Learning From Imbalanced Data: Rank Metrics and Extra Tasks

Rich Caruana
Departments of Computer Science and Radiology, UCLA
Center for Automated Learning and Discovery, CMU
5000 Forbes Avenue, Pittsburgh, PA 15213
caruana@cs.cmu.edu

Abstract

Imbalanced data creates two problems for machine learning. First, even if the training set is large, the sample size of smaller classes may be small. Learning accurate models from small samples is hard. Multitask learning is one way to learn more accurate models from small samples that is particularly well suited to imbalanced data. A second problem when learning from imbalanced data is that the usual error metrics (e.g., accuracy or squared error) cause learning to pay more attention to large classes than to small classes. This problem can be mitigated by careful selection of the error metric. We find rank based error metrics often perform better when an important class is under-represented.

INTRODUCTION

Consider a data set with 10,000 cases. Many would consider this a large data set. Suppose, however, that this data set is imbalanced: the positive class occurs only 1% of the time. This large dataset contains only 100 positive cases.

Consider a second dataset with only 200 cases, 50% of which are positive. Most would consider this a small dataset. But this small dataset also contains 100 positive cases. If a large dataset and a small dataset contain the same number of positive instances, is the large dataset really larger than the small data set? Perhaps a better measure of dataset size is the size of the smallest important class.

So one problem of learning from imbalanced data is the problem of learning from small data. Learning good models from small samples is difficult, and care must be taken to avoid overfitting. The problem is further amplified with imbalanced data because the model selection criteria managing the bias/variance tradeoff often are more sensitive to larger classes than to smaller classes.

Imbalanced data poses additional problems beyond those arising from small data. Because cases from smaller classes are surrounded by a sea of cases from larger classes, most learning methods typically spend most of their effort learning about the predominant classes. For example, a small increase in the false positive rate on a large class easily swamps changes made in predictions for smaller classes. Most learners focus most of their effort on getting the larger classes right before learning much about smaller classes. One reason for this is that most error metrics favor small improvements in accuracy on predominat classes over the ability to differentiate between classes, small or large.

Imbalanced data gives us two problems to address:

- how to re-focus learning and model selection so that they pay as much attention to small classes as they do to large classes.
- how to learn models that generalize well from small samples

We discuss two approaches to these problems. One is to use rank metrics instead of accuracy metrics for training and model selection. Rank metrics are more concerned with the relative ordering of cases than with making absolute predictions for cases. This yields two advantages for dealing with imbalanced data. First, relative models ofen are simpler (i.e., less nonlinear) than absolute models, and thus usually are easier to learn from small data sets, and imbalanced data sets often effectively are small. Second, and perhaps more importantly, rank metrics place more emphasis on learning to distinguish classes than on learning the internal structure of classes, and thus place more emphasis on learning how small classes differ from large classes than learning internal structure about the larger classes.

The second approach to dealing with imbalanced data is to use inductive transfer to maximize what can be learned from small data sets. The approach we follow is multitask learning (MTL). MTL trains the model on extra tasks related to the important main task while using a shared representation. If what is learned for these extra tasks is useful to the main task, the main task will be learned better. MTL allows us to amplify the effective size of the under-represented class by adding extra training signals to the data set.

RANK METRICS

Consider a data set with two classes, pos and neg. Suppose the data set has 10,000 instances, with 1/data set suffers from data imbalance: it contains 9,900 neg instances but only 100 pos instances.

The Standard Approach

Suppose we train an artifical neural net on this problem, with an output coding of 0 for neg and 1 for pos. The first thing the net learns is to adjust the bias weight on the output unit so that it predicts values near zero for all cases. The net predicts small values for all cases, including the few pos cases in the training set.

This net has two qualitatively different directions learning can go: it can learn to predict larger values for pos cases, or it can learn to predict values closer to zero for neg cases. Given that there are 100 times as many neg cases, a backprop net usually will learn more about driving neg cases towards zero than driving pos cases towards one. This disparity in what the net learns about the two classes continues for a long time. Eventually, the net is so good at learning to drive neg cases to values near zero that it places more emphasis on learning to drive pos cases to larger values. Unfortunately, most of what the net learned to drive neg values near zero probably is not relevant to distinguishing pos and neg cases. Even worse, if the net can learn something useful for the pos cases, but even slightly harmful to the near-zero predictions for the neg cases, gradient descent on squared error will not learn it. Things learned for pos cases must not hurt neg cases.

