Quantifying Counts, Costs, and Trends Accurately via Machine Learning

Article · January 2007

1 author:

George Forman

74 PUBLICATIONS **6,559** CITATIONS

SEE PROFILE

Quantifying Counts, Costs, and Trends Accurately via Machine Learning

George Forman
Business Optimization Lab
HP Laboratories Palo Alto
HPL-2007-164 (R.1)
March 26, 2007*

supervised machine learning, classification, prevalence estimation, class distribution estimation, cost quantification, quantification research methodology, minimizing training effort, detecting and tracking trends, concept drift, class imbalance, text mining

In many business and science applications, it is important to track trends over historical data, for example, measuring the monthly prevalence of influenza incidents at a hospital. In situations where a machine learning classifier is needed to identify the relevant incidents from among all cases in the database, anything less than perfect classification accuracy will result in a consistent and potentially substantial bias in estimating the class prevalence. There is an assumption ubiquitous in machine learning that the class distribution of the training set matches that of the test set, but this is certainly not the case for applications where the goal is to measure changes or trends in the distribution over time. The paper defines two research challenges for machine learning that address this *distribution mismatch problem*. The 'quantification' task is to accurately estimate the number of positive cases (or class distribution) in an unlabeled test set via machine learning, using a limited training set that may have a substantially different class distribution. The 'cost quantification' task is to estimate the total cost associated with the positive class, where each case is tagged with a cost attribute, such as the hours of labor needed to resolve the case. Obtaining a precise quantification estimate over a set of cases has a very different utility model from traditional classification research, whose goal is to obtain an accurate classification for each individual case. For both forms of quantification, the paper describes a suitable experiment methodology and evaluates a variety of methods. It reveals which methods give more reliable estimates, even when training data is scarce and the testing class distribution differs widely from training. Some methods function well even under high class imbalance, e.g. 1% positives. These strengths can make quantification practical for business use, even where classification accuracy is poor.

* Internal Accession Date Only Approved for External Publication
To be published in international journal Data Mining and Knowledge Discovery in a special issue on Utility-Based Data Mining

Quantifying Counts, Costs, and Trends Accurately via Machine Learning

George Forman
Hewlett-Packard Labs
Palo Alto, CA

ABSTRACT

In many business and science applications, it is important to track trends over historical data, for example, measuring the monthly prevalence of influenza incidents at a hospital. In situations where a machine learning classifier is needed to identify the relevant incidents from among all cases in the database, anything less than perfect classification accuracy will result in a consistent and potentially substantial bias in estimating the class prevalence. There is an assumption ubiquitous in machine learning that the class distribution of the training set matches that of the test set, but this is certainly not the case for applications where the goal is to measure changes or trends in the distribution over time.

The paper defines two research challenges for machine learning that address this *distribution mismatch problem*. The 'quantification' task is to accurately estimate the number of positive cases (or class distribution) in an unlabeled test set via machine learning, using a limited training set that may have a substantially different class distribution. The 'cost quantification' task is to estimate the total cost associated with the positive class, where each case is tagged with a cost attribute, such as the hours of labor needed to resolve the case.

Obtaining a precise quantification estimate over a set of cases has a very different utility model from traditional classification research, whose goal is to obtain an accurate classification for each individual case. For both forms of quantification, the paper describes a suitable experiment methodology and evaluates a variety of methods. It reveals which methods give more reliable estimates, even when training data is scarce and the testing class distribution differs widely from training. Some methods function well even under high class imbalance, e.g. 1% positives. These strengths can make quantification practical for business use, even where classification accuracy is poor.

Keywords

supervised machine learning, classification, prevalence estimation, class distribution estimation, cost quantification, quantification research methodology, minimizing training effort, detecting and tracking trends, concept drift, class imbalance, text mining.

1. Introduction

At Hewlett-Packard our employees have trained thousands of text classifiers to examine the written notes of our voluminous technical support logs. We then use these classifiers to quantify the prevalence of specific support issues and monitor for changes or trends in the class distribution over time (Forman, Kirshenbaum and Suermondt, 2006). For example, in the iPAQ handheld product line, one classifier focuses on detecting customer calls where the battery is not holding a charge, while another focuses on cracked screens. By quantifying the prevalence for each of twenty to fifty issues for different product lines, we can determine their relative and absolute priorities. By tracking the prevalence of each type of issue over time, we can identify trends even among lesser issues, such as an increase in cracked screens reported after the July 4th US Independence holiday. More importantly, rising problems can be identified before they become

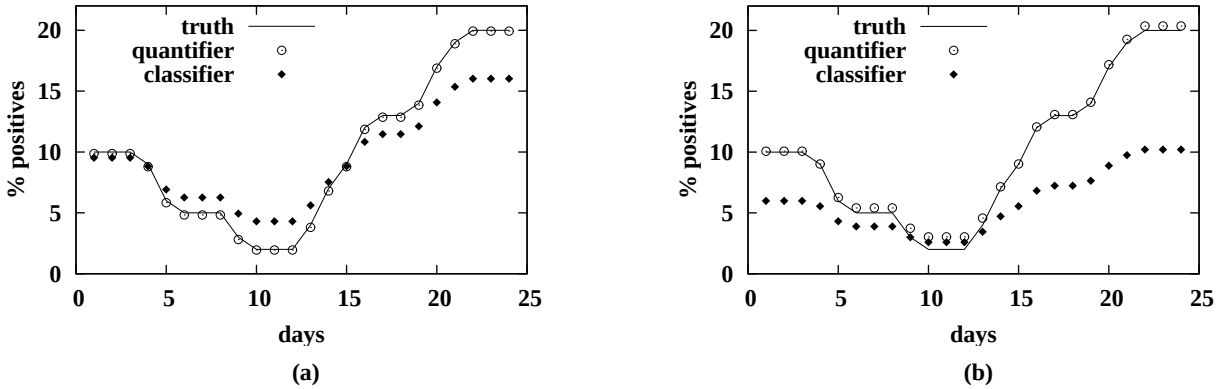

Figure 1. A fictitious trend tracked by a normal classifier and by a quantifier. (a) Each is trained with 100 positives and 900 negatives sampled from day 1, and the classifier averaged 94% correct on the rest of day 1. (b) Each is trained with just one tenth of the effort: 10 positives and 90 negatives. The classifier still averaged 93% correct on day 1.

epidemics, and the Pareto ordering of issues enables the business to efficiently improve support and products. Such analyses are needed in many business and science applications, e.g. monitoring for disease epidemics or for trends in customer demand. In most cases, such analyses assume that the individual cases have been explicitly labeled with the correct class, or that these class labels can be accurately generated by a classifier. To determine the class distribution then, one simply counts the number of cases in each class, yielding a histogram.

However, for many classification tasks, it is either impossible or costly to develop an accurate classifier. The cost of having domain experts label sufficiently many training cases can be substantial, especially when there are thousands of classifiers to train. Real-world concepts can be difficult to distinguish even for state-of-the-art classifiers. The effort to develop special purpose features or classifiers could increase the cost significantly, with no guarantee of an accurate classifier. Thus, an imperfect classifier is often all that is available.

Using an imperfect classifier to generate a class histogram will result in some degree of error that leads, perhaps surprisingly, to a systematic and sometimes large bias if the test class distribution changes from that of the training set. Although this *distribution mismatch* is expected and fundamental to applications that track trends, the vast majority of machine learning methods assume that the training set is a random sample of the test distribution. We illustrate the severity of the resulting bias with a brief example. Figure 1a shows a trend line for a single fictitious issue. It begins at 10% prevalence of the daily case volume for the first three days, decreases to 2% over the next few days, and eventually climbs to 20%. On day 1 we train a state-of-the-art SVM text classifier with a training set of 100 positives and 900 negatives. On the remaining test cases of day 1, it achieved good accuracy: 94% correct predictions, with balanced precision and recall. (Each measurement here is averaged over 100 randomly selected training sets.) The solid diamonds (♦) show the average percentage of cases predicted positive each day. Observe at the low point that it overestimated by a factor of two, and later substantially underestimated the peak. One may suggest that some other learning model might have been more accurate, but the general problem remains. We rarely obtain 100% accurate classifiers, and the imperfection leads to a systematic bias that overestimates when the prevalence goes down, and worse, underestimates during an epidemic (proof in Section 2.1). For some applications, simply detecting any change or knowing its direction may be enough. But for most business applications, precise estimates are desired, especially when comparing many non-exclusive classes against each other, such as different technical support issues.

It is sufficient but not necessary to have a perfect classifier in order to estimate the class distribution well. If the number of false positives balances against false negatives, then the overall count of predicted positives is nonetheless correct. Intuitively, the estimation task is easier for not having to deliver accurate predictions on *individual* cases. An insurance company can estimate how many cars will have accidents next year, without having to predict which ones. The nature of the uncertainty is shifted from the individual cases to the aggregate count of cases. To illustrate, the hollow circles (○) in Figure 1a show the class distribution estimates by one of the quantifier methods described in Section 2.2. It approximates the ground truth curve much more closely. Moreover, the benefits for business use are even greater—it also performs well with less training data. Figure 1b shows the same experiment, but using only one tenth the training effort: just 10 positives and 90 negatives. Compared with the previous figure, the class distribution estimates by the quantifier method are almost as good, but those of the classifier have degraded considerably, even though it still has 93% classification accuracy on day 1. Advanced methods stand to improve estimates as well as reduce the demand for training data, which is often proportional to labeling effort by domain experts. Moreover, some of the methods we propose are able to operate effectively even under high class imbalance, e.g. 1% positives, which has traditionally been very difficult for machine learning classification.

For many applications that track trends in business or science, the objective is *to accurately estimate the count or cost of cases that belong to each class*, with minimal burden. Note that this utility is quite different from needing an accurate classification for each individual case, regardless whether measured by error rate, F-measure, etc. Altogether, the radically different utility, the opportunity for improvement with less training data, and the ability to address the distribution mismatch problem even under class imbalance bring about a valuable challenge for ongoing research. We define two forms of the task:

1. The *quantification* task for machine learning: given a limited training set with class labels, induce a *quantifier* that takes an unlabeled test set as input and returns its best estimate of the number of cases in each class. In other words, the quantification task is to accurately estimate the test class distribution via machine learning, but without assuming a large training set that is sampled at random from the test distribution. The input to a quantifier is a *batch* of cases, whereas a traditional classifier takes a single case at a time and predicts its single class (or a distribution of classes reflecting its uncertainty about that one case).

   The recognition of this as an important and distinct research objective may have been delayed for several reasons. First, on the surface, this task might seem trivial—although our experiences have taught us otherwise. Second, the problem does not naturally surface in machine learning research. The ubiquitous practices of random sampling and cross-validation ensure that the training class distribution matches the test class distribution, a fundamental assumption in most machine learning theory. Third, quantification research calls for a more complicated evaluation methodology, which we address in Section 3. Finally, the extent of machine learning research where the test distribution differs from training has focused on improving the correctness of the individual classifications, not on the estimate of the class distribution itself. The literature has successfully applied ROC analysis from the medical statistics field of diagnostic test evaluation in order to adjust the classification threshold once the test distribution is known, e.g. by a labeled calibration sample from the test cases (e.g. Provost & Fawcett, 2001; Fawcett, 2003). While this research is related, it is not directly applicable for measuring the test class distribution.

The second form of the quantification task is for a common situation in business where a cost or value attribute is associated with each case. For example, a customer support log has a database field to record the amount of time spent to resolve each individual issue, or the total monetary cost of parts and labor used to fix the customer's problem. Note that this type of cost has nothing to do with any of the misclassification costs or attribute measurement costs identified by Turney (2000).

2. The *cost quantification* task for machine learning: given a limited training set with class labels, induce a *cost quantifier* that takes an unlabeled test set as input and returns its best estimate of the total cost associated with each class. In other words, return the subtotal of cost values for each class.

   For example, given a trained cost quantifier for customer support logs, one could determine the total time spent by technical support dealing with calls for each type of issue. Sometimes an expensive support issue that occurs rarely can exceed the total cost of a common, inexpensive issue. Thus, whenever an appropriate cost attribute is available, cost quantification may be preferred over simply quantifying the *number* of cases in each class.

   The obvious solution is to train a classifier and subtotal the cost attributes according to the hard class label predictions. For a binary problem, this simply sums the costs of test cases predicted to be positive. This naïve approach performs very poorly. Furthermore, the problem is more challenging than basic quantification, because false positives and false negatives do not cancel each other out unless their costs are identical. We address this task and propose methods in Section 4, comparing their performance empirically in Section 5.

We believe this vein of research may have considerable economic value. A major benefit of sophisticated methods for quantification is that a much less accurate classifier can be used to obtain reasonably precise quantification estimates. This enables some applications of machine learning to be deployed where otherwise the raw classification accuracy would be unacceptable or the training effort too great. Reducing the number of training cases required may not seem a substantial savings, however, for a large company like Hewlett-Packard with many services and product lines, multiplied by the many issues to track within each, the savings in training effort should not be undervalued. Further, the training needs may be ongoing because of concept drift in class character, as well as newly emerging support issues and the introduction of new product lines. Therefore, labor savings from quantification improvements continue to accumulate over time. For these reasons, research for quantification should focus on situations where labeled training data is limited, and should not concentrate on saving computer training time, which is relatively inconsequential and becoming more so with Moore's Law and algorithmic improvements in induction. Operating an enterprise with accurate business intelligence can lead to tremendous opportunities for optimization and/or new profit, dwarfing any cost savings in reduced training labor. Nonetheless, it is well to minimize the training labor, since it is an up-front cost for uncertain future benefits. If the up-front cost appears too great, then the endeavor will likely not be undertaken.

Real-world situations can also involve arbitrary concept drift, where the target concept may gradually or suddenly change. We limit our scope to situations where the class distribution P(c) varies, but the conditional probability of the feature space given the class P(**x**|c) does not. Fawcett and Flach (2005) provide an insightful discussion and examples of such situations, which they refer to as "Y → X" domains. We do not attempt to address concept drift in general.

We have defined the quantification tasks generally enough to cover multi-class settings. However, we primarily focus on binary tasks: a positive class versus a majority negative class. This simplifies the exposition and resolves a key subproblem for decomposing a multi-class setting into a series of binary "one class versus others" problems, as discussed in Section 6.1. When there are many classes, such a decomposition naturally leads to binary class distributions that are highly imbalanced—one reason why research should focus on imbalanced situations. This also corresponds with our experience in business settings, where it is often desirable to track many narrowly defined issue types with a limited number of positive training examples for each class.

