Class-Oriented Reduction of Decision Tree Complexity

José-Luis Polo, Fernando Berzal, and Juan-Carlos Cubero

Department of Computer Sciences and Artificial Intelligence
University of Granada. 10871 Granada, Spain
{jlpolo,fberzal,jc.cubero}@decsai.ugr.es

Abstract. In some classification problems, apart from a good model, we might be interested in obtaining succinct explanations for particular classes. Our goal is to provide simpler classification models for these classes without a significant accuracy loss. In this paper, we propose some modifications to the splitting criteria and the pruning heuristics used by standard top-down decision tree induction algorithms. This modifications allow us to take each particular class importance into account and lead us to simpler models for the most important classes while, at the same time, the overall classifier accuracy is preserved.[1]

1 Introduction

Traditional classification techniques treat all problem classes equally. This means that classification models are built without focusing on particular problem classes. In practice, however, not all problem classes are equally important. Obtaining a simpler model for the important classes, even when it might be slightly less accurate, might be of interest. In this paper, we introduce *class-complexity-sensitive* classification techniques, CCS classifiers for short, to address this issue.

The aim of CCS algorithms is to build classification models that, being as simple as possible for the most important classes, preserve the global classification accuracy their traditional counterparts provide. It should be noted that, given their goal, CCS algorithms will not treat all class values equally during the classifier construction process.

CCS classification models can be useful in the following scenarios:

- **Extreme class values**, when experts are specially interested in succinctly describing classes whose importance is premium within the decision making process (a common situation when classes are ranked).
- **Ontologies relating different problem classes**, when classes can be organized somehow (i.e. focusing on a subset of related classes that can be specially meaningful for the expert).
- **Typical binary classification problems**, when obtaining a proper description of one of the classes is more important for the user than having a typical classification model where both problem classes are equally treated.

[1] Work partially supported by grant TIN2006-07262.

It should also be noted that the class values we are interested in might change depending on our personal goals, even for the same problem. We incorporate the relative importance of each problem class into the classification model building process. We consider *relative weights* representing each class relative importance. When all the problem classes are equally important, we assign a relative weight of 1 to each one of them. When a class is 50%, 100%, 150%, 200%, 400%, and 800% more important than other, its relative weight should be 1.5, 2, 2.5, 3, 5, and 9 times the weight of the less important class. Afterwards, the resulting values can always be normalized to fall within the [0,1] interval while preserving the relative values they define in their ratio scale. The relative importance for the class values must be set empirically, according to each problem.

The rest of our paper is organized as follows. Section 2 describes existing work where special class features are taken into account. Section 3 briefly describes standard decision trees and how to modify them in order to build CCS classification models. Section 4 proposes how to evaluate the resultant models from a CCS perspective. Experimental results are provided in Section 5. Finally, Section 6 includes some conclusions.

2 Related Work

All classes should not be equally treated in all classification problems. Class differences make traditional classification models ineffective when class features are ignored. Some techniques have been proposed to deal with particular class features:

- **Imbalanced learning methods** [1] do not ignore the less frequent classes when there are very frequent classes that would lead traditional methods astray. Unlike CCS models, which are focused on classifier complexity, imbalanced learning methods are focused on the resulting classifier accuracy.
- **Cost-sensitive classification** [4] techniques take into account that the cost of misclassification is not the same for all the problem classes. Unlike CCS models, these learners are mainly focused on classifier accuracy (to avoid high-cost classification mistakes).
- **Subgroup discovery** [5] tries to find interesting subgroups within the training data set with respect to a target class value from a statistical point of view. They only provide a model for the target class, whereas CCS learners build complete classification models.

3 CCS Decision Trees

Decision trees [8] are one of the most widely used classification models. The availability of efficient and scalable decision tree learners [9,6] makes them useful for data mining tasks. Moreover, their interpretability makes them specially suitable for CCS classification.

We have modified the standard top-down induction of decision trees algorithm (TDIDT) to address CCS classification by modifying both its heuristic splitting criterion and the tree pruning strategy.