One way to avoid this is to artificially reduce the disparity between the sizes of the two classes by subsampling the larger class. But it is unfortunate to throw away so much data. (A better approach would be to subsample from the larger class many times, learn many models, and combine those models by averaging.) Another way to avoid this problem is to increase the weight applied to errors on the less frequent pos cases. This forces learning to pay more attention to errors on the pos cases early in learning. This is a good approach. Unfortunately, it has a problem: if the number of pos cases is small (e.g., only 100 pos cases), learning almost certainly will overfit the pos cases as it tries to learn models that makes small errors on these few, but highly weighted, cases.

Benevolent Spirits

Can we force learning to pay similar attention to both classes at all stages of learning without provoking massive overfiting to the smaller class? Often we can. The real source of the problem is that we used an output coding of 0 for neg cases and 1 for pos cases. Moreover, we trained the net using squared error. This is very unfortunate output coding and error metric for imbalanced data.

Most real-world problems exhibit a range of neg cases. Many of the neg cases will be very different from the pos cases. Many more will be moderately different from the pos cases. And some of the neg cases will be very similar to the pos cases. (Some neg cases may be indistinguishable from pos cases, and vice-versa.)

When we use an output encoding of 0/1 squared error, we force learning to treat all the neg cases the same way. It is penalized for allowing some neg cases to have predicted values near the value coding for pos cases. This is not so serious a problem when the number of pos and neg cases are balanced, because the penalty is similar for pos cases with predicted values near the value coding for neg cases. The two penalties roughly balance each other as the net learns models that discriminate between the pos and neg cases.

But when the data is imbalanced, these two penalties no longer balance each other. The net becomes far more concerned about making the predictions for neg cases be near the target value for neg cases (0) because there are so many more neg cases than pos cases. It is far less concerned about pos cases predicted to be the neg target value. The real source of this problem is that the net is not allowed to treat some neg cases differently from other neg cases – it needs to drive all kinds of neg cases near the zero coding because there are so many neg cases, that even the small proportion of neg cases that are similar to the pos cases will outnumber the pos cases. The net spends more effort learning to predict the same coded value for all neg cases than it does learning to predict different values for pos and neg cases.

One solution to this problem is to introduce intermediate codings for some neg cases that allow neg cases similar to pos cases to have target values closer to the target values for pos cases. Ideally, some benevolent spirit would take all the neg cases, and label them with values between 0 and 0.99 indicating how similar they are to the pos cases, which are still labelled as 1.0. Given this benevolent relabelling, many of the problems of learning from imbalanced data go away. Of course the real world rarely is this benevolent. Rank metrics, however, allow us to create intermediate valuations for the neg cases. And as a bonus, the models that have to be learned typically are simpler (i.e., less non-linear), and thus are easier to learn from small data sets.

Benevolent Ranks: Rankprop

Several years ago we introduced a method called *rankprop* that learns to predict the target values for the neg cases at the same time it learns a model to discriminate bwetween the pos and neg cases (Caruana, Baluja and Mitchell 1994). Rankprop uses a rank metric instead of the absolute 0/1-based metric above. The basic approach in rankprop is as follows: rank the training set by the true class, neg and pos. This ranks all neg cases below all pos cases. Assign to each training case a target value between 0 and 1 based on its position in this ranking. (Note that because all neg cases get the same rank, they all get assigned a value of 0.495. Similarly, the pos cases all get assigned 0.995.)

Train the backprop net on these target values for one epoch. So far there is little to distinguish rankprop from training on 0/1 targets except that the targets are not 0 and 1 (which isn't significant). Here's where a difference arises: After performing one epoch of backprop, re-rank the training cases using the net's predictions as secondary keys in the ranking. All neg cases are still ranked below all pos cases (the class is the primary key), but the predictions for each case allow ties between neg cases to be split (and similarly for ties between pos cases, but this is less important since their values will always fall between 0.99 and 1.0). Using this new ranking, re-assign the target values for the training cases. Now some neg cases have target values near 0.0, and others have target values near 0.99. Using the new target values, perform a second epoch of backprop. After this epoch, once again use the net's predictions to re-rank the pos and neg cases, and assign new target values to the training cases based on the new rankings. Do another epoch. Repeat.