The bulk of the paper focuses on quantification of a single test set. In order to track trends over time, the natural problem decomposition is to partition the dataset into discrete time buckets and quantify each bucket as a separate test set. There are a few nuances, which we discuss in Section 6.2.

One purpose of this paper is the empirical comparison of many methods against each other, which cannot be determined well from a series of papers each introducing one new method. For this reason, a number of quantification methods are introduced, and they are tested under an array of experimental conditions over a baseline of 25 binary text classification problems. Omitting some methods would make the exposition simpler, but would leave an incomplete statement about known methods and their performance.

The structure of the paper is as follows. Sections 2 and 3 describe a series of quantification methods of increasing complexity, and respectively evaluate them with an appropriate experiment methodology. Likewise, sections 4 and 5 lay out a series of cost quantification methods, and subsequently evaluate them. Section 6 briefly discusses the generalization to multi-class settings and trending over time. Section 7 describes related work, and Section 8 gives conclusions and directions for future work.

2. Quantification Methods

In this section, we present a series of methods for quantification. We include discussions of their shortcomings to help explain the motivation for subsequent methods of increasing complexity. In Section 3 we evaluate these methods experimentally. Until Section 6, we restrict our scope to binary tasks and the quantification of a single test set only.

2.1 Basic Methods

The obvious method for quantification is to train the best available classifier on the training set and count its positive class predictions on the test set. We call this method **Classify & Count (CC)**. The observed count P' of positives from the classifier will include both true positives and false positives, P' = TP + FP, as characterized by the standard 2x2 contingency table:

Actual:	Classifier Predictions:	
P positives	TP = $tpr \cdot$ P	FN
N negatives	FP = $fpr \cdot$ N	TN

where *tpr* is the *true positive rate* characteristic of the classifier, the probability that the classifier outputs a positive prediction given the case is actually positive, P(+ | pos) = TP / P, and *fpr* is its *false positive rate*, P(+ | neg) = FP / N. For the domains we consider, the *tpr* and *fpr* characteristics are independent of changes to the class distribution, per Fawcett and Flach (2005). Unless the classifier is perfect (*tpr=1* and *fpr=0*), which we shall assume is not the case henceforth, this simple CC method is not a good quantifier. For example, if the number of positives P increases, only the fraction *tpr* of this increase will be observed in P'. We state the following theorem:

Theorem: For an imperfect classifier, the CC method will underestimate the true proportion of positives p in a test set for p > p*, and overestimate for p < p*, where p* is the particular proportion where the CC method estimates correctly.

Proof: The probability that the binary classifier outputs a positive prediction on a random item of the test set is:

$$P(+) = P(+ \mid pos) \cdot P(pos) + P(+ \mid neg) \cdot P(neg)$$
$$= tpr \cdot P(pos) + fpr \cdot (1 - P(pos))$$

where P(pos) reflects the ground truth prevalence of positives in the test set, hereafter renamed p.
We can write as a function of p the expected prevalence p' of positive classifier outputs over the test set:

$$p'(p) = tpr \cdot p + fpr \cdot (1 - p)$$

If the classifier correctly estimates the prevalence for a particular value p*, i.e. p'(p*) = p*, then for a strictly different prevalence p + Δ, where Δ ≠ 0, it does not produce the correct prevalence:

$$p'(p^* + \Delta) = tpr \cdot (p^* + \Delta) + fpr \cdot (1 - (p^* + \Delta))$$
$$= p'(p^*) + (tpr - fpr) \cdot \Delta$$
$$= p^* + (tpr - fpr) \cdot \Delta$$

Since the classifier is imperfect, (*tpr* – *fpr*) is a fraction less than one. Therefore, if Δ is positive, the estimate p' is less than p* + Δ, and if Δ is negative, the estimate is greater than p* + Δ. QED

Hence, such a simple quantifier underestimates when the prevalence goes up, and overestimates when the prevalence goes down, as was observed in Figure 1. However, we can derive an improved quantifier method as follows:

$$p'(p) = tpr \cdot p + fpr \cdot (1 - p)$$
$$= (tpr - fpr) \cdot p + fpr$$

Solving for p, the sought quantity, we get:

$$p = \frac{p'(p) - fpr}{tpr - fpr} \qquad (1)$$

Using this, we can estimate the true proportion of positives p from the observed proportion p'. This derivation is known in the medical statistics literature, where sensitivity (*tpr*) and specificity (1-*fpr*) of a medical diagnostic test are determined from prior studies in which ground truth medical conditions are known (Zhou, Obuchowski and McClish, 2002, p.389). In our case, these distribution-independent characteristics must be estimated for the learned classifier. One option is to split the available training set: induce a classifier from one portion, and then estimate *tpr* and *fpr* from the held-out set. However, the reduction in training cases results in poorer learning, especially when there are not many positive training cases available (≤100 is our research focus). Instead, we can train on 100% of the labeled data and estimate the characteristics of the learned classifier via standard cross-validation. For 10-fold cross-validation, each fold classifier trains on 90% of the data. This difference could be substantial in the early part of the learning curve, i.e. when there is little training data

relative to the difficulty of the target concept. For this reason, we use 50-fold cross-validation so that the difference in training set size is only 2%. One could potentially use leave-one-out cross-validation. Recall that we desire to minimize the cost of obtaining labeled training cases from domain experts, rather than saving computation time, which is relatively inexpensive and becoming more so.

By assumption, there should be no fundamental change in the *tpr* and *fpr* characteristics between the training and testing distributions. However, one should expect some amount of discrepancy between their estimates from the limited training set and their ground truth values on the finite test sample. Particularly if either set is small, these discrepancies lead to error in the final estimation of p and can occasionally lead to infeasible estimates, e.g. negative values. Thus, as a final step, we clip the estimate into the range 0% to 100%. Altogether, we call this procedure the **Adjusted Count (AC)** quantification method (Forman, 2005): learn a binary classifier from the entire training set, estimate its characteristics via many-fold cross-validation (*tpr*=TP/P and *fpr*=FP/N), apply the classifier to the test set, count the number of test cases on which the classifier outputs positive, estimate the true percentage of positives via equation (1), and finally, clip the output to the feasible range.

The AC method can estimate the class distribution well in many situations, but its performance degrades severely when the training class distribution is highly imbalanced (Forman, 2006). If the positive class is rare enough in the training set, the classifier will learn to always vote negative, i.e. *tpr*=0%. Although this gives optimal *classification* accuracy for such high class imbalance, it is useless for quantification. Backing off from this extreme, either by training with less imbalance or by optimizing for F-measure instead, the classifier would predict at least a few true positives, but would still remain conservative (low *tpr*) to avoid misclassifying a great many negatives. And this results in a small denominator (*tpr* – *fpr*) in equation (1), making the quotient highly sensitive to any errors in the estimate of *tpr* or *fpr*. This problem is amplified if the training set is small, but would occur even on a very large training set having high class imbalance.

The need to operate well under these conditions is important. Class imbalance is widespread in practice, especially in business applications where the rare class is often most interesting or profitable. In some situations, a surfeit of negative training cases is freely available. A well known and often successful technique is to select only a limited sample of the available negatives to reduce the training imbalance, either by random or systematized undersampling (Van Hulse, Khoshgoftaar and Napolitano, 2007). Except where this yields a perfect classifier, we can do better for the purpose of quantification. Having investigated the reason for the degradation, we next introduce quantification methods devised to be resilient under class imbalance.

2.2 Imbalance Tolerant Methods via Classifier Threshold Selection

The solution to the class imbalance problem lies in recognizing that trying to minimize the expected loss of individual classifications has little bearing on the objective of quantification. Rather than use a classifier decision threshold optimized for accuracy, we should explicitly select the threshold to provide better quantifier estimates from equation (1). This means avoiding thresholds where estimates of *tpr* and *fpr* have greater variance or where the denominator (*tpr* – *fpr*) is small. Under class imbalance, the default threshold may yield zero or extremely few true positives or false positives in cross-validation, which leads to greater uncertainty in the *tpr* and *fpr* estimates. Instead, we will select a threshold that admits more true positives and many more false positives, yielding worse classifier accuracy but better quantifier accuracy.

The remaining question is what policy to use for selecting the threshold. To consider the possibilities, see the illustration in Figure 2, which shows a typical tradeoff between *tpr* and *fpr*. The x-axis represents the spectrum of thresholds, i.e. the scores generated by the raw classifier. In general, they are uncalibrated and may take any range, e.g. the signed distance from the separating hyperplane of an SVM, or the probability output by Naïve Bayes, which is notorious for its poor probability calibration. Any method that calibrates or normalizes the classifier scores merely changes the x-axis scale, but otherwise has no effect on the curves.

The descending curve shows the false positive rate *fpr* and the ascending curve shows the true positive rate, inverted (1 – *tpr*) to better visualize the tradeoff with *fpr*. For a perfect classifier, there would exist thresholds where the curves do not overlap. This never occurred in the many text classification tasks we studied. These particular curves represent an SVM classifier whose natural threshold delivers 92% classification accuracy for a training set having 50 positives and 1000 negatives. Because negatives abound, the classifier naturally

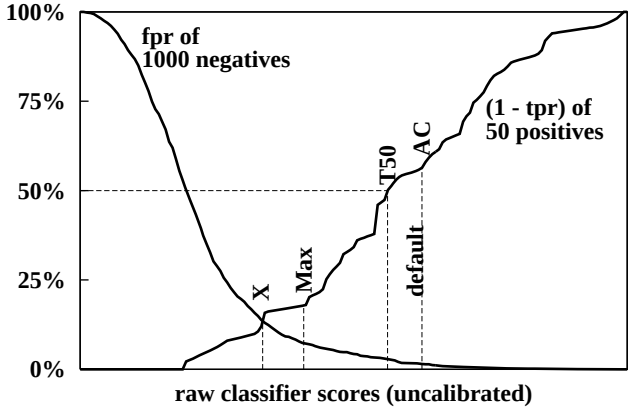

Figure 2. Various threshold selection policies.

optimized for a very low false positive rate, even at the cost of a 'few' misclassified positives: 28. This default threshold is used by the basic AC method. If the number of negatives were much greater, it would select a threshold further to the right in the tails of the curves, where estimates of *fpr* and *tpr* are less certain.

An intuitive policy for selecting a threshold is where the two curves cross, where *fpr* = 1-*tpr* (labeled X). This variant of AC we shall call **method X**, which avoids the tails of both curves. Considering the earlier discussion of small denominators, another likely policy is where the denominator (*tpr*–*fpr*) is maximized: **method Max**. More traditionally, a Neyman-Pearson criterion would select the threshold at a particular false positive rate or true positive rate. We have empirically tested various such policies, but only report here on a particularly successful one. The **method T50** selects the threshold where *tpr*=50%, avoiding the tails of the *tpr* curve, which is especially important to do for the positive class, since it has many fewer data samples than the *fpr* curve. Thus, we might expect this method to perform well when training examples are scarce.

Any threshold selection method runs the risk that the *tpr* and *fpr* estimates from cross-validation *at the chosen threshold* do not happen to match the actual rates due to sampling variation. For this reason, we consider an additional approach: obtain an estimate by equation (1) at *every* threshold, and return a mean or median of these estimates. Median is preferred, as it is less sensitive to outliers. Specifically, the **Median Sweep (MS)** method performs cross-validation to estimate *tpr* and *fpr* for all thresholds on the training set, and then on the test set it computes the estimate via equation (1) for all thresholds, and returns the median of these. This bears some relation to the *area under the ROC curve* (AUC) measure, which considers every classification threshold. A theoretical justification for expecting MS to perform well is that by taking the median of many threshold estimates, it gets the variance-reduction benefits of a bootstrap method. One detail of the method remains. Since we know that some of these thresholds yield poor estimates when the denominator (*tpr*–*fpr*) becomes small, we exclude thresholds where the denominator is less than ¼. We validated that this exclusion improved the estimates, and no experimental tuning of this constant was performed.

2.3 Mixture Model Method

For thoroughness and comparison, we also include a robust quantification method from our earlier work, which is based on very different principles. The full details of its design choices are given in Forman (2005), as well as a lesion study where we systematically disabled or varied each design choice and confirmed that performance was worse in each case. The **Mixture Model (MM)** method trains a binary classifier on the training set, discarding the decision threshold. It later applies this classifier to each test case and records the distribution D_U of raw classifier scores. This observed distribution on the unlabeled test set is modeled as a mixture of two distributions: the distribution of classifier scores on positive cases, D_+, and that of negative cases, D_-. These two distributions are determined via many-fold cross-validation performed once at training time. Figure 3 illustrates these three sample distributions to communicate the graphical intuition. The method outputs the estimate of p that gives the best match between the observed distribution D_U and the mixture distribution—that is, where $D_U \approx p \cdot D_+ + (1-p) \cdot D_-$ and the distributions are each normalized to percentages.

Determining the optimal value of p requires a search and, more significantly, a distance metric for comparing the observed distribution to the mixture distribution. While Kolmogorov-Smirnov is a logical choice for comparing cumulative distributions, the MM method uses a new metric we developed that performs better: PP-Area. Given two cumulative distribution functions (CDFs), a well-known visual comparison method is to plot one against the other while varying their input threshold, yielding a Probability-Probability plot, a.k.a. P-P plot. An example is illustrated in Figure 4, shown as a wavy curve. If the two CDFs yield the same probability at each threshold, then they generate a perfect 45° line. By sighting down this line, one can get an intuitive feel for the level of agreement between two CDFs. This is commonly done to decide whether an empirical distribution matches a parametric model of the distribution. In order to distill this visual linearity test to computation, it would be natural to measure the mean-squared-error (MSE) of the points on the PP curve versus the 45° ideal line. But MSE is highly sensitive to the maximal difference, as is Kolmogorov-Smirnov. The PP-Area metric measures the difference between two CDFs as the area where the PP curve deviates from the 45° line, as indicated in the figure. This has well defined behavior. The curve always begins at (0,0), ends at (1,1), and is monotonic in both x and y. It also has the intuitive property of being commutative, unlike MSE or mean error. And unlike Kolmogorov-Smirnov, it is sensitive to the entire shape of the curve, rather than just the maximal difference.

There is one additional detail to the MM method. Some test cases may generate a score that is greater than (or less than) any score observed in cross-validation. In this case, they are removed from the D_U distribution and treated separately as positives (or negatives) that are completely certain with respect to the final quantification estimate.