3.1 Splitting Criteria

A heuristic splitting criterion is used to decide how to branch the tree [2]. Quinlan's *Gain Ratio* criterion [8], for instance, chooses the attribute maximizing the information gain ratio with respect to the class attribute:

$$GainRatio(A) = \frac{H(C) - \sum_{j=1}^{J_A} p(a_j) \cdot H(C|a_j)}{-\sum_{j=1}^{J_A} p(a_j) log_2(p(a_j))}$$

where $H(C)$ is the entropy of the class, J_A corresponds to the number of different values for the A attribute, and $p(a_j)$ is the relative frequency of value a_j. $H(C|a_j)$ represents the class entropy for the a_j value of the A attribute:

$$H(C|a_j) = -\sum_{k=1}^{K} p(c_k|a_j) \cdot log_2(p(c_k|a_j))$$

where K is the number of classes and $p(c_k|a_j)$ is the relative frequency of the k-th value of the class given the j-th value of the attribute A.

Standard splitting criteria measure how good an attribute is for separating the examples belonging to different classes, but they do not take the relative importance of each class into account. In a CCS classification context, however, we should bias the heuristic criteria towards nodes with a better representation of examples belonging to the most important classes.

Criteria based on class entropy average the contribution of each class. We could include CCS information in this averaging process. The resulting criterion family, or CCS evaluation criteria, E_f, consider the relative importance of each class and can be formalized as follows:

$$E_f(C|a_j) = \sum_{k=1}^{K} f(w_k) \cdot E(c_k|a_j)$$

where $f(w_k)$ is the function used to aggregate the contributions from each class according to its relative importance, which is uniquely determined by its weight.

$E(c_k|a_j)$ is the function we use to compute how good the j-th value of the attribute A is in determining the k-th value of the class. Here, we can resort to the value given by the standard entropy-based splitting criteria, that is

$$E(c_k|a_j) = -p(c_k|a_j) \cdot log_2(p(c_k|a_j))$$

Please note that, when $f(w_k) = 1$, $E_f \equiv H$. We propose two alternative splitting criteria using two different aggregation functions:

Simple CCS Evaluation (E_I). We could directly resort to the class weights to aggregate the contributions from each class, i.e., using the identity function, $f(w_k) \equiv I(w_k) = w_k$. The resulting criterion is then:

$$E_I(C|a_j) = \sum_{k=1}^{K} w_k \cdot E(c_k|a_j)$$

Weighted Premium CCS Evaluation (E_{WP}). In some cases, the previous criterion could lead to classifiers that would tend to ignore the least important classes as we increase the relative importance of some classes. In some sense, this is similar to the problem imbalanced learning methods try to address and it should be avoided. Hence we propose a different criterion that uses a softened aggregation function: *Weighted Premium* (WP). For a given class weight w_k, its weighted premium is

$$\mathrm{WP}(w_k) = 1 + \frac{w_k - min\_weight}{max\_weight} \tag{1}$$

where *min_weight* corresponds to the weight of the least important class and *max_weight* represents the most important class weight. Therefore, the weighted premium is 1 for the least important classes and it is greater than one for more important classes. It favors the most important classes without ignoring the least important ones.

The normalized version of weighted premiums can then be used as the $f(w_k)$ aggregation function to define a Weighted Premium Evaluation criterion, E_{WP}:

$$E_{\mathrm{WP}}(C|a_j) = \sum_{k=1}^{K} \left[\frac{\mathrm{WP}(w_k)}{\sum_{i=1}^{K} \mathrm{WP}(w_i)} \right] \cdot E(c_k|a_j)$$

For instance, in a 2-class problem where one class is nine times more important than the other, the relative weights for the two classes would be $\frac{9}{10}$ and $\frac{1}{10}$. After the WP transformation, the resulting values would be softened: 0.65 and 0.35, respectively.

3.2 Tree Pruning

Tree pruning is necessary in TDIDT algorithms to avoid overfitting. Quinlan's pessimistic pruning [8], for instance, performs a postorder traversal of the tree internal nodes in order to decide, for each subtree, if it should be replaced for a single leaf node, which would then be labeled with the most common class in that subtree.

From a CCS classification perspective, a subtree should be pruned if the errors that tree pruning introduces correspond to examples belonging to the less important classes. In order to perform CCS tree pruning, we define a CCS error rate that takes class weights into account:

$$CCSError = \frac{\sum_{k=1}^{K} w_k \cdot e_k}{\sum_{k=1}^{K} w_k \cdot n_k}$$

where e_k is the number of misclassified training examples from the k-th class and n_k is the support of the k-th class in the training set. In other words, a misclassified example is taken into account according to its relative class weight. Please note that no smoothing function, such as WP, is needed when defining the CCSError rate because it is a direct measure of the CCS model quality. For splitting criteria, however, the smoothing function was needed for the algorithm not to ignore the least important classes.