Rankprop allows neg cases to acquire target values that place them nearer or farther from the few pos cases. In fact, almost 50% of the neg cases will have target values above 0.5! Moreover, there is no pressure for the net to assign any particular value to any particular neg case. The net is allowed to discover an ordering of the neg cases that is most compatible with the discrimination model it is trying to learn to distinguish beteen the pos and neg cases. This means that the rankprop net is not driven to spend most of its effort driving the neg cases towards an unrealistic target value of zero, eliminating the pressure caused by imbalanced data that had caused the net to learn more about neg cases than about pos cases.

A second benefit of this approach is that for many problems, the target values that correspond to the ranked data form a smoother function than the 0/1 targets. Since it is easier to learn smoother, less nonlinear functions from limited data, rankprop models often overfit the pos samples less. The combination of these two effects can yield dramatic improvements. For example, on a medical problem where training sets contained 1000 cases, and where the pos class occurred only 10/performance. (Much of this benefit would disappear if the data sets were *very* large, and if the data were balanced.)

Soft Ranks

Rankprop is a complex method that learns soft target values for the neg cases while it is learning a model to discriminate between the pos and neg cases. Many error metrics based on rankings (instead of absolute target values) confer similar advantages when learning from imbalanced data, even if they don't explicitly train on iteratively reassigned target values. For example, if you learn to optimize the ROC area for imbalanced data, the imbalance problems that rankprop sidesteps also will be sidestepped. We take advantage of this observation to introduce a more general rank-based learning procedure.

Rankprop is effective, but it is not clear how to adapt Rankprop to learning algorithms other than backprop. We developed a new ranking mathod called Soft Ranks that is more flexible than rankprop and can be adapted for most learning methods. More importantly, soft ranks allow us to use any rank metric as an error metric for learning.

Ranking takes continuous data and converts it to a discrete ordering. Small changes in the predicted value of a case usually do not affect the ranking unless they are large enough to cause that case to be ranked to a different palce. This makes it difficult to use metrics defined on ranks with learning methods such as gradient descent: the large number of plateaus in the error surface cripples gradient descent. For example, if the predicted value of B changes from 0.13 to 0.14 in the data set shown below, it still ranks 2nd in the set.

```
ITEMS          TRADITIONAL RANKS

A: 0.25          E: 0.08  ->  1
B: 0.13          B: 0.13  ->  2
C: 0.54    =>    A: 0.25  ->  3
D: 0.27          D: 0.27  ->  4
E: 0.08          C: 0.54  ->  5
```

Soft Ranks is a generalization of standard discrete ranks that gives ranks a continuous flavor, making it easier to create differentiable error metrics based on ranks. The main problem solved by soft ranks is to make small changes to the predicted value of a case have a small affect on the ranking of that case. This eliminates the plateaus that can hinder learning. A small modification to the traditional rank procedure eliminates the plateau problem while preserving rank semantics. Order the data as usual and temporarily assign to each item the traditional rank. Then, post-process the traditional ranks as follows:

$$SR(i) = TR(Prev(i)) + 0.5 + \frac{V(i) - V(Prev(i))}{V(Post(i) - V(Prev(i))}$$

where $TR(i)$ is the traditional rank of i, $SR(i)$ is its continuous rank, $V(i)$ is i's value, and $Prev(i)$ and $Post(i)$ denote the items that rank just before and just after i, respectively.

```
TRADITIONAL RANKS           SOFT RANKS

E: 0.08 ->  1          E: 0.08 ->  1 ->  1
B: 0.13 ->  2          B: 0.13 ->  2 ->  2.088
A: 0.25 ->  3    =>    A: 0.25 ->  3 ->  3.300
D: 0.27 ->  4          D: 0.27 ->  4 ->  3.569
C: 0.54 ->  5          C: 0.54 ->  5 ->  5
```

Item A's value is closer to item D's value than to item B's value. The soft rank reflects this by assigning to A a soft rank closer to the soft rank of D than to the soft rank of B. Moreover, if changes in the values cause neighbors to swap, the soft ranks reflect this in a smooth way.

