Counting Positives Accurately Despite Inaccurate Classification

George Forman
HP Laboratories Palo Alto

HPL-2005-96(R.1)
August 3, 2005*

supervised machine learning, estimation, mixture models, shifting class prior, non-stationary class distribution

Most supervised machine learning research assumes the training set is a random sample from the target population, thus the class distribution is invariant. In real world situations, however, the class distribution changes, and is known to erode the effectiveness of classifiers and calibrated probability estimators. This paper focuses on the problem of accurately estimating the number of positives in the test set–*quantification*–as opposed to classifying individual cases accurately. It compares three methods: classify & count, an adjusted variant, and a mixture model. An empirical evaluation on a text classification benchmark reveals that the simple method is consistently biased, and that the mixture model is surprisingly effective even when positives are very scarce in the training set–a common case in information retrieval.

* Internal Accession Date Only
Published in and presented at the 16th European Conference on Machine Learning (ECML'05), 3-7 October 2005, Porto, Portugal http://ecmlpkdd05.liacc.up.pt/

Counting Positives Accurately Despite Inaccurate Classification

George Forman

Hewlett-Packard Labs
Palo Alto, CA 94304 USA
ghforman@hpl.hp.com

Abstract. Most supervised machine learning research assumes the training set is a random sample from the target population, thus the class distribution is invariant. In real world situations, however, the class distribution changes, and is known to erode the effectiveness of classifiers and calibrated probability estimators. This paper focuses on the problem of accurately estimating the number of positives in the test set—*quantification*—as opposed to classifying individual cases accurately. It compares three methods: classify & count, an adjusted variant, and a mixture model. An empirical evaluation on a text classification benchmark reveals that the simple method is consistently biased, and that the mixture model is surprisingly effective even when positives are very scarce in the training set—a common case in information retrieval.

1 Motivation and Scope

We address the problem of estimating the number of positives in a target population, given a training set from which to learn to distinguish positives from negatives. This could be used, for example, to estimate the number of news articles about terrorism each month, or the volume of advertising by a competitor over time. Unlike previous literature in machine learning, our end goal is not to determine a classification for each item, but only to estimate the number of positives—*quantification* as opposed to *classification*. This is an important problem in real-world situations where the class

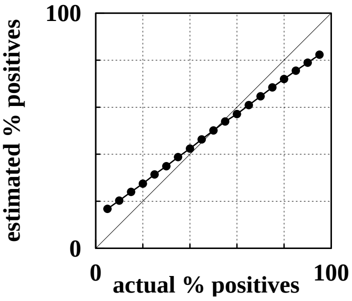

Fig. 1. Counting positives via a classifier trained with 100 positives and 100 negatives yields a poor estimate of the count as we vary the test class distribution, even though the classifier achieves nearly 90% F-measure in cross-validation.

Table 1. Summary of parameters considered in the empirical comparison

P = 10...200	Positives in training set	Counting Methods:	
N = 100...1000	Negatives in training set	CC	Classify & Count
p = 5...95%	Percent positives in test set	AC	Adjusted CC
Benchmark: 21 binary text classification tasks		MM	Mixture Model
Learning Algorithms:		Performance Metrics:	
SVM	Support Vector Machine	Err	estimated p – actual p
NB	Naive Bayes	AbsErr	|Err|
NBM	Multinomial Naive Bayes	CE	Normalized Cross-Entropy

distribution may shift over time in the target population. It is then needed to track, detect and report noteworthy shifts in the class distribution, to calibrate probability-estimation classifiers, and to select a binary classification threshold to optimize F-measure or misclassification costs on an ROC curve [Fawcett, Weiss]. It is also needed to calibrate classifiers that are used on different target populations, such as in medical settings where the training set does not represent a random sample of each target population.

The obvious solution is to train a binary classifier from the training data, and count its positive predictions on the test set. For example, Fig. 1 shows the result of this method as we vary the proportion of positives in a particular test set. The classifier consistently overestimates/underestimates the positives when the test set deviates from the balanced class distribution used in training, even though this classifier achieves nearly 90% F-measure in cross-validation testing. Unfortunately, as we demonstrate later, this method consistently leads to poor results—unless the classifier is extremely accurate, which can require substantial cost to get enough training data and may never be feasible for some tasks.