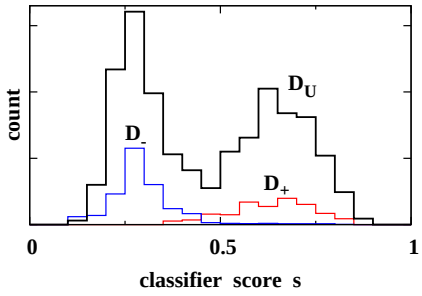

Figure 3. Distributions of classifier scores

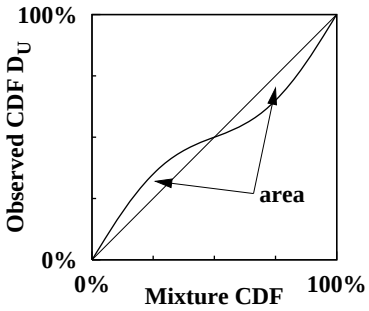

Figure 4. P-P plot comparing two CDFs

2.4 Non-Solution: Summing a Probability Estimating Classifier

It is sometimes suggested that quantification might be performed by a small variant of Count & Classify: train a *calibrated,* probability-estimating classifier, which estimates P(+|x) for a given case x, and then sum these probability estimates for all cases in the test set. This has an intuitive appeal compared with merely counting positives from a binary classifier, whose hard decisions lose information about the uncertainty of individual predictions. Nonetheless, this is an ill-conceived method. The calibration of such a classifier depends critically on the class distribution, and its estimates are no longer calibrated if the test class distribution changes. For example, for a well calibrated classifier, the output value y=70% indicates that, among the training cases that score similarly, approximately 70% were positive. Supposing a large test set contains the same proportion of positives, then among the cases that score y=70% ± ε, roughly 70% would be positive. But if we repeat the test with most of the negative cases removed at random, then the proportion of positives among the remaining test cases scoring in this bin (y=70% ± ε) would be much greater. Thus, the output of 70% would greatly underestimate the new P(+|y=70%); and likewise for every bin. The end effect is again that this method would underestimate at high prevalence, and overestimate at low prevalence, just as the CC method. Moreover, most calibration techniques add the additional complexity of determining an appropriate bin granularity and, additionally, have difficulties calibrating with small training sets or under high class imbalance. We therefore do not include this method in the experiments.

3. Empirical Evaluation of Quantification Methods

In this section we experimentally evaluate the quantification methods introduced in the previous section. There are three subsections. We begin by explaining the need for a new experiment methodology and describe that which we used. The central subsection details the results of the many experiments, highlighting methods that estimate well under a variety of circumstances. We conclude with a more general discussion that steps back from the many graphs and individual findings.

3.1 Experiment Methodology

As mentioned in the introduction, the methodology to evaluate quantification methods is substantially different than that of traditional classification research. In traditional classification, an individual prediction for a test case can be judged right or wrong independently of others, and the overall success of a method can be judged on a test set of cases. In contrast, quantification outputs a *single* number for a whole set of test cases. Many such estimates over subsets of a large benchmark must be aggregated to evaluate a method. Hence, the research methodology is unusual. It bears a superficial resemblance to the batching of cases used in research for probability estimating classifiers and in calibrating classifiers. For example, if a perfect meteorologist predicts the chance of rain at 20% on certain days of the year, we expect rain on 20% of those days. The correctness of a prediction on a single day cannot be judged in isolation. Some methods even require examination of the entire test set before producing any output. However, probability estimation, like traditional classification, continues to make individual predictions on each item and then judges them in aggregate. By contrast, quantification makes a single prediction based on an entire batch—a single scalar for binary tasks. This batching requirement calls for a unique research methodology. In particular, the class distribution must be varied independently and dramatically between the training set and the test set. Further, since each test set produces only a single estimate, we must test on many different batches and aggregate their measurements of error to identify which methods are better than others. The ideal quantification method will generate accurate estimates, despite wide variation in training and testing conditions.

To vary the *training* conditions, we randomly select P=10…100 positive training cases and N=100 or 1000 negative training cases from the benchmark dataset at hand. These sizes are selected to cover common operating ranges of interest to our business applications, and are reasonable for many other situations. The larger number of negatives represents a common multi-class case where we consider one class at a time against many others that each has 10…100 example cases.

Our prior study measured performance on test sets ranging from p=5%…95% positive. While reasonable scientifically, this does not focus on the area of interest for typical business problems: 1…20% positives is a more challenging and more important range in which to estimate well. Why? First, it is common to have several *mutually exclusive* classes, e.g. each news article is printed in only a single section of the newspaper. If there are many classes, most of them must be relatively rare in order to sum to 100%. Second, for datasets

Table 1. Benchmark classification tasks.

#	Dataset	Class	Positives	Negatives	%Positive	Total
1	fbis	3	387	2076	16%	2463
2	fbis	7	506	1957	21%	2463
3	fbis	10	358	2105	15%	2463
4	la1	0	354	2850	11%	3204
5	la1	1	555	2649	17%	3204
6	la1	2	341	2863	11%	3204
7	la1	3	943	2261	29%	3204
8	la1	4	273	2931	9%	3204
9	la1	5	738	2466	23%	3204
10	la2	0	375	2700	12%	3075
11	la2	1	487	2588	16%	3075
12	la2	2	301	2774	10%	3075
13	la2	3	905	2170	29%	3075
14	la2	4	248	2827	8%	3075
15	la2	5	759	2316	25%	3075
16	ohscal	0	1159	10003	10%	11162
17	ohscal	1	709	10453	6%	11162
18	ohscal	2	764	10398	7%	11162
19	ohscal	3	1001	10161	9%	11162
20	ohscal	4	864	10298	8%	11162
21	ohscal	5	1621	9541	15%	11162
22	ohscal	6	1037	10125	9%	11162
23	ohscal	7	1297	9865	12%	11162
24	ohscal	8	1450	9712	13%	11162
25	ohscal	9	1260	9902	11%	11162

where positives are the majority, one may merely reverse the naming of positive and negative to draw insight from this study. Third, when positives are <10% prevalent, class imbalance is a greater problem for traditional classifiers, and yet we show that quantifiers can operate quite successfully in this region. Finally, the labor savings of quantification by machine learning compared with traditional manual labeling is greater under class imbalance. As imbalance increases, the number of items that need to be manually labeled increases greatly in order to maintain tight confidence intervals on the final estimate.

To vary the *testing* conditions, we select positives and negatives from the remaining benchmark dataset such that the percent of test positives matches our target p. For example, for a dataset having 864 positives and 10298 negatives, we first remove 100 positives and 1000 negatives for training (subsetting these for the varied training situations), leaving 764 positives and 9298 negatives. When targeting p=20% positive, we test with all 764 positives and a random subset of 3056 negatives. When targeting p=1%, we use a random subset of 94 positives against all 9298 negatives. At the extremes, this can lead to having only a few training cases in one class or the other, and it requires large datasets for reasonable experimentation. The business datasets we work with at Hewlett-Packard often have >100,000 cases to quantify, but they are not publishable for research.

The benchmark text classification tasks are drawn from OHSUMED abstracts (ohscal), Los Angeles Times articles of 1989 and 1990 (la), and the Foreign Broadcast Information Service (fbis) (Han and Karypis, 2000). The feature vectors are publicly available for download from the Journal of Machine Learning Research (Forman, 2003). For this suite of experiments, we consider the binary classification tasks of one class versus all others. Some of the fbis classes had too few labeled positives to suit our needs, but we still use them as negatives for the other classes. Table 1 above shows the number of positive and negative cases available for each binary task. The natural percentage of positives averages 14% and the central two quartiles range from 9% to 16%, which corroborates our earlier discussion about the importance of studying quantification for < 20% positives. As an indicator of the difficulty of these text classification problems, SVM classifiers average in the mid-70's with respect to F-measure, the harmonic mean of precision and recall.

A natural error metric for quantification is the estimated percent positives minus the actual percent positives. By averaging across benchmark tasks, we can determine whether a method has a positive or negative bias. But even a method that guesses five percentage points too high or too low equally often will have zero bias. For this reason, *absolute* error is also a useful measure. But it is unsatisfactory in this way: estimating 41% when the ground truth is 45% is not nearly as 'bad' as estimating 1% when the ground truth is 5%. Hence, cross-entropy is often used as an error measure. To be able to average across different test class distributions, however, it needs to be normalized so that a perfect estimate yields zero error. We use normalized cross-entropy, defined as:

$$\text{normCE}(p,q) = \text{CE}(p,q) - \text{CE}(p,p)$$
$$\text{CE}(p,q) = -p \log_2(q) - (1-p) \log_2(1-q) \qquad (2)$$

where q is the estimate of the actual percent positives p in testing. Cross-entropy goes to infinity as q goes to 0%. When this happens in an experiment, it would overwhelm the average, yielding no useful data. Instead, we back off of zero by half a count. That is, we

Table 2. Summary of quantification methods evaluated

Method	Training	Testing
CC: Classify & Count	Train a binary classifier BC with all P positives and N negatives	$p' = \frac{1}{\lvert test \rvert} \sum_{x \in test} BC(x)$, the proportion of the test set on which BC outputs positive
AC: Adjusted Count	Also perform 50-fold cross-validation, estimating *tpr* = TP / P and *fpr* = FP / N	$p'' = \frac{p' - fpr}{tpr - fpr}$ Output 0 if p'' < 0, 1 if p'' > 1, p'' otherwise
T50: *tpr* = 50	Determine *tpr* and *fpr* for each potential decision threshold Set the decision threshold of BC where *tpr* = 50%	Same as AC
X: *tpr* crosses *fpr*	Same as T50, but set the threshold where (1 – *tpr*) = *fpr*	Same as AC
Max: max *tpr* - *fpr*	Same as T50, but set the threshold where *tpr* – *fpr* is maximized	Same as AC
MS: Median Sweep	Determine tpr_t and fpr_t for each threshold *t* and record the values where $tpr_t - fpr_t > ¼$	Same as AC, but compute p'' for each recorded threshold *t* Output the median p'' estimate
MM: Mixture Model	Train a binary classifier BC that outputs a real-valued score. Perform 50-fold cross-validation, recording the distribution of scores for the positives D_+ and for the negatives D_-	Apply BC to test set, saving score distribution D_U Output p' for which the PP-Area metric is minimized between D_U and the mixture distribution $p' \cdot D_+ + (1 - p') \cdot D_-$

substitute q=0.5/(size of test set). We avoid the same problem with q=100% likewise. Matching our intuition, this back-off will increasingly penalize a method for estimating zero positives for larger test set sizes. It is worse to mistakenly estimate zero positives among thousands of test cases than among ten.

The Adjusted Count variants and the Mixture Model all require cross-validation on the training set to generate the distribution of scores for positives and negatives, used to characterize *tpr* and *fpr* at each threshold. We chose 50-fold stratified cross-validation. Although this computational cost may seem undesirable, in practice we usually find that computer time is inexpensive compared to the employee time to label training cases and compared to the overall business benefit of quantification. If very large training sets were available, one could instead split the training data and use the held-out portion to determine *tpr* and *fpr*.

The various quantification methods we evaluated are summarized in Table 2. As the base classifier, we use the linear Support Vector Machine (SVM) implementation provided by the WEKA library v3.4 (Witten and Frank, 2005). We repeated the experiments using the multinomial Naïve Bayes classifier, which is also used regularly in text classification. However, we do not present its results because every quantification method under every condition performed substantially better with SVM. It is well established that SVM usually obtains better text classification accuracy than Naïve Bayes, but this finding further suggests its *tpr* and *fpr* characteristics might be more stable as well.

3.2 Results of the Quantification Experiments

Table 3 summarizes the dimensions that we varied in our study of the quantification methods. Given its high dimensional nature, we break down the results into sections where we hold some conditions constant as we vary others. For each figure, we will have a pair of graphs varying the degree of class imbalance in the training set: N=100 negatives and N=1000 negatives. We take care that the y-axis range is identical in each pair for easy comparison. Every data point represents an average performance over 250 evaluations: 25 benchmark text classification tasks times 10 random splits.

Table 3. Parameters varied in the quantification experiments

P = 10...100	Positives in training set
N = 100...1000	Negatives in training set
p = 1...95%	Percent positives in test set
Benchmark =	25 binary text classification tasks, x 10 splits
Learning Algorithms:	
SVM	linear Support Vector Machine
NB	multinomial Naive Bayes
Performance Metrics:	
Abs.Err	|estimated p – actual p|
Bias	estimated p – actual p
CE	normalized Cross-Entropy

3.2.1 Stability as the Class Distribution of the Training Data is Varied

We begin by examining how resilient the various quantification methods are to wide variations in the training set, while we hold the *test* conditions fixed at p=5% positive. Figure 5 shows the average error of the quantification estimates, as measured by absolute error from the 5% test target. The x-axis varies the number of positive training cases P=10…100. Overall we see the Median Sweep (MS) method dominates, giving consistently good results even under great variations in the training set. Note the absolute scale: on average it estimated within two percentage points given only P=30 positives & N=100 negatives (left), and within one percent given P=30 & N=1000 (right). Because the curve is relatively flat after P=40, there is little benefit to adding additional positives to the training set. This property can be leveraged for significant labor savings in building up training sets.

By contrast, we see the Classify & Count (CC) method is unstable and often gives very poor estimates. There is a small region (far left) where CC happened to achieve the least error, and it also appears to be competitive when given a large enough training set (far right), but this is illusory. As we demonstrate later, deeper examination reveals that it simply underestimates for smaller P and overestimates for greater P, as we might expect from our earlier analysis. Having more training positives simply yields more positive predictions on the test set, and at some point it happens to be tuned with our current test prevalence.

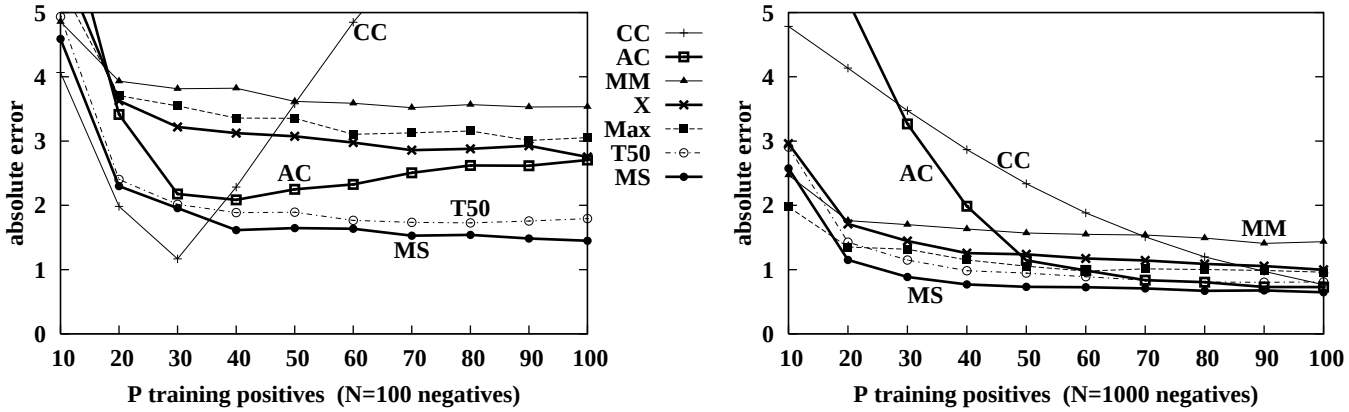

Figure 5. Absolute error for test target p=5% positives only.

