Feature Shaping for Linear SVM Classifiers

George Forman, Martin Scholz, Shyamsundar Rajaram

HP Laboratories
HPL-2009-31R1

Keyword(s):
text classification machine learning, feature weighting, feature scaling, SVM

Abstract:
Linear classifiers have been shown to be effective for many discrimination tasks. Irrespective of the learning algorithm itself, the final classifier has a weight to multiply by each feature. This suggests that ideally each input feature should be linearly correlated with the target variable (or anti-correlated), whereas raw features may be highly non-linear. In this paper, we attempt to re-shape each input feature so that it is appropriate to use with a linear weight and to scale the different features in proportion to their predictive value. We demonstrate that this pre-processing is beneficial for linear SVM classifiers on a large benchmark of text classification tasks as well as UCI datasets.

External Posting Date: May 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: May 6, 2009 [Fulltext]

Published and presented at 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'09) Paris France, June 28-July 1, 2009

Feature Shaping for Linear SVM Classifiers

George Forman
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304 USA
ghforman@hpl.hp.com

Martin Scholz
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304 USA
scholz@hp.com

Shyamsundar Rajaram
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304 USA
shyam.rajaram@hp.com

ABSTRACT

Linear classifiers have been shown to be effective for many discrimination tasks. Irrespective of the learning algorithm itself, the final classifier has a weight to multiply by each feature. This suggests that ideally each input feature should be linearly correlated with the target variable (or anti-correlated), whereas raw features may be highly non-linear. In this paper, we attempt to re-shape each input feature so that it is appropriate to use with a linear weight and to scale the different features in proportion to their predictive value. We demonstrate that this pre-processing is beneficial for linear SVM classifiers on a large benchmark of text classification tasks as well as UCI datasets.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design methodology—*Feature evaluation and selection*; H.2.8 [**Database Management**]: Applications—*Data mining*; H.3.3 [**Information Storage & Retrieval**]: Information Search & Retrieval—*Information filtering*

General Terms

Algorithms, Performance, Experimentation

Keywords

machine learning, feature weighting, feature scaling, linear Support Vector Machine, SVM, text classification

1. INTRODUCTION

Linear classifiers are extremely popular for their simplicity and proven effectiveness. The output of such a classifier is a weighted average of the input feature vector. Yet, given an arbitrary training set with binary class labels and numeric features, does it not seem far fetched to expect that the various features are each *linearly* correlated with the target concept? This is the presumption of a linear classifier: a delta increase of any variable causes a linear effect on the output, positive or negative. Of course, this is unsuitable for many situations, for example, the maxim that a driver's risk of death in an accident doubles for each 10 mph over 50. Clearly with manual intervention and domain knowledge, such a feature can be transformed to a more linearly predictive input. But the field of machine learning is about getting the computer to better handle any sort of input feature without careful, manual data preparation by an expensive domain expert.

This paper is concerned with pre-conditioning each input feature separately so as to improve the prediction ability of a downstream linear classifier, such as a linear Support Vector Machine (SVM). The SVM measures the distance between feature vectors via a simple dot-product in which all features are treated equally. Raw input features are usually not suited to be used with such a classification scheme. This is a multi-faceted problem that covers (i) the inclusion of irrelevant features, (ii) the inappropriate scale of some feature ranges, (iii) non-linear relationships between the feature and the target concept, such as the driving speed example, and even (iv) non-monotonic relationships with the target concept, such as an increased risk of death for negative speeds as well. While points (i) and (ii) are well known to compromise the predictive power of SVMs and are widely addressed via feature selection and scaling methods, this paper studies techniques for resolving all four issues, in particular (iii) and (iv). An ideal feature for linear classifiers should intuitively be linearly correlated with the class label. We want to study the effects of transforming features for this purpose.

As an example for all four issues above, let us look at a feature like "blood pressure" in a hypothetical disease prediction problem. We would expect this feature to show a concentration around the normal interval 60–120, mostly containing patients labeled as "healthy," and a few unhealthy patients that have measurements spread out over a larger range. The conditional distributions might look very different when conditioning on "healthy" and on "unhealthy," but say that (a) there is no linear dependency between label and feature, and (b) values above *and* below the normal interval might tend to be labeled "unhealthy" with higher probability. Finally, (c) the feature is represented in the range of 50–230, so an SVM would spend far more attention to this feature than to, say, the blood glucose level that happens to range from 3 and 8, regardless of which feature is truly more predictive. The cookbook advice for this case would be to make it much more complicated than a simple linear kernel, e.g., to experiment with multiple polynomial

KDD'09, June 28–July 1, 2009, Paris, France.

and radial-basis kernels, along with tuning each of their parameters, until one achieves good results. Most users would routinely scale all feature to a common range, or to zero mean and standard deviation of 1.0, in order to be sure the distance computations are not dominated by a single feature with an excessive range.

Yet, in most classification problems, some features are more predictive than others. Even if one applies feature selection to discard the weakest features, among those remaining, the strongest features deserve to have a greater effect on the distance dot-product than the weaker features. That is, the better features should be granted a wider dynamic range.

Our intuition suggests that a reasonable and much easier first step could just look at the conditional distributions of the feature and transform it into a feature that ranges from "predictive for positive" to "predictive for negative," even if that involves non-monotone mappings.

We will refer to the relative positioning of data points along each feature, with and without conditioning on the class label, as the *feature shape*. In the spirit of computationally cheap feature selection and scaling, we want to discuss methods that are of similar computational complexity as the typical filter approach to feature selection, looking at individual features one at a time. The goal is to help the classifier induction step by transforming raw feature values into a representation well suited for linear classifiers.

Our proposed methods will attempt to condition each input feature independently so that (1) it is more linearly correlated with the target concept, and (2) the better features have a wider range so that they have greater influence over the kernel distance. In this work, we restrict our attention to binary classification problems and demonstrate the benefit of feature scaling on the performance of linear SVMs. Such conditioning may also have benefits for other classifiers that apply dot-products to the features. This includes text classifiers that use cosine-similarity or k-Nearest Neighbor classifiers that measure distance between cases taking all features into account. For this paper, however, we limit our study to performance benefits such conditioning provides to a linear SVM classifier, which is already well established as a leading technique in various domains, especially text classification.