Although accurate classification is *sufficient* for estimating the count accurately, it is not *necessary*. Even with a mediocre classifier, the count can be accurate if the false positives are canceled out by a balanced number of false negatives. This raises the question of whether estimating the count alone can be accomplished more accurately and/or with less training data.

We describe and evaluate two such superior methods: one based on adjusting the binary classifier's count, the other based on a mixture model of the distribution of classifier scores, as described in section 3. We empirically compare their ability to accurately track varying test class distributions under a variety of training set compositions. In such a study it is important to vary the training and testing class distributions as independent parameters, in contrast with most classification research practice, which assures the class distribution is the same in training and testing via random sampling or cross-validation. We are especially interested in the situation where there are a small number of positives to train from—a common case in information retrieval and bioinformatics where positives are rare and obtaining labels costs human effort. Table 1 provides an overview of the range of parameters we studied. The experiment protocol and its results are described in sections 4 and 5, respectively. Next, we complement this introduction with a discussion of related work to help scope this work.

2 Related work

The great majority of the machine learning literature assumes the class distribution is invariant between training and testing. Some work focuses on improving classification accuracy when the target class distribution is imbalanced, usually by over-sampling the minority class or under-sampling the majority class to balance the training set, where induction algorithms are more effective. The work of Weiss & Provost [Weiss], for example, carefully studies the effect of varying the training distribution to optimize classification accuracy for a given test set.

Many works mention the need to adjust the class priors to match the test distribution. This is usually assumed to be done via foreknowledge or a manual inspection of a random sample. Even in such papers, their performance goal is only to improve binary classification accuracy or probability estimation [Bennet], and not to accurately count positives in test sets as here. Some works specifically seek to factor out the effect of class distribution by, for example, evaluating classification performance via *balanced accuracy* or the area under the ROC curve (AUC).

Finally, some work attempts to detect class drift—when the character of a class with respect to its feature space changes over time, suddenly or gradually. Class drift falls outside the scope of this paper. We only consider shifts in the relative populations of positives and negatives.

3 Theoretical Framework of Counting Methods

In this section, we describe the theoretical framework of three methods for estimating the count of positives. We also list two intuitive methods that do not work.

3.1 CC: Classify & Count

This is the obvious method. First, we learn a binary classifier from the training data, such as a Support Vector Machine (SVM) or Naïve Bayes model. We then apply it to each item of the test set, and count the number of times it predicts positive. If the predictions are nearly perfect, then the count will be nearly accurate, no matter what the test class distribution. This should be successful where the two classes are very well separated, e.g. distinguishing news articles written in German vs. English.

If the classes are not well separated, then there will be some number of false positives and false negatives, and it is unlikely that these would be closely balanced. Moreover, any induction algorithm that is designed to maximize its accuracy on the training set will prefer the negative class if positives are the minority, a common case. It leads to systematically underestimating the count. (This suggests artificially balancing the training set to achieve a balance between false positives and false negatives. This is ineffective and even ill-conceived, as discussed in section 3.4.)

3.2 AC: Adjusted Count

This method is an extension to the straightforward *classify & count* method. Although the class labels are not available on the test set, consider the counts that would appear in the 2x2 confusion matrix below:

	Prediction:	
Actual Class:	Positive	Negative
Positive	TP	FN
Negative	FP	TN

The observed count is the sum of true positives TP and false positives FP. We can model each of these counts separately as:

$$\begin{aligned} &\text{observed\_count} = \text{TP} + \text{FP} && \textbf{(1)} \\ &\text{TP} = \text{TPR} * \text{actual\_positives} \\ &\text{FP} = \text{FPR} * \text{actual\_negatives} = \text{FPR} * (\text{total} - \text{actual\_positives}) \end{aligned}$$

where TPR is the true positive rate of the classifier, P(predict +|actual +), and FPR is its false positive rate, P(predict -|actual -). These are estimated from the training set, as discussed below. Solving this system of equations, we obtain:

$$\text{actual\_positives} = (\text{observed\_count} - \text{FPR} * \text{total}) / (\text{TPR} - \text{FPR}) \qquad \textbf{(2)}$$