Qualitatively, soft ranks behave like traditional ranks, but are continuous: small changes to item values yields small changes in the soft ranks. It is possible to use soft ranks in most error metrics that use traditional ranks. Most error metrics defined on soft ranks are similar to those defined on traditional ranks, except that soft ranks will not make the error metric discontinuous like traditional. This means it is easier to apply gradient descent to error metrics based on soft ranks.

As we shall see in the next section on multitask learning, soft ranks allow rank-based error metrics to be optimized using gradient descent for learning.

FOCUSSING LEARNING WITH EXTRA TASKS

Multitask Learning (MTL) is an inductive transfer method that uses the information contained in the training signals of *related* tasks to improve the generalization performance of the main task. It does this by learning the extra tasks in parallel with the main task while using a shared representation – what is learned for the extra tasks can help the main task be learned better. In artificial neural nets, multitask learning is done by training all the tasks on one neural net using a single hidden layer shared by the tasks. The shared hidden units allow some of the features learned for the extra tasks to be used by the outputs for the main task.(Abu-Mostafa 1989)(Suddarth and Holden 1991)(Caruana 1993)(Baxter 1994)

MTL helps learning from imbalanced data in two ways:

1. Because MTL improves generalization performance when learning from small data sets, and imbalanced data sets effectively often are small data sets, MTL makes learning from them more effective.
2. The extra tasks used in MTL can be used to focus the learner's attention towards the under-represented classes.

MTL is equally effective with case-based methods such as kernel regression and k-nearest neighbor. Since these methods do not have a hidden layer to share between tasks, we use an error metric that combines the performance on the main task with the performance of the extra tasks. This causes models to be learned that perform well on both the main task and the extra tasks.

Weighted Euclidean Distance

The distance metric most commonly used for kernal regression and k-nearest neighbor is weightedEuclidean distance:

$$DIST(c1, c2) = \sqrt{\sum_{i=1}^{D} W_i * (F_{i,c1} - F_{i,c2})^2}$$

where $c1$ and $c2$ are two cases, D is the number of feature dimensions, $F_{i,c}$ is feature i for case c, and W_i is a weight for each feature dimension that controls how important that dimension is to the distance computation. In simple *unweighted* Euclidean distance, $\forall i$, $W_i = 1.0$.

Kernel Regression

Kernel regression (also known as locally weighted averaging) uses the distance metric to weight the contribution of each training case to the prediction:

$$PRED = \frac{\sum_{i=1}^{N} VAL_i * e^{-DIST(c_i, c_{test})/KW}}{\sum_{i=1}^{N} e^{-DIST(c_i, c_{test})/KW}}$$

where N is the number of cases in the case base, $DIST(c_i, c_{new})$ is the distance between case c_i and the test (probe) case, and KW is the kernel width that determines the scale of the neighborhood that has most effect. See (Atkeson, Moore and Schaal 1997) for more detail about locally weighted averaging.

Feature Weights and the Distance Metric

The performance of case-based methods like kernel regression depends on the quality of the distance metric. Finding good feature weights is essential to optimal performance. The search for good feature weights can be cast as an optimization problem. Gradient descent is used with leave-one-out cross validation (CV) to search for good feature weights. Feature weights are initialized to a starting value such as 1.0, and the CV performance of the initial weights is calculated. The gradient of the CV performance with respect to the feature weights is calculated, and a step is taken down the gradient. If the updated feature weights improve performance as measured by CV, the step is accepted. If not, the step is rejected and the step size is reduced. This process is repeated until a local minimum in CV performance is found.

Unfortunately, if this process is applied to imbalanced data using 0/1 codings and a squared error (or accuracy) error metric, the same problem arises as when backprop nets were trained this way: gradient descent spends more time learning feature weights that are good at pushing neg cases towards their target coding than at learning weights that distinguish between pos and neg cases. Instead of rankprop (which isn't applicable to case-based methods), we use the soft ranks to force learning to pay more equal attention to the pos and neg cases. We also use extra tasks to force learning to learn models that pay more attention to the less frequent class.

