrapper method (*Powell20*) that directly searches for good input weights gave excellent results across a wide range of domains and loss matrices.

Table 6: Algorithms that yielded the best performance for the standard 0/1 loss matrix

Domain	Methods
Splice	Powell20, Uniform
Waveform	All methods
Iris	ClassFreq, Uniform
Lymphography	Uniform
Vehicle	Powell20, Uniform
Annealing	Uniform
Hypo	Powell10, Powell20
Glass	Powell20, Eval10

The paper also defined a complexity measure for loss matrices of size higher than 3, the *cost-irregularity*. The experiments have shown that this measure can help in deciding whether the costly wrapper method is likely to give better weights than simple loss-matrix methods.

References

Bradford, J. P., Kunz, C., Kohavi, R., Brunk, C., & Brodley, C. E. (1998). Pruning decision trees with misclassification costs. In *ECML-98. Lecture Notes in Artificial Intelligence*. Springer Verlag.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.

Dietterich, T. G., Kearns, M., & Mansour, Y. (1996). Applying the weak learning framework to understand and improve C4.5. In Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 96–104 San Francisco, CA. Morgan Kaufmann.

Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. *Data Mining and Knowledge Discovery*, *1*(3).

Haykin, S. (1994). *Neural Networks. A Comprehensive Foundation*. Macmillan Publishing Company.

Kubat, M., Holte, R., & Matwin, S. (1997). Learning when negative examples abound. In *ECML-97, Lecture Notes in Artificial Intelligence*, Vol. 1224, pp. 146–153. Springer Verlag.

Kubat, M., Holte, R. C., & Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, *30*(2/3).

Kubat, M., & Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided sampling. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186. Morgan Kaufmann.

Merz, C. J., & Murphy, P. M. (1996). UCI repository of machine learning databases. Tech. rep., U.C. Irvine, Irvine, CA. [`http://www.ics.uci.edu/~mlearn/MLRepository.html`].

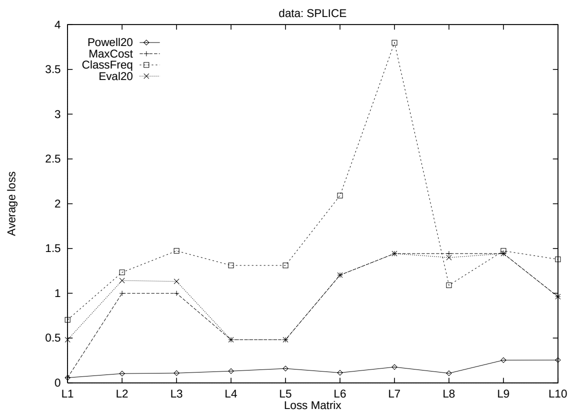
Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T., & Brunk, C. (1994). Reducing misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 217–225. Morgan Kaufmann.

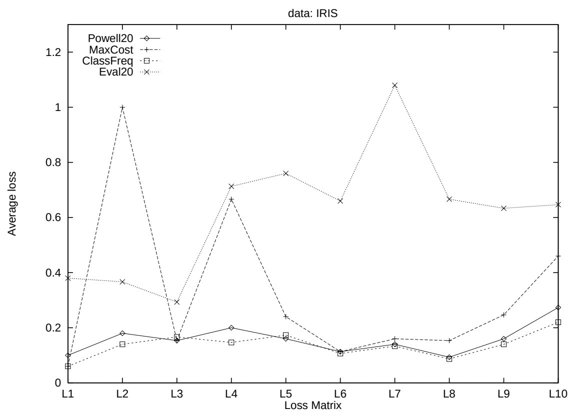
Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C - The Art of Scientific Computing, 2nd ed.* Cambridge University Press.

Provost, F., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for comparing classifiers. In *Proceedings of the Fifteenth International Conference on Machine Learning.* Morgan Kaufmann, San Francisco, CA.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Francisco, CA.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730 Cambridge, MA. AAAI Press/MIT Press.





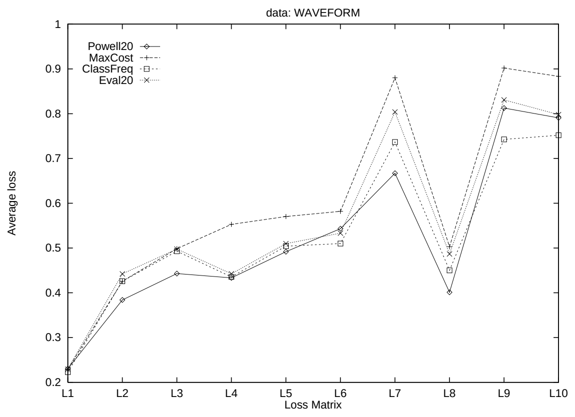


Figure 1: Performance of the algorithms on 3-class problems.