This adjustment to the count is the essence of this method, but there are a few additional points. First, observe that the denominator could go to zero. This would only happen with a worthless classifier that is equally likely to predict positive for either class. Normally TPR >> FPR, so the denominator is positive and somewhat less than 1.0. But if TPR < FPR, then the classifier is more likely to predict positive for negative items than for positive items. In this situation, one could reverse the outputs of the classifier. Re-deriving for this case ends up with the exact same equation, so it may be used without special casing for a negative denominator. In these situations, FPR*total is likely to exceed the observed count of positives, so the numerator will also be negative. Finally, the adjusted count may under some situations predict a negative number of positives or more positives than the total test cases. Thus, we limit its output to the range [0, total].

To estimate TPR and FPR for a given classifier, standard techniques may be used, such as stratified 10-fold cross-validation on the training set (divide the training set into 10 subsets, testing each one with a classifier trained on the other 9; stratification ensures that the minority class is evenly distributed among the folds). It yields a 2x2 confusion matrix, and we compute TPR = TP/(TP+FN), and FPR = FP/(FP+TN).

3.3 MM: Mixture Model

Many binary classifier models consist of a scoring mechanism and a threshold on the score to choose between predicting positive or negative. The induction algorithm has to learn both how to score positives higher than negatives and how to pick the threshold well. In the MM method, we eliminate this second step, and only use the scoring portion. We then consider the *distribution* of scores generated by the classifier. After training the classifier, we determine the empirical probability

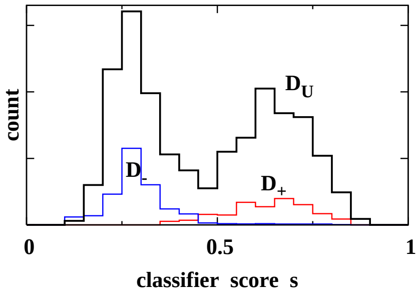

Fig. 2. Histograms of classifier scores

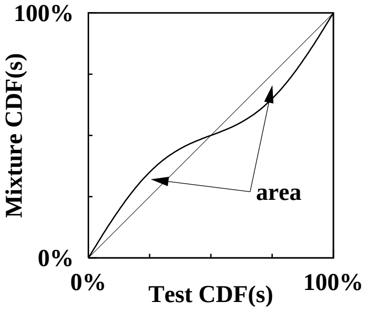

Fig. 3. P-P plot comparing two CDFs

distribution D_+ of scores that it generates on the positive training examples, and separately D_- for the negative training examples. Then, during testing, we model the observed distribution D_U of scores on the unlabeled data as the mixture (see Fig. 2):

$$\text{total} * D_U \ = \ \text{actual\_positives} * D_+ \ + \ \text{actual\_negatives} * D_- \qquad \textbf{(3)}$$

Finally, to estimate the positive count, we determine which mixture of positives and negatives would yield the closest fit to D_U. This is the essence of the method, but there remain several design choices:

1. *How to obtain the distributions D_+ and D_- from the training set*: If we train on all the training data, and then observe the classifier scores on the training data, the separation between positive and negative scores will be overly optimistic compared with the actual test distribution. Instead we use stratified f-fold cross-validation, and gather the scores from each fold into one distribution. Strictly speaking, these scores were generated from f different classifiers, but if f is large, then these classifiers share most of their training data in common.
2. *Whether the empirical distributions found during training should be reduced to a parametric model of a distribution*: Based on the range of variation, we decided not to try to fit the distributions to parametric models, which also avoids adding parameters to the algorithm that may need to be optimized.
3. *Whether to characterize the distributions by their empirical probability density function (PDF) or their cumulative distribution function (CDF)*: Using the PDF requires discretizing the counts into artificial bins. If the bin size is too small, then the estimates in each bin become noisy. This would create additional parameters for tuning. We selected CDF.
4. *Whether to give special treatment to test scores that fall outside the range of scores observed during training*: Optionally, test items that score higher than any score observed during training could be treated separately, i.e. surely included in the final positive count, and excluded from the mixture model fitting. Likewise scores smaller than any observed score in training could be treated separately as a negative. We include this refinement.
5. *How to measure the goodness of fit between D_U and the mixture model*: Given two CDFs, the standard way to measure their difference is the Kolmogorov-Smirnov statistic, which measures the maximum difference between the two for all scores. While common, this coarse metric does not consider finer differences in the shape of the fit. For this reason, we developed another difference metric we call PP-Area,