Multitask Learning in Kernel Regression

The goal is to find feature weights that yield good performance on both the main task and on the related extra tasks. We do this by using an error metric that combines the performance on the main task with the performance on the extra tasks. This causes gradient descent to learn feature weights that perform well on both the main task and the extra tasks. If the extra tasks are selected to help the model better learn distinctions between the imbalanced classes, it will help focus the model on characteristics of the under-represented classes.

We weight the contribution of the extra task performances because it is possible for many extra tasks to swamp the error signal of the main task if the extra tasks are too dissimilar. The contribution of the extra tasks to the combined error metric is controlled by a single weight λ:

$$MTL\_ERF = (1-\lambda)MAIN\_ERF + \lambda \sum_{t=1}^{TASKS} ERF(t)$$

where MTL_ERF is the multitask learning criterion being minimized by gradient descent on feature weights, $MAIN\_ERF$ is the performance on the main task, $ERF(t)$ is the performance on extra task t, and λ is defined on $[0.0, 1.0]$. When $\lambda = 0$, all weight is given to the main task and the extra tasks are ignored. This is traditional learning of a single task. When $\lambda = 0.5$, equal weight is given to the main task and to each extra task. When $\lambda = 1.0$, all weight is given to the extra tasks and the main task is ignored.

Pneumonia Risk Prediction

Here we apply MTL to a pneumonia problem that has imbalanced data. The goal is to predict which patients have the *least* risk of dying from pneumonia (all patients have already been diagnosed with pneumonia). Approximately 11% of the hospitalized patients in the database died of pneumonia. The measurements available for most patients include 30 basic measurements acquired prior to hospitalization such as age, pulse, blood pressure, and 33 lab tests such as blood counts or gases usually will not available until after hospitalization. The database also indicates whether each patient lived or died.

The most useful medical tests for predicting pneumonia risk usually require hospitalization. But we'd like to predict risk before patients are hospitalized so that those at low risk can be considered for outpatient care. Before hospitalization, we have available just 30 basic measurements such as age, gender, and blood pressure. Because all patients in our database were hospitalized, however, we also have in the database the results of the 33 lab tests that usually are only available for hospitalized patients. We can use these extra tests to help mitigate some fo the effects of imbalanced data. Although the lab test results will not be available when the model is used (before patients are admitted), multitask learning can be used to improve performance on the main task by using the extra lab test results available in the training set.

Learning these extra tests help mitigate the imbalanced data problem by forcing learning to learn more complete models. Some of these tasks are particularly useful because they are designed to present normal values for most well patients, and to show more unusual values for more seriously ill patients. By learning to predict these results, the model has to learn to differntiate between patients of different risk, *even if the aditional risk did not cause the patient to die.* Because high risk is more common than death, the extra tasks mitigate some of the problems of imbalanced data by creating extra tasks that are less imbalanced, but related to the main task that unfortunately is imbalanced.

Loss Functions

The main prediction task is mortality risk. Kernel regression predicts mortality risk by examining whether the close neighbors in the case-base to the new probe case lived or died. The optimization error metric we adopt for this task is the sum of the soft ranks for all patients in the sample *who live.* Successfully ordering all patients who live before all patients who die minimizes this sum. We scale the sum of soft ranks so that 0 indicates this ideal performance. Random ordering of the patients yields soft rank sums around 0.5.

The extra tasks include predicting the white blood cell count, hematocrit, albumin level, or partial pressure of oxygen in the blood. The error metric we use for the extra tasks is the standard sum-of-squares loss. Predictions for the extra tasks are not used to predict patient risk. Learning to minimize the sum-of-squares error of the extra tasks is useful only if it helps the model learn feature weights that improve performance on the main risk task.

Empirical Results

Feature weights for the 30 input features are initialized to 1.0. Gradient descent with line searches is done using CV on the training set. We use $KW = 1.0$, a value preliminary experiments indicate performs well on this problem. The value KW is not trained via gradient descent because training the feature weights can provide a similar effect.