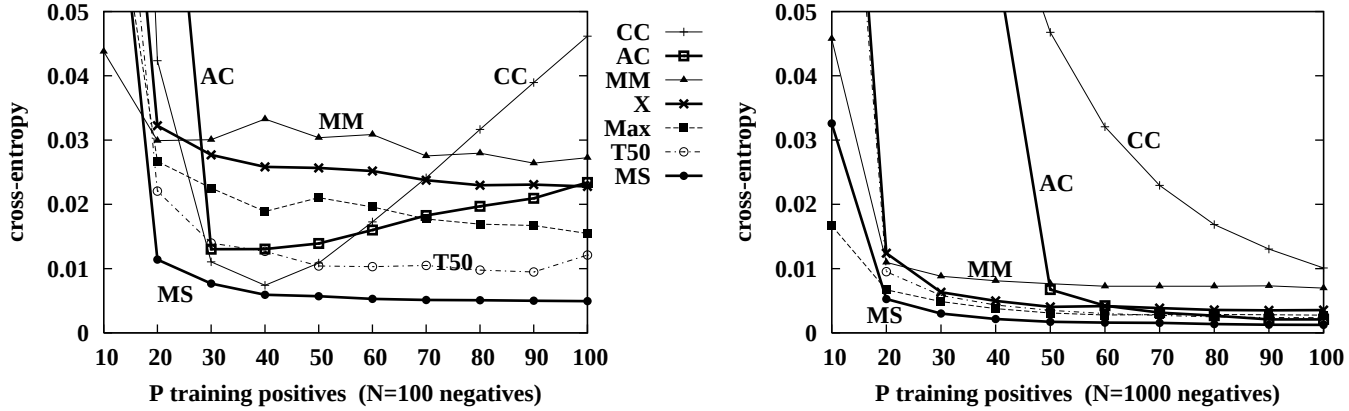

Figure 6. Cross-entropy error for test targets p=1…20% averaged.

The analysis so far is for a single target of p=5% test positives. To draw more general conclusions, in Figure 6 we average the performance over the range p=1…20%. To average over different values of p requires that we use normalized cross-entropy given in equation (2), as discussed in the methodology section. Although the y-axis scale is different and less interpretable than the absolute error shown in Figure 5, the curve shapes and rankings are qualitatively similar. Median Sweep continues to dominate. Its superior performance is statistically significant in both graphs. Its standard error is quite small: for P=50, it is 0.0002 for N=100 and 0.00005 for N=1000—too small to see in the graph.

As we anticipated, the basic Adjusted Count (AC) method degrades rapidly under very high class imbalance, esp. for N=1000. Here it selects a decision threshold with low *tpr,* making the denominator in equation (1) small and sensitive to variance in the *tpr* and *fpr* estimates. And we expect especially poor estimates of *tpr* when there are few positive training cases. By contrast, T50 (recall *tpr*=50%) mitigates the problem with a small denominator and avoids the tails of the *tpr* curve. We find its performance is nearly competitive with Median Sweep. We return to this point in the discussion in Section 3.3.

Our prior work observed the remarkable stability of the Mixture Model (MM) in comparison to baselines AC and CC, even under great training imbalance (Forman, 2005). However, with the broader suite of methods being compared here, we see that MM is consistently dominated by other methods. These sorts of results would be impossible to predict analytically, partly because the MM method operates so differently from the other methods.

3.2.2 Accuracy over a Wide Range of Class Distributions in the Test Set

Next we vary the percentage of positives p in the test set, holding the training set fixed. In each figure beginning with Figure 7, its first graph fixes the training set at P=100 and N=100, while its second graph fixes it at P=100 and N=1000 (9% positive). Since we found that Median Sweep had the best performance over a wide range, we expanded the study up to 95% test positives to see what happens with extreme differences in the class distribution between training and testing. Even so, in order to keep the focus on the important lower p region, we use a logarithmic x-axis in the figures. Since we are not averaging results across different values of p, the y-axis again shows the absolute error, which is more intuitive than cross-entropy.

In the low p range, Median Sweep dominates other methods, especially with a balanced training set. Its absolute error climbs for the high p range, where instead the Max and X methods excel consistently. (But in the analysis of bias below we shall see that the Max method suffers from a strong systematic bias, much like CC. Thus, the X method performed better overall for high p.)

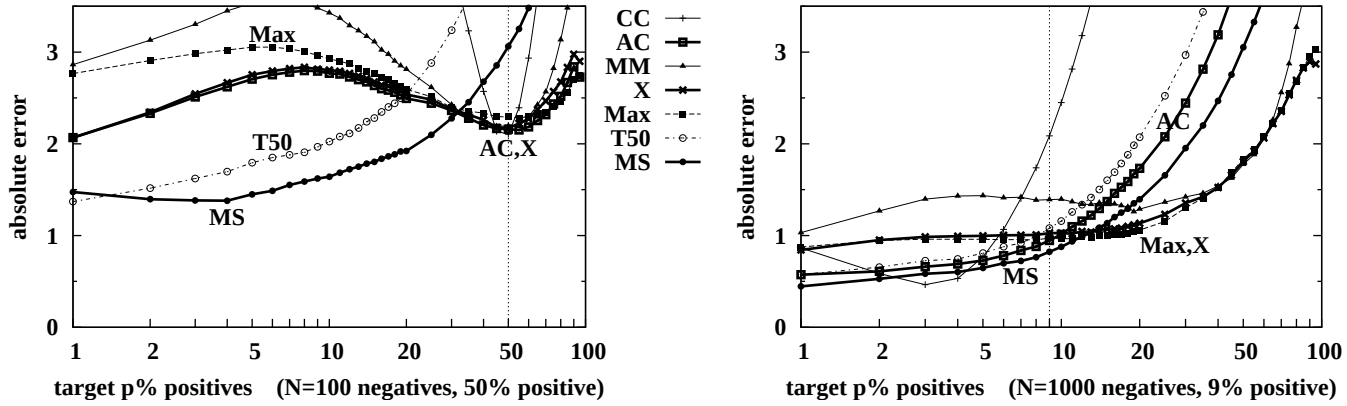

Figure 7. Absolute error for targets p=1…95% individually. P=100 training positives. The vertical line marks where the test class distribution matches the training class distribution.

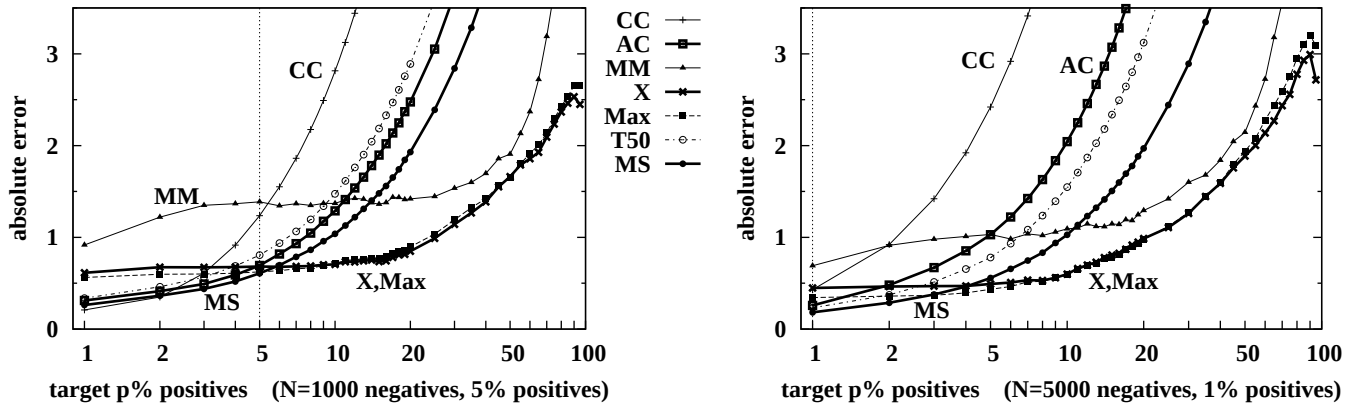

Figure 8. Same as Figure 7, but on 17 classes of a held-out dataset allowing greater training class imbalance. P=50 positives.

Given the dominance of the Median Sweep method in the low p region of most interest, we would like to validate whether its performance continues for situations with even greater class imbalance in training (~1%), as we sometimes face in practice. This study so far has been limited to N=1000 training negatives, in order to have 25 benchmark tasks for study. Although we could increase class imbalance by reducing P, small samples result in degenerate classifiers. Instead, we require a greater number of negatives. In addition, we want to validate these results against other classification problems. For these two purposes, we use a separate dataset: new3 from Han and Karypis (2000). It has 9558 text cases partitioned into 44 classes. We repeated our study on 17 of its classes that have at least 200 positives, setting aside 5000 negatives for training. Whereas the two graphs of Figure 7 present results for 50% and 9% training positives, those of Figure 8 present results on the new dataset for 5% and 1% (P=50, N=5000). Considering all the curves, the gestalt is much like Figure 7. This validates that the results are not particular to the individual text classification problems in the benchmark. And it is encouraging that the earlier findings generalize to greater training imbalance. The Median Sweep method continues to estimate well for low p: it has absolute error < 1% for $p \leq 10\%$. The Max and X methods generally become more competitive in the low p region with the increased training imbalance. Finally, observe that the MM method continues to be remarkably stable through a wide range of training and testing situations, though it is dominated by newer methods. Hereafter we return to the previous benchmark datasets in Table 1.

Interestingly, observe in Figure 7 that the curves of the competitive methods can be clustered into two shapes: concave upward (MS,T50) and S-curve (Max,X,MM). The AC method under balanced training (left) belongs to the S-curve group, whereas under high imbalance (right, and all of Figure 8) it belongs to the concave upward group. As discussed previously, the AC method trained under high class imbalance uses thresholds with low *tpr*, i.e. closer to T50, which is in the concave upward group. But under more balanced conditions, AC prefers thresholds with high *tpr*, closer to the X crossover point in Figure 2, which results in its S-curve grouping. Looking now at MS, its grouping with T50 suggests the median may come from estimates derived from *tpr* rates nearer to 50% than near the cross-over point X.

3.2.3 Bias & Failure Analysis

Next we analyze the bias component of accuracy by measuring the average *signed* error for each method. A perfect method would have zero bias across a broad range of conditions. Figure 9 shows the bias under varied training (left vs. right) and testing conditions (p% positives on x-axis). We abandon the log-scale here in order to show the strongly linear bias of two methods: Max and Classify & Count. For the classifier trained with 50% positives (P=N=100, at left), the CC method progressively overestimates when p<50%, and underestimates when p>50%, as expected. When trained with 9% positives (P=100, N=1000, at right), this balance point is shifted accordingly, but not proportionately—it is unbiased only at p=3% instead of 9%. This is known behavior. SVM exaggerates the training class imbalance in testing even when the test class distribution matches. Although the frequent suggestion is to bias the SVM cost penalty or complexity parameters, it generally proves ineffective and has been better addressed recently by Wu and Chang (2005). For more general classifier models, one can set the decision threshold of a classifier via ROC analysis so that it matches the class distribution of the *training* set (Lachiche and Flach, 2003). However, such adjustments do not avail quantification.

It is surprising that the Max method, being an Adjusted Count variant, also exhibits a linear bias, albeit to a lesser degree than CC. This means that it consistently finds thresholds such that the *tpr* and *fpr* characterization is inaccurate and biased. For example, if it selects a threshold that shows *tpr*=1 in cross-validation on the training set—the extreme of the tail—it may nevertheless produce some false negatives on the test set, yielding lower recall than estimated. Most of the other methods have a relatively stable bias over a wide range, mostly positive and less than 1%.

In all cases, we see greatly increasing bias at the tails, which is expected. If a method's estimates vary by a few percent and p is close to zero, any infeasible negative estimates are clipped to 0%, resulting in a positive bias, and likewise as we approach p=95% positives. To provide test samples with 1% positives we end up providing small samples of positives, which leads to greater variance in the experiments. This compounds with the bias effect of clipping.

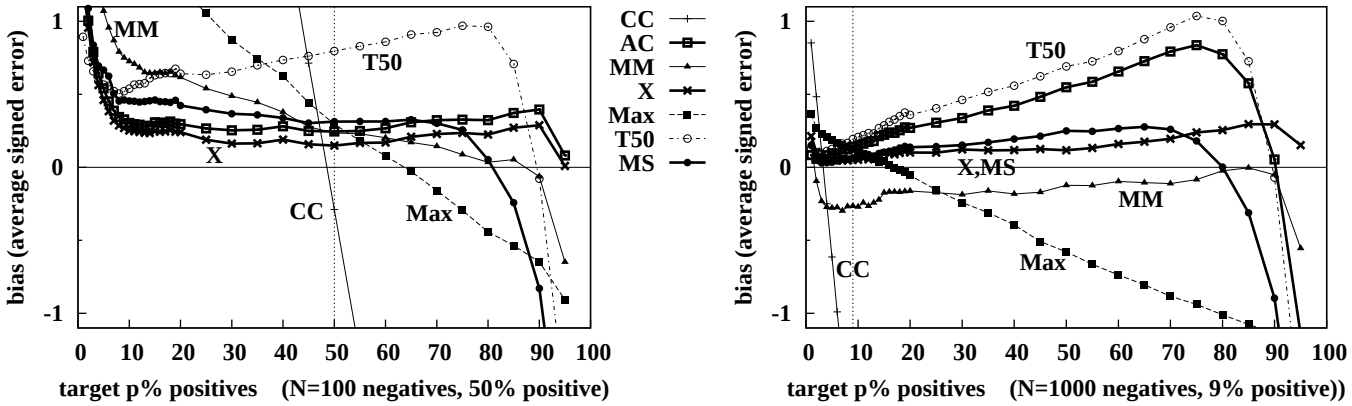

Figure 9. Bias. Average signed error for targets p=1..95% individually. P=100 training positives. The vertical dotted line marks where the test class distribution matches the training class distribution.