described below. Another choice would be the standard Anderson-Darling statistic, but it is known to emphasize the tails of the distribution, which is not what we need for this application. (The well-known Chi-Squared statistic is appropriate only for discrete PDFs.)

6. *How to determine the mixture that optimizes the fit*: For research purposes, we compute the goodness of fit for each value from 0% to 100% positives stepping by 0.5% returning the best, but in practice one could use hill-climbing methods.

We name this particular collection of design choices the "Countess" method.

PP-Area: a difference metric for two CDFs.
Given two CDFs, a well-known method for visually comparing them is to plot one vs. the other while varying their input threshold, yielding a Probability-Probability plot, or P-P plot (see Fig. 3.). If the two CDFs yield the same probability at each input, 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, commonly to decide whether an empirical distribution matches a parametric distribution. To reduce this linearity test to computation, it would be natural to measure the mean-squared-error (MSE) of the points on the PP curve to the 45° line. But MSE is highly sensitive to the maximal difference, as is Kolmogorov-Smirnov.

Our solution is to measure the difference between two CDFs as the area where the PP curve deviates from the 45° line. This has well defined behavior partly because the curve always begins at (0,0), ends at (1.0,1.0), and is monotonic in both x and y. It also has the intuitive property of being commutative, unlike MSE or mean-error.

3.4 Non-Solutions

If the classes are not well separated by a classifier, then a tradeoff must be made between precision vs. recall (false negatives vs. false positives). This tradeoff is manifested in the threshold used by a binary classifier. During training it is optimized for accuracy in risk minimization methods, such as SVM. One ill-conceived idea is to try to adjust this threshold at training time so as to balance false positives and false negatives. This is not possible because balancing these two depends explicitly on the class distribution, which may vary in testing.

Another ill-conceived idea is the following: Rather than have the classifier output a hard binary decision despite its uncertainty, use a classifier that outputs a probability estimate for each item. Then, estimate the positive count as the sum of probabilities over the test set. Again, this cannot work because the probability estimates depend explicitly on the class distribution; the calibrated probabilities would become uncalibrated whenever the test class distribution varies.

4 Experiment Protocol

To compare these methods, we conducted an empirical evaluation. The standard methodology of cross-validation to obtain training and testing sets is not appropriate.

Table 2. Benchmark data sets, the specific classes used as positive, and their sizes

Dataset	Source	Cases	Classes	# Positives in Each Class
fbis	TREC	2463	3,7,10	387,506,358
la1	LA Times	3204	0,1,3,5	354, 555, 943, 738
la2	LA Times	3075	0,1,3,5	375, 487, 905, 759
ohscal	OHSUMED	11162	0…9	1159,709,764,1001,864, 1621,1037,1297,1450,1260

Instead, we must independently vary the class distribution in the training set and the test set to determine how well various methods can track the test distribution, despite variations in the training set they are given. To this end, we randomly drew 200 positives and 1000 negatives from each benchmark classification task as the maximum training set. We then trained with various subsets of this data, reducing the number of positives and negatives independently. Likewise, from the remaining data, we randomly removed positives or negatives to achieve various desired testing class distributions. We varied the test distribution from 5% positive to 95%, stepping by 5% increments.

Datasets: We used publicly available text classification datasets and used 21 "one vs. all other classes" classification tasks that had sufficient positives and negatives to cover the variety of experimental conditions (see Table 2) [Forman]. These datasets contain from 2000 to 31,000 binary word features.

