A Loss Function Analysis for Classification Methods in Text Categorization

Fan Li, Yiming Yang HUSTLF@CS.CMU.EDU, YIMING@CS.CMU.EDU

Carnegie Mellon Univ, 4502 NSH, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

This paper presents a formal analysis of popular text classification methods, focusing on their loss functions whose minimization is essential to the optimization of those methods, and whose decomposition into the *training-set loss* and the *model complexity* enables cross-method comparisons on a common basis from an optimization point of view. Those methods include Support Vector Machines, Linear Regression, Logistic Regression, Neural Network, Naive Bayes, K-Nearest Neighbor, Rocchio-style and Multi-class Prototype classifiers. Theoretical analysis (including our new derivations) is provided for each method, along with evaluation results for all the methods on the Reuters-21578 benchmark corpus. Using linear regression, neural networks and logistic regression methods as examples, we show that properly tuning the balance between the training-set loss and the complexity penalty would have a significant impact to the performance of a classifier. In linear regression, in particular, the tuning of the complexity penalty yielded a result (measured using macro-averaged F1) that outperformed all text categorization methods ever evaluated on that benchmark corpus, including Support Vector Machines.

1. Introduction

Text categorization is an active research area in machine learning and information retrieval. A large number of statistical classification methods have been applied to this problem, including linear regression, logistic regression (LR), neural networks (NNet), Naive Bayes (NB), k-nearest neighbor (kNN), Rocchio-style, Support Vector Machine (SVM) and other approaches (Yang & Liu, 1999; Yang, 1999; Joachims, 1998; McCallum & Nigam; Zhang & Oles, 2001; Lewis et al., 2003). As more methods are published, we need to have a sound theoretical framework for cross-method comparison. Recent work in machine learning focusing on the *regularization* of classification methods and on the analysis of their loss functions is a step in this direction.

Vapnik (Vapnik, 1995) defined the objective function in SVM as minimizing the *expected risk* on test examples, and decomposed that risk into two components: the *empirical risk* that reflects the training-set errors of the classifier, and the inverse of *margin width* that reflects how far the positive and negative training examples of a category are separated by the decision surface. Thus, both the minimization of training-set errors and the maximization of the margin width are the criteria used in the optimization of SVM. Balancing between the two criteria has been referred as the regularization of a classifier; the degree of regularization is often controlled by a parameter in that method (section 2). SVM have been extremely successful in text categorization, often resulting in the best performance in benchmark evaluations (Joachims, 1998; Yang & Liu, 1999; Lewis et al., 2003).

Hastie et al. (Hastie et al. 2001) presented a more general framework for estimating the potential of a model in making classification errors, and used a slightly different terminology: *loss* or *generalization error* corresponding to the *expected risk*, *training-set loss* corresponding to the *empirical risk*, and *model complexity* corresponding to the margin-related risk in SVM. Using this framework they compared alternative ways to penalize the model complexity, including the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), and the Minimum Description Length (MDL) criterion. More interestingly, they compared the differences in the training-set loss functions for SVM, LLSF, LR and AdaBoost, in a way such that the sensitivity of those methods with respect to classification errors on training examples can be easily compared (section 2).

It would be valuable to analyze a broader range of classification methods in a similar fashion as presented by Hestie et al., so that the comparison among methods can be made explicitly in terms of their inductive biases with respect to training examples, or in terms of their penalty functions for model complexity. For this we need a formal analysis on the optimization criterion of each method, in the form of a loss function that decomposes into the training-set error term and the model complexity term. Such a formal analysis, however, often is not available in the literature for popular text categorization methods, such as Nave Bayes, kNN and Rocchio-style classifiers.

The primary contribution we offer here is a loss-function based study for eight classifiers popular in text categorization, including SVM, linear regression, logistic regression, neural networks, Rocchio-style, Prototypes, kNN and Nave Bayes. We provide our own derivations for the loss function decomposition in Rocchio-style, NB, kNN and multi-class prototypes (Prototypes), which have not been reported before. We also show the importance of properly tuning the amount of regularization by using controlled examinations of LLSF, LR and NNet with and without regularization. Finally, we compare the performance of the eight classifiers with properly tuned regularization (though validation) using a benchmark corpus (Reuters-21578) in text categorization.

The organization of the remaining parts of this paper is as follows: Section 2 outlines the classifiers and provides a formal analysis on their loss functions. Section 3 describes the experiment settings and results. Section 4 summarizes and offers the concluding remarks.

2. Loss functions of the classifiers

In order to compare different classifiers on a common basis, we need to present their loss functions in the unified form: $L_c = g_1(y_i f(\vec{x}_i, \vec{\beta})) + g_2(\vec{\beta})$. We call the first term $g_1(y_i f(\vec{x}_i, \vec{\beta}))$ the *training-set loss* and the second term $g_2(\vec{\beta})$ the *complexity penalty* or the *regularizer*. The following notation will be used in the rest of this paper:

- The training data consists of N pairs of $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_N, y_N)$
- Vector $\vec{x}_i = (x_{i1}, \ldots, x_{ip})$ represents the values of the p input variables in the ith training example.
- Scalar $y_i \in \{-1, 1\}$ (unless otherwise specified) is the class label: "1" for positive examples and "-1" for negative examples of the category in consideration[1].
- Vector $\vec{\beta} = (\beta_1, \ldots, \beta_p)^T$ consists of the parameters in a linear classifier, which are estimated using the training data.
- Scalar $f(\vec{x}_i, \vec{\beta})$ is the classifier's output given input $\vec{x}_i$, and the quantity $y_i f(\vec{x}_i, \vec{\beta})$ shows how much the system's output agrees with the truth label: if y_i and $f(\vec{x}_i, \vec{\beta})$ agree completely, then this quantity is a large positive number; if $f(\vec{x}_i, \vec{\beta})$ is a negative number with a large absolute value, it indicates a poor prediction by the classifier.
- A linear mapping $f(\vec{x}_i, \vec{\beta}) = \vec{x}_i \vec{\beta}$ is a special case of the mapping from the input to the output, which will be the focus in our analysis. Accordingly, $g1(y_i \vec{x}_i \vec{\beta})$ is the form we use to present the training-set losses for different classifiers.
- The 2-norm of $\vec{\beta}$ is represented as $\|\vec{\beta}\|$ and the 1-norm of $\vec{\beta}$ is represented as $\|\vec{\beta}\|_1$

Note that we purposely chose to define $\vec{x}_i$ as a horizontal vector and $\vec{\beta}$ as a vertical vector, so that we can conveniently write $\vec{x}_i \vec{\beta}$ for the dot product $\sum_{k=1}^{p} x_k \beta_k$ (and vice versa), which will be frequently seen in our derivations.

2.1. SVM

SVM has been extremely successful in text categorization. Multiple versions of SVM exist; in this paper we only use linear SVM for our analysis, partly for clarity and simplicity of our analysis, and partly because linear SVM performed as well as other versions of SVM in text categorization evaluations(Joachims, 1998). SVM emphasizes the generalization capability of the classifier (Hastie et al. 2001), whose loss function (for class c) has the form of

$$L_c = \sum_{i=1}^{n} (1 - y_i \vec{x}_i \vec{\beta})_+ + \lambda \|\vec{\beta}\|^2 \qquad (1)$$

in which the training-set loss on a single training example is defined to be

$$l_c = (1 - y_i \vec{x}_i \vec{\beta})_+ = \begin{cases} 1 - y_i \vec{x}_i \vec{\beta} \quad when \;\; y_i \vec{x}_i \vec{\beta} \le 1 \\ 0 \quad otherwise \end{cases}$$

The first term in the right hand side of formula 1 is the cumulative training-set loss and the second term is the complexity penalty and both are functions of vector $\vec{\beta}$. The optimization in SVM is to find $\vec{\beta}$ that minimizes the sum of the two terms in formula 1. In other words, the optimization in SVM is not only driven by the training-set loss, but also driven by the 2-norm of vector $\vec{\beta}$, which is determined by the squared sum of the coefficients in $\vec{\beta}$ and reflects the sensitivity of the mapping function with respect to the input variables. The value of λ controls the trade-off between the two terms, that is, it is the weight (algorithmically determined in the training phase of SVM) of the second term relative to the first term. Formula 1 can be transformed into dual form and solved using quadratic programming.

This kind of analysis on the loss function in SVM is not new, of course. In fact, it is a part of the SVM theory, and has been presented by other researchers (Vapnik, 1995; Hastie et al. 2001). Our point here is to start with a good framework and carry out the formal analysis for the other classifiers chosen for this study in a consistent fashion; some of those classifiers have not been formally analyzed in this manner.

[1] For simplicity we only use 2-way classification for our analysis in this paper; however, our methods and conclusions can be generalized to m-way classification problems.

2.2. Linear Least Squares Fit (LLSF)

Linear regression, also called Linear Least Squares Fit (LLSF) in the literature, has performed competitively to SVM and other high performing classifiers (including kNN and logistic regression) in text categorization evaluations (Yang, 1999). LLSF is similar to linear SVM in the sense that both learn a linear mapping $\hat{f}(\vec{x},\vec{\beta}) = \vec{x}\vec{\beta}$ based on the training data. Its optimization criterion in estimating $\vec{\beta}$, however, is strictly the minimization of the training-set error in terms of the sum of squared residuals. The loss function is defined to be: $L_{llsf} = \sum_{i=1}^{n}(y_i - \vec{x}_i\vec{\beta})^2$. Expanding the right hand side and rearranging, we obtain the equivalent formula in the desired form of $g1(y_i\vec{x}_i\vec{\beta})$:

$$\begin{aligned} L_{llsf} &= \sum_{i=1}^{n} y_i^2 + (\vec{x}_i\vec{\beta})^2 - 2y_i\vec{x}_i\vec{\beta} \\ &= \sum_{i=1}^{n} 1 + (y_i\vec{x}_i\vec{\beta})^2 - 2y_i\vec{x}_i\vec{\beta} \\ &= \sum_{i=1}^{n}(1 - y_i\vec{x}_i\vec{\beta})^2 \end{aligned}$$

Adding the regularizer $\lambda\|\beta\|^2$ to the training-set loss, we obtain the loss functions of the regularized LLSF (which is also called Ridge Regression):

$$L_c = \sum_{i=1}^{n}(1 - y_i\vec{x}_i\vec{\beta})^2 + \lambda\|\vec{\beta}\|^2. \qquad (2)$$