The original dataset is randomly subsampled to create training, halt, and test sets. Because the search for feature weights repeatedly uses the same training set, overtraining is likely. A halt set is used to prevent this. Because halt set performance is often not monotonic, premature early stopping can be a problem. To prevent this, gradient descent is run for a large number of steps, and the feature weights yielding best performance on the halt set is found by examining the entire training curve. The weights learned at this point are then tested on the independent test set.

This first experiment examines how much attention learning should pay to the extra tasks. Is it better to ignore the extra tasks and optimize performance only on the main task, or is better performance on the main task achieved by optimizing performance on all tasks?

Figure 1 shows the mean rank sum error of 50 trials of learning as a function of λ for training and halt sets containing 500 patterns each. Each data point has error bars indicating the standard errors. The horizontal line at the top of the graph is the performance of kernel regression when all feature weights are 1.0, before any training has been done. Because no learning has

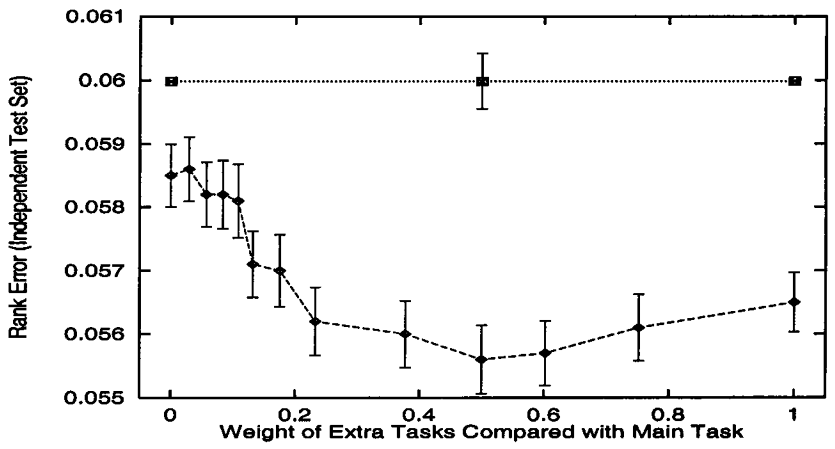

Figure 1: Rank Sum Error as a Function of λ

occurred, and because performance is measured only on the main mortality risk prediction task, this performance is independent of λ and thus forms a horizontal line on the graph.

The curve in Figure 1 below the horizontal line shows how varying λ affects performance on the main risk prediction task. As expected, training feature weights with gradient descent improves performance. The point at $\lambda = 0$ is the performance of learning with traditional single task learning; the extra tasks are completely ignored by gradient descent when $\lambda = 0$. Training the feature weights on the single main risk task reduces rank sum error about 2.5% compared with weights initialized to 1.0 and not trained.

The points for $\lambda > 0$ are multitask learning. Larger values of λ give more weight to the extra tasks.[1] From the graph it is clear that learning yields better performance on the main task if it searches for feature weights that are good for both the main task and for the extra tasks. Note that multitask learning is not using any extra training patterns, it just has more training signals in each pattern. Multitask learning is training and testing on the same training patterns used to train feature weights on the main task by itself.

The optimal value of λ is between 0.2–0.8. Interestingly, these λs place near equal weight on the main task and on each extra task. At $\lambda = 0.5$ all tasks – *including the main task* – are given equal weight. At $\lambda \approx 0.5$, multitask learning reduces error about 5–10%.

[1]The vertical axis shows performance on only the main risk prediction task. The contribution of the extra tasks to the error metric is not shown – though its effect on the error for the main task is apparent. In this problem we are not concerned with how accurately the extra tasks are predicted. The reason for including the extra tasks is so that they will help better feature weights be learned and thus improve performance on the main task.

How Much Does MTL Help?

Multitask kernel regression outperforms traditional single task kernel regression about 5% on this domain when trained with 1000 cases (500 train plus 500 halt). How many more training examples would be needed to yield this much improvement without multitask learning? We ran experiments using training sets containing 200, 400, 800, 1600, and 3200 training patterns, using single task learning ($\lambda = 0.0$) and multitask learning with $\lambda = 0.5$.