We propose two pruning strategies that take CCSError into account:

– The first pruning strategy, ***CCS Pruning***, adapts Quinlan's pessimistic error estimate by replacing *Error* with *CCSError*. However, there are some situations in which *CCS pruning* is not effective enough. Let us imagine a node whose examples mainly belong to unimportant classes, maybe with some occasional examples belonging to a very important class. When the relative importance of the important class is very high, pruning will not be performed. However, it is clear that pruning might be beneficial in this case, since complexity would be reduced while accuracy would not be severely affected. This leads us to a second pruning strategy:
– ***Double Pruning*** addresses the aforementioned situation by allowing the use of the *CCSError* rate to perform *additional* pruning, apart from Quinlan's standard pruning. In other words, we will prune a subtree if the pessimistic estimation of the CCS error for the subtree is higher than the CCS error for a single leaf node, even when the standard estimation of the leaf node error is greater than the standard estimation of the subtree error.

4 CCS Classifier Evaluation

In order to evaluate CCS models, we have used three different metrics corresponding to three different aspects we would like a CCS classifier to address: simplicity with respect to important classes (that is, the main driver behind CCS classification), a good overall accuracy (since the resultant model should still be useful in practice), and a small false positive rate (to check that the complexity reduction does not come at the cost of too many false positives for the important classes).

4.1 Classifier Weighted Complexity

AvgCSDepth is defined as the CCS version of the average tree depth typically used to measure decision tree complexity. The average tree depth, without taking class weights into account, can be computed as

$$AvgDepth = \frac{\sum_{k=1}^{K} \sum_{l=1}^{L} n_{kl} \cdot l}{\sum_{k=1}^{K} n_k}$$

where K is the number of classes, L is the number of tree levels (i.e. its overall depth), n_k is the number of examples belonging to the k-th class, and n_{kl} is the number of examples belonging to the k-th class that end up in a leaf at the l-th tree level.

The average class-sensitive depth, $AvgCSDepth$, is computed as a weighted average by taking class weights into account, so that each class importance determines that class contribution to the CCS depth:

$$AvgCSDepth = \frac{\sum_{k=1}^{K} w_k \sum_{l=1}^{L} n_{kl} \cdot l}{\sum_{k=1}^{K} w_k n_k}$$

This way, a decision tree with important class leaves nearer to the root will have a lower CS depth. When all weights are the same, the classifier CCS depth measure is equivalent to the standard average tree depth.

4.2 Classifier Accuracy

Even though our goal is to achieve simpler models for the most important classes, we must still consider the resulting classifier accuracy. The simplest classification model, from a CCS classification perspective, would be useless if it misclassified too many examples. Hence, we will also include the overall classifier accuracy in our experiment results:

$$Accuracy = \frac{\sum_{k=1}^{K} TP_k}{N}$$

where TP_k is the number of true positives belonging to the k-th class and N is the total number of examples.

4.3 F Measure: False Positives and False Negatives

Finally, we also check that the complexity reduction we achieve does not come at the cost of an inordinate number of false positives. For this, we resort to the F measure typically used in information retrieval. This measure computes the harmonic mean of the resulting classifier precision and recall. In particular, we resort to the macro-averaging F measure [10] that is defined as follows:

$$MacroF = \frac{2 \cdot Pr^M \cdot Re^M}{Pr^M + Re^M}$$

where Pr^M and Re^M represent the macro-averaged precision and recall measures (i.e. the average of the individual measurements performed for each one of the problem classes).

5 Experimental Results

We have tested CCS versions of the standard C4.5 TDIDT algorithm [8] with some well-known classification problems from the UCI Machine Learning Repository [7]. Table 1 shows the data sets we have used in our experiments.

Table 1. Data sets used to evaluate CCS decision trees

Dataset	Records	#Attr.	Classes
ADULT	48842	15	2
AUSTRALIAN	690	15	2
AUTOS	205	25	6
BALANCE-SCALE	625	4	3
BREAST	699	9	2
CAR	1728	7	4
CHESS	3196	36	2
CMC	1473	9	3
FLAGS	194	29	8
GLASS	214	9	6
HAYESROTH	160	5	3
HEART	270	14	2
IMAGE	2310	18	7
IRIS	150	5	3
MAGIC	19020	10	2
MUSHROOM	8124	23	2
NURSERY	12960	9	5
PIMA	768	9	2
SPAMBASE	4601	57	2
SPLICE	3175	61	3
TICTACTOE	958	10	2
TITANIC	2201	4	2
VOTES	435	17	2
WAVEFORM	5000	22	3
WINE	178	14	3