Learning algorithms: The bulk of our experiments were conducted with linear Support Vector Machines (SVM), which is considered state of the art for text classification. We later replicated the experiments for Naïve Bayes and Multinomial Naïve Bayes [McCallum]. Given that feature selection has shown to improve SVM results, we also replicated the experiments with feature selection via Bi-Normal Separation [Forman].

Error metrics: In order to be able to average across tasks with different numbers of positives, a natural error metric is the estimated percent positive minus the actual percent positive. By averaging across conditions, we can determine whether a method has a positive or negative bias. But suppose a method guesses 0% or 100% randomly; it would also have a zero bias on average. Thus, it is also important to consider an unsigned measure of error. Absolute error is one candidate, but estimating 41% when the ground truth is 45% is not nearly as 'bad' as estimating 1% when the ground truth is 5%. For this reason, 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 always yields zero error. Hence, the normalized cross-entropy is defined as follows:

$$\begin{aligned} \text{normCE}(p,x) &= \text{CE}(p,x) - \text{CE}(p,p) \\ \text{CE}(p,x) &= -p \log_2(x) - (1-p)\log_2(1-x) \end{aligned} \quad \textbf{(4)}$$

where x is the estimate of the actual percent positives p in testing. Since cross-entropy goes to infinity as x goes to 0% or 100%, if a method estimates zero positives, we adjust its estimate to half a count out of the entire test set for purposes of evaluating its normalize cross-entropy error. Likewise, if a method estimates that all the test items are positive, we back it off by half a count. (Note that this error metric will increasingly penalize a method for estimating zero positives as the test set size

```
for each of 21 benchmark tasks—distinguishing positive class c in dataset d:
|  set aside 200 positives and 1000 negatives for training, the rest for testing
|  for P = 10,20,50,100,200 training positives:
|     for N = 100,200,500,1000 training negatives:
|     |  for each classifier C: SVM, NB, NBM; with and without feature selection
|     |     train C on training set (size P+N)
|     |     perform 50-fold cross-validation to estimate TPR, FPR, $D_+$ and $D_-$
|     |     for p = 5% to 95% by 5%:
|     |     |  select maximal test set such that p% are positive
|     |     |  apply C to test set
|     |     |  for each of the methods:
|_    |_    |_    estimate x% positives, and record result & error.
```

Fig. 4. Overall experiment procedure in pseudo-code

grows. Intuitively it is worse to estimate zero positives among thousands of test cases than among ten.)

Cross-validation folds for calibration: The adjustment method and the mixture model both require cross-validation to generate calibrated values during training. We chose f=50 folds for this study. (Note that if there are fewer than f positives in the training set, some of the test folds will contain no positives. We experimented with using only min(f,P,N) folds, but generally found no improvement.)

Experiment Procedure: The overall experiment procedure is shown in Fig. 4. In total there were over 20,000 experiment jobs consuming over a hundred CPU days. These ran in parallel on the HP Utility Data Center in a few days. We used the WEKA software in Java to provide the base classifiers [Witten].

5 Experiment Results

We begin by evaluating how the estimates of each method are biased by the composition of the training set, holding the test set fixed. Fig. 5 shows each method's estimate, averaged over all benchmark tasks, as we vary the training set. The test set for each task is fixed with p=20% positives. (Except where stated otherwise, hereafter the classifier is a linear SVM with all features, calibrated via 50-fold cross-validation on the training set.) Though MM tends to overestimate by a small amount, it is