2.3. Rocchio-style

Rocchio-style classifiers are widely used in text categorization for their simplicity and relatively good performance(Lewis et al., 2003). They construct a prototype vector for each category using both the centroid of positive training examples and the centroid of negative training examples. When classifying a new document, the Rocchio-style classifier computes either the dot product or cosine value between the new document (a vector) and the prototype vector of the category, and then thresholds on this value for a classification decision (yes or no) with respect to that category. For simplicity of the analysis, we restrict our discussion to dot product, which is equivalent to using cosine similarity after both input document and the class prototype vectors are normalized. Using this assumption, Rocchio-style is a linear classifier with the scoring function $\hat{f}(\vec{x},\vec{\beta}) = \vec{x}\beta$ where $\vec{\beta}$ is the prototype vector. Let $\vec{u}$ be the centroid of the positive training examples of class c, and $\vec{v}$ be the centroid of the negative training examples then the prototype is defined to be:

$$\begin{aligned} \vec{\beta}^T &= \vec{u} - b\vec{v} = \frac{1}{N_c}\sum_{\vec{x}_i \in c_j}\vec{x}_i - \frac{b}{N_{\bar{c}}}\sum_{\vec{x}_i \notin c}\vec{x}_i \\ &= \frac{1}{N_c}\sum_{y_i=1} y_i\vec{x}_i + \frac{b}{N_{\bar{c}}}\sum_{y_i=-1} y_i\vec{x}_i \qquad (3) \end{aligned}$$

where b is a parameter in the Rocchio-style method, whose value is the weight of the negative centroid relative to the positive centroid, and can be empirically determined using a held-out subset of training data. Now we show that the regularized loss function in the Rocchio-style classifier is

$$L_c = -\sum_{y_i=1} y_i\vec{x}_i\vec{\beta} - \frac{bN_c}{N_{\bar{c}}}\sum_{y_i=-1} y_i\vec{x}_i\vec{\beta} + \frac{N_c}{2}\|\vec{\beta}\|^2 \qquad (4)$$

In order to minimize the loss function, we need to take the first order derivative of formula 4 with respect to $\vec{\beta}$ and set it to zero,

$$\frac{dL_c}{d\vec{\beta}} = -\sum_{y_i=1} y_i\vec{x}_i - \frac{bN_c}{N_{\bar{c}}}\sum_{y_i=-1} y_i\vec{x}_i + N_c\vec{\beta} = 0$$

It is easy to see that $\vec{\beta}$ in formula 3 is just the solution. In other words, formula 4 is the loss function that the Rocchio-style classifier is trying to minimize. Presenting its loss function in this fashion enables us to compare the Rocchio-style approach with other classification methods on the same basis, i.e., loss-function based analysis.

Observing formula 4 is interesting. The loss function consists of three parts, instead of two as in the other classifiers we analyzed so far. The first part is the training-set loss on positive examples; The second part is the training-set loss on negative examples; the third part is the complexity penalizer $\|\beta\|^2$.

The training-set loss on a single training example depends on whether it is a positive or negative example. That is,

$$l_{ci} = \begin{cases} -y_i\vec{x}_i\vec{\beta} & \text{when } y_i = 1 \\ -\frac{bN_c}{N_{\bar{c}}} y_i\vec{x}_i\vec{\beta} & \text{when } y_i = -1 \end{cases}$$

2.4. Multi-class Prototype Classifier

Multi-class Prototype classifier, or just "Prototype" as an abbreviation, is even simpler than Rocchio-style. It is the same as Rocchio-style except that only positive examples are used to construct the prototype of each category. That is, the method is defined by setting the parameter b to zero in the formula 3 and 4. Accordingly, the regularization loss in the Prototype method is:

$$L_c = -\sum_{y_i=1} y_i\vec{x}_i\vec{\beta} + \frac{N_c}{2}\|\vec{\beta}\|^2 \qquad (5)$$

and the training-set loss on a single training example is:

$$l_{ci} = \begin{cases} -y_i\vec{x}_i\vec{\beta} & \text{when } y_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

Including Prototype in this study enables us to compare the differences of using positive training examples only (in Prototype) as opposed to using both positive and negative examples (in Rocchio-style). It also allows us to compare the differences of using a fixed centroid (in Prototype) versus using a varied centroid for each text example in kNN, which we describe next.

2.5. kNN

kNN has been popular in text categorization, both for its simplicity and for the good performance in benchmark evaluations(Yang & Liu, 1999; Lewis et al., 2003). kNN is very similar to Prototype except that only the training examples inside of the neighborhood local to each test example have a non-zero loss. The nearness of each neighbor in kNN is often measured using the cosine similarity between the test example and the training example, which is equivalent to using dot product after both vectors are normalized. For simplicity of analysis, we restrict our discussion under the assumption of using normalized vectors. Under this assumption, kNN has a locally linear classification function with the vector of coefficients

$$\vec{\beta}_x^T = \sum_{\vec{x}_i \in c \wedge \vec{x}_i \in R_k(\vec{x})} \vec{x}_i \qquad (6)$$

where $R_k(\vec{x})$ is the k training examples nearest to test example $\vec{x}$, and $\vec{\beta}_x$ is the local centroid of the positive examples in category c. The classification decision on test example $\vec{x}$ is obtained by thresholding on the dot product $\vec{x} \cdot \vec{\beta}_x$. Now we need to formally analyze exactly what kNN is optimizing. Defining a loss function in the following form