Finally, we consider failures. Although an induced classifier should learn to separate cases well enough that its true positive rate *tpr* is greater than its false positive rate *fpr*, they nonetheless fail sometimes. This usually happens under great class imbalance in training. For example, for one of the ten splits on one of the tasks trained with P=10 and N=100, the induced classifier's natural threshold gave *tpr=fpr=0.* It learned to classify everything as negative, which results in a troublesome zero denominator in the adjusted count method. The commonness of this problem was partly the impetus for this research: *tpr* was less than or equal to *fpr* in 623 of 10,000 trial training sets for AC. In most of these, *tpr* and *fpr* were both zero, but in a handful of cases *tpr=0* and *fpr* was slightly greater than zero, e.g. *fpr*=0.002, training on P=20 and N=1000. In progressively decreasing occurrence of failure, we have: AC, T50 and X. The Max method never experienced a failure, exactly because it seeks to maximize the denominator. This is not a problem for CC and the Mixture Model, which do not involve equation (1).

3.3 Discussion

One motivation for research in quantification is to reduce the training effort needed to obtain a given level of accuracy in quantification. We would like to emphasize that this research has led to methods that estimate more accurately *and* with less training data (unlike active learning research where all methods produce the same classification accuracy at the end of the learning curve). For example, observe in Figure 6 that Median Sweep performs better than the older Mixture Model and Adjusted Count methods, even with less training data. Furthermore, it is remarkable how little training data is required at minimum. Median Sweep performs decently with as few as P=20 positives and N=100 negatives for training. By contrast, supplying more training data does not lead to better estimates by the simple Count & Classify method.

The reason that Median Sweep works so well is that instead of relying on the accuracy of the *tpr* and *fpr* estimate at a single threshold, it uses information from all the thresholds. In some sense, it has the advantage of bootstrapping, without the heavier computational cost. True bootstrapping for this application would involve repeating the 50-fold cross-validation for many different random samplings of the available training set. That said, if it turns out that bootstrapping can yield significantly better estimates, then the computer time may easily be worthwhile for many business and scientific applications.

Unfortunately, the Median Sweep method does involve some programming complexity, as well as storing the complete *tpr* and *fpr* curves with the quantifier. For these reasons, some programmers may prefer the T50 method, which is simple and performed decently for the low p range, albeit with more bias than some methods.

In cases where an application calls for tracking targets around 50% prevalence, one may wish to use the X method—or the simple AC method if the training set is sure to be balanced and unlikely to stray far from 50%. If the target prevalence is very high, then one may simply reverse the naming of positive and negative and use the Median Sweep method with a training set roughly matching the target distribution.

Finally, although we are pleased to have reduced the absolute error of the estimate to less than 1% in many situations, some applications need to quantify much rarer events, where the bias and the relative error both grow. To conduct experiments in the tail of the distribution requires much larger labeled datasets available for research, and they must also have very few errors in the class labels in order to draw correct conclusions.

4. Cost Quantification Methods

Estimating the *number* of cases belonging to a class may not accurately reflect its importance. A relatively uncommon class having a very high cost can be more important overall than a frequent class with low cost. In cost quantification applications, each case is tagged with a cost attribute, e.g. the cost of parts and labor to resolve the case. It can sometimes happen that the prevalence of the positive class stays constant over time, but the cost attribute climbs substantially, e.g. when suppliers increase the cost of the needed parts. In either of these cases, *cost quantification* is the appropriate task, not just quantification of the prevalence

Let C^+ stand for the unknown average cost per positive case in the test set. If one knows a historical value for C^+ and it is believed to be unchanged in the test set, then it can simply be multiplied by the quantifier's estimate of the number of positives Q in order to estimate the total cost of positive cases. But this is unsatisfactory if the cost may change over time. Also, C^+ is commonly not known in advance, and we need to determine it by analyzing the cost attribute attached to only the positive cases.

Occasionally one's goal is to determine the average cost of positives C^+ from data, without concern for the total cost. This is needed in forecasting, for example, where we have models that predict future volume, and need to parameterize them with the average cost in order to predict future total costs. For the purposes of discussion, we will focus on determining the total cost. However, most of the methods below estimate the average cost first, and then multiply by the best estimate available from a quantifier. The experiments in Section 5.2 evaluate the methods both with respect to total cost and with respect to average cost.

4.1 Basic Methods

We describe several methods that are straightforward. With each, we also describe some of its shortcomings, which motivate the more complex methods that follow.

Classify & Total (Simple): The obvious, simple solution to cost quantification, akin to Classify & Count, is to train a binary classifier and to compute the total of the cost attribute values associated with all cases that are classified as positive, i.e. the sum

$S = \sum_{x \in test} c(x) * BC(x)$, where c(x) is the cost attribute of case x, and BC(x) is an indicator function representing the binary classifier. Unless the classifier is perfectly accurate, it will result in poor cost estimates that are systematically biased, for the reasons shown previously regarding Classify & Count. Also, false positives and false negatives no longer cancel each other out in this domain, because their cost attribute values are in general not equal.

Grossed-Up Total (GUT): The next obvious solution is to compute the total S as above, but then to adjust it upwards or downwards according to the best quantifier estimate Q available. That is, GUT estimates the total cost $S' = S \cdot Q / \sum BC(x)$. For example, if the binary classifier predicted 502 positives and the quantifier estimated Q=598.3 positives, then the cost total from the Simple method would be grossed up by the factor 598.3/502 = 119% to reflect the 19% positives that were missed by the binary classifier. But this method suffers from similar problems as AC. It runs the risk that the binary classifier may select zero or very few cases to include in the total, if positives happen to be rare in the training set *or* the test set. On the other hand, if positives were overly common in the training set, then the induced classifier will liberally include many negatives in its total, polluting the average cost. This pollution occurs even if the class distribution has not changed, as long as there are some false positives.

Conservative Average * Quantifier (CAQ): We can reduce the false-positive pollution above by setting the classifier decision threshold to be more conservative—a classic precision-recall tradeoff. Using a smaller set of predictions with high precision, we average their costs to estimate C^+, and then multiply it by Q, the estimated size of the class from a quantifier—mathematically equivalent to the GUT method. Ideally we would like a threshold with 100% precision, but sometimes there is no such threshold. Furthermore, a highly conservative threshold may predict only a few cases as positive, especially if positives are rare. Given very few positive items to average over, the uncertainty of the C^+ estimate will be large. To avoid small samples, one might set the classifier threshold to take the top, say, 100 most strongly predicted positives. But this does not ensure high precision. The classifier may only have high precision in the top 30. Or the test set may have only 50 positives.

In the experiments in Section 5.2 we evaluate two variants: **CAQ30** takes only the top 30 predicted positives, and for the purposes of a controlled experiment, we ensure there are always more than 30 positives available in the test sets. The other variant is **CAQhalf**, which takes the top half of the positives predicted by the binary classifier. Depending on the test set and the classifier, CAQhalf may sometimes take fewer than 30 predicted positives to compose its average.

4.2 Precision Correction Methods

Trying to drive the classifier to have 100% precision is an ill-suited approach. It may not be possible in many cases, and it leads to instability when few or even zero cases are predicted positive. Instead, we will attempt to characterize the imperfect precision and to correct for it, analogous to the Adjusted Count method.

Precision-Corrected Average * Quantifier (PCAQ): To correct for imperfect precision, we will require that we have an estimate of the precision of the classifier on the *test* set. Traditionally this has been done via cross-validation using the training set, but such a characterization is dependent on the class distribution. Because of the class distribution mismatch problem in our applications, estimating the classifier's precision on the *test* set is a multi-step process. First, we characterize the *distribution-independent* measures *tpr* and *fpr* of the classifier via cross-validation on the training set, as described previously. Then we estimate of the test class distribution $q \in [0,1.0]$ via the best quantifier technology available. Finally, we estimate the precision Pr of the classifier on the test set by modeling the percentage of expected true positives divided by the expected true positives and false positives:

$$\Pr = \frac{q \cdot tpr}{q \cdot tpr + (1-q) \cdot fpr} \tag{3}$$

Given this, we can now model the average cost C of cases that are predicted positive by the classifier as the weighted average

$$C = \Pr \cdot C^+ + (1 - \Pr) \cdot C^- \tag{4}$$

where C^- is the average cost of negatives. C^+ and C^- are both unknown. If only Pr were 100% precise, then C would equal C^+. We next write an equivalent weighted average for the entire test set. The average cost C_{all} of all cases in the test set is

$$C_{all} = q \cdot C^+ + (1 - q) \cdot C^- \tag{5}$$

Both C and C_{all} are easily computed directly. Given the two equations (4) and (5), and the two unknowns C^+ and C^-, we can solve for C^+ and eliminate C^- from the formula. First we solve equation (5) for C^-, and then substitute it into equation (4):

$$C = \Pr \cdot C^+ + (1 - \Pr) \cdot [\, (C_{all} - q\ C^+) / (1 - q) \,]$$

We solve this equation to obtain the precision-corrected average C+:

$$C^+ = \frac{(1-q)\,C - (1-\Pr)\,C_{all}}{\Pr - q} \tag{6}$$

Finally, to estimate the total cost, this estimate of the average is then multiplied by the quantifier's estimate Q of the number of positives.

But there remains a familiar problem. Under high training class imbalance, the classifier's default decision threshold may again select only a few cases to include in the average C. Furthermore, if the classifier's threshold is highly conservative, then the estimation of the true positive rate *tpr* may depend on the cross-validation results of very few of the training positives. That is, it may be an unreliable estimate.

In perfect analogy to the first part of this paper, we solve this problem by selecting a less conservative classification threshold, which has worse precision, but has more stable characterization of its precision, as well as providing a greater number of predicted positives over which to compute the average C. This works because under PCAQ we can adjust for the false-positive pollution of lower precision.

The remaining design decision is which threshold policy to use—for example, the T50 or X thresholds shown in Figure 2. T50 uses the threshold where *tpr*=50%, which avoids the specific concern mentioned above about unreliable *tpr* estimates. We suggest not using the Max threshold, given our earlier bias discussion about its choosing thresholds having poor *tpr* and *fpr* characterization. For the experimental section ahead we selected the X threshold (where $fpr \approx 1\text{-}tpr$, the false negative rate), given its good experimental performance for quantification.

Median Sweep of PCAQ (mPCAQ): Rather than use a single threshold and hope that its precision characterization is reliable, we can instead sweep over many classifier decision thresholds, and return the median of the many PCAQ estimates for C^+. The variables above that depend on the threshold t are tpr_t, fpr_t, Pr_t, and C_t, the average cost of cases predicted positive up to the threshold t. The others remain scalars: C, C_{all}, and q.

As with Median Sweep earlier, we expect this to have similar benefits as bootstrapping. Furthermore, just as the MS method excludes estimates that are likely to have high variance, a potential variant on this method might exclude estimates from thresholds where (a) the number of predicted positives falls below some minimum, e.g. 30, (b) the confidence interval of the estimated C^+ is overly wide, and/or (c) the precision estimate P_t was calculated from fewer than, say, 30 training cases predicted positive in cross-validation. For our experiments below, we used the median estimate of all thresholds.

Mixture Model Average * Quantifier (MMAQ): Finally, rather than try to determine an estimate at each threshold, we can model the shape of the C_t curve over all thresholds as the mixture model in equation (4), and solve for C^+ and C^- using linear algebra. Specifically, if we re-write equation (4) in the form $y = m \cdot x + b$:

$$\frac{C_t}{Pr_t} = C^- \cdot \left(\frac{(1 - P_t)}{P_t} \right) + C^+ \tag{7}$$

then we can solve for C^+ (the y-intercept) using basic linear regression over the many (x_t,y_t) data points.

The same thresholds omitted by Median Sweep could be omitted here as well, in order to eliminate some outliers that may have a strong effect on the linear regression. Alternately, one may use regression techniques that are less sensitive to outliers, e.g. those that optimize for L1-norm instead of mean squared error. We used neither of these extensions in the experimental evaluation.

4.3 Non-Solution: Regress and Sum

An alternative method that suggests itself for cost quantification involves regression rather than classification. One would train a (calibrated) regression model for the cost attribute c(x), with the cost of negative training cases set to zero. This trained regressor would then be applied to each test item, and the total cost estimate is the sum of these regression estimates. This solution, however, is ill-conceived, in analogy to the method of quantifying with a calibrated classifier discussed in Section 2.4. Unless the regressor is able to distinguish positives and negatives perfectly, its output reflects the uncertainty between the two classes and their relative proportion in the training set. A given output reflects the average cost of mixed training positives and negatives that have "similar" feature vectors. If the proportion of positives changes in the test set, the regressor is no longer calibrated. Moreover, if the average cost of a positive C^+ rises drastically in testing, the regressor will be blind to this—just as a classifier must not use the class label field in testing. We exclude this method from further consideration.

5. Empirical Evaluation of Cost Quantification Methods

The next logical step is to evaluate all these cost quantifier methods against one another on a large benchmark. Any such empirical experiment depends strongly on the distribution of the cost attributes for positives versus negatives, in addition to the factors that affect basic count quantification. For this, we would like to have a large publishable benchmark with real-world cost values of reasonable interest to a family of applications. This is an open invitation to the scientific and business communities.

In this section we begin by describing our methodology, including how we create a suitable dataset in lieu of a publically available dataset including appropriate cost attributes. The second subsection details the results of the experiments, and the third subsection contains a discussion of these results at a higher level, and describes practical issues in their application.

5.1 Experiment Methodology

We supply artificial cost attributes to the cases of our text classification benchmark. In order to limit the dimensionality of the experiments and simplify the interpretation of results, we set $C^+ = 1.0$ and $C^- = 0.5$. These being constants, the variance of the results is reduced somewhat, helping to expose the consistent differences between methods.

The experiment methodology proceeds much as before, but with the addition of the cost attributes. For each of the 25 binary text classification tasks x 10 random splits, we select a limited training set of P=10…100 positives and N=100…1000 negatives. We vary the testing percent positives p=1%...45% by removing as few positives or negatives from the remaining cases. (We tried to expand out to p=1…95% to match the quantification study above. But when training with low prevalence and then testing with high prevalence, we encountered stability problems with the precision-corrected methods, because the denominator in equation (6) became zero or negative with large q estimates. We choose to simply avoid this region, since the most interesting region for business purposes tends to be below 20% anyway.)