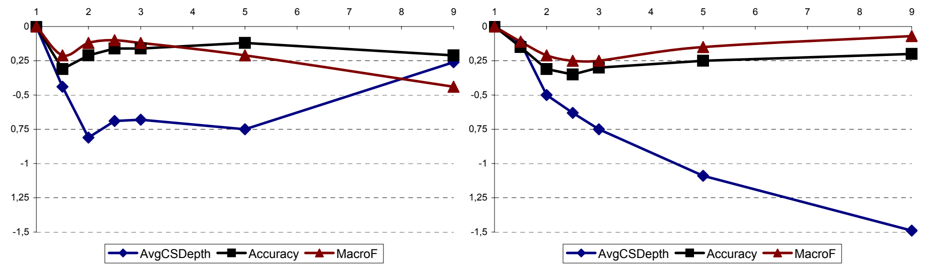
For each classification problem, we have performed an experiment suite for each class value. In each suite, a different class is chosen to be the most important one whereas the others are equally important among them. Each suite, itself, includes experiments with seven different relative weights. We have tested the algorithms performance when the higher weight is 1, 1.5, 2, 2.5, 3, 5, and 9 times the lower weight, where 1 corresponds to the non-weighted case, 1.5 corresponds to a 50% premium, and so on. Since each particular weight assignment is evaluated using a 10-folded cross validation, that leads to $10 \cdot 7 \cdot K$ experiments for each algorithm tested on a particular data set, where K is the number of different classes in the data set. Average results for each problem will be used in order to draw conclusions.

In addition, further statistical analysis [3] have been performed in order to ensure the validity of the conclusions drawn from our experiments. In our results, the number of wins-ties-losses and the average value each measure will be deemed as significant according to the Sign Test [11] and Wilcoxon's test [12], respectively.

Figures 1-3 compare the results we have obtained with CCS decision trees with the results standard decision trees would achieve. The charts show the changes that the use of CCS heuristics cause for the three metrics described in the previous section: *average class-sensitive tree depth* (`AvgCSDepth`), *classifier accuracy* (`Accuracy`), and *macro-averaged F measure* (`MacroF`). In these charts, the results for different class weights are shown along the X axis, while the Y axis corresponds to the percentage variation CCS techniques introduce for each metric. The X axis corresponds to the results we would achieve by using a traditional decision tree classifier, which are the results we will always obtain with equal class weights ($w = 1$).

Splitting criteria. Figure 1 shows the results we obtain when we use the E_I and E_{WP} splitting criteria instead of the C4.5 gain ratio criterion, always using the standard C4.5 pessimistic pruning strategy [8].

Regarding E_I, no effective CCS depth reduction is observed. Moreover, neither `Accuracy` nor `MacroF` are reduced. In fact, only the `MacroF` result for $w = 9$ is

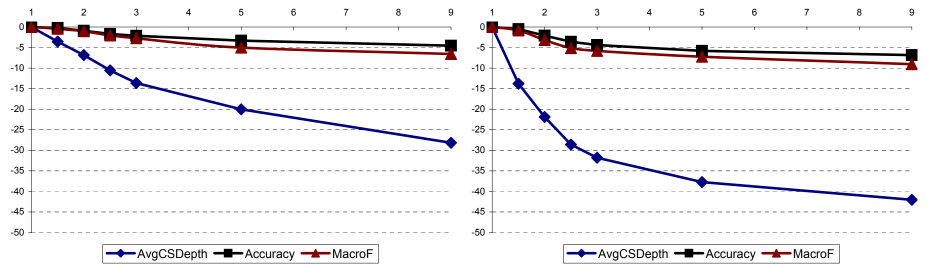


E_I + C4.5 Pessimistic Pruning E_{WP} + C4.5 Pessimistic Pruning

Fig. 1. Results obtained using CCS splitting criteria

significant according to Wilcoxon's test. The remaining differences are never significant according to this statistical test.

With respect to E_{WP}, the average CCS depth is progressively reduced when w is increased, while both `Accuracy` and `MacroF` are hardly reduced. However, the differences are never significant according to Wilcoxon's test.