$$L_c = - \sum_{y_i = 1 \wedge \vec{x}_i \in R_k(\vec{x})} y_i \vec{x}_i \vec{\beta}_x + \frac{1}{2} \|\vec{\beta}_x\|^2 \qquad (7)$$

and setting the first order derivative of the right hand side to zero yields the coefficient vector in formula 6. This is to say that the optimization criterion in kNN is the minimization of loss function L_c in formula 7 which has both the training-set error component (the first term) and the complexity penalization component (the second term). Accordingly, the training-set loss on a single training example is:

$$l_{ci} \quad = \quad \begin{cases} -y_i \vec{x}_i \vec{\beta}_x & \text{when } y_i = 1 \wedge \vec{x}_i \in kNN(\vec{x}) \\ 0 \quad \text{Otherwise} & \end{cases}$$

Analyzing kNN's optimization criterion in the form of the loss function presented above has not been reported before, to our knowledge. Note that we use $\vec{\beta}_x$ instead of $\vec{\beta}$ to emphasize the local nature of the classification in kNN. The loss function depends on each test example, which strongly differentiates kNN from the other classifiers.

2.6. Logistic Regression (LR)

Logistic regression methods have also shown good performance (competitive to SVM, LLSF and kNN) in the evaluations on benchmark collections(Yang, 1999; Zhang & Oles, 2001). It estimates the conditional probability of y given $\vec{x}$ in the form of

$$P(y|\vec{x}) = \pi(y\vec{x}\vec{\beta}) \overset{def}{=} \frac{1}{1 + exp(-y\vec{x}\vec{\beta})}$$

and learns the regression coefficients $\vec{\beta}$ in order to maximize $\prod_{i=1}^{n} P(y_i|\vec{x}_i)$. This is equivalent to minimizing the training-set loss defined in the logarithmic form:

$$L_c = \sum_{i=1}^{n} \log \frac{1}{\pi(y_i \vec{x}_i \vec{\beta})} = \sum_{i=1}^{n} \log(1 + exp(-y_i \vec{x}_i \vec{\beta}))$$

The regularized version of LR (Zhang & Oles, 2001) has the loss function in the form of

$$L_c = \sum_{i=1}^{n} \log(1 + exp(-y_i \vec{x}_i \vec{\beta})) + \lambda \|\vec{\beta}\|^2 \qquad (8)$$

2.7. Neural Networks (NNet)

Neural networks have also shown competitive performance in text categorization evaluations(Yang & Liu, 1999; Yang, 1999). We restrict our analysis to a two-level (no hidden layers) neural network in this paper. NNets without hidden layers are very similar to LR in the sense that they estimate $P(y = 1|\vec{\beta}, \vec{x})$ in the form of

$$P(y = 1|\vec{\beta}, \vec{x}) = \frac{1}{1 + exp(-\vec{x}\vec{\beta})} = \pi(\vec{x}\vec{\beta})$$

However, the objective function is to minimize $L_c = \sum (y_i' - \pi(\vec{x}_i \vec{\beta}))^2$ where y_i' is 1 or 0. To make its loss function in a form consistent and comparable to those in other classifiers, we need to write it using y_i instead of y_i' where $y_i = 1$ when $y_i' = 1$ and $y_i = -1$ when $y_i' = 0$. The training-set loss is:

$$\begin{aligned} L_c \quad &= \quad \begin{cases} \sum_{i=1}^{n} (1 - \pi(\vec{x}_i \vec{\beta}))^2 & \text{when } y_i = 1 \\ \sum_{i=1}^{n} (0 - \pi(\vec{x}_i \vec{\beta}))^2 & \text{when } y_i = -1 \end{cases} \\ &= \quad \begin{cases} \sum_{i=1}^{n} (1 - \pi(\vec{y}_i x_i \vec{\beta}))^2 & \text{when } y_i = 1 \\ \sum_{i=1}^{n} (\pi(-y_i \vec{x}_i \vec{\beta}))^2 & \text{when } y_i = -1 \end{cases} \\ &= \quad \sum_{i=1}^{n} (1 - \pi(y_i \vec{x}_i \vec{\beta}))^2 \end{aligned} \qquad (9)$$

Adding an regularization term $\lambda \|\vec{\beta}\|^2$ yields the loss function of the regularized NNet:

$$L_c = \sum_{i=1}^{n} (1 - \pi(y_i \vec{x}_i \vec{\beta}))^2 + \lambda \|\vec{\beta}\|^2 \qquad (10)$$

2.8. Naive Bayes (NB)

We restrict our analysis to the most popular multinomial NB classifier(McCallum & Nigam). It estimates the posterior probability of test document D as a member of category c using the following formula:

$$p(c|D) = \frac{P(c)\prod_{k=1}^{p} P(W_k|c)^{n(W_k,D)}}{p(D)} \tag{11}$$

where $P(c)$ is the prior probability of the category, $P(D)$ is the probability of D occurring by chance, $P(W_k|c)$ is the probability of word W_k conditioned on category c, and $n(D, W_k)$ is the count of word W_k in document D. Taking the logarithm of both sides of formula 11 yields an alternative scoring function for category c with respect to document D:

$$\begin{aligned} \log P(c|D) &= \sum_{k=1}^{p} n(W_k, D) \log P(W_k|c) \\ &+ \mathit{log}(P(c)) - \mathit{log}(P(D)) \end{aligned} \tag{12}$$