For each of these train and test pairs, each cost quantification method is run. For these experiments we used the same base learner as before, a linear SVM, and 50-fold stratified cross-validation to estimate tpr_t and fpr_t at each threshold t. Most of the cost quantification methods also rely on a quantifier subroutine to estimate the class distribution. For this we used Median Sweep (MS) solely, since it performed well in the empirical evaluation. We also repeated the experiment using an Oracle quantifier that always supplies the true test distribution q. This gives an upper bound on how much better these cost quantifier methods might do with future advances in quantification methods to estimate the counts.

Table 4 summarizes the methods we evaluated. We compare them based on (a) their total cost estimate, and (b) their estimate of the average cost C^+ of positives. In order to average together different benchmark runs with different total costs, we first normalize the estimate by dividing by the ground-truth total cost. For example, 0.95 represents an under-estimate of the total cost by 5%. To evaluate the bias of a method, we use the average of the signed estimates. While useful, it does not give the whole picture. Zero bias may also be achieved by a method that equally often over-estimates and under-estimates by 5%. We would prefer a method with the same perfect bias and less variance. Given so many methods and experimental conditions, it becomes quite difficult to present the standard deviations of all these averages. Instead, we opt to show a much more readable representation: graphs of their average absolute errors, which detect both bias and variance problems. Note that we do not recommend *mean squared error*, for it is highly sensitive to outliers among the estimates. We would prefer to know which method(s) work well most of the time, than to rule out any method that occasionally makes a very poor estimate.

Table 4. Summary of cost quantification methods evaluated

Method	Training	Testing
Simple: Classify & Total	Train a binary classifier BC with all P positives and N negatives	Output S = $\sum_{x \in T} c(x)$, where T is the set of all test cases for which BC predicts positive
GUT: Grossed-Up Total	Also train a quantifier Q(·)	Output S · Q(test set) / |T|
CAQhalf: Conservative-Average · Q using top half	Same as GUT	Output C^+ · Q(test set), estimating the average cost as $C^+ = \frac{1}{\|T'\|} \sum_{x \in T'} c(x)$ where T' contains half the cases of T for which BC predicts more strongly positive
CAQ30: Conservative-Average · Q using top 30	Same as GUT	Same as CAQhalf, but where T' contains only the top 30 test cases according to BC
PCAQ: Precision-Corrected Avg. · Q	Also perform 50-fold cross-validation, estimating *tpr* = TP / P and *fpr* = FP / N	Output C^+ · Q(test set), estimating C^+ by equation (6)
mPCAQ: Median Sweep PCAQ	Same as PCAQ, but estimate tpr_t and fpr_t at each occurring classifier threshold *t*	Estimate via PCAQ at each threshold *t*, then output the median
MMAQ: Mixture Model Average · Q	Same as mPCAQ	Output C^+ · Q(test set), estimating C^+ via linear regression on the values in equation (7)

5.2 Cost Quantification Experiment Results

There are multiple views to consider in such a high-dimensional experiment. We break down the results into sections where we hold some conditions constant as we vary others. As before, each figure has a pair of graphs with matching y-axes: N=100 training negatives on the left, and N=1000 training negatives on the right, representing greater imbalance. Every data point represents an average performance over the 25 benchmark text classification tasks times 10 random splits.

5.2.1 Stability as the Class Distribution of the Training Data is Varied

We begin by holding the percentage of positives in the target set fixed at p=10% and vary the number of positive training cases from P=10 to 100. Hence, the percentage of training positives varies from 9% to 50% with N=100 (Figure 10 left), and from 1% to 9% with N=1000 (right). The y-axis shows the total cost, after normalizing to 1.0, averaged over the 250 benchmark runs. This reveals the bias of each method. The ideal method would have a score near 1.0, and be as flat as possible across the whole range, even though the training set varies substantially. Even if this is not possible, we would at least like the performance to consistently converge near 1.0 as we add more positive training examples.

Three of the methods did not converge as we added training positives: Simple, GUT (Grossed-Up Total) and CAQhalf (Conservative Average). This is an undesirable property and nearly rules them out for real-world use. Each of these methods is designed with an assumption of high precision of the classifier, yet as we add more training positives, they accept more false positives. For the Simple method, adding false positives increases the total cost estimate. With the imbalanced training set (right), Simple is far below the chart and climbs with increasing P. For GUT and CAQhalf, admitting more false positives into the average brings their total estimates down. Each negative has half the cost of a positive (C^+=1, C^-=0.5), so their average cost estimates plummet. By contrast, CAQ30 does not accept more false positives, since it only takes the top 30 positives it can find in the test set.

The methods that use the precision-corrected average (PCAQ, mPCAQ, MMAQ) performed similarly. They generally over-estimated the cost slightly, and they tended toward less bias as more training positives are given. These methods, rather than depending on high precision classification, depend on stable *estimates* of their precision. Having more training positives helps establish a more confident estimate of *tpr*. Errors from the underlying quantifier damage the estimate of precision within these methods, as well as affecting the total cost after multiplying by Q. For an "upper" bound comparison, we replaced the fallible quantification estimates of MS by an Oracle.

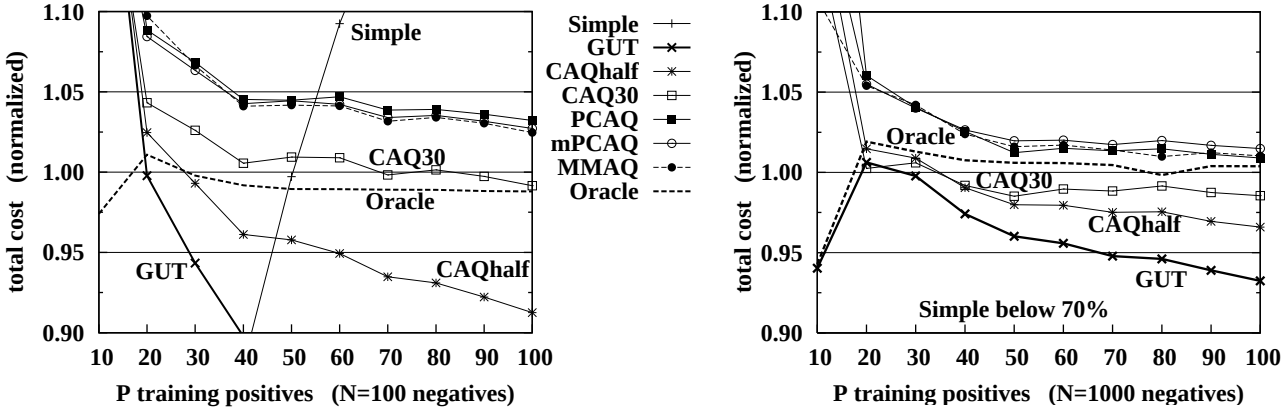

Figure 10. Total cost estimate, normalized to 1.0, as we vary the training set. The test set has p=10% positives.

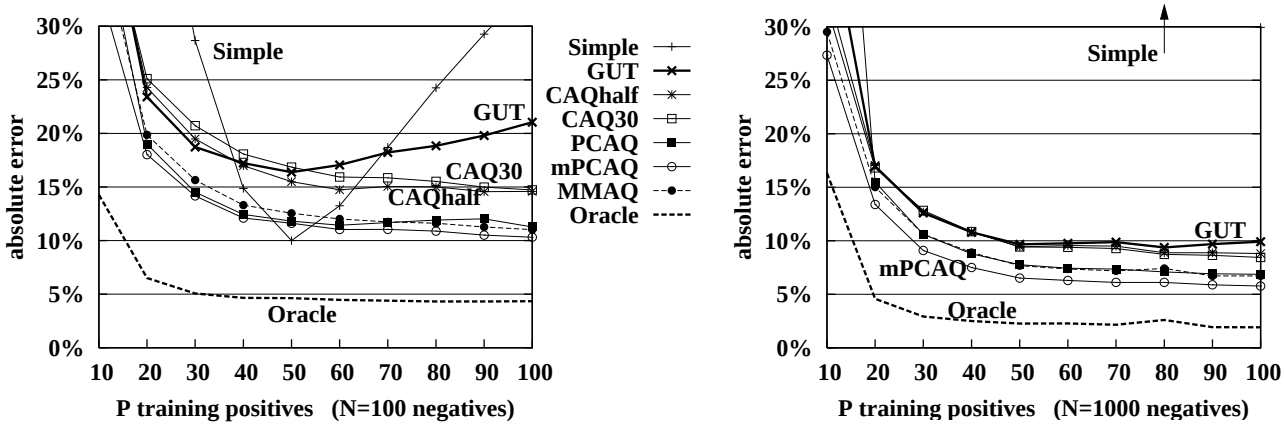

Figure 11. Same as previous figure, but the y-axis is now the average absolute error of the normalized cost.

Rather than clutter the display with all the results, we only show the Oracle for the Mixture Model Average method (MMAQ, shown with a dotted line), which tended to perform among the best in estimating the average. It converges to within 1% of a perfect score on the left in Figure 10 and within 0.5% on the right. Even without the input from the Oracle quantifier, MMAQ averaged within ~3% (left) and ~1% (right).

So far we have only examined bias. To get a perspective on both bias and variance together, we graph the absolute error averaged over the 250 runs in Figure 11, which is otherwise like the previous figure. Most striking is how poorly the Simple method performed versus all the others; it is unstable and off the chart at the right. All four methods that do not compensate for imperfect classifier precision exhibited the worst absolute error: Simple, GUT, CAQhalf, and CAQ30. Although CAQ30 showed very low bias in the previous figure, here it reveals poor absolute error. Thus, it exhibited substantial variance centered on the true cost. Having large variance is expected with its approach of depending on just a few items for its average.

The precision-corrected methods form a tight cluster. The Median Sweep idea emerged as a consistent winner, just as it did for count quantification: mPCAQ showed a slight improvement over the others. Given just P=50 positives, it achieved within 12% of the true total cost (left) and within 6% (right). Even so, we see a substantial margin between all these methods and the Oracle-MMAQ performance, which achieved ~4% average absolute error on the left and ~2% on the right. This indicates that improvements in technology for quantification could substantially improve cost quantification as well.

5.2.2 Accuracy over a Wide Range of Class Distributions in the Test Set

In this section we fix the number of training positives at P=50 and vary the percentage p of positives in the test set along the x-axis. This naturally varies the total cost for each test set, but we normalize each to 1.0 for easier evaluation and proper averaging. Figure 12 shows the average absolute error on the y-axis. The left has 33% positives in training (N=100) and the right has 5% (N=1000). Again, most striking is how poorly and erratically the Simple method performed. With N=1000 its very conservative decision threshold made it badly underestimate the total cost, putting it far off the chart.

We see the precision-corrected methods in a tight cluster, dominating the conservative average methods. Once again, the median sweep method mPCAQ outperformed the others by a small amount over the entire range, especially with N=1000. Nonetheless, there is still a substantial performance gap to the Oracle method, which simply corrects the quantifier estimate for MMAQ. Also to note, all methods do much worse as the percentage p of testing positives gets small, even with the Oracle. At p=1%, some of the smaller datasets lead to test situations that have only a dozen positives to count. This creates very difficult "hit-or-miss" situations, depending strongly on a stable characterization of the classifier. This is, nonetheless, a useful region for many real-world purposes, so we include it in the x-axis range partly to inspire future research.

Next we turn from estimating the total cost of positives to estimating the *average* cost of positives, C^+. Why study this? First, some applications seek the average instead of the total, e.g. when the total population of the target is not known. Second, it gives a perspective on how well the methods estimate the average before they multiply by the quantifier's count Q. This analysis factors out the uncertainty contributed by the quantifier for most methods, but not all. The methods that estimate their precision via equation (3) still require a quantifier. We continue to provide the Median Sweep (MS) method as the quantifier subroutine. Thus, its quantification errors will continue to affect the PCAQ, mPCAQ, and MMAQ methods.

Figure 13 shows the average absolute error for estimates of C^+. In comparison with the previous figure, a major difference is that the precision-corrected methods no longer dominate the other methods. Another difference is that the Simple method is far less erratic. Indeed, although the Simple method is dominated by more advanced methods, note that it becomes competitive to the right in the right-hand graph. In this region, where it has been trained to be conservative and yet test positives are abundant to select from, the classifier yields high precision and hence the correct average. This high precision comes at the cost of low recall, but since C^+ is a constant for these experiments and here we are only concerned with the average cost, low recall does not harm the estimate. This analysis explains why CAQ30 appears to be the dominant method for estimating average cost—its top thirty positives tend to be true positives.

A further observation can be made from Figure 13. The line labeled "Oracle" is the MMAQ method but with the ground truth quantification used in estimating the precision correction. We see the Oracle and MMAQ curves converge at p=45% test positives. Here the Oracle quantifications provide no benefit over the basic MMAQ method. But as p becomes small, we see a growing and substantial performance gap. By this comparison, we see what proportion of the error might be eliminated by better count quantification technology alone.

In an analysis of bias (not shown), the Simple, GUT and both the CAQ methods tended to substantially under-estimate the average, which is reflective of admitting some false positives into the average—in this experiment, false positives have half the cost of true positives. Despite this bias, the CAQ30 method appears to dominate consistently by having very low variance. Unfortunately, this is somewhat an artifact of the unusual testing conditions, as discussed next.

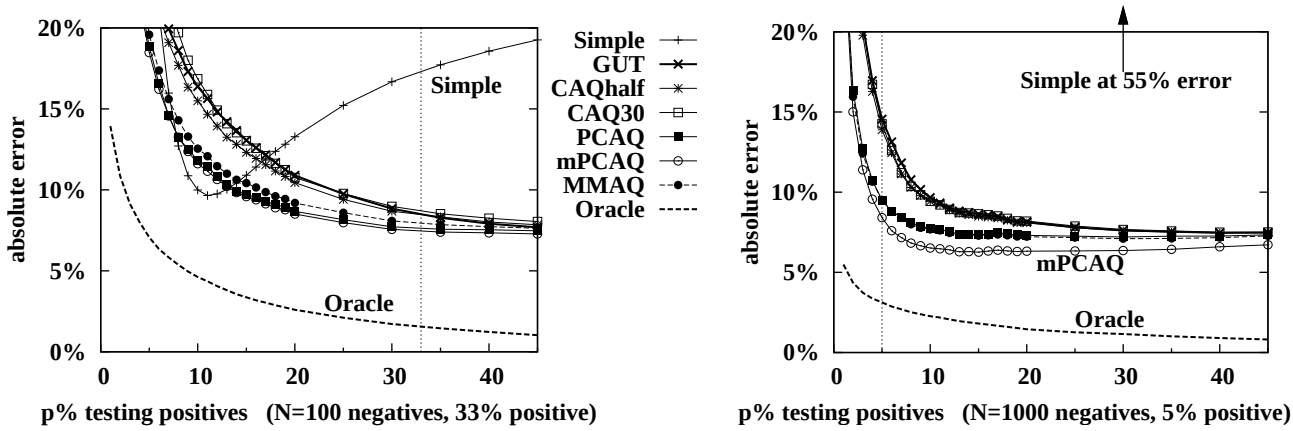

Figure 12. Absolute error as we vary the percentage of positives in testing p=1..45%. (P=50) The vertical line marks where the test class distribution matches the training class distribution.