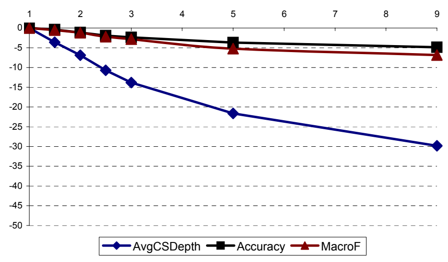


Gain Ratio + CCS Pruning Gain Ratio + Double Pruning

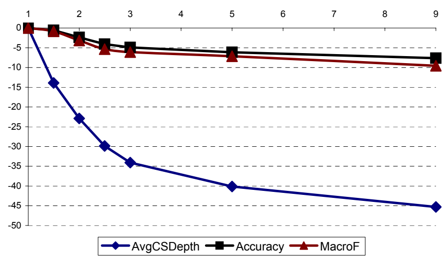
Fig. 2. Results obtained using CCS pruning techniques

Pruning techniques. Figure 2 depicts the summarized results we have obtained when using CCS pruning and double pruning. Both pruning techniques lead to much simpler models from a CCS classification point of view, with relatively small accuracy and precision losses.

Double pruning has proved to be more effective than simple CCS pruning, since greater reductions are achieved even for smaller relative weights. The average CCS depth obtained by using double pruning is obviously never worse than what we would achieve using the standard pessimistic pruning and the differences are always statistically significant, even though they are less pronounced for accuracy and MacroF than for tree depth.



E_{WP} + CCS Pruning



E_{WP} + Double Pruning

Fig. 3. Results obtained when combining WP Criterium and CCS pruning techniques

Combined methods. Figure 3 summarizes the results we have obtained by combining WP criterium and CSS pruning techniques (similar results are obtained by considering E_I instead of E_{WP}). In all cases, CCS depth is significantly reduced (according to Wilcoxon's test) with minor reductions on classifier accuracy and precision. Both splitting criteria (E_I and E_{WP}) always achieve, in combination with CCS pruning, simpler models than the standard splitting criterion, but CCS pruning is the main driver behind the observed differences.

If we consider the number of times that the average CCS tree depth is reduced, we consistently obtain significant improvements according to the Sign Test [11]. Out of the 81 experiments we performed, simple CCS pruning reduces the classifier complexity in 51 to 72 situations depending on the relative weights we use (the higher w, the higher the number of wins). Double pruning achieves even better results: from 71 to 76 wins out of 81 tests (consistently reducing classifier complexity even for small values of w).

6 Conclusions

Classification model complexity is very important since it is closely related to its interpretability. In many real-life problems, some class values are, somehow, more important than others. In such situations, experts might be interested in building succinct models for the mnost important classes. Class-complexity-sensitive (CCS) classification techniques are designed to address this issue and they build simpler models for the most important classes without incurring into high accuracy losses.

In this paper, we have introduced several heuristics that let us adapt standard decision tree learning algorithm in order to take class importance into account. Both splitting criteria and pruning strategies have been devised to deal with CCS classification, thus providing the mechanisms needed to build CCS decision trees using the standard TDIDT algorithm.

Our experimental results show that CCS splitting criteria do not provide significant improvements with respect to their traditional counterparts, a result that is consistent with prior research on splitting criteria [2].

However, CCS pruning techniques help us achieve considerable reductions in CCS model complexity within a reasonable accuracy loss. Depth and accuracy/precision are traded off, as expected, when weights are introduced into the standard TDIDT model. Combining both CCS splitting criteria and CCS pruning techniques leads us to even smaller CCS classification models.

References

1. Batista, G.E.A.P.A., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations 6(1), 20–29 (2004)
2. Berzal, F., Cubero, J.C., Cuenca, F., Martín-Bautista, M.J.: On the quest for easy-to-understand splitting rules. Data and Knowledge Engineering 44(1), 31–48 (2003)
3. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30 (2006)
4. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 155–164 (1999)
5. Gamberger, D., Lavrac, N.: Expert-guided subgroup discovery: Methodology and application. Journal of Artificial Intelligence Research 17, 501–527 (2002)
6. Gehrke, J., Ramakrishnan, R., Ganti, V.: Rainforest - a framework for fast decision tree construction of large datasets. Data Mining and Knowledge Discovery 4(2/3), 127–162 (2000)
7. Blake, C.L., Newman, D.J., Merz, C.J.: UCI repository of machine learning databases (1998)
8. Ross Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
9. Rastogi, R., Shim, K.: PUBLIC: A decision tree classifier that integrates building and pruning. Data Mining and Knowledge Discovery 4(4), 315–344 (2000)
10. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. 34(1), 1–47 (2002)
11. Sheskin, D.J.: Handbook of parametric and nonparametric statistical procedures. Chapman and Hall/CRC, Boca Raton (2000)
12. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80–83 (1945)