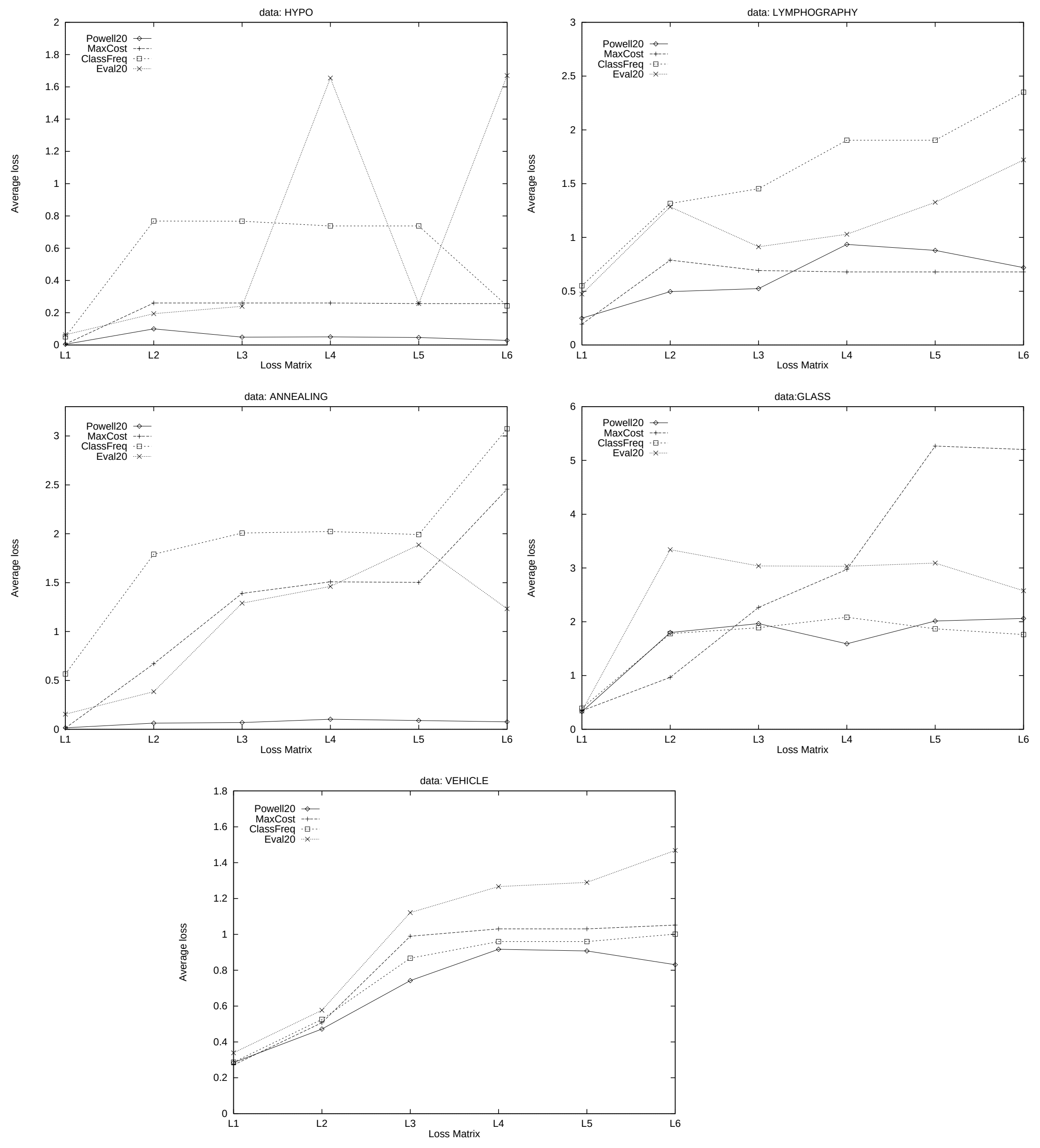


Figure 2: Performance of the algorithms on the datasets with a number of classes larger than 3. The loss matrices are sorted on the x-axis based on their class-irregularity value.