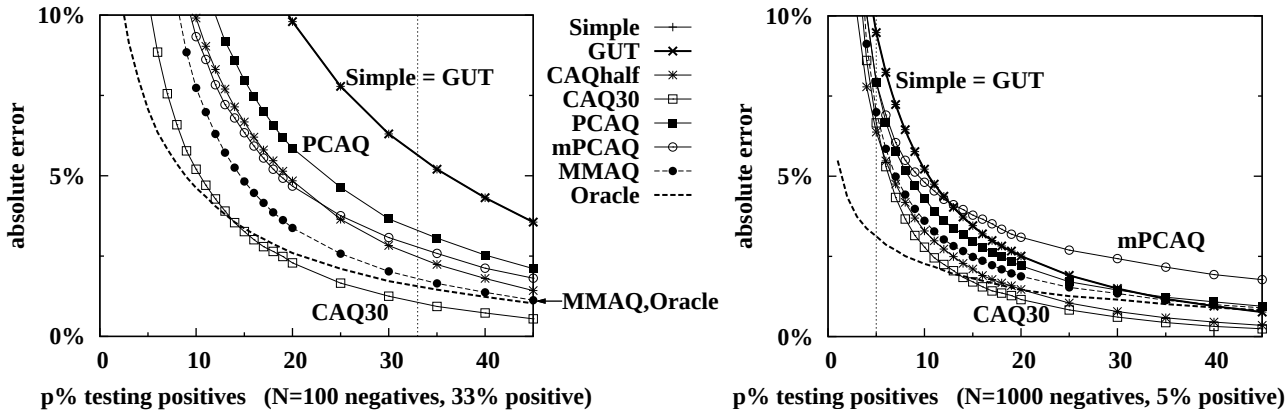

Figure 13. Same as previous figure, but for the task of quantifying the *average cost* C+ of positives in the test set. (P=50)

5.3 Discussion

Stepping back from the many detailed results of the experiments, we generally have succeeded in producing methods that are much more stable for cost quantification than the baseline method Simple. That said, the complexity of some of the methods does not appear particularly worthwhile, since the uncomplicated CAQ30 method performs well. But because the design of our experiment had no variance in the cost of positives ($C^+ = 1.0$), CAQ30 faced no disadvantage by having only 30 cases from which to determine its average cost. In real-world settings with substantial variance and perhaps errors in the cost attributes, averaging only 30 items can give a relatively uncertain estimate for C^+. Thus, we hypothesize a practical advantage for methods that consider the cost attributes of many more cases, such as mPCAQ and MMAQ. We suggest this vein of study for future work. It could be performed with large, realistic datasets, or else by supplementing labeled datasets with generated costs as we have done, but also systematically varying the means and variances of positives and negatives. Such a high dimensional study could test our hypothesis that the estimates of simple methods would deteriorate much more rapidly under increasing variance than methods mPCAQ or MMAQ.

To see whether the results are insensitive to the constant costs we used, we repeated our experiment with $C^- = 2.0$. That is, positives are half the cost of negatives. As expected, the false positive pollution of Simple, GUT, CAQhalf and CAQ30 each caused these methods to over-estimate cost. The results were mostly similar and so we omit them. Overall, MMAQ tended to perform well with both sets of constant costs.

When computing the precision-corrected average, it is possible to have a negative or zero denominator in equation (6) whenever the estimated precision Pr is less than or equal to the class distribution q. This happens especially often for the median sweep method mPCAQ, which tries many thresholds. We tested a variant method that ignored thresholds yielding negative estimates for C^+, since all costs are positive, as in most business datasets. But this only resulted in a strong positive bias for the median. In situations where Pr equals q, we get a division by zero error. We leave these out of the set of thresholds considered by mPCAQ and MMAQ. But if it happens at the single threshold chosen for the PCAQ method, then some fallback position needs to be taken. It happens seldom, so it has little effect for an empirical study of this nature, but deployed software must be ready to cope with such situations. We had PCAQ fall back to another method (GUT) where Pr=q exactly. Perhaps ideally it might fall back to using a different threshold, but if this threshold also fails, then further software complexity proliferates to avoid failure.

5.3.1 Cost-Confounded Prediction

The methods above implicitly assume that the cost of positive cases is not correlated with the prediction strength of the base classifier. As an assurance, one may check the correlation between cost and the classifier scores over the positive cases of the training set. If the classifier predicts the most expensive positives strongest, then the methods above will overestimate badly, especially CAQ. Likewise, negative correlation results in underestimates. This problem could also arise if the classifier's scores have substantial correlation with cost for *negative* cases, so these can be checked separately.

To avoid these problems, we recommend the cost attribute not be given as a predictive feature to the classifier. If the average cost for the positive class C^+ is similar to the overall average, then this attribute will generally be non-predictive. But in the interesting case where it is substantially different from the background, this feature will be strongly predictive, e.g. for a rare but relatively expensive subclass. In this case, it is tempting to provide the cost attribute as a predictive feature to improve the classifier. However, it is better not to. The methods are explicitly designed to compensate for imperfect classifiers, but cannot compensate for a bias that prefers the more expensive positives, which would likely underestimate the total cost and average of positives. Even so, if the cost attribute is extremely predictive and helps induce a near perfect classifier, then it would seem justified to include the cost as a predictive feature. Determining at what point it is beneficial could be an avenue for future work. In any case, a predictive cost attribute may be employed as an input feature for non-cost *quantifiers*, as well as for the quantifier subroutine used by the cost quantifier to estimate the prevalence of the positive class.

5.3.2 Missing Costs

In some settings, especially those where datasets are gathered across worldwide enterprises, cost values may be missing or detectably invalid for some cases. Given that most of the above methods begin by estimating the average cost for positives C^+, such cases with missing cost may be omitted from the analysis. That is, the estimate of C^+ is determined by the subset of cases having valid cost values, and the count is separately estimated by a quantifier applied to all the cases, including those with missing costs.

This treatment assumes the costs are *missing at random*. To test this assumption, one can train a binary classifier to predict which cases do not have a valid cost value; if it can achieve high precision and recall based on the rest of the record's feature vector, then the process is surely not random. Unfortunately, if such a binary classifier is inaccurate, it does not necessarily mean the costs are missing at random.

6. Extensions

Up to this point we have attacked only the problem of quantifying a single positive class of interest in a single test set. Yet in all settings where we have applied this technology in practice, there have been many different issues to be quantified, and not only for a single test set (Forman, Kirshenbaum and Suermondt, 2006). Instead, they are multi-class tasks, and we often wish to perform the quantification over time in order to look for changes or monitor trends. For example, among many different types of support issues that occur with a particular product, a rarely occurring issue that bears significant business costs may be becoming more frequent month to month. Naturally, it is important to detect this trend early, long before the issue becomes one of the most frequent issues.

In the first subsection below we describe how we have generalized from the binary setting to multi-class settings. In the second subsection we describe how we have applied the methods to track trends over time—the original impetus for this work. These extensions apply to both count quantification and cost quantification. The treatment is not exhaustive, but expected to be useful guidance for other practitioners and includes discussion of pitfalls to avoid.

6.1 Multi-Class Quantification

There are two types of multi-class tasks in traditional classification. *Single-label* tasks (also known as "1-of-n tasks") have exhaustive, mutually exclusive classes, where each item belongs to exactly one class. *Multi-label* tasks allow cases to be labeled with multiple classes. These two task types carry across to both quantification tasks and cost quantification tasks. The deciding property for single-label versus multi-label *quantification* tasks is whether the sum of the counts or costs for all classes must add up to the total of the whole test set. In many settings, the classes are not mutually exclusive and do not necessarily cover the entire dataset. In our applied experience, multi-class quantification tasks commonly have many cases that belong to none of the classes. For example, when tracking a variety of issues in technical support logs at Hewlett-Packard, there may be 30 recognized issue types for a product line, but there may also be many one-off incidents in the logs that never or rarely re-occur and do not warrant a separate class. For some complex product lines, these can amount to nearly half the incidents. In such cases, it would not be worthwhile to define and build up training sets for enough classes to cover the whole dataset. To treat such multi-label quantification tasks, each of the classes can simply be treated as a separate binary quantification problem, mirroring the practice in multi-label classification.

Nonetheless, there are situations where the single-label setting is called for, e.g. when segmenting user populations into different classes of customers (quantification), or estimating the total spent for the different customer segments (cost quantification). In such cases, we want the quantification to sum to 100%. The obvious multi-class extension for a single-label version of Classify & Count would train a single-label classifier, such as C4.5 or any other model, and count its test set predictions in each class. This bears instability problems similar to the binary Classify & Count method. In a single-label setting with many classes, some classes are bound to be relatively rare, and just as we saw for binary class imbalance, the default decision thresholds of a single-label classifier will tend to under-predict rare classes and over-predict common classes. To illustrate this, Figure 14 (left) shows the true prevalence of 44 mutually exclusive classes in dataset

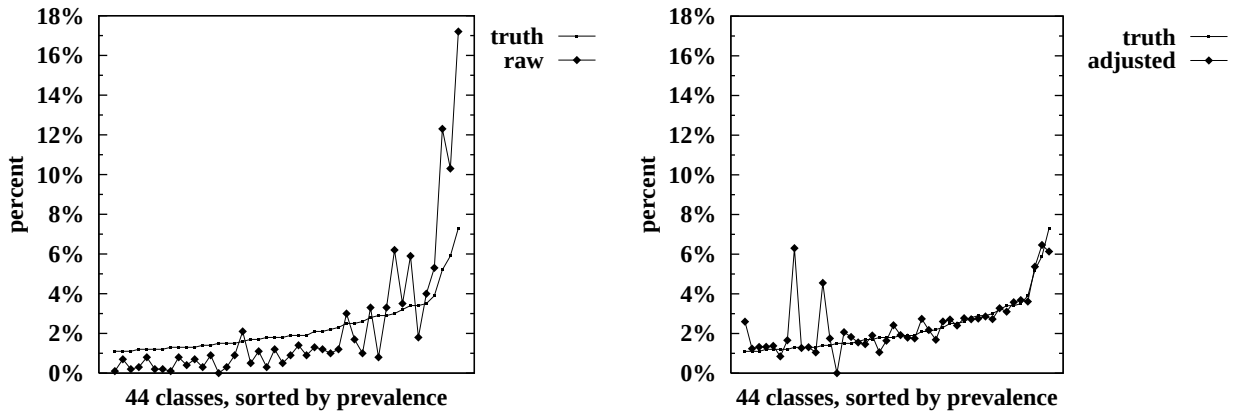

Figure 14. A problem with multi-class classifiers used for quantification: rare classes are underestimated and *vice versa*.

new3, and the percentage that are predicted in each class by a standard single-label Naïve Bayes classifier. SVM exhibits a similar trend. Observe that the rarer classes are predicted even more rarely. And this example is with a perfectly stratified split of the dataset, so the test distribution exactly matches the training distribution.

Next, one may consider adapting the Adjusted Count method for single-label quantification tasks. This extension consists of the following steps. Induce a single-label classifier on the training set, and estimate its *tpr* and *fpr* characteristics for each class via cross-validation on the training set. Once given the test set, count the number of test cases predicted in each class, and finally adjust each of the class counts using equation (1). We illustrate the results of this method in the right-hand graph of Figure 14. This extension does not work well for three key reasons:

- As illustrated above, a single-label classifier selects very conservative thresholds for the many rare classes. Thus, the *tpr* and *fpr* estimates for those classes are determined in the tail of the distribution, where there are few true positives or false positives. Such estimates have high variance, yielding instability in the quantification. Observe in the right-hand graph that the adjusted estimates are much less stable for the rare classes. Although most class estimates are reasonably accurate, four of the rarer classes have erratic estimates.
- In some situations, the single-label classifier may never predict a certain class. In training, this may occur in cross-validation, yielding $tpr=fpr=0$ for that class, which results in a division by zero in equation (1). In testing, if a zero count for a class is observed in the test set and $fpr>0$ in training, then the numerator of equation (1) becomes negative. In the right-hand graph, one class exhibited the former, division-by-zero type of error, depicted as a 0% estimate. Such an error is clearly independent of the test set.
- Although the expected behavior of this single-label method should yield a total sum of 100% on average with large data sets, it may produce sums that do not total 100%, especially for smaller data sets where the estimates are less certain. Nothing in the method forces the sum to 100%. In the example, the sum of the adjusted counts is 109% of the number of test cases. While the counts could be normalized back to 100% by a simple division, we see from the wildly high predictions for some of the rare classes in the example that normalization will bias the whole curve, even though we already have good precision for the larger classes. Generally, erratic estimates for rare classes are biased on average to greatly over-predict. Negative predictions are clipped at zero and underestimates for a rare class make little difference to the total. Conversely, large over-estimates can amount to a significant portion of the total. Thus, if we normalize the estimates, we will tend to under-estimate almost all classes.

For binary quantification, we saw the solution was to set the classifier threshold to be more liberal in identifying positives, in order to obtain better estimates of *tpr* and *fpr* and also avoid having too few predictions for the minority class in the test set. But for the multi-class case, there is no way to cause a single-label classifier to increase the number of positives simultaneously for all the classes. Increasing predictions for one class comes at the expense of others.

For theses reasons, we recommend performing independent binary quantifications for each class versus all others, and then normalizing the final estimates so that they sum to 100%. In this way, each quantification compensates independently for its imperfect classification. These binary subproblems may have high class imbalance, particularly if there are many classes. Fortunately, dealing with class imbalance is a relative strong point for quantification. Thus, it is not a concern to decompose single-label problems into many binary 1-vs-other-classes subproblems, unlike traditional classification. That said, other methods for treating single-label quantification tasks might be devised that avoid the imbalance by pitting several classes together against all others, in analogy with error correcting code (ECC) matrices for multi-class classification (Ghani, 2000). This is an open area for future work.

6.2 Quantifying Trends over Time

One of the primary motivations for our research was to accurately measure trends over time. We refer solely to measuring trends in historical data, rather than forecasting into the future before data is available. Forecasting is an important business application, and can be done with traditional time series methods. In any case, both forecasting and analyzing past trends require historical measurements over time—our focus.