2. RELATED WORK

The literature on selecting, scaling and transforming features is deep and broad. Here we describe these areas to set context and to establish how this work is different.

2.1 Feature Selection

It is widely accepted that an initial step of feature selection [7] can improve the predictive performance of classifiers, including SVMs (see e.g., [4, 6]). There are different flavors of feature selection: wrappers and filters, supervised and unsupervised. For an extensive overview see [9] and [10]. Some approaches are less practical for high dimensional datasets, such as text or bioinformatics domains with tens of thousands of features or millions. Wrapper approaches attempt to search for a subset of features that obtains the best performance in cross-validation, but it is completely impractical on high dimensional datasets, because its search space is enormous and because it invokes the learning algorithm many times to guide its search. In contrast, the typical filter approach to feature selection requires just a single data scan and treats each feature independently, yielding an efficient computation, if myopic. Filter methods are very useful and popular in practice. The approach in this paper is more aligned with supervised filter approaches, in the way that we shape each feature independently, and leave the feature-interdependent analysis to the downstream classifier.

2.2 Feature Scaling

The result of feature selection is that it makes a hard decision to either include or exclude each feature. This idea can be generalized to weighting or scaling each feature with respect to its predictiveness toward the class variable [3, 12]. Empirical studies have shown that the additional degree of freedom of weighting features can achieve even stronger improvement than with feature selection alone. In one approach, the range of each feature is normalized to the range $[0, x]$, where x is a statistical measure computed from the corresponding feature column in the input data set. Our past research found, in a study of 15 different measures of predictiveness, that the best gain in text classification performance was seen by normalizing the range to the Bi-Normal Separation (BNS) score of the feature, i.e. $x = |NormalCDF^{-1}(tpr) - NormalCDF^{-1}(fpr)|$, where tpr and fpr are the true positive rate and false positive rate of the feature, respectively [5]. Subsequent unpublished research found that Hellinger distance [1] produced very similar results to BNS, but still slightly inferior for text classification. Using the tpr and fpr rate of a feature implies that it addresses only binary features, such as whether a given word occurs or not, and is especially appropriate for text. Yet most datasets include real-valued features, and it is not clear how to scale such features. In this work we leverage the idea of scaling each feature to a range, and extend the idea of BNS scaling to handle real-valued features.

Principle Component Analysis (PCA), Latent Semantic Indexing (LSI), and other variants in the family transform all the input features together into a new set of features that are each linear combinations of existing features. This in effect puts a weight on each feature and is loosely related to feature weighting. The contrast of these methods to ours serves to distinguish them. First, PCA and many variants are unsupervised—discovering the strong correlations between features, whether or not they have any pertinence to the classification task at hand. Second, such algorithms are generally heavyweight transformations that must deal with the entire dataset together, rather than treating each feature column independently, as we do. Finally, all such transforms produce a *linear* weighted sum of the other features, whereas our method reshapes the features in a way that can be highly non-linear.

2.3 Feature Shaping vs. Kernelized SVMs

In this work, we study an even less constrained approach than feature scaling: the transformation of individual features using different functions that take as an input conditional probability estimates of the class with respect to the input feature, and output a new input feature value. We try to generalize and combine ideas that have proven useful in either feature scaling or feature selection.

Note that this approach is not restricted to monotone transformations. Reordering data instances with respect to their conditional distributions along each axis can hence

be considered an alternative to using kernels as part of the learner, especially in cases where there is no good intuition at hand about which kernel and which parameters to use. The approach transforms the input features using informed local guesses (one feature at a time). One way of justifying this local strategy is to assume "mild" conditional independence. For example, the success of Naive Bayes for textual data proves that the violations of the conditional independence assumptions for this kind of real-world data are usually not in an overly critical range. Another justification is the success of feature selection and scaling, despite also making decisions for each feature in isolation. The broader concept of building global models (e.g., an SVM) on top of local patterns (e.g., individual predictive features) has recently gained some attention. See [11] for an overview.

Clearly, linear SVMs are able to optimize with respect to feature interactions, but they require the labels to be linearly separable (for the most part) in a space the user has to provide a mapping into in the first place. For making the decision which space is appropriate, most users rely on experimentation and manual beam search in parameter space. In contrast, our technique takes advantage of observing the conditional class distributions along each feature axis independently, so in the case of "mild independence" it transforms the data to be linearly separable. A linear SVM will usually still be able to handle violations of this assumption, but benefit from a feature landscape that is more of a linear nature than the raw data.

2.4 Kernel Learning

The field of kernel learning provides a very powerful alternative to the cumbersome process of manually selecting and fine-tuning kernels. This active field of research has recently attracted much attention, e.g., [2].

The general idea is to *learn* the kernel matrix (or the similarity between examples) along with the final classifier. There are different approaches towards this goal, that share the burden of complex computations. In fact, kernel learning techniques are at the outer end of the spectrum when it comes to computational costs, so we consider them promising and theoretically valuable, yet impractical in most settings.

The general idea either involves user-provided candidate or basis kernels, from which the kernel learning method will compute an optimal combination (see e.g. [13, 16]), or so called hyper-kernels [14]. Kernel learning leads to expensive optimization problems, most prominently in the framework of semi-definite programming. The main qualitative difference between our approach and kernel learning is that we aim at improving classifiers via cheap local optimizations, while kernel learning aims at computing an optimal global kernel matrix.

3. FEATURE SHAPING

Our goal in this work is to precondition each individual feature in isolation in a way that fosters its utility for linear classification. Let us formalize the framework before going into details.

3.1 Formal Background

Our algorithms work on sets of labeled examples $\mathcal{E} = \{\langle x^{(1)}, y^{(1)}\rangle, \ldots \langle x^{(m)}, y^{(m)}\rangle\}$, where each $x^{(i)} \in \mathcal{X}$ is a d-dimensional vector consisting of continuous, integer, and/or binary attributes, while $y^{(i)} \in \mathcal{Y} = \{+1, -1\}$ is a binary class label. Nominal attributes can easily be transformed into binary attributes and are hence supported. Examples in training and testing are assumed to be sampled i.i.d. from an unknown underlying distribution. The goal is to pick a function $c : \mathcal{X} \to [0, 1]$ based on the sampled training data that gives us (in expectation) a good estimate of $P(y = +1|x)$ on subsequently sampled test examples $\langle x, y\rangle$. Maximum-margin methods are dominant in linear classification, so we will focus on linear SVMs as our underlying classification scheme.

Soft margin SVMs solve the optimization problem

$$\min_{w,b,\xi} w^T w + C \sum_{i=1}^{m} \xi_i \tag{1}$$

$$\begin{aligned} s.t. \quad & w^T x + b \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \qquad\qquad\quad i = 1, \ldots, m \end{aligned}$$

The decision function of SVMs does not natively provide well calibrated soft classifications in the range $[0, 1]$, but can be calibrated in a post-processing step to do so [15].

We decided to consider soft classifiers in this work, because tuning those covers both the ranking and standard classification tasks.

Before looking into changing feature shapes, we want to discuss which kind of transformations matter:

- Linear SVMs depend on both the order and distance of examples along each feature axis, so any transformation – monotone and even more so non-monotone – will usually have an effect on the optimal hyperplane.
- As discussed in Section 2.2, the range of features (scale) determines the importance. This can be seen from (i) the objective function (eqn. (1)), since a feature x_i with a larger scale requires less of the regularized weight w_i to achieve the same effect, but also (ii) by simply considering the effect on distances in the Euclidean space the SVM operates in. Note that scaling *all* features by a common factor just has the same effect as changing the parameter C, so the important aspect is to scale features relative to each other.
- The only kind of transformations that will *not* affect an SVM are affine transformations $x_i \to x_i + c$, i.e., adding a constant c to the values of any feature x_i; the offset b in the decision functions of SVMs is not regularized, so if $w^T x + b$ is the SVMs decision function before this transformation, then the optimal decision function $f'(x)$ afterward is known to be $f'(x) = w^T x + b + c$.

The whole process of feature reshaping can be broken down into coherent data processing blocks that form a pipeline. The next subsections will describe those blocks in natural sequential order.

3.2 Estimating local conditional distributions

The goal of reshaping is to transform each feature x_i based on the label information given in the training set. More precisely, we want to achieve a linear correlation between attribute and class label, as motivated in Section 1. To this end, we start with a step of estimating the conditionals $P(y = +1 \mid x_i)$ with respect to the true underlying distribution, based on our (often small) training sample.

Attributes might be of type integer, real-valued, or nominal. For nominal attributes, conditional class probabilities can be estimated in a straighforward way by computing the fraction of positives seen for each attribute value. The case of continuous attributes requires more attention. Continuous features might be concentrated in a small region or spread out over a large range, and may present themselves in any arbitrary shape. Since we are – by design – not introducing any further assumptions, we use a broadly applicable local probability estimator at this point: For each threshold $x_i = v$ of interest, we compute the fraction of positives in up to n neighboring examples from above, and n from below after projecting our example set to feature x_i. We use fewer examples whenever fewer than n examples are available on either side at any point. For our experiments, we fixed n so that we averaged over a total of (up to) 30 neighboring examples, the rationale being that 30 gives us "near normal" behavior.[1] To prevent extreme estimates that some subsequent techniques are not able to handle well, we used standard Laplace correction.

Note that it is important to use the same amount of neighbors from above and below rather than just $2n$ nearest neighbors based on distance, because by balancing, one avoids the assumption that absolute distance corresponds to similarity, and therefore that x_i and the class label are already nicely correlated.

We used the balanced nearest neighbor approach during training with the set of x_i training values, and computed estimates $\tilde{p}_i(v) \approx P(y = +1 \mid x_i = v)$ for each of these thresholds. On the test set, we identified the closest values $v_l < v < v_u$ for any given threshold v, and defined $\tilde{p}_i(v)$ by linearly interpolating $\tilde{p}_i(v_l)$ and $\tilde{p}_i(v_u)$. For values above the maximum and below the minimum observed in training we just substitute the values for maximum and minimum, respectively. The resulting function $\tilde{p}_i$ is the output of this phase.

Clearly, instead of using a general purpose estimator, we might instead utilize any available domain knowledge for picking and fine-tuning a parametric model, a kernel density estimator etc. We abstained from complex techniques in this phase, in order to enable a broad, general study, but expect that better techniques would only improve our subsequently reported results if a user is willing to spend time on this issue. In practice, this might often still be easier than to directly adapt the kernel function of an SVM.

3.3 Reshaping Features

For any given feature x_i, this phase starts from a function $\tilde{p}_i : \mathbb{R} \to [0, 1]$ that yields estimates of the conditional probability $P(y = +1|x_i = v)$ for any $v \in \mathbb{R}$. The goal is to transform the input feature x_i towards a linear dependency on the class label. This suggests to use transformations T that map each value $v \in \mathbb{R}$ of attribute x_i to a new value $v' \in \mathbb{R}$ by referring to $\tilde{p}_i \in [0, 1]$ and ignoring the value v itself. In this case, T would be a function of the form $T : [0, 1] \to \mathbb{R}$ and usually describe a non-monotonic feature transformation.

To achieve linearity, the most straightforward choice for T is the identity function. We will use the term *local probability shaper (LP)* for this method. More precisely, with LP we will refer to the method that estimates $\tilde{p}_i$ using the balanced nearest neighbor approach (cf. Sec. 3.2) and then replaces each feature value v by $\tilde{p}_i(v)$. Despite its simplicity and non-monotonicity, it consistently gave good results in our experiments, so we will include this method into the experimental part of this work.

There are many other reasonable choices for T, like the log odds function, which we have seen to work even a bit better in some cases. However, none of these methods performed consistently well on a large subset of our benchmark datasets, so we will just discuss the stable LP shaper in subsequent sections.

Besides trying different functions that operate on $\tilde{p}_i$ only, we also experimented with monotone feature transformations, mostly based on ROC analysis. We did not see any substantial improvements over the baseline with any of these methods.

3.4 Feature Scaling

Some of the reshaping options discussed in the last subsection scale features in a way that indicates their predictive power. Other reshapers have a fixed range though, in particular the LP shaper we are mostly looking at. Those fixed range shapers might benefit from subsequently applying one of the feature scaling techniques discussed in Section 2.2. In those cases, we included the option of scaling *after* reshaping in our pool of experiments.

Some of the previously discussed scaling techniques do not directly apply though, because shaping usually yields continuous features. Scaling presupposes binary features. We resolved this issue by trying all thresholds for binarizing, and by picking the best scaling score for each feature that we have seen for any threshold.

We found BNS scaling to perform very similar or better than the alternatives on all data sets, so we will focus on this scaling technique in the experimental section.

3.5 Additional Options

The previous subsections discussed the major building blocks of feature shaping. We identified a few other options that we believe to be essential.

We start with the set of additional options we used for text. First of all, we noticed that the value of zero deserves some special treatment. Feature shaping up to this point and without any further post-processing would destroy the sparsity of documents: In the raw format the vast majority of words counts for each document are zero, and do not have to be stored explicitly, but after shaping the value of zero usually gets mapped to a different value. This has an adverse effect of the runtime and memory consumption. As a counter-measure, we keep track of the value that zero gets mapped to, and subtract it after shaping. The effect is that the value of zero gets mapped to itself and does not have to be stored explicitly. Please recall from Section 3.1 that the SVM is not affected by this kind of affine transformation.

In this context we noticed that word counts of zero behave quite differently, so we put them into a separate bin that never gets mixed with non-zero values when estimating local conditional probabilities, see Section 3.3.

We included a further option from the standard catalog of text mining techniques, that gets applied last in the pipeline: We tried L2 normalization to project document vectors on the unit hyperball, which has the effect that dot products

[1] We did not try to optimize n, but rather treated it as a parameter-free estimator.

Table 1: Summary of Benchmark Datasets.

Dataset	Cases	Classes	Min–Max	Features
Text:				
la1s	3204	6	8.5–29.4%	13195
la2s	3075	6	8.1–29.4%	12432
oh0	1003	10	5.1–19.3%	3182
oh10	1050	10	5.0–15.7%	3238
oh15	913	10	5.8–17.2%	3100
oh5	918	10	6.4–16.2%	3012
ohscal	11162	10	6.4–14.5%	11465
re0	1504	13	0.7–40.4%	2886
re1	1657	25	0.6–22.4%	3758
tr11	414	9	1.4–31.9%	6429
tr12	313	8	2.9–29.7%	5804
tr21	336	6	1.2–68.8%	7902
tr23	204	6	2.9–44.6%	5832
tr41	878	10	1.0–27.7%	7454
tr45	690	10	2.0–23.2%	8261
wap	1560	20	0.3–21.9%	8460
UCI:				
credit-g	1000	2	30.0–70.0%	20
diabetes	768	2	34.9–65.1%	8
ecoli	336	8	0.6–42.6%	7
glass	214	6	4.2–35.5%	9
iris	150	3	33.3–33.3%	4
letter	20000	26	3.7– 4.1%	16
optdigits	5620	10	9.9–10.2%	64
sonar	208	2	46.6–53.4%	60
vehicle	846	4	23.5–25.8%	18
vowel	990	11	9.1– 9.1%	13
wine	178	3	27.0–39.9%	13

used by the SVM reflect cosine similarities between documents. We also evaluated L1 normalization.

For the non-textual datasets that we included in our study, attributes include all kinds of continuous features, so any special treatment of zero is unmotivated. We deactivated the corresponding option described above. We still evaluated L2 normalization, since it is applied in more general settings.

4. EXPERIMENTS

We conducted an extensive suite of experiments to evaluate the proposed methods, both on text classification datasets, as well as UCI datasets. Table 1 lists the datasets and some of their characteristics, including the number of classes defined with an indication of their class distribution. The text classification datasets come from a variety of sources: Reuters, Los Angeles Times, OHSUMED abstracts, TREC challenges, and Web data [8]; we have previously made their pre-processed feature vectors of word counts available for download at the Weka site.[2] The subset of publicly available UCI datasets was selected for having nominal class labels and numerical attributes.

For each multi-class dataset, we consider the binary tasks of classifying each 'positive' class vs. all others, avoiding any tasks that have fewer than 50 positives. For UCI datasets that were already binary, we selected the minority class as the positive class. This yields 104 binary text tasks and 64 UCI tasks, which we analyze in separate groups in the following subsections.

We measure performance via Accuracy, the Area Under the ROC curve (AUC), the Precision@20 (percentage of positives among the twenty test cases the classifier scores highest) and F-measure (the harmonic average of precision and recall). Accuracy and F-measure both depend on the classifier to make a good choice of threshold, whereas AUC and Precision@20 do not. The latter measure is especially important for search applications, where we desire the greatest number of 'hits' in our top few search results. For each task and setting, we perform 10-fold cross-validation using consistent random split seeds. Note that to correctly evaluate AUC with cross-validation, we compute the AUC for each fold and then average, since the scores output by the classifier under different folds are not necessarily comparable. (As of v3.6.0, beware that Weka's Evaluation class sorts the scores from all the different folds together.)

During our preliminary software development and experimentation, we were using 4-fold cross-validation in order to reduce computation time. Surprisingly, when we prepared the final measurements for this paper using 10-fold cross-validation, we saw our margin of improvement diminish significantly. The primary difference being that with 4-folds 75% of the dataset is used for training, whereas with 10-folds 90% is used for training. Given the additional training cases, all methods were getting very good accuracy; classification problems that are already easily learnable with the raw features provide little possibility to improve the feature space by shaping. We then conducted a suite of learning curve experiments to systematically reduce the training set. This gives us a much broader picture of the performance benefits and yields additional insight beyond a typical K-fold cross-validation study. It also shows that the superior performance of our method is robust across a wide range of training set sizes and does not collapse when training data is scarce—a real-world concern for our intended applications.

We generate the learning curves as follows: for each of the ten folds, we first produce the test and train datasets, then we sub-sample only the training set. In this way, the performance measurements are comparable to traditional 10-fold cross-validation, since the test set is identical with 10-fold cross-validation at each data point. We are careful to ensure that each successive point on the learning curve includes a superset of the previous training cases, rather than just a larger independent sample. This helps to reduce experimental variance.

Except where otherwise stated, we used as the base classifier the Weka v3.6 SMO implementation of a linear SVM [17]. We used its default options, except that we had to disable its internal feature range normalization (`-N 2`), and we needed to enable logistic scaling in order to obtain probability scores for evaluation, rather than just hard binary classifications (`-M -V 2`). In addition, we tuned the SVM complexity parameter C by selecting the apparent best value from the set of values {0.01,0.1,1,10,100}, determined via 5-fold cross-validation on the sub-sampled training set only. That is, for each of the training sets of the ten folds, we perform an additional, smaller cross-validation to tune the parameter for each measurement. The chosen value is then used to train an SVM on the entire sub-sampled training set. Note that

[2]http://www.cs.waikato.ac.nz/ml/weka/

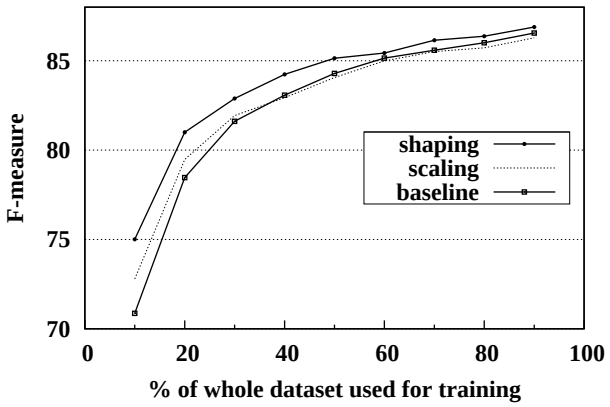

Figure 1: F-measure learning curves, each point averaged over 104 binary text classification problems.

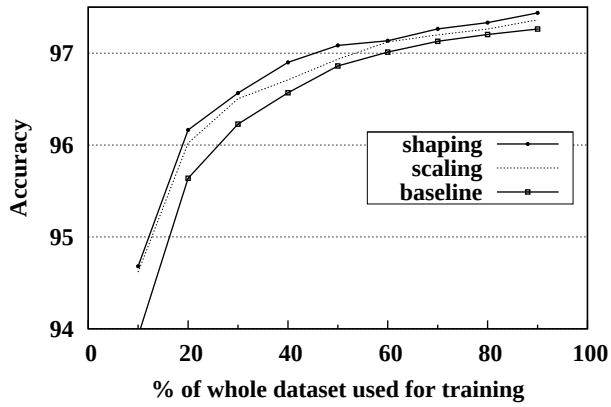

Figure 2: Accuracy learning curves for the text benchmark.

the implementation of the logistic scaling within the SMO code performs an additional 2-fold cross-validation to calibrate its output scores. It is partly because of this nested folding that a minimum of 50 positives was the criterion for selecting binary tasks. Though this nested complexity seems undesirable, tuning C and delivering calibrated SVM scores are worthwhile objectives to ensure that this research is conducted with highly competitive classifiers; there have been too many papers about improvements to Rocchio. Fortunately we have available to us a large batch cluster of HP BL460 blade servers on which to compute the millions of inductions that were required for a study of this scale and complexity.

4.1 Text datasets

We begin with the analysis of the text benchmark. Since text datasets are very sparse, we enable the special treatment of zero values, as explained in the methods section. Without it, the 'ohscal' dataset, for example, expands from 5MB to nearly a gigabyte.

During our preliminary studies, we experimented to determine the strongest baseline against which to compare our work. We found that using binary word features with L2-normalized feature vector lengths gave a stronger baseline than variations without L2-normalization or with using word count features or discretized word counts. This will be referred to as 'baseline' in this section. Likewise, we determined that LP method benefited substantially from BNS scaling and L2-normalization (e.g., a ten point drop in F-measure without them). Thus, for the 'shaping' method used on the text datasets, we refer to reshaping the raw word counts via LP, followed by BNS scaling, followed by L2-normalization. It would be far too cumbersome to present all the variations; many of them are so poor, that the parameterization choice is clear.

Figure 1 shows the learning curves for F-measure, which is the accepted metric used for text evaluation studies, because of their typically high class imbalance. Every point represents a (macro-)average performance over all 104 binary tasks. The x-axis shows the percentage of the whole dataset that was used for training; naturally, for 10-fold cross-validation, 90% is the maximum possible— no subsampling.

As we expect, we see the performance drop significantly as the size of the training set is reduced to the left. There are two findings in this graph. One is that reshaping the features produces a benefit for the linear classifier, and the other is that the performance gap widens towards the left as the task becomes very difficult to learn. In terms of relative improvement, at the leftmost the benefit exceeds 5%, while at the rightmost there is no worthwhile benefit for this benchmark. The difference in performance between the LP method and the baseline method is statistically significant at 50% training—a paired T-test yields p=0.01. The significance increases to the left and decreases to the right.

Consider the 'devil's advocate' hypothesis that all of the benefit is due to the BNS scaling, and not to the feature reshaping. To evaluate this, we compared these results with an augmented baseline that includes BNS scaling without reshaping. It uses binary word features with BNS scaling (shown previously to be a stronger combination than a wide variety of other representations [5]), with the addition of L2-normalization and tuning of the SVM C parameter, both of which tend to improve performance. We see that this curve, labeled 'scaling,' gets part of the benefit of reshaping, but not all of it. Thus, we conclude that the reshaping is indeed beneficial for a wide variety of text classification problems, especially when there is a paucity of training data. We speculate that it may also be helpful even for text classification tasks that have ample training data, but are fundamentally difficult. For the record, we examined the subset of tasks that got less than 80% F-measure to see if these benefited more from shaping, but a significance test refuted the idea.

An analysis of the AUC performance showed very similar findings, and Precision@20 showed no appreciable gain. These graphs are included in the additional graphs section in the appendix. We do, however, include here the learning curve for Accuracy, as it shows a slightly different aspect. In Figure 2 we see that the marginal benefit of feature shaping was retained across the entire spectrum of training set sizes. A paired T-test does not rule out the null hypothesis at the rightmost point (p=.20), but we note that the AUC of the baseline method averaged 0.984 AUC and exceed 0.990 in 45 of the 104 tasks. Thus, these datasets leave little margin for significant improvement, and our enthusiasm is undaunted by cautious null hypothesis tests, partly because of the results that follow.

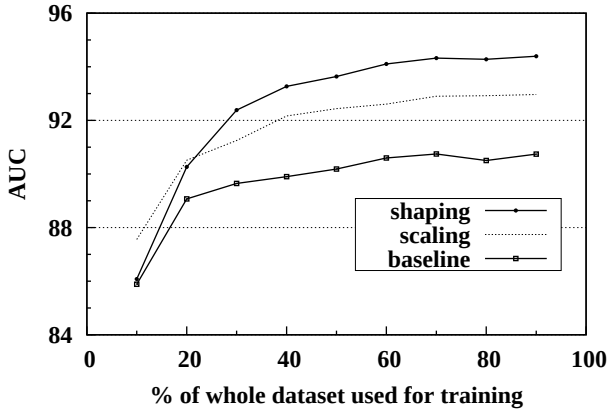

Figure 3: AUC performance across the learning curve, each point averaged over 64 binary UCI classification tasks.

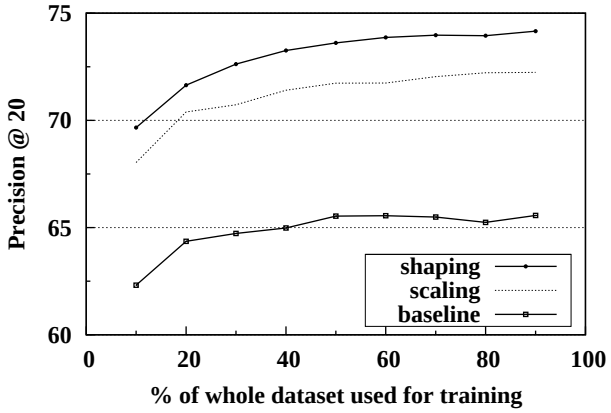

Figure 4: Precision@20 learning curves for UCI.

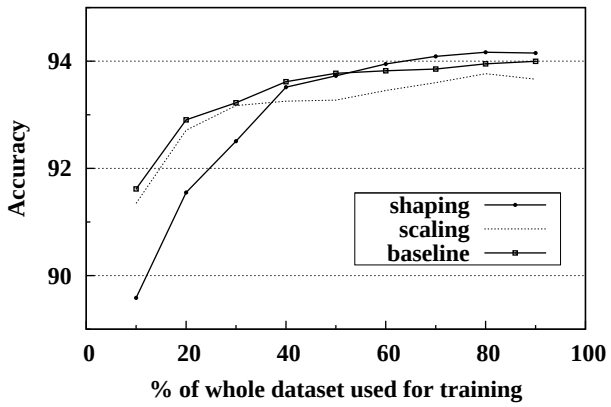

Figure 5: Accuracy learning curves for UCI.

4.2 UCI datasets

Next we turn to the UCI datasets. Since these datasets have only comparatively few features and are not sparse, we did not use the special treatment of zero values. Again, we began with a preliminary study to determine a strong parameterization for the baseline classifier. We found that for these datasets, L2-normalization significantly hurt its performance, so we did not use it. Further, using no normalization of the input features causes the SVM solver to run inordinately slow, e.g. over 24 hours for some individual data points. Therefore, we re-enabled the default Weka option to normalize all features to the range $[0.0, 1.0]$ for the baseline; for all other methods it was disabled, and the feature reshaping provided the necessary variable conditioning to make the SVM solve quickly.

Figure 3 shows the learning curves for AUC, averaged over the 64 one-vs.-others discrimination tasks on the UCI datasets. Here we see the benefit of feature reshaping more strongly. The curve labeled 'shaping' uses LP reshaping, BNS scaling, and finishes with L2-normalization of the feature vector lengths. Here we find that the substantial benefit extends all the way to the right, i.e. regular 10-fold cross-validation.

Again we ask whether the benefit could be almost entirely due to the BNS scaling, and not the shaping. To answer this question, we performed another lesion study: we removed the reshaping component and used only the BNS scaling with the L2-normalization. These results are labeled 'scaling.' We find by its middling performance that the scaling explains only a portion of the benefit. We also performed the opposite lesion, where we removed the BNS scaling from the 'shaping' method, and we found similarly middling results. (Not shown to reduce clutter.) Thus we conclude that the benefit of conditioning the variables derives both from reshaping the features to be more correlated with the positive class, and from scaling each feature in proportion to its predictive value.

The AUC results reflect an ability to improve the overall ranking; this is also reflected in the Precision@20 measure, which is shown in Figure 4. The only significant difference is that the performance benefit continues all the way to the left. In absolute terms, this improvement is quite large and is significant at p=0.002 in a paired T-test. The results for F-measure show an outstanding improvement of 18% from 0.6 to over 0.7 F-measure, but we include it only in the online appendix, as this is not a customary measure for UCI datasets.

Next, we examine the Accuracy learning curves shown in Figure 5, which tell a somewhat different story. For the larger training sets, feature shaping produced no significant benefit. But for the smaller training sets, feature shaping hurt accuracy substantially, unlike all the other learning curves in this paper. The 'scaling' lesion variant does not show such a performance drop, so the reshaping itself must be at fault. After deeper analysis on particular datasets, we found that the reshaping is overfitting local regions. We leave it to future work to determine the extent and develop effective safeguards.

Clearly no single method can be expected to be best for all classification tasks. We illustrate this variation in Figure 6, which shows the 10-fold cross-validation AUC performance for the 64 binary classification tasks derived from the UCI data. The tasks are sorted by difficulty, according to the baseline method. This view makes it clear that the change of representation can provide great performance improvements for many tasks, but not all. For the tasks at the far right, there is no headroom to improve learning, since the baseline linear classifier nearly achieves perfection.

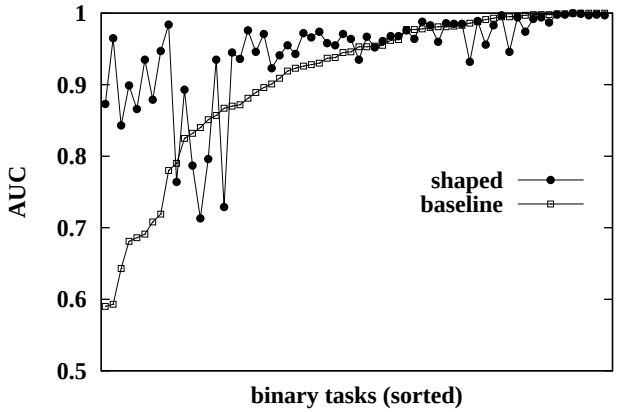

Figure 6: AUC for each of the 64 binary UCI tasks.

The take-home message is that while the proposed shaping method improved performance on average over a wide variety of problems, it certainly is not a panacea. In particular, feature shaping may only be needed for particular features, but in this study we simply apply the same shaping method to all features. We leave it to future work to determine policies for applying shaping more selectively.

4.3 Lesion Study

One way to determine the relative importance of the various stages of a method is to systematically disable one at a time and measure performance under each variant. We performed such a lesion study on the 'shaping' method that appears in the previous figures. Recall that its processing stages are (1) reshaping the input feature via LP, which outputs local probability estimates, (2) scaling the output to the range [0.0,BNS], where BNS is the maximum Bi-Normal Separation score over all decision thresholds on the reshaped training data, and (3) normalizing each feature vector row so that its total L2-length is 1.0. This transformed dataset is then given to the SVM for learning.

We performed the lesion study via 10-fold cross-validation for all 64 binary classification tasks from the UCI datasets, and then we averaged the results separately for each lesion variant. Figure 7 shows the results for all four performance measurements, with the lesion variants sorted along the x-axis by their AUC score. They are in order as follows:

L1-norm: The change that caused the largest loss in performance was using an L1-norm in place of the L2-norm. This normalizes each feature vector so that the *sum* of its feature values is 1.0, rather than normalizing the Euclidean length of the vector to 1.0.

no shaping: The second most damaging change was to remove the shaping step altogether. This is the variant labeled 'scaling' in the previous figures.

Log Odds: This variant on shaping returns the log odds ratio $(\log \tilde{p} - \log(1-\tilde{p}))$ of the local probability estimate $\tilde{p}$.

no scaling: This variant simply eliminates the BNS scaling step. Note that without it, the minimum value may not be zero, and hence sparse text datasets would balloon in memory.

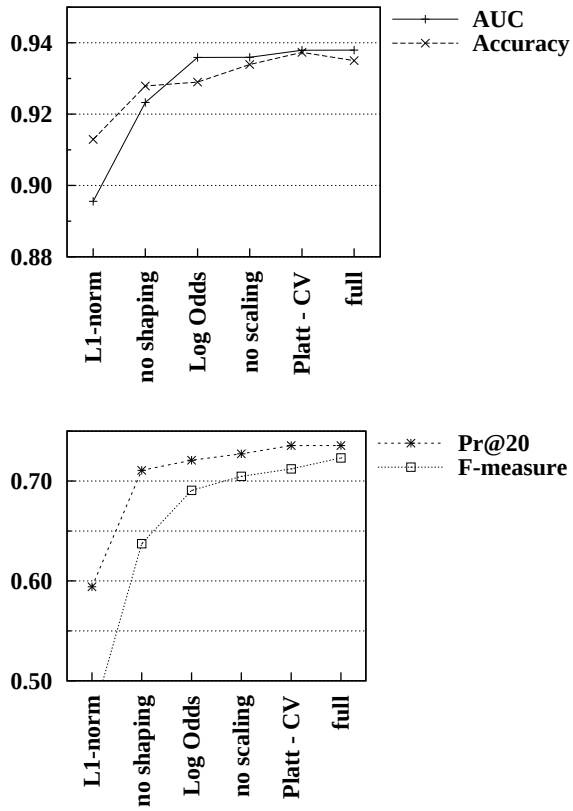

Figure 7: Lesion study on UCI datasets, using 10-fold cross-validation.

Platt - CV: Weka's SVM provides an option for whether or not to use cross-validation when training the final stage of logistic regression, due to Platt [15]. For this stage, we use 2-fold CV throughout this paper. That is, the SVM output scores that are given as input to the logistic regression are generated by two separate invocations of the SMO so that the scores that are given for training are not overly optimistic. This variant eliminates the cross-cross validation, so that only a single SVM training step is performed instead of three (2CV plus the final training on the whole training set). Looking at the upper graph in Figure 7, we see that the Accuracy actually benefited slightly from having this cross-validation step eliminated. It had no visible effect on AUC or Precision@20, and a slight loss for F-measure. Depending on one's goals and the relative importance of computational load, it may be worth turning off this extra cross-validation step.

To summarize, the results of the lesion study confirm that the importance of shaping for the UCI datasets is one of the most substantial contributors to the performance of the full method. This supports the hypothesis in this paper that independently reshaping each feature may lead to valuable performance gains for the downstream linear classifier.

5. DISCUSSION / FUTURE WORK

We set out with an intuition that reshaping each input feature independently could bring performance benefits to a downstream classifier. The experiments bear this hypothesis out, and by studying learning curves and multiple objective functions, we gain additional perspective on when shaping is most beneficial. In particular, the shaping method continued to work well even in the early parts of the learning curve. Depending on the objective, this benefit sometimes diminished when the training set was sufficiently large. It is perhaps surprising that feature shaping does benefit text classification at all. By our design, the many zero entries are mapped to zero in order to keep the dataset sparse. Thus, shaping can only have an effect on a sparse minority of the feature values, and yet the difference can be substantial viewed through the eyes of an SVM.

Our approach of reshaping each feature independently of the others affords both disadvantages and advantages. It is well known that the 'XOR-problem' cannot be solved by looking at features independently, yet we might ask how often this problem arises in practice. Advantages include simplicity, computational tractability, and the opportunity for parallel computation—of growing importance with the long heralded arrival of many-core computers, on which sequential programs will no longer speed up with Moore's Law.

The current limitations of the methods described here are plenty. The encouraging results of these experiments open the doors to other types of reshaping and to relax current restrictions. Future avenues for work includes multi-class reshaping, treating missing values, semi-supervised feature reshaping, processing other types of features, and experimentation with various other linear and non-linear classifiers. And most profoundly, different features in general datasets may call for different sorts of techniques—future research should explore *automated policies* for deciding what may be best to be done with each feature. This emulates a labor intensive process today, where an expert examines the raw data and proposes useful, domain-specific transforms to provide high quality, or at least worthwhile, features for the classifier.

A caveat is in store for researchers heading down this path. There are many, high-dimensional design decisions that ultimately depend, not on reason or theorems, but on their practical ability to be helpful on many 'typical' datasets. Of course, this requires performing one's research over many realistic datasets in order to *hill-climb* in one's research to the most useful ideas, as opposed to local optima for one or two datasets. But with this comes the threat of overfitting one's research to the available benchmarks. To safeguard against this, one must insist on studying many datasets in various domains, and on having very strong baselines. This in turn requires a concurrent search for very strong baseline results, which can itself be a daunting, high-dimensional search.

6. REFERENCES

[1] D. A. Cieslak and N. V. Chawla. Learning decision trees for unbalanced data. In *European Conference on Machine Learning / Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD'08)*. Springer, 2008.

[2] C. Cortes, A. Gretton, G. Lackriet, M. Mohri, and A. Rostamizadeh, editors. *NIPS Kernel Learning Workshop*. 2008. http://www.cs.nyu.edu/learning_kernels/.

[3] F. Debole and F. Sebastiani. Supervised term weighting for automated text categorization. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 784–788, New York, NY, USA, 2003. ACM.

[4] G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003.

[5] G. Forman. BNS feature scaling: an improved representation over TF-IDF for SVM text classification. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)*, pages 263–270, NY, 2008.

[6] E. Gabrilovich and S. Markovitch. Text categorization with many redundant features: Using aggressive feature selection to make SVMs competitive with C4.5. In *Proc. of International Conference on Machine Learning (ICML'04)*, pages 321–328, 2004.

[7] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[8] E. Han and G. Karypis. Centroid-based document classification: Analysis and experimental results. In *Proc. of the 4th European Conference on the Principles of Data Mining and Knowledge Discovery*, pages 424–431, 2000.

[9] H. M. Huan Liu. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[10] H. M. Huan Liu. *Computational Methods of Feature Selection*. Chapman & Hall/CRC, New York, NY, USA, 2008.

[11] A. Knobbe, B. Cremilleux, J. Fürnkranz, and M. Scholz. From local patterns to global models: The LeGo approach to data mining. In *Proc. of ECML/PKDD-08 Workshop: From Local Patterns to Global Models (LeGo'08)*, 2008. http://www.ke.informatik.tu-darmstadt.de/events/LeGo-08/.

[12] M. Lan, C. L. Tan, J. Su, and Y. Lu. Supervised and traditional term weighting methods for automatic text categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(2), 2008.

[13] G. Lanckriet, N. Cristianini, P. Bartlett, L. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[14] C. S. Ong and A. J. Smola. Machine learning with hyperkernels. In *Proc. of International Conference on Machine Learning (ICML'03)*, pages 568–575, 2003.

[15] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[16] M. Szafranski, Y. Grandvalet, and A. Rakotomamonjy. Composite kernel learning. In *Proc. of International Conference on Machine Learning (ICML'08)*, pages 1040–1047, 2008.

[17] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition, June 2005.

APPENDIX

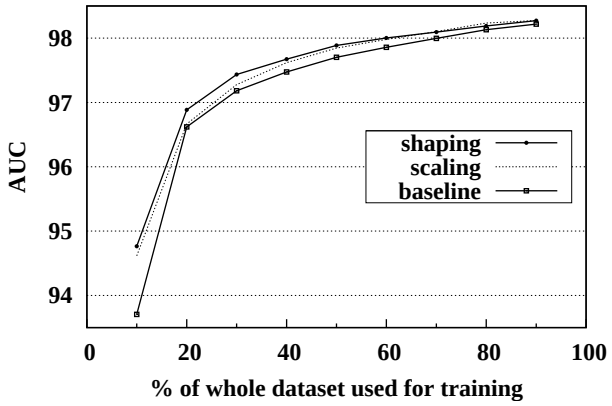

Figure 8: AUC learning curves, each point averaged over 104 binary text classification problems.

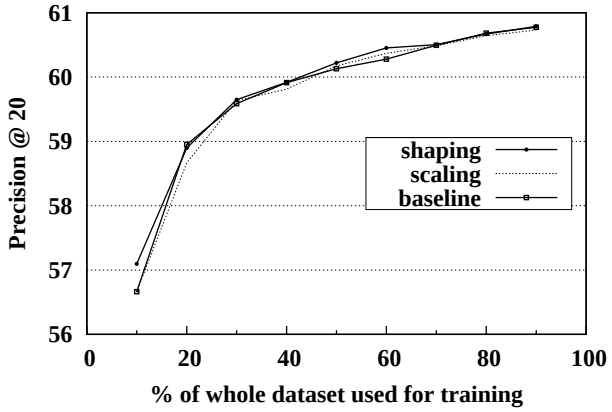

Figure 9: Precision@20 learning curves, each point averaged over 104 binary text classification problems.

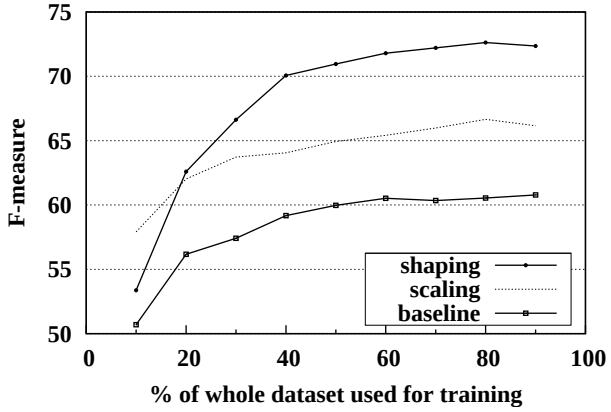

Figure 10: F-measure learning curves for the UCI benchmark.