Rewriting formula 12 using $x_k = n(W_k, D)$, $\theta_k = P(W_k|c)$ and $\beta_k = \log\theta_k$, we have

$$\begin{aligned} \log P(c|D) &= \sum_{k=1}^{p} x_k \log\theta_k + \mathit{log}(P(c)) - \log P(D) \\ &= \sum_{k=1}^{p} x_k\beta_k + \mathit{log}(P(c)) - \log P(D) \\ &= \vec{x}\vec{\beta} + \mathit{log}(P(c)) - \log P(D) \end{aligned} \tag{13}$$

The first term shows that NB is a linear classifier with respect to the input vector $\vec{x}$; the second term (the logarithm of the prior probability) varies by category, making common categories more preferable than rare categories but is invariant with respect to $\vec{x}$; the third term does not effect the scoring or ranking of categories given $\vec{x}$ and thus can be eliminated from consideration.

Optimization in NB regards the estimation of the model parameters based on training data [2] : $\hat{P}(c) = \frac{N_c}{N}$ and $\hat{\theta_k} = \frac{F_{ck}}{S_c}$ where N_c is the number of positive training examples for category c, F_{ck} is the frequency of word W_k in those positive training examples, and $S_c = \sum_{k=1}^{p} F_{ck}$ is the total number of word occurrences in category c.

We now show how to relate the parameter estimates in NB to a loss function in the form of $L = g_1(y_i\vec{x_i}\vec{\beta}) + g_2(\vec{\beta})$ so that we can compare NB with other classifiers on the same basis. Let us use vector $\vec{x_i}$ to represent the ith training document whose elements are the within-document term frequencies of individual words, the vector sum $\vec{F_c} = \sum_{x_i \in c} \vec{x_i}$ to represent the within-category term frequencies in category c, and θ_k to denote $P(W_k|c)$. We define the loss function in the following form:

$$L_c = -\sum_{k=1}^{p} F_{ck} \log\theta_k + S_c \sum_{k=1}^{p} \theta_k \tag{14}$$

To minimize this loss function, we take the first-order partial derivative with respect to θ_k and set it to zero:

$$\frac{\partial L_c}{\partial\theta_k} = -F_{ck}\frac{1}{\theta_k} + S_c = 0$$

Clearly, $\theta_k = \frac{F_{ck}}{S_c}$ is just the solution. This means that the loss function in formula 14 is the optimal objective function NB is using to estimate its parameters $P(W_k|c)$(it is also equivalent to maximizing the likelihood $\prod_{k=1}^{p} \theta_k^{F_{ck}}$ subject to $\sum_{k=1}^{p} \theta_k = 1$).

We now rewrite formula 14 as a function of $\vec{F_c} = (F_{c1}, \ldots, F_{cp})$ and $\vec{\beta} = (\beta_1, ..., \beta_p)^T$ where $\beta_k = \log\theta_k$. Since θ_k is a word probability, all the elements in $\vec{\theta}$ are positive numbers. This means $\sum_{k=1}^{p} \theta_k = \|\vec{\theta}\|_1$ is the norm-1 of vector $\vec{\theta}$. Substituting those terms in 14 yields the loss function in the form of

$$L_c = -\vec{F_c}\vec{\beta} + S_c\|\vec{\theta}\|_1 \tag{15}$$

Furthermore, from $\vec{F_c} = \sum_{\vec{x_i} \in c} \vec{x_i}$ we have $\vec{F_c}\vec{\beta} = \sum_{\vec{x_i} \in c} \vec{x_i}\vec{\beta}$, and from $\vec{\beta} = \log\vec{\theta}$ we have $\vec{\theta} = e^{\vec{\beta}}$ where $e^{\vec{\beta}} \overset{def}{=} (e^{\beta_1}, \ldots, e^{\beta_p})$. Substituting those terms in 15 yields the loss function in the form of

$$\begin{aligned} L_c &= -\sum_{\vec{x_i} \in c} \vec{x_i}\vec{\beta} + S_c\|e^{\vec{\beta}}\|_1 \\ &= -\sum_{y_i = 1} y_i\vec{x_i}\vec{\beta} + S_c\|e^{\vec{\beta}}\|_1 \end{aligned} \tag{16}$$

Now we have successfully decomposed NB's loss function into the form of $L = g_1(y_i\vec{x_i}\vec{\beta}) + g_2(\vec{\beta})$. Note that we only discussed NB without any smoothing which is known to be important for the effectiveness of NB. It is easy to see in the second term of formula 16 that $\|e^{\vec{\beta}}\|_1$ would be overly sensitive to estimation errors in the elements $\beta_k = \log P(W_k|c)$ if those numbers (negative) have large absolute values, that is, when the word probabilities are near zero.

We now present the loss function for NB with Laplace smoothing which is common in NB. Here the estimate of θ_k is $\theta_k = \frac{1+F_{ck}}{p+S_c}$. let us use $\vec{1}$ to represent vector (1,1,...,1). Note that the elements in β are all negative numbers because $\beta = \log\theta$ and θ are probabilities. So $-\vec{1}\vec{\beta} = \|\vec{\beta}\|_1$. Then we have the loss function for NB as the following:

$$L_c = -\sum_{k=1}^{p}(1 + F_{ck})\log\theta_k + (p + S_c)\sum_{k=1}^{p}\theta_k$$

[2] Now we only consider NB without smoothing for simplicity. We will consider NB with Laplace smoothing next.

Table 1. The training-set loss functions and the regularizers of eight classifiers

Classifier	training-set loss: $g_1(y_i\vec{x}_i\vec{\beta})$	regularizer: $g_2(\vec{\beta})$
Regularized LLSF	$\sum_{i=1}^{n}(1-y_i\vec{x}_i\vec{\beta})^2$	$\lambda\|\vec{\beta}\|^2$
Regularized LR	$\sum_{i=1}^{n}\log(1+exp(-y_i\vec{x}_i\vec{\beta}))$	$\lambda\|\vec{\beta}\|^2$
Regularized 2-layer NNet	$\sum_{i=1}^{n}(1-\pi(y_i\vec{x}_i\vec{\beta}))^2$	$\lambda\|\vec{\beta}\|^2$
SVM	$\sum_{i=1}^{n}(1-y_i\vec{x}_i\vec{\beta})_+$	$\lambda\|\vec{\beta}\|^2$
Rocchio	$-\sum_{y_i=1}y_i\vec{x}_i\vec{\beta}-\frac{bN_c}{N_{\bar{c}}}\sum_{y_i=-1}y_i\vec{x}_i\vec{\beta}$	$\frac{N_c}{2}\|\vec{\beta}\|^2$
Prototype	$-\sum_{y_i=1}y_i\vec{x}_i\vec{\beta}$	$\frac{N_c}{2}\|\vec{\beta}\|^2$
kNN	$-\sum_{y_i=1\wedge\vec{x}_i\in R_k(\vec{x})}y_i\vec{x}_i\vec{\beta}_x$	$\frac{1}{2}\|\vec{\beta}_x\|^2$
NB without smoothing	$-\sum_{y_i=1}y_i\vec{x}_i\vec{\beta}$	$S_c\|e^{\vec{\beta}}\|_1$
NB with Laplace smoothing	$-\sum_{y_i=1}y_i\vec{x}_i\vec{\beta}$	$(p+S_c)\|e^{\vec{\beta}}\|_1+\|\vec{\beta}\|_1$

$$\begin{aligned} &= -(\vec{1}+\vec{F}_c)\vec{\beta}+(p+S_c)(\|\vec{\theta}\|_1) \\ &= -(\vec{1}+\sum_{y_i=1}y_i\vec{x}_i)\vec{\beta}+(p+S_c)\|e^{\vec{\beta}}\|_1 \\ &= -\sum_{y_i=1}y_i\vec{x}_i\vec{\beta}+(p+S_c)\|e^{\vec{\beta}}\|_1+\|\vec{\beta}\|_1 \quad (17) \end{aligned}$$

Comparing this to formula 16 of NB without smoothing, we can see the correction by Laplace smoothing in the third term, which prevents the coefficients in $\vec{\beta}$ from being too large. In other words, it prevents the classification decisions from being overly sensitive to small changes in the input.

Also, both formulas 16 and 17 show a unique property of NB classifiers, that is, the influence of term S_c in the loss functions, which causes the amounts of regularization to vary for different categories. To our knowledge, this is the first time that this property is made explicit in loss-function based analysis. Whether this is a theoretical weakness or a desirable property of NB requires future research.

2.9. Comparative Analysis

The loss functions of the eight classifiers are summarized in Table 1. All the regularized classifiers, except NB, have their regularizers in the form of the vector 2-norm $\|\beta\|^2$ multiplied to a constant or a weight (category-specific). Among those, regularized LLSF, NNet and LR have exactly the same regularizer as that in SVM, so the differences among those methods are only in their training-set loss functions. Prototype and NB, on the other hand, are exactly the same in terms of their training-set loss, but fundamentally different in their regularizer terms.

The curve of the training-set loss on individual training examples is shown for each classifier in Figure 1; the 0-1 misclassification loss is also shown for comparison. The Y axis represents the loss, and the X axis is the value of $y_i\vec{x}_i\vec{\beta}$. Examples with $y_i\vec{x}_i\vec{\beta}\geq 0$ are those correctly categorized by a classifier assuming the classification decisions are obtained by thresholding at zero; examples with $y_i\vec{x}_i\vec{\beta}<0$ are those misclassified. Examples with $y_i\vec{x}_i\vec{\beta}=1$ are those perfectly scored in the sense that the scores ($\vec{x}_i\vec{\beta}$) by the classifier is in a total agreement with the true scores of y_i.

From Figure 1, we can see that LLSF gives the highest penalty to the misclassified examples with a negative and very large absolute value of $y_i\vec{x}_i\vec{\beta}$ while NNet gives those errors the lowest penalty. In other words, LLSF tries very hard to correctly classify such outliers (with relatively small scores) while NNet does not focus on those outliers. As for the correctly classified examples with a large positive value of $y_i\vec{x}_i\vec{\beta}$, LLSF is the only method which penalizes them heavily. SVM, NNet and LR tend to ignore these examples by giving them zero or near zero penalties. On the other hand, Rocchio, NB, Prototype and kNN give these examples minus loss rather than neglecting them.

It should be noticed that we have two lines for Prototype and NB: a linear function with a non-zero slope for the positive examples, and the other with a flat slope for the negative examples. This reflects the fact that only the positive training examples of each category are used to train the category-specific models in those methods. kNN is similar in this sense except that its loss functions are local, depending on the neighborhood of each input example; we omit the lines of kNN in this figure. Rocchio-style, on the other hand, uses both positive and negative examples to construct the category prototype, should have two linear lines (with non-zero slopes) as its loss functions. For convenience, we show a specific case of Rocchio-style when parameter $b=\frac{N_{\bar{c}}}{N_c}$ in this figure, i.e., the two lines for positive and negative examples become the same.

3. Empirical Evaluation

We conducted two sets of experiments: one set was for the global comparison of the eight classifiers in text categorization using a benchmark collection, the Reuters-21578 corpus ApteMod version(Yang & Liu, 1999) (http: //www-2.cs.cmu.edu/~yiming), and the other set was for examining the effectiveness of regu-

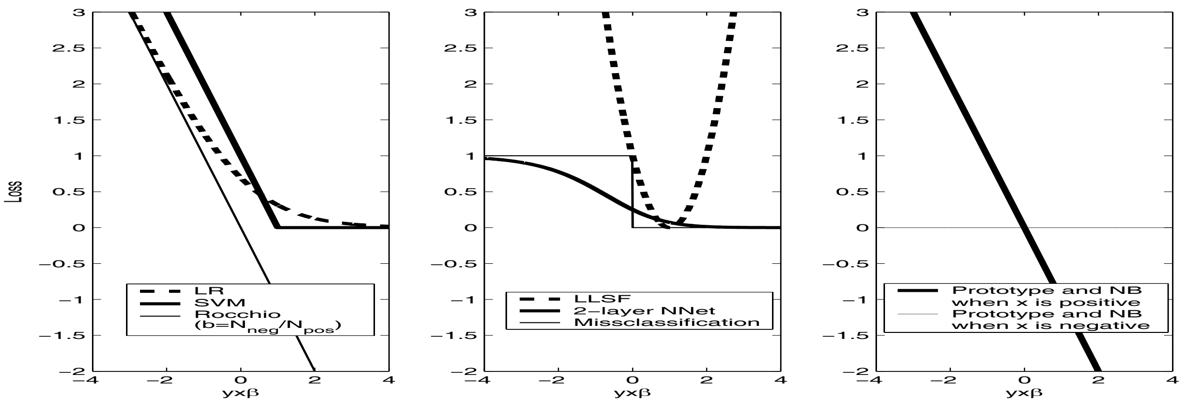

Figure 1. The training-set loss functions of eight classifiers

larization in individual classifiers. For the latter, we chose LLSF, LR and NNet because the regularizer can be easily implemented as a "plugged-in" term in the non-regularized versions of those classifiers, while how to add a tunable regularization component in the other classifiers (for example, in SVM) is not obvious.

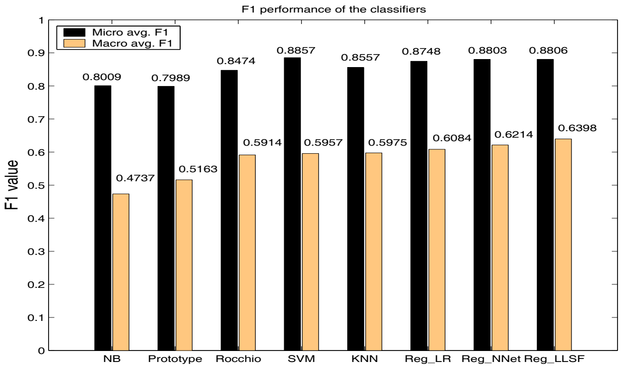

Figure 2. Performance of eight classifers on Reuters-21578

Figure 2 shows the results of the eight classifiers on the test set of the Reuters corpus. Both macro- and micro-averaged F1 are reported, which have been the conventional performance measures in text categorization evaluations (Yang & Liu, 1999). All the parameters are tuned using five fold cross-validation on the training data. Feature selection was applied to documents as a preprocessing step before the training of the classifiers; the χ^2 criterion was used for feature selection in all the classifiers. For NB, we selected the top 1000 features. For Rocchio-style (implemented using the version in (Lewis et al., 2003)) we used the top 2000 features and set parameter $b = -2$. For Prototype we used the top 2000 features. For kNN we set $k = 85$ and used the top 2000 features that when micro-avg. F1 was the performance measure, and the top 1000 features when macro-avg. F1 was the performance measure. For regularized LLSF, LR, NNet (2-layer) and SVM, we used all the features without selection. We used cross-validation to tune the classification threshold for every category (this is probably the main reason that our SVM results are better than those reported in (Yang & Liu, 1999; Joachims, 1998)). We also used T-test to compare the macro F1 scores of regularized LLSF and SVM and found regularized LLSF was significantly better.

Figure 3 shows the performance curves of the regularized LLSF, NNet and LR on a validation set (a held-off subset of the training data), with respect to the varying value of λ that controls the amount of regularization. Clearly, the performance of those classifiers depends on the choice of the value for λ: all the curves peak at some λ values larger then zero. For LLSF and NNet, in particular, having regularization (with a properly chosen λ) can make a significant improvements over the cases of no regularization ($\lambda = 0$). Based on macro-averaged F1 curves, we chose $\lambda = 10^{-4}$ for regularized LLSF and $\lambda = 10^{-7}$ for regularized NNet and LR for the evaluation of those methods on the test set. We also heuristically tuned the relative weight between positive examples and negative examples for the three classifiers: first, we set the learning rate of positive examples as 0.1 and the learning rate of negative examples as 1; second, we further duplicated the positive examples in each category so that the number of positive examples and negative training examples became equal.

Figure 4 compares our results of the regularized LLSF and regularized NNet with the published results of LLSF and NNet without regularization on the same data collection(Yang & Liu, 1999). Clearly, our new results are significantly better than the previous results of those methods, and the regularization played an important role in making the difference. Note that in (Yang & Liu, 1999), the LLSF used truncated SVD to get the solution and the NNet had 3 layers. Thus, those scores are not directly comparable, but rather just indicative.

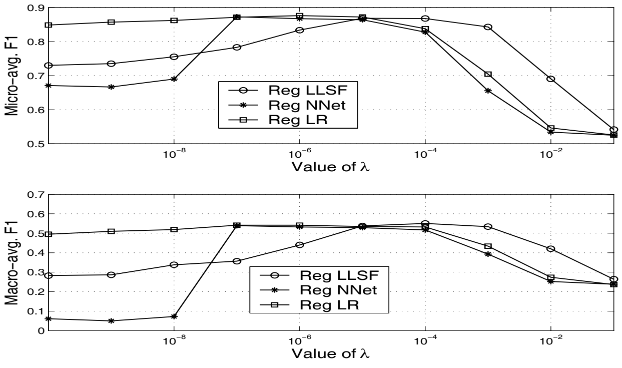

Figure 3. Performance of classifers with respect to varying amounts of regularization

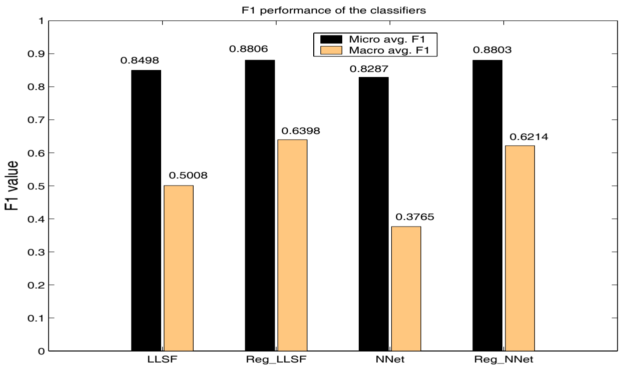

Figure 4. Perfromance of LLSF and NNet with and without regularization

4. Concluding Remarks

In this paper, we presented a loss-function based analysis for eight classifications methods that are popular in text categorization. Our main research findings are:

- The optimization criteria in all the eight methods can be presented using a loss function that consists of two terms: the training-set loss and the regularizer (i.e., the complexity penalty). The proofs for four of those methods, Rocchio, Prototype, kNN and NB, are new in this paper. Such decomposition enables an insightful comparison of classification methods using those two terms.

- Regularized LLSF, NNet and LR have exactly the same regularizer as that in SVM, so the differences among those methods are only in their training-set loss functions. Our evaluation results on the Reuters corpus show that the performance of the four methods are quite competitive, in both macro- and micro-averaged F1 scores, despite the theoretical differences in their loss functions.

- Regularization made significant performance improvements in LLSF and NNet on Reuters. Regularized LLSF, in particular, performed surprisingly well although its training-set loss function is not monotonic, which has been considered as a weakness of this method in some theoretical analysis(Hastie et al. 2001). Its macro-averaged F1 performance (0.6398) is the best score ever reported on the Reuters-21578 corpus, statistically significantly outperforming SVM that was the best until this study.

- Our new derivation shows that NB has the regularizer in a form of $(p + S_c)\|e^{\vec{\beta}}\|_1 + \|\vec{\beta}\|_1$, which is radically different from the $\|\vec{\beta}\|^2$ regularizer in the other classification methods. Whether or not this would be an explanation for the suboptimal performance of NB requires future research.

References

Hastie T., Tibshirani, R. & Friedman, J. (2001) The Elements of statistical learning, data mining, Inference, and Prediction. In *Springer Series in Statistics.*

Joachims, T. (1998) Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Proceedings of the European Conference on Machine Learning (ECML), Springer.

Lewis, D., Yang, Y., Rose, T. & Li, F. (2002) RCV1: A New Text Categorization Test Collection to be appeared in Journal of Machine Learning Research.

McCallum, A. & Nigam, K. (1998) A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization.*

Vapnik, V. (1995) *The nature of statistical learning Theory.* Springer, New York.

Yang, Y. & Chute, C.G. (1994) An example-based mapping method for text classification and retrieval. In *ACM Transactions on Information Systems.*

Yang, Y. & Liu, X. (1999) A re-examination of text categorization methods. *ACM Conference on Research and Development in Information Retrieval*, pp 42–49.

Yang, Y. (1999) An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, Vol 1, pp 67–88.

Zhang, T. & Oles, F.J. (2001) Text Categorization Based on Regularized Linear Classification Methods. In *Information Retrieval* 4(1): 5-31.