So far, we have only discussed quantification with respect to a single test set. To use a quantifier for trending is relatively straightforward by traditional methods. The date-stamped test cases are partitioned into discrete bins—for example, daily or monthly batches—and a single trained quantifier is applied to each batch separately to estimate the prevalence of positives in each. These measurements may then be plotted together in one time series graph, optionally with a conventional moving average filter or fitted trend line to smooth the data or project into the future.

In such applications, a common subproblem is to select an appropriate bin-width for the data set. If the bin-width is too coarse, little information is revealed and sudden changes cannot be detected. If the bin-width is too granular for the available data, the number of samples in each bin becomes small or even zero. This is a common issue in constructing a traditional histogram. But for quantification, there is an additional factor at work here. Not only may there be high variability between adjacent bins due to the underlying phenomenon being inconstant, but additionally the quantification will naturally have higher variance when estimating from only a few samples in a bin. The quantification methods we presented are intended to work on large test sets and will produce noisy estimates on small batches. Thus, to avoid narrow bin-widths in situations where the business desires a fine-grained trend line, we have used a sliding window technique to aggregate cases from adjacent bins into each test set batch. This provides smoothing similar to a moving average, and we have used it to mask uninteresting weekend variations. That is, each day is a bin and each test batch consists of 7 adjacent bins. If instead one were to quantify each bin separately and *then* use a 7-period moving average to smooth over the weekends, then the very small number of incidents reported on Sundays can sometimes produce highly erratic estimates of the prevalence, which disturb the moving average.

Another issue that has come up in our use of trending with quantifiers is that additional training cases can become available later. For example, after a new batch of a month's data arrives, a domain expert may label some of its cases. While we would like to use the larger, cumulative training set to possibly produce a more accurate quantifier, it is important to recognize that the new quantifier may be uncalibrated with respect to the previous quantifier. Thus, its estimates on new batches of data should not be appended to pre-existing time series produced from previous quantifiers. It is best to throw out the old data, and apply the new quantifier over the entire time period. The additional training data may improve the quantifier's estimates on the old bins as well.

When trending over longer periods of time, general concept drift is often involved and can be difficult to cope with. For example, old products may undergo name changes, new product models may be released under the same general product name, and the sets of issues associated with a product may change drastically with major operating system upgrades, especially newly arising issues. In order to accurately quantify a trend over time, we need the quantification estimate from each time bin to be reasonably calibrated with each other. But under substantial concept drift, we may need to employ different quantifiers for different time periods. Each may be produced by a training set appropriate to the class concept in effect for its time period, which would naturally require ongoing training data. For such a difficult task, this may be the best that is possible.

7. Related Work

The majority of the literature in machine learning classification focuses on optimizing the correctness of individual predictions, whether measured by error rate, F-measure, misclassification cost, etc. The problem of learning classifiers under highly imbalanced class distributions is well known and various methods have been devised to improve individual predictions in these situations (e.g. Van Hulse, Khoshgoftaar and Napolitano, 2007; Weiss and Provost, 2003; Vucetic and Obradovic, 2001; Lachiche and Flach, 2003). Regardless of the degree of imbalance, in most machine learning literature, the class distribution for training and testing is assumed equal, and moreover, enforced to be equal within experiments via random sampling and cross-validation. In less common work that acknowledges that the test distribution may differ, the estimate of the class distribution is used only *in order to* improve the individual classifications, e.g. via ROC analysis (Provost and Fawcett, 2001; Fawcett, 2003; Mei and Zhai, 2005; and Saerens, Latinne and Decaestecker, 2002).

The overall objective in all such research has been the correctness of the individual classifications. Except where improved methods achieve perfect accuracy, they are not sufficient for the large family of quantification applications whose objective utility is different: precise estimation of the class distribution. To our knowledge, there has not been work to empirically compare and determine machine learning methods that excel in quantification, nor any work in cost quantification via classification. This paper extends our recent publications (Forman, 2005; Forman, 2006) with superior methods, a more focused experiment protocol, and empirical evaluation of cost quantification methods. Note that improvements in quantification methods can be used as a subroutine for the traditional purposes of optimizing the classification decision threshold via ROC analysis and in calibrating probability estimating classifiers.

Outside the field of machine learning, the problem of estimating the class distribution of a target population is not new. In the medical statistics literature, an imperfect binary diagnostic test with known sensitivity and specificity for a disease can be used to estimate the prevalence of the disease in a test population (Zhou, Obuchowski and McClish, 2002, p.389). In fact, research by Valenstein (1990) considers the problem of determining the sensitivity and specificity of a test when only noisy ground truth labels are available. By contrast, our work leverages the flexibility of machine learning methods, and allows us to handle highly imbalanced class distributions more gracefully. This also enables one to decompose single-label multi-class tasks into many highly imbalanced binary subproblems.

In biology, the capture-mark-recapture methods are well known for estimating the population size of animals—most famously, butterflies (Seber, 1982). The Lincoln-Petersen method estimates the population of a species (number of positives) as n*M/R, where n is the total captured on recapture day, M is the number of specimens marked initially, and R is the number of recaptured specimens, recognized by their marks. In contrast with quantification, this procedure does not require knowing the total test set size, and it assumes a perfect classifier. That is, no false positive mistakes are made in capturing specimens of unrelated species. Variations in biology include estimating site occupancy of a species, e.g. what percentage of square acres contain a particular variety of toad. Work by MacKenzie *et al.* (2002) addresses the problem of estimating site occupancy when the classifier—typically a biologist in the field—may make false negative errors. In such a setting, the probability of false positives is assumed zero.

The cost quantification task is new to machine learning and data mining, despite the large body of literature that deals with various types of cost, e.g. misclassification costs per class or per case. Where in the literature the 'cost' refers to some penalty that an algorithm needs to minimize, in cost quantification it carries no connotation of cost avoidance. Turney (2000) developed an extensive taxonomy of types of cost in machine learning, but none of the costs are related to the cost attribute in cost quantification. Our general concern for the labor cost to label training examples he refers to as the "cost of teacher," but unlike the literature in this area, we do not attempt to simultaneously model this cost in conjunction with any sort of misclassification cost.

Regarding trending, there exists *unsupervised* work in tracking topic distributions as they shift over time (e.g. Mei and Zhai, 2005; and Havre, et al., 2002). The idea is to cluster items, then plot the prevalence of the time-stamped cases over time. This can be used to identify the ebb and flow of news topics, for example. Being unsupervised, this approach naturally has uncalibrated cluster boundaries and thus has little bearing on the *supervised* task of quantification, which works very hard to get the quantity inside the boundary correct. As a side note, we point out a potential connection between tracking trends over time and the growing field of learning from data streams (e.g. Hulten, Spencer and Domingos, 2001). In most of these situations concept drift is involved, and this leads to the need to obtain additional labeled training examples and retrain classifiers. While additional training examples are not required in the straightforward quantification task, it is natural that in real-world settings we would expect some degree of concept drift. Most research to date deals with either detecting when a concept has drifted, or in adapting the classifier to track the current target concept.

8. Conclusions and Future Work

We have articulated two new research challenges that are important to many business and scientific applications: (1) inducing a *quantifier* to accurately estimate the test class distribution, and (2) inducing a *cost quantifier* to estimate the cost distribution—the per-class subtotal of a cost value associated with each case. These capabilities are particularly needed for applications that track trends over time. For both types of quantification, we have proposed a variety of methods, and compared them all experimentally with a new methodology suitable for quantification research. The experiments showed the consistent strength of median sweep methods (MS and mPCAQ): returning the median of estimates produced from many different thresholds of a classifier. This method smooths over estimation errors made at any single decision threshold, much as the well known method of statistical bootstrapping.

It is fortunate that quantification and cost quantification can be made to compensate for the inaccuracy of a classifier, yielding substantially more precise and less biased estimates. Moreover, this requires only small amounts of training data, which can reduce labor costs compared with having to train highly accurate classifiers. These factors may lead to greater acceptance of machine learning technology for business use, where paid labor is necessary to generate training sets at the outset, often for an uncertain business benefit. We have been promoting machine learning within our company for years, but have never before experienced the business interest we find for quantification (Forman, Kirshenbaum and Suermondt, 2006). To data mining researchers who desire to apply and develop advanced predictive models, this comes as some surprise, since the task of counting seems so simple in comparison—at least on the surface.

Quantification technology and future improvements thereon can be folded back in to support traditional threshold selection via ROC analysis, as well as calibrating probability estimating classifiers that have fallen out of calibration because of a change in the class distribution. The estimates can also be used to calibrate certain detectors, e.g. to control the expected false alarm rate in certain monitoring applications. The availability of effective and robust quantifiers can enable business and scientific tracking, and with little effort on the part of domain experts. For example, imagine training a quantifier to track customer sentiment in blog posts over time about the products of one's company. This application involves a classification task that will never be perfect, yet quantification may provide good overall estimates despite the inability to classify individuals well.

Future work will involve new, improved methods as well as experimentation under greater degrees of class imbalance. To set an ambitious goal, we propose that within the next five years, quantification methods will be able to estimate within half a percent of the target across a reasonably wide range of prevalence, as well as estimating well down to one part per thousand. Chemists easily talk about *parts per million*, but machine learning is currently nowhere near up to the task. To perform research far into the tail of the probability distribution will require very large benchmark datasets, ideally publishable ones for experimentation and repeatability by others. Studying high class imbalance requires that the data set labels not have any mistakes, for the conclusions are more sensitive to any noise in the labels used as ground truth. Ideally, such a dataset would include individual costs to support research in *cost quantification*, where estimates are especially difficult when positives are rare. The most effective methods may depend strongly on the characteristics of the data, so hopefully such a dataset would suit a popular family of applications in the real-world. And on datasets where the cost attribute is a highly differentiating feature of the positive class, research will be needed to determine when and how to leverage it as a predictive feature and yet avoid the problems of cost-confounded bias in the prediction, as we discussed. Other future research directions include advanced multi-class methods, class hierarchies, missing cost attributes (often not missing-at-random), and quantification under various constraints,

such as having less tolerance for *under*estimating the size or cost of a subclass, as needed by some business applications. Finally, trending over time naturally introduces general concept drift, which is a challenging and important area for future research. Currently we assume that the distribution of features given the class P(x|class) is the same between training and testing, but future methods may further loosen this assumption.

Acknowledgements

I wish to thank my colleagues Jaap Suermondt, Evan Kirshenbaum, Jim Stinger, Tom Tripp, Farzana Wyde and Jamie El Fattal for contributions in conceiving and developing this application. Thanks also to Bin Zhang, Hsiu-Khuern Tang and Shyam Rajaram for their input. Finally, I would like to thank Gary Weiss and the anonymous reviewers for their extensive work to review and improve this paper.

References

Fawcett, T. ROC graphs: notes and practical considerations for data mining researchers. Hewlett-Packard Labs, Tech Report HPL-2003-4, 2003. www.hpl.hp.com/techreports/2003/

Fawcett, T. and Flach, P. A response to Webb and Ting's 'On the application of ROC analysis to predict classification performance under varying class distributions.' *Machine Learning*, 58(1):33-38, 2005.

Forman, G., Kirshenbaum, E., and Suermondt, J. Pragmatic text mining: minimizing human effort to quantify many issues in call logs. In *Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (KDD, Philadelphia):852-861, 2006.

Forman, G. Quantifying Trends Accurately Despite Classifier Error and Class Imbalance. In *Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (KDD, Philadelphia):157-166, 2006.

Forman, G. Counting positives accurately despite inaccurate classification. In *Proc. of the 16th European Conf. on Machine Learning* (ECML, Porto):564-575, 2005.

Forman, G. An extensive empirical study of feature selection metrics for text classification. *J. of Machine Learning Research*, 3(Mar):1289-1305, 2003.

Ghani, R. Using Error-Correcting Codes for Text Classification. In *Proc. of the 17th Int'l Conference on Machine Learning* (ICML):303-310, 2000.

Han, E. and Karypis, G. Centroid-based document classification: analysis & experimental results. In *Proc. of the 4th European Conf. on the Principles of Data Mining and Knowledge Discovery* (PKDD): 424-431, 2000.

Havre, S., Hetzler, E., Whitney, P., and Nowell, L. ThemeRiver: visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9-20, 2002.

Hulten, G., Spencer, L., and Domingos, P. Mining time-changing data streams. In Proc. of the 7th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD, San Francisco):97-106, 2001.

Lachiche, N., Flach, P.A. Improving Accuracy and Cost of Two-class and Multi-class Probabilistic Classifiers Using ROC Curves. In *Proc. of the 20th Int'l Conf. on Machine Learning* (ICML, Washington DC): 416-423, 2003.

MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., and Langtimm, C. A. Estimating site occupancy rates when detection probabilities are less than one. *Ecology*, 83:2248–2255, 2002.

Mei, Q. and Zhai, C. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining* (KDD, Chicago): 198-207, 2005.

Provost, F. and Fawcett, T. Robust Classification for Imprecise Environments. *Machine Learning*, 42:203-231, 2001.

Saerens, M., Latinne, P., and Decaestecker, C. Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. *Neural Computation*, 14(1):21–41, 2002.

Seber, G. A. F. *The Estimation of Animal Abundance and Related Parameters*, 2nd Edition. Blackburn Press, New Jersey, 1982.

Turney, P.D. Types of cost in inductive concept learning. *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning* (WCSL, ICML, Stanford University), 2000. *Computing Research Repository*, vol. cs.LG/0212034.

Valenstein, P. Evaluation diagnostic tests with imperfect standards. *American Journal of Clinical Pathology*, 93:252-58, 1990.

Van Hulse, J., Khoshgoftaar, T. M., and Napolitano, A. Experimental perspectives on learning from imbalanced data. In *Proc. of the 24th Int'l Conference on Machine Learning* (ICML, Oregon):935-942, 2007.

Vucetic, S. and Obradovic, Z. Classification on data with biased class distribution. In *Proc. of the 12th European Conf. on Machine Learning* (ECML, Freiburg):527-538, 2001.

Weiss, G. M., & Provost, F. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

Witten, I. and Frank, E., *Data mining: Practical machine learning tools and techniques (2nd edition)*, Morgan Kaufmann, San Francisco, CA, 2005.

Wu, G. and Chang, E. KBA: kernel boundary alignment considering imbalanced data distribution. *IEEE Trans. on Knowledge and Data Engineering*, 17(6):786-795, 2005.

Zhou, X.-H., Obuchowski, N. A. and McClish, D. K. *Statistical Methods in Diagnostic Medicine*, Wiley, New York, 2002.