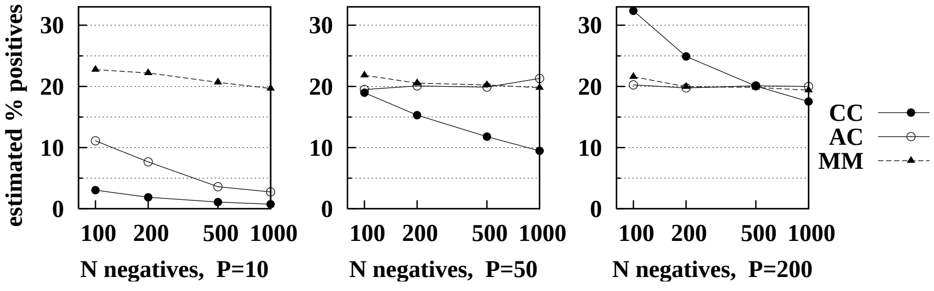

Fig. 5. Sensitivity to training set composition, with actual test positives p=20%

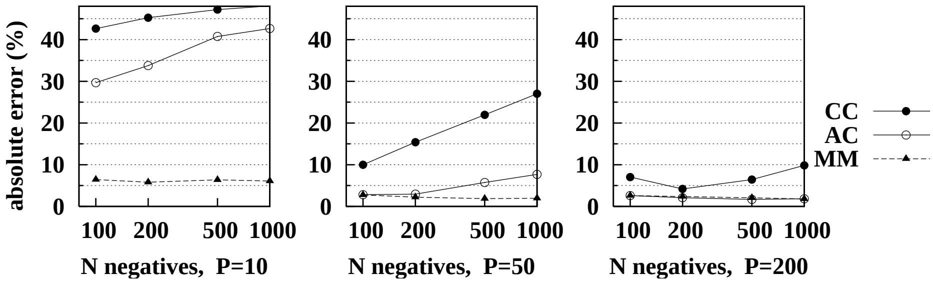

Fig. 6. Absolute error averaged over all test conditions p=5–95% and all tasks

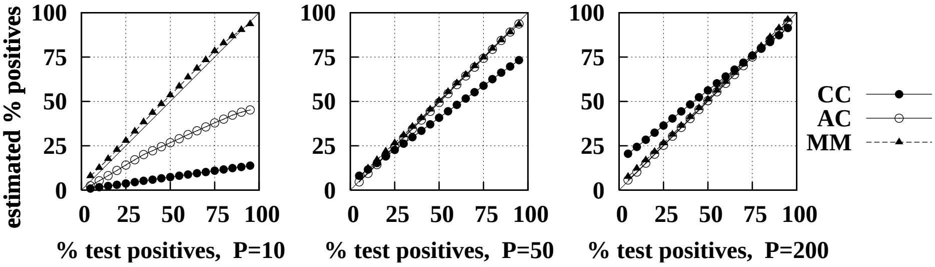

Fig. 7. Ability to track % testing positives, training with N=100 negatives

striking that it is so close to the target p=20% when there are only P=10 training positives. CC and AC are not competitive with P=10, and they underestimate more strongly as the imbalance of training negatives N increases. In contrast, additional training negatives help MM converge for each value of P.

With more positives, we see AC also converge to 20%. But more training data do not make CC converge—it is always tuned for a specific percentage of positives. Popular wisdom suggests the best classification performance for this test set should be when the training distribution matches 20%; however, CC performed best with ~30% training positives (P:N = 50:100, 200:500, and, not shown, 100:200). Hence, were it magically possible to always match the training class distribution to that of testing, it would still not make CC an effective method of counting positives.

The results presented so far were for a single fixed percentage of test positives. Next we average over all tasks and all testing situations: p=5%–95% positives. Instead of averaging the error, which reveals positive or negative bias, we average the absolute error to determine how far the estimate lies from the true answer on average. Fig. 6. shows this as we vary the training set as before. Again, MM performs surprisingly well given only P=10 training positives, achieving ~6% absolute error on average, regardless of the number of training negatives N. With P=50 or 200 training positives, MM achieves ~2% absolute error on average. With P=200 training positives, AC is competitive. Recall the ideal method should be as insensitive as possible to the training set size and class distribution: strongly recommending MM.

Next we hold the *training* set fixed and measure the ability of each method to track various percentages of positives in the test set. Fig. 7 shows for each method the estimate, averaged across all benchmark tasks, as we vary the percentage of positives

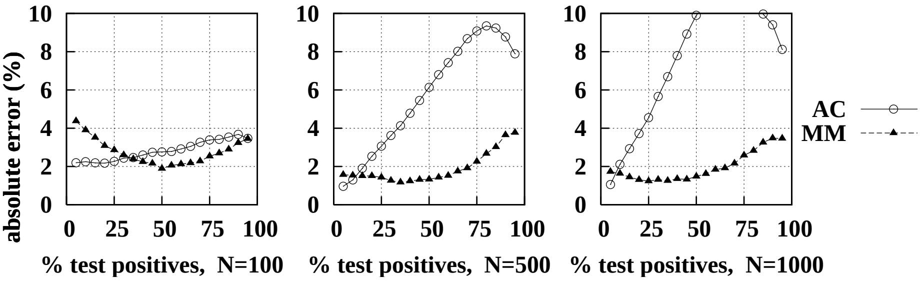

Fig. 8. Absolute error averaged over all tasks, comparing AC and MM at P=50

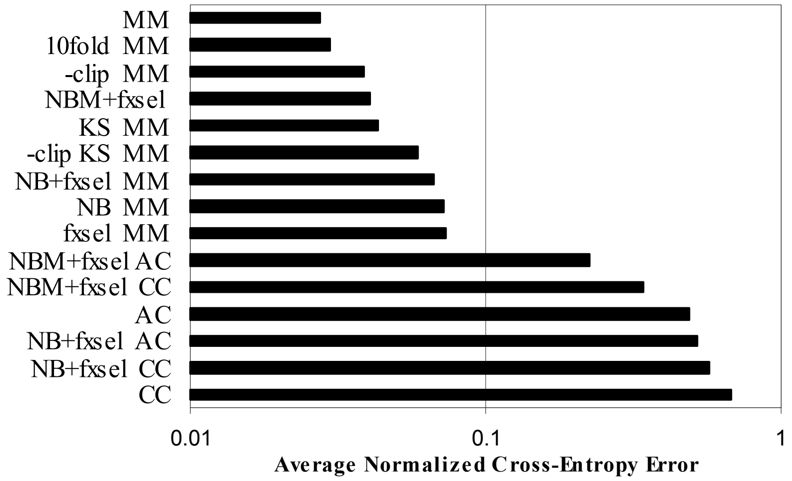

Fig. 9. Lesion study results, averaged over tasks, training, and testing situations

in the test set p=5% to 95%. We use N=100, which was least favorable to MM in Fig. 5. For P=10 we see that MM is alone effective and AC becomes competitive with enough positives. MM shows a slight positive bias.

While the view in Fig. 7 gives a good overview of the biases, next we zoom in to examine the differences in absolute error between MM and AC. We show this only for P=50, as the results are uninteresting at P=10 and undifferentiated at P=200. We see a consistent picture in Fig. 8. When the testing set contains a high percentage of positives, MM estimates better than AC, especially as the number of negative training examples grows for a fixed number of training positives. However, when testing with a small percentage of positives, the MM estimates worse than AC, especially with few training negatives. Recall that MM exhibits a small but consistent positive bias; this systematically hurts its estimate when there is a small percentage of testing positives.

Lesion Study: The MM method includes a number of design choices. We performed a lesion study to evaluate other choices. Fig. 9 summarizes the effect of each of these independent changes by showing the normalized cross-entropy averaged over all benchmark tasks, all training set compositions, and all test situations. In every case, the changes resulted in worse estimation (our choices were made prior to the study, with the exception of range clipping, which was suggested by preliminary results). We describe these lesions in ranked order: **10fold MM** uses 10-fold cross-validation, rather than 50-fold—a small loss in performance for one fifth the training time, *if* that one-time computational cost is significant in one's application. **-clip MM** does not use range clipping—clipping benefits substantially when there are few training positives. **NBM+fxsel MM** uses the Multinomial Naïve Bayes model with feature selection, in place of SVM; without feature selection NBM failed sometimes. **KS MM** uses the standard Kolmogorov-Smirnov statistic in place of our PP-Area metric; **-clip KS MM** is similar, but without clipping as well. **NB+fxsel MM** uses Naïve Bayes with feature selection, instead of SVM; **NB MM** uses Naïve Bayes with all features. **fxsel MM** adds feature selection to SVM. (Whenever feature selection was applied, the 200 best features were selected via Bi-Normal Separation.)

