Implementation of Probabilistic Outputs for Support Vector Machine

陳奕瑋 B88506052
江岳軒 B88506054
李沛倫 B89902094
吳亭範 B89902098
林虹佑 B89902103

January 9, 2003

Abstract

The output of a classifier should be a calibrated posterior probability to enable post-processing. Standard SVMs do not provide such probabilities. In [1] they introduce a method by training the parameters of an additional sigmoid function to map the SVM outputs into probabilities. This method yields probabilities of good quality, without much effort after training SVMs. Using python [2], we implement this method with LIBSVM [3], and test several data sets.

1 Introduction

Constructing a classifier to produce a posterior probability $P(class \mid input)$ is very useful in practical recognition. Posterior probabilities are also required when a classifier is making a small part of an overall decision, and the classification outputs must be combined for the overall decision. However, Support Vector Machines [4] (SVMs) produce an uncalibrated value indicating which class data "belongs to", rather than a probability.

Some works are proposed for mapping the outputs of SVMs to probabilities. In [5] Hastie and Tibshirani suggests that class-conditional densities $p(f|y = 1)$ and $p(f|y = -1)$ should be fitted to Gaussians. They use a single tied variance to estimate both Gaussians. The posterior probability is thus a sigmoid, whose slope is determined by the tied variance. However, the single parameter derived from the variance may not accuarately model the true posterior probability.

One can also use a more flexible version of the Gaussians fit to $p(f|y = \pm 1)$. The mean and the variance for each Gaussian are determined from a data set. Bayes' rule can be used to compute the posterior probability via:

$$p(y=1|f) = \frac{p(f|y=1)P(y=1)}{\sum_{i=-1,1} p(f|y=i)P(y=i)}, \tag{1}$$

where $P(y=i)$ are prior probablities that can be computed from the training set. In this formulation, the estimated posterior probability is an analytic function of f with form:

$$P(y=1|f) = \frac{1}{1+exp(af^2+bf+c)}. \tag{2}$$

There are two issues with this model. First, the posterior estimate above is not monotonic in f. There's a very strong prior for considering the probability $p(y=1|f)$ to be monotonic in f since intuitively with larger f the data should be more away from the separating hyperplane, thus more possible to be a positive example. Second, the assumption of Gaussian class-conditional densities is sometimes failed.

Following we will introduce how [1] do modifications to SVMs, an efficient algorithm for implementation, and finally results to several data sets applied by this method.

2 Fitting a Sigmoid After the SVM

In [1], instead of estimating the class-conditional densities $p(f|y)$, they use a parametric model to fit the posterior $P(y=1|f)$ direclty. The parameters of the model are adapted to give the best probability outputs. Observing empirical data, the form of the parametric model are inspired. They found that the class-conditional densities between margins are usually exponential. By applying the Bayes' rule on two exponentials we can get a parametric form of a sigmoid:

$$P(y=1|f) = \frac{1}{1+exp(Af+B)} \tag{3}$$

Now we want to train a sigmoid by some SVM decision function values f. We use maximum likelihood estimation to fit the parameters A and B. Given a sigmoid training set (f_i, y_i), we define a new training set (f_i, t_i), where the t_i are target probabilities defined as:

$$t_i = \frac{y_i+1}{2}, \tag{4}$$

so that $t_i = 0$ while $y_i = -1$; $t_i = 1$ while $y_i = 1$. According to maximum likelihood estimate, we maximize: $\prod_i \hat{p}_i$, where:

$$\hat{p}_i = \begin{cases} \frac{1}{1+exp(Af_i+B)} & , y_i = 1 \\ 1 - \frac{1}{1+exp(Af_i+B)} & , y_i = -1 \end{cases} \tag{5}$$

This is the same as minimizing the negative log likelihood, which is a cross-entropy error function:

$$\min_{A,B} -\sum_i t_i \log(p_i) + (1-t_i)\log(1-p_i), \tag{6}$$

where

$$p_i = \frac{1}{1+exp(Af_i+B)}. \tag{7}$$

Therefore we have to do this two-parameter minimization to get A and B.

3 Levenberg-Marquardt Algorