Neighborhood-Based Local Sensitivity

Paul N. Bennett

Microsoft Research, One Microsoft Way, Redmond WA 98052, USA
paul.n.bennett@microsoft.com

Abstract. We introduce a nonparametric model for sensitivity estimation which relies on generating points similar to the prediction point using its k nearest neighbors. Unlike most previous work, the sampled points differ simultaneously in multiple dimensions from the prediction point in a manner dependent on the local density. Our approach is based on an intuitive idea of locality which uses the Voronoi cell around the *prediction point*, *i.e.* all points whose nearest neighbor is the prediction point. We demonstrate how an implicit density over this neighborhood can be used in order to compute relative estimates of the local sensitivity. The resulting estimates demonstrate improved performance when used in classifier combination and classifier recalibration as well as being potentially useful in active learning and a variety of other problems.

1 Introduction

Consider the following tasks often faced during peer review: (1) Make a recommendation accept/reject; (2) Rate from 0 to 5, where 0 is definitely reject and 5 is definitely accept; (3) State your confidence on a 0 to 5 scale in your review. When a reviewer answers the first question, he is classifying the paper. The answer to the second question is an implicit measure of the posterior probability regarding "Accept/Reject". As with a posterior and a classification decision, a consistent reviewer can use the rating to provide information that both summarizes and subsumes the classification decision.

Next, the reviewer states his confidence — which intuitively is a self-assessment of his expertise (previous familiarity with topic, perceived completeness of study, *etc.*) and mathematically is a statement about how strongly he believes the posterior he gave is correct. In other words it is a second-order summary of the uncertainty the reviewer has about his classification.

Given only the feedback and rating from a single reviewer, we cannot use this secondary confidence information if immediately forced to make a decision. However, if one of our choices is to consult another expert, then suddenly we are faced with a *value of information* problem, and presumably, we will consult an additional expert when the confidence of the first expert is low. Likewise when combining the opinions of several reviewers, instead of directly averaging the acceptance ratings we could weight them by confidence or treat low confidence reviews as abstentions regardless of the rating.

Furthermore, given access to a past history of the reviewer's ratings and the final decisions regarding the papers, we might conclude this reviewer is either too

harsh or too lenient and opt to post-calibrate the reviewer's ratings. Additionally, the calibration function may vary depending on confidence; for example, the reviewer may be perfectly calibrated when his confidence is high but increasingly uncalibrated as his confidence decreases.

Finally, when attempting to improve his own ratings, the reviewer may opt to perform active learning by requesting what the decisions should be from an oracle. However, rather than simply requesting decisions on papers that have borderline ratings (some papers simply are borderline!) he could request decisions where his confidence is low to identify areas where his expertise could be improved.

Similarly, confidence information can be used in similar ways for value of information estimation, classifier combination, post-learning recalibration, and active learning when provided by automated systems. However, most previous work has targeted estimating only the posterior at a particular point rather than estimating both the posterior and confidence. This paper partially addresses that limitation by demonstrating the link between sensitivity estimates of the stability or rate of change of the learned posterior function and the confidence in the posterior. Rather than obtaining sensitivity estimates by tweaking a point in a single dimension or independently in several dimensions, we use the neighborhood around the prediction point and sample from this neighborhood in a way that exploits the local covariance of the input and is not constrained to relying on global estimates of the covariance. The resulting estimates are demonstrated to be of use in both post-learning recalibration over synthetic datasets and in combining classifiers over text classification corpora.

Before explaining our approach for sensitivity estimation, we first further distinguish the concepts of posterior and confidence in the posterior and demonstrate the latter's relationship to sensitivity. Next, we motivate and explain the use of neighbors to estimate the sensitivity and related terms of a learned model. Using several synthetic datasets, we demonstrate how these quantities can be used to improve post-learning recalibration of the estimates provided by the kNN classifier. We then summarize several ensemble learning experiments over text classification corpora where these quantities play a key role. Finally, we conclude with a discussion of related work and a summary of our contributions.

2 Variance and Sensitivity

Returning to our motivating example, consider if instead of making the reviewer specify his uncertainty, we allowed him to specify an entire distribution expressing his belief, $p(P(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})$, where $\boldsymbol{x}$ is the item being classified, c is a particular class, and z is a specific value of the random variable $P(c \mid \boldsymbol{x})$. Then when he is asked to summarize his uncertainty via a rating, the typical approach is to predict the expected value of the distribution: $\hat{P}(c \mid \boldsymbol{x}) = \int z\, p(P(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})\, dz$.

However, the mean of a distribution does not fully summarize the distribution. Presumably, as the reviewer receives more information or perceives he has

all necessary information because of his expertise, his confidence that the expected value fully summarizes his uncertainty will become quite high. Therefore a reasonable measure for confidence is to treat it as an (inverse) measure of the variance of $p(P(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})$. Since "confidence" is often used in the literature to refer to the posterior distribution estimate, $\hat{P}(c \mid \boldsymbol{x})$, we will avoid this confusion for the remainder of the paper by referring to the "posterior distribution", $P(c \mid \boldsymbol{x})$, and the "confidence distribution", $p(P(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})$.

Rather than estimating the variance of the true posterior according to the confidence distribution, $p(P(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})$, we will instead consider the variance of the output of the classifier under the distribution $p(\hat{P}(c \mid \boldsymbol{x}) = z \mid \boldsymbol{x})$. The variance of this second distribution is the local *sensitivity* of the classifier, and we will demonstrate how it can be useful in several of the scenarios where we previously motivated the importance of the confidence distribution.

Consider again observing a past history of the reviewer's ratings and the final decisions on a set of papers in order to learn a recalibration function for the reviewer's ratings. For the remainder of the paper, it will be convenient to work with log-odds estimates, $\log \frac{\hat{P}(c|\boldsymbol{x})}{1-\hat{P}(c|\boldsymbol{x})}$. However, the derivation below holds for any function $\hat{\lambda}(\boldsymbol{x})$ such that $\mathrm{VAR}_{\Delta}\left[\hat{\lambda}\right] \neq 0$ and thus can be used for computing the sensitivity of a variety of functions. Now, given our reviewer's uncalibrated estimate of the acceptance rating $\hat{\lambda}$, we will attempt to recalibrate it with a locally weighted recalibration function:

$$\hat{\lambda}^*(\boldsymbol{x}) = W_0(\boldsymbol{x}) + W_1(\boldsymbol{x})\hat{\lambda}(\boldsymbol{x}). \tag{1}$$

We can determine the optimal weights in a simplified case by assuming we are given "true" log-odds values, λ, and a family of distributions $\boldsymbol{\Delta}_{\boldsymbol{x}}$ such that $\boldsymbol{\Delta}_{\boldsymbol{x}} = p(\boldsymbol{s} \mid \boldsymbol{x})$ encodes what is local to $\boldsymbol{x}$ by giving the probability of drawing a point $\boldsymbol{s}$ near to $\boldsymbol{x}$. We use $\boldsymbol{\Delta}$ instead of $\boldsymbol{\Delta}_{\boldsymbol{x}}$ for notational simplicity. Since $\boldsymbol{\Delta}$ encodes the example-dependent nature of the weights, we can drop $\boldsymbol{x}$ from the weight functions. To find weights that minimize the squared difference between the true log-odds and the estimated log-odds in the $\boldsymbol{\Delta}$ vicinity of $\boldsymbol{x}$, we can solve a standard regression problem, $\mathrm{argmin}_{w_0,w_1} E_{\boldsymbol{\Delta}}\left[\left(w_1\,\hat{\lambda} + w_0 - \lambda\right)^2\right]$. Under the assumption $\mathrm{VAR}_{\boldsymbol{\Delta}}\left[\hat{\lambda}\right] \neq 0$, this yields:

$$w_0 = E_{\boldsymbol{\Delta}}[\lambda] - w_1 E_{\boldsymbol{\Delta}}\left[\hat{\lambda}\right] \qquad w_1 = \frac{\mathrm{COV}_{\boldsymbol{\Delta}}[\hat{\lambda},\lambda]}{\mathrm{VAR}_{\boldsymbol{\Delta}}[\hat{\lambda}]} = \frac{\sigma_{\lambda}}{\sigma_{\hat{\lambda}}}\rho_{\lambda,\hat{\lambda}} \tag{2}$$

where σ and ρ are the standard deviation and correlation coefficient under $\boldsymbol{\Delta}$, respectively. The first parameter w_0 is a measure of calibration that addresses the question, "How far off on average is the estimated log-odds from the true log-odds in the local context?" The second parameter w_1 is a measure of correlation, "How closely does the estimated log-odds vary with the true log-odds?" Note that the second parameter depends on the local sensitivity of the base classifier, $\mathrm{VAR}_{\boldsymbol{\Delta}}^{1/2}\left[\hat{\lambda}\right] = \sigma_{\hat{\lambda}}$. Although we do not have true log-odds, we *can* introduce local density models to estimate the local sensitivity of the model.

3 Neighborhood-Based Locality

To compute the local sensitivity, we define a simple nonparametric method based on the k nearest neighbors. Since we are concerned with how the decision function changes locally around the current prediction or *query* point, it is natural to use a neighborhood-based definition. In particular, consider all points in the input space whose nearest neighbor is the query (*i.e.* the Voronoi cell of the query).

Next, we can either explicitly define a density over the Voronoi cell of the query or implicitly do so by defining a procedure that allows us to draw S samples from the cell. We do the latter by finding k neighbors of the query $\boldsymbol{x}$ in the training set and then draw $S = k$ samples by using the difference vector between the query and its neighbor to interpolate a sample point where the vector intersects the boundary of the cell. This is illustrated in Figure 1.

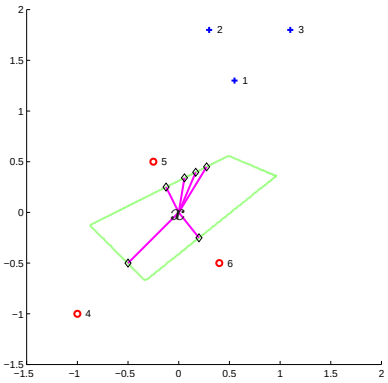

Fig. 1. Given the numbered points as the neighbors of the query point $\boldsymbol{x}$, the figure illustrates the points that would be sampled within the Voronoi cell of $\boldsymbol{x}$.

There are multiple advantages to defining the points to sample within the Voronoi cell in terms of the neighbors of the query point. First, we need not explicitly compute the Voronoi cell of the query point. Instead after determining the k neighbors, we can directly compute the sample points. Second, by using the k neighbors to guide sampling, we sample in locally dense directions. For example in Figure 1 we do not sample by shifting the query point directly to the right because no data supports variance in this direction.

More formally, we can consider this as shifting $\boldsymbol{x}$ toward each one of k neighbors with equal probability. Let $\boldsymbol{x}_i$ denote the query after it has been shifted by a factor β_i toward its ith neighbor, $\boldsymbol{n}_i$. That is, $\boldsymbol{x}_i = \boldsymbol{x} + \beta_i(\boldsymbol{n}_i - \boldsymbol{x})$. To determine β_i we choose the largest β_i such that the closest neighbor to the new point is the original example. Thus, the new point cannot be more than halfway to the neighbor, and β_i will not exceed 0.5. Furthermore, we can find it within a small ϵ efficiently using a simple bisection algorithm.

In terms of computational cost, since every bisection step halves the remaining range of β_i, this will terminate in at most $\lceil \log_2 \frac{0.5}{\epsilon} \rceil$ iterations. Thus to find β_i within $\epsilon = 0.001$ for a given neighbor would require at most 9 iterations. Furthermore, for the standard Euclidean space, only neighbors closer than the neighbor we are currently interpolating between can have an insulating effect that reduces β_i. Thus, we need only check our candidate interpolation point against the closer neighbors to see if we must move the interpolated point. Therefore, after finding the k neighbors, the sample points can be computed relatively efficiently.

Returning to our quantities of interest, let Δ be a uniform point-mass distribution over the shifted points. Given this definition of Δ and an output function or classifier $\hat{\lambda}$, it is straightforward to compute $\mathrm{VAR}_{\Delta}\left[\hat{\lambda}\right]$ and $E_{\Delta}\left[\hat{\lambda}\right]$. Rather than computing exactly these quantities, for comparison to previous work we compute the closely related quantities $\mathrm{VAR}_{\Delta}\left[\hat{\lambda}(\boldsymbol{s}) - \hat{\lambda}(\boldsymbol{x})\right]$ and $E_{\Delta}\left[\hat{\lambda}(\boldsymbol{s}) - \hat{\lambda}(\boldsymbol{x})\right]$. Note, since $\hat{\lambda}(\boldsymbol{x})$ is constant with respect to Δ, $\mathrm{VAR}_{\Delta}\left[\hat{\lambda}\right] = \mathrm{VAR}_{\Delta}\left[\hat{\lambda}(\boldsymbol{s}) - \hat{\lambda}(\boldsymbol{x})\right]$ and $E_{\Delta}\left[\hat{\lambda}(\boldsymbol{s}) - \hat{\lambda}(\boldsymbol{x})\right] = E_{\Delta}\left[\hat{\lambda}(\boldsymbol{s})\right] - \hat{\lambda}(\boldsymbol{x})$.

Although based on the k nearest neighbors, the neighborhood-based sensitivity estimates can be used to estimate the sensitivity of any function. All that is required is an input space where interpolation is meaningful and a function $\hat{\lambda}$ that can be applied to any point in the input space. For example, the input space could be the bag-of-words representation of documents in a text corpus and $\hat{\lambda}$ could be the margin score using a linear model learned by an SVM algorithm.

4 Empirical Analysis

To understand the impact sensitivity estimates can have in practice, we studied the use of the estimates in two settings: post-learning recalibration of the output of the kNN classifier and combining classifiers for text classification. The bulk of our analysis focuses on the recalibration task to isolate the behavior of the neighborhood-based estimates. We then summarize the ensemble learning work which uses neighborhood-based estimates as well as other similar quantities to improve text classification — full details of the ensemble study are in [1].

4.1 Post-Learning Recalibration

The post-learning recalibration problem attempts to take either poor probability estimates from a classifier or a more general rating and learn a function that will output good probability estimates or an improved thresholding [2–5]. In Platt recalibration [2], we perform nested cross-validation over the training set to collect a set of data for which we know both the class labels and model predictions. These collected pairs $\left\langle \hat{\lambda}(\boldsymbol{x}), c(\boldsymbol{x}) \right\rangle$, where $c(\cdot)$ denotes the actual class of the example, are then used as a training set by logistic regression to learn a and b such that our final recalibrated estimate is:

$$\hat{\lambda}^*(\boldsymbol{x}) = a\hat{\lambda} + b \tag{3}$$

As noted above, we can compute neighborhood-based sensitivity estimates for a variety of classifiers, but for this study we will focus on the case where the classifier to be recalibrated is the kNN classifier. Our expectation is that a kNN classifier that is input to a local recalibration function should perform well in many sets where adaptive kNN methods work well.

Given the similarity between Equation 3 and Equation 1, an obvious experiment is one in which we compare the performance of three systems:

1. kNN — use the estimates from kNN directly;
2. Recalibrated kNN — Use the log-odds estimates from kNN obtained by nested cross-validation as inputs to logistic regression to produce improved estimates that depend only on the original kNN estimations;
3. Sensitivity Recalibrated — Use both the log-odds estimates from kNN and sensitivity-related estimates obtained by nested cross-validation as inputs to logistic regression to produce improved estimates that use both the original estimates and the sensitivity estimates.

Synthetic Datasets Friedman [6] introduced a series of synthetic datasets to highlight the differences between various flexible kNN approaches and other adaptive methods. These datasets form an interesting case study since we expect a locally recalibrated kNN classifier to demonstrate many of the strengths of adaptive kNN methods. We note that Friedman's work and this work could complement each other by extending the work here to more general metric spaces and learning a metric before estimating the sensitivity using that metric.

We present results here for the first five synthetic problems in [6]. Each problem is binary classification and varies as to whether each dimension is equally informative. Because these are binary classification tasks, we refer to the positive and the negative class. We use the same number of training/testing points per problem as used in Friedman's original study. Ten train/test sets are drawn for each problem and the error averaged across the ten runs is computed. Since we expect the base kNN to produce poor log-odds estimates, we do not compare probability quality and instead compare improvement in error.

Problem 1 The number of input dimensions equals ten, $d = 10$. A class label is drawn with $P(c = +) = P(c = -) = 0.5$. For the negative class, the examples are drawn from a standard normal $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{1})$. For the positive class the examples are drawn from a normal $\boldsymbol{x} \sim N(\mu, \boldsymbol{\Sigma})$ where $\mu_i = \frac{\sqrt{i}}{2}$, $\boldsymbol{\Sigma}_{ii} = \frac{1}{\sqrt{i}}$, and $\boldsymbol{\Sigma}_{ij} = 0$ for $i \neq j$ and $i, j = 1, \ldots, d$. Thus the higher number dimensions have both means that are further separated and variances that are smaller and are generally more informative than the lower dimensions. For each run, 200 training and 2000 testing points are generated.

Problem 2 The number of input dimensions equals ten, $d = 10$. A class label is drawn with $P(c = +) = P(c = -) = 0.5$. For the negative class, the examples are drawn from a standard normal $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{1})$. For the positive class the examples are drawn from a normal $\boldsymbol{x} \sim N(\mu, \boldsymbol{\Sigma})$ where $\mu_i = \frac{\sqrt{d-i+1}}{2}$, $\boldsymbol{\Sigma}_{ii} = \frac{1}{\sqrt{i}}$, and

$\mathbf{\Sigma}_{ij} = 0$ for $i \neq j$ and $i, j = 1, \ldots, d$. Thus the higher number dimensions have means that are closer together, but the variance binds the values more closely to the means. Whereas, the lower dimensions have means that are well separated but also have higher variances. Thus all dimensions are informative. For each run, 200 training and 2000 testing points are generated.

Problem 3 The number of input dimensions equals ten, $d = 10$. All examples are drawn from a standard normal $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{1})$ and are labeled with a class by the rule: if $\sum_{i=1}^{d} \frac{x_i^2}{i} \leq 2.5$ then negative else positive. The resulting class distribution marginalizes to approximately $P(c = +) = 0.51$. Unlike the previous two problems, the optimal error here is zero. Again, the dimensions contribute unequally to the final determination of the class label although the values of the features vary uniformly in the space. The decision surface is quadratic. For each run, 200 training and 2000 testing points are generated.

Problem 4 The number of input dimensions equals ten, $d = 10$. All examples are drawn from a standard normal $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{1})$ and are labeled by the rule: if $\sum_{i=1}^{d} x_i^2 \leq 9.8$ then negative else positive. The resulting class distribution marginalizes to approximately $P(c = +) = 0.46$. Again the optimal error here is zero, but now the dimensions contribute equally to the final determination of the class label. For each run, 500 training (instead of 200) and 2000 testing points are generated.

Problem 5 The number of input dimensions equals ten, $d = 10$. All examples are drawn from a standard normal $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{1})$ and are labeled with a class by the rule: if $\sum_{i=1}^{d} x_i \leq 0$ then negative else positive. The resulting class distribution is $P(c = +) = P(c = -) = 0.5$. Again the optimal error here is zero, and the dimensions contribute equally to the final determination of the class label. The decision surface is linear. For each run, 200 training and 2000 testing points are generated.

kNN Classifier We use a standard way of performing a distance-weighted vote of the neighbors to compute the output for the kNN classifier [7]. k is set to be $2\lceil \log_2 N \rceil + 1$ where N is the number of training points.[1] The score used as the uncalibrated log-odds for being in a class y is:

$$\hat{\lambda}_{k\mathrm{NN}}(\boldsymbol{x}) = \sum_{\boldsymbol{n} \in k\mathrm{NN}(\boldsymbol{x}) | c(\boldsymbol{n}) = y} K(\boldsymbol{x}, \boldsymbol{n}) \quad - \quad \sum_{\boldsymbol{n} \in k\mathrm{NN}(\boldsymbol{x}) | c(\boldsymbol{n}) \neq y} K(\boldsymbol{x}, \boldsymbol{n}). \tag{4}$$

where K is a kernel (similarity) function. For text classification, this is typically cosine similarity. For the synthetic datasets, we use a Gaussian RBF kernel with $\sigma = 1$ since the problems imply a translation invariant kernel is desirable. It is reasonable to expect this score to behave like an uncalibrated log-odds estimate

[1] This rule is motivated by theoretical results which show such a rule converges to the Bayes optimal classifier as the number of training points increases [8].

since it is similar to SVM's margin score, $\sum \alpha_i y_i K(\boldsymbol{s}_i, \boldsymbol{x})$, which has been shown to empirically behave like an uncalibrated log-odds estimate [2, 4].

Efficient Approximation Because identifying the k neighbors of a point in kNN is computationally expensive, we desire a more efficient approach than separate neighbor retrievals for the $k+1$ classifications. In this case, we perform a single retrieval around the query point to obtain a cache of size $K \geq k$ such that the k neighbors of a sample point are approximated by taking the k closest of the K. We use $K = 2k$ as a heuristic derived from the fact that retrieving K neighbors by finding all points within twice the *radius*, r, of the original example (the farthest neighbor of the k from the query) would guarantee that the k closest neighbors of a sample point are contained within the K. This is because the sample point is within $0.5r$ of the original example, and therefore the original query's k neighbors are within $1.5r$ of the sample. Thus, the sample has radius $\leq 1.5r$ and therefore $2r$ from the original will contain all its neighbors.

For benchmarking, we ran over a text classification corpus using a kNN algorithm with an inverted index, the same algorithm but retrieving $2k$ neighbors, and the version that computes both predictions and sensitivity estimates. The experiments were run on a machine with an Intel Pentium® 4 CPU, 3GHz clock speed, and 1 GB of RAM. As can be seen in Table 1, a slowdown of three times the baseline is experienced instead of the naïve slowdown of $k + 1 = 30$ times the baseline. For the synthetic experiments below, we do not use this caching approximation, but for the experiments over the text corpora we do.

Table 1. Effect on running time of computing the kNN sensitivity estimates for the Reuters 21578 corpus (9603 training examples, 3299 testing examples, 900 features).

Method	Total Run Time (s)	Ratio to Baseline
Sparse *(k)*	69.1	1
Sparse *(2k)*	80.07	1.16
Sparse w/Sensitivity *(2k)*	196.37	2.84

Recalibration Results and Discussion Table 2 summarizes results over the synthetic datasets giving the average error for each method over the ten runs as well as the standard deviation across runs. Table 3 presents a summary of the two recalibration methods in terms of the relative reduction in error they yield over the baseline kNN method. When the best result is statistically significant according to a two-tailed paired t-test with p=0.01, it is underlined.

To ensure that our baseline error is not artificially poor, we list the average error reported by Friedman for his baseline kNN method. Our baseline performs either approximately the same or far better (Problems 1 and 2); the latter probably results from our use of a distance-weighted kNN vote rather than Friedman's unweighted vote.

Examining the performance of Recalibrated kNN, we see that simple recalibration gains large reduction in errors in all problems but one — Problem 5

where it loses slightly relative to the baseline. Examining the Sensitivity Recalibrated results, we see that not only does this method always achieve large wins over the baseline method, but it also yields large improvements over Recalibrated kNN in all but one case — Problem 4 where it improves over Recalibrated kNN by only a slight amount.

In the fifty different runs (10 runs * 5 problems), the Sensitivity Recalibrated method only has a worse error on 3/50 runs vs. Recalibrated kNN and on 4/50 runs vs. kNN; furthermore, no more than one of these runs occurs on the same Problem (explaining the paired t-test significance). Relative to the baseline, the Sensitivity Recalibrated method reduces the error anywhere from 13% to 65%. Altogether, the evidence argues that not only are the sensitivity estimates useful in recalibrating kNN, but they provide information beyond that which is captured in the original log-odds estimates.

Table 2. The average error of each method in the synthetic datasets as well as one standard deviation. The best result in each problem is in bold.

	Problem 1		Problem 2		Problem 3		Problem 4		Problem 5	
	Error	Stdev	Error	Stdev	Error	Stdev	Error	Stdev	Error	Stdev
Friedman kNN [6]	13.2	—	10.3	—	35.9	—	34.0	—	17.4	—
kNN	2.57	0.2973	2.81	0.5868	36.22	1.7335	38.16	0.6919	17.07	1.5592
Recalibrated kNN	2.23	0.6601	2.40	0.3752	21.81	0.9459	21.52	1.4870	18.00	1.9490
Sensitivity Recal.	**1.33**	0.2679	**1.81**	0.3354	**20.97**	1.0061	**13.19**	0.9562	**14.85**	1.6264

Table 3. Relative Reduction in Error, 1 - Error(Method) / Error(kNN).

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Recalibrated kNN	0.13	0.15	0.40	0.44	-0.05
Sensitivity Recal.	0.48	0.36	0.42	0.65	0.13

4.2 Local Classifier Combination

We have seen that sensitivity estimates can boost the performance of recalibration techniques. However, it seems likely that they should also be useful in classifier combination. Returning to our peer review problem, when faced with opinions from three reviewers, we might want the weight on a reviewer's acceptance rating to depend on the reviewer's confidence.

The STRIVE metaclassification approach [9] extended Wolpert's stacking framework [10] to use reliability indicators. In this framework a metaclassifier has access to both the outputs of base classifiers as well as some auxiliary reliability indicators. If each of the base classifiers outputs a log-odds like estimate, then a linear metaclassifier has a natural interpretation as both recalibrating and weighting each classifier according to its class-conditional information content [11, 1]. Note that this is the direct extension of the recalibration paradigm in the previous section to multiple classifiers. A metaclassifier that recalibrates a single

base classifier is now replaced by a stacking approach which applies a metaclassifier to the outputs of multiple base classifiers. Likewise recalibrating with the additional sensitivity information is now replaced by a striving approach which applies a metaclassifier to the outputs of multiple base classifiers and additional sensitivity information.

Our goal in this section is to illustrate that computing the neighborhood-based sensitivity estimates is both practical in "real-world" datasets and yields increased performance. To demonstrate this, we compare the performance of the base classifiers to both a stacked model of the base classifier outputs and to striving with the base classifier outputs and sensitivity information.

For the base classifier outputs, we obtain log-odds like estimates from five standard classifiers (kNN, linear SVM, decision trees, multivariate naïve Bayes, and multinomial naïve Bayes), and use a linear SVM as a metaclassifier to learn a stacked model. For the striving model, we apply a linear SVM metaclassifier to the base classifier outputs and the sensitivity estimates described above for kNN as well as a variety of other variables tied to sensitivity as motivated in [1]. Since an **S**VM is used as a metaclassifier in both cases, we refer to the stacking model as *Stack-S* and the striving model as *STRIVE-S*. Space prevents us from providing a full description of these experiments, but see [1].

We examined performance over several topic-classification corpora including: the *MSN Web Directory*; two corpora drawn from the *Reuters* newswire; and the *TREC-AP* corpus. We selected these corpora because they offer an array of topic classification problems from very broad topics over web pages to very narrow topics over financial news stories. In terms of scale, the number of training documents across these corpora varies from 9603 to 142791 while the number of testing documents varies from 3299 to 781265.

4.3 Results and Discussion

For space, only the base SVM — which typically performs best among the base classifiers — is explicitly listed. We note that our implementation of the base SVM is on par with the best results in previously reported literature [12, 9]. In Figure 2, we display the changes in the three components that determine error and F1: false positives, false negatives, and correct positives. Not only does *STRIVE-S* achieve considerable reductions in error of 4-18% (*left*) and 3-16% (*right*), but it also nearly always improves beyond those gains achieved by *Stack-S*. Furthermore, *STRIVE-S* never hurts performance relative to the *SVM* on these performance measures as *Stack-S* does over RCV1-v2 on the far left. Examining a micro-sign, macro-sign, and macro t-test, *STRIVE-S* significantly improves ($p < 0.05$) over the base SVM classifier (except for one collection and one performance measure, F1 for RCV1-v2), and over *Stack-S* for the majority of collections and performance measures (see [1] for more detail).

Finally, the STRIVE model augmented with sensitivity information not only demonstrates improved and robust performance across a variety of corpora, but in post-analysis, backward selection reveals that the primary variables of interest are the neighborhood-based sensitivity related estimates [1].

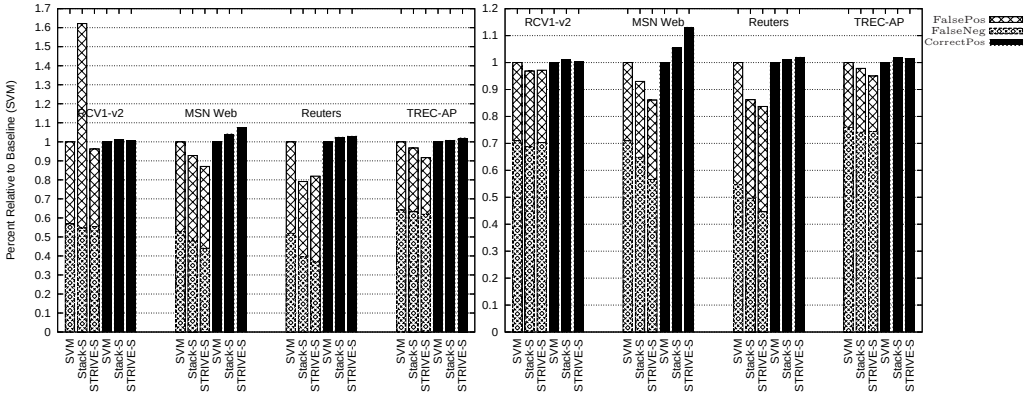

Fig. 2. For *Stack-S* and *STRIVE-S* change relative to the best base classifier — the *SVM* classifier — over all the topic classification corpora. On the *left*, we show the relative change using thresholds learned for F1, and on the *right*, we show the relative change using thresholds learned for error.

5 Related Work, Future Work, and Summary

In addition to the related work mentioned throughout the paper, several approaches merit mention. The STRIVE metaclassification approach [9] extended Wolpert's stacking framework [10] to use reliability indicators. In recent work, Lee *et al.* [13] derive variance estimates for naïve Bayes and tree-augmented naïve Bayes and use them in a combination model. Our work complements theirs by laying groundwork for how to compute variance estimates for models such as kNN that have no obvious probabilistic component in addition to being able to use the same framework to compute estimates for any classification models.

A variety of other work has examined stability and robustness related to sensitivity. Several researchers have attempted to obtain more robust classifiers by altering feature values [14] or training multiple classifiers over feature subsets [15]. Similarly, [16] compresses a more complex model into a simpler model by using the former to label instances generated by combining attributes of a point with its nearest neighbor. Finally, [17] contains a variety of related work that focuses on sensitivity analysis to determine the significance of input dimensions while [18] examines the stability of the algorithm with respect to small changes in the training set. Where much of the previous work focuses on change in a single dimension, we focus on characterizing the change around the current prediction point according to the local density across all dimensions.

There are a variety of interesting directions for future work. One of the most interesting directions is extending the neighborhood-based estimates for learned metrics. Additionally, exploring other methods of sampling from the Voronoi cell and characterizing how these methods differ in the quality of estimates they yield could be quite useful. Such a study could also analyze how critical staying within the Voronoi cell is. In our experiments, we found performance over the synthetic

datasets was less sensitive to staying within the Voronoi cell while it was more important over the text datasets. Formally characterizing this dimension will be important in advancing the understanding of sensitivity estimators.

In conclusion, we motivated the use of sensitivity information in a variety of different scenarios including active learning, recalibration, and classifier combination. We then demonstrated how a neighborhood-based sensitivity estimator can sample efficiently from the points in the input space near the prediction point to estimate sensitivity. Finally, the resulting estimates were demonstrated to be useful in improving performance in both recalibration over synthetic data and classifier combination over standard text classification corpora.

References

1. Bennett, P.N.: Building Reliable Metaclassifiers for Text Learning. PhD thesis, CMU (2006) CMU-CS-06-121.
2. Platt, J.C.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Smola, A.J., Bartlett, P., Scholkopf, B., Schuurmans, D., eds.: Advances in Large Margin Classifiers. MIT Press (1999)
3. Zadrozny, B., Elkan, C.: Reducing multiclass to binary by coupling probability estimates. In: KDD '02. (2002)
4. Bennett, P.N.: Using asymmetric distributions to improve text classifier probability estimates. In: SIGIR '03. (2003)
5. Zhang, J., Yang, Y.: Probabilistic score estimation with piecewise logistic regression. In: ICML '04. (2004)
6. Friedman, J.H.: Flexible metric nearest neighbor classification. Technical report, Department of Statistics, Stanford University (1994)
7. Yang, Y.: An evaluation of statistical approaches to text categorization. Information Retrieval **1** (1999) 67–88
8. Devroye, L., Györfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer-Verlag, New York, NY (1996)
9. Bennett, P.N., Dumais, S.T., Horvitz, E.: The combination of text classifiers using reliability indicators. Information Retrieval **8** (2005) 67–100
10. Wolpert, D.H.: Stacked generalization. Neural Networks **5** (1992) 241–259
11. Kahn, J.M.: Bayesian Aggregation of Probability Forecasts on Categorical Events. PhD thesis, Stanford University (2004)
12. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: Rcv1: A new benchmark collection for text categorization research. Journal of Machine Learning Research **5** (2004) 361–397 http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf.
13. Lee, C.H., Greiner, R., Wang, S.: Using query-specific variance estimates to combine bayesian classifiers. In: ICML '06. (2006)
14. O'Sullivan, J., Langford, J., Caruana, R., Blum, A.: Featureboost: A meta-learning algorithm that improves model robustness. In: ICML '00. (2000)
15. Bay, S.D.: Combining nearest neighbor classifiers through multiple feature subsets. In: ICML '98. (1998)
16. Bucila, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: KDD '06. (2006)
17. Saltelli, A., Tarantola, S., Campolongo, F., Ratto, M.: Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models. John Wiley & Sons Ltd (2004)
18. Bousquet, O., Elisseeff, A.: Stabilitiy and generalization. JMLR **2** (2002) 499–526