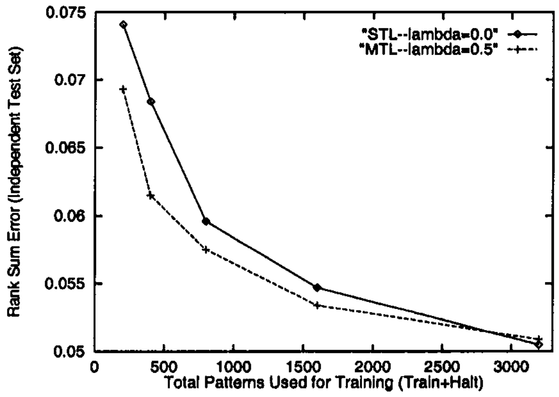

Figure 2: Rank Sum Error of STL ($\lambda = 0$) and MTL ($\lambda = 0.5$) vs. Training+Halt Set Size

Figure 2 shows the performance of 25 trials of single task kernel regression ($\lambda = 0$) and multitask kernel regression with $\lambda = 0.5$ as a function of training set size. Between 200 and 1000 cases, the improvement due to multitask learning is equivalent to a 50–100% increase in the number of training patterns. The benefit of multitask learning, however, decreases when the sample size is large. With training sets larger than 3200, the performance with $\lambda = 0.5$ may be worse than with $\lambda = 0$. The extra tasks are not helpful if the sample size is large enough to insure excellent performance from optimizing the main task alone. $0 < \lambda < 0.5$ however, may yield better performance than single task learning ($\lambda = 0$) even with training sets this large.

SUMMARY

Imbalanced data creates several problems for machine learning. One problem is that imbalanced data sets often act is if they are small data sets because some classes often have few cases. The second problem is that learning often places more emphasis on learning about the predominant classes than about learning to distinguish between large and small classes. This paper discussed two approaches to these problems. One of these is to use rank metrics such as rankprop or soft ranks to prevent learning from placing too much focus on learning about the larger classes at the expense of learning to differntiate all classes, both large and small. The other is to use extra tasks that are related to the class distinctions that must be learned, but which suffer from less imbalance. This multitask learning approach focuses learning towards more complete models that better capture features related to important inter-class distinctions. We combine both of these methods (soft ranks and MTL) and test them on a pneumonia risk prediction problem that has imbalanced data. Earlier results (not included in this paper), show that rank methods improve performance over the standard non-rank methods 30-50% on this problem. MTL improves this performance an additional 5-10%. Some of the problems of learning from imbalanced data can be mitigated by careful choice of error metric and by forcing learning to learn more complete models.

References

Y. Abu-Mostafa, "Learning From Hints in Neural Networks," *Journal of Complexity* 6:2, pp. 192-198, 1989.

C. Atkeson, A. Moore, and S. Schaal, "Locally Weighted Learning," *Artificial Intelligence Review,* (in press).

Baxter, J., "Learning Internal Representations", Ph.D. Thesis, The Flinders Univeristy of South Australia, Dec. 1994.

R. Caruana, "Multitask Learning: A Knowledge-Based Source of Inductive Bias," *Proceedings of the 10th International Conference on Machine Learning,* pp. 41-48, 1993.

R. Caruana, "Using the Future to "Sort Out" the Present: Rankprop and Multitask Learning for Medical Risk Prediction," *Advances in Neural Information Processing Systems 8,* 1996.

G. Cooper, et al., "An Evaluation of Machine Learning Methods for Predicting Pneumonia Mortality," *AI in Medicine,* 1997.

M. Fine, D. Singer, B. Hanusa, J. Lave, and W. Kapoor, "Validation of a Pneumonia Prognostic Index Using the MedisGroups Comparative Hospital Database," *American Journal of Medicine,* **94** 1993.

Pratt, L. Y., Mostow, J. & Kamm, C. A. (1991) Direct Transfer of Learned Information Among Neural Networks. *Proceedings of AAAI-91.*

S. C. Suddarth and A. D. C. Holden, "Symbolic-neural Systems and the Use of Hints for Developing Complex Systems," *International Journal of Man-Machine Studies* **35**:3, pp. 291-311, 1991.

S. Thrun and T. Mitchell, "Learning One More Thing," CMU TR: CS-94-184, 1994.