For comparison, we also include the other major methods **AC** and **CC**. These are also shown with alternate learning methods that generated a somewhat better balance between false positives and false negatives. Although they improve the estimates, they are not competitive with MM or any of its variants.

6 Discussion

In most machine learning research, where the objective is accurate classifications, each item in the test set provides an additional test result, which may contribute to an average. In contrast, a single test item by itself is not sufficient for evaluating a method for quantification. This must be done on an entire *batch* of test items at a time, and yields only a single scalar estimate. For this reason, evaluation requires concocting many different test situations over many benchmark tasks. In this way, research on counting requires more experimental design. We hope that the test conditions we designed provide useful guidance for others. In our framework, we varied the percentage of test positives from p=5%–95% as a reasonable experimental gamut. However, we recognize that in many situations the estimator will not be called on to span this entire gamut well. For the common situation of rare positives, it may be that other methods will excel. But obtaining statistical significance in this realm may prove difficult, as it requires a much larger benchmark to evaluate the tails properly.

Stepping outside the box of machine learning, another way to estimate the positives in a population is to have a person—an expensive, slow, but presumably perfect classifier—manually count in a random sample. Let us sketch the labor required for comparison. Supposing you wish to have the width of the 95% confidence interval be half of the estimate. For example, if the estimate were 40%, then the person would need to classify 100 items to get the confidence interval down to 40%±10%. Alternately, for a confidence interval of 4%±1% one would need to examine 1500 items—the labor increases greatly and non-linearly for rarer classes of positives. For obtaining a single count, certainly the labor may be dwarfed by the effort to set up a machine learning solution. But if a count or many such counts must be performed every day, the complexity of a computerized solution may be quickly amortized. Likewise, if hundreds or thousands of different classes need to be counted just once, again the machine learning solution can greatly reduce the effort spent by the person to classify items. Indeed, the manual classification can serve both as a rough estimate, and as a training set for a machine learning quantifier that can examine the complete dataset or can be applied to next month's dataset.

7 Conclusion

This paper highlights the problem of assessing the number of positives in a population via machine learning—quantification. It is a valuable real-world task, but is commonly overlooked for the natural goal of improving classification accuracy. The issue is made invisible by common machine learning research methodology, which selects the training set and testing set so the class distribution is the same. We laid out an evaluation framework, which varies the training and testing distributions independently to determine which method minimizes error measured as normalized cross-entropy. We described and evaluated three methods. The straightforward and probably most common method of classifying and counting positives should only be used when an extremely accurate classifier can be learned with available training data.

Opportunities for future work include: evaluating non-text benchmark domains, extending to multi-class classification tasks, and inventing superior methods. We identified two avenues for future work which are ill-conceived and cannot succeed: calibrating the threshold at training time and calibrated probability estimation. Finally, the most successful methods can be folded back in to calibrate classifiers at testing time to improve their accuracy or probability estimation.

References

1. Bennett, P.: Using Asymmetric Distributions to Improve Text Classifier Probability Estimates. Proc. ACM SIGIR Conference on Research and Development in Information Retrieval, July-August (2003)
2. Fawcett, T.: ROC graphs: Notes and practical considerations for data mining researchers. Tech report HPL-2003-4. Hewlett-Packard Laboratories, Palo Alto, CA, USA (2003)
3. Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification. Journal of Machine Learning Research 3 (2003) 1289-1305
4. McCallum, A., Nigam, K.: A Comparison of Event Models for Naive Bayes Text Classification. AAAI/ICML Workshop on Learning for Text Categorization (1998) 41-48
5. Weiss, G., Provost, F.: Learning when Training Data are Costly: The Effect of Class Distribution on Tree Induction. J. of Artificial Intelligence Research 19 (2003) 315-354
6. Witten, I.H., Eibe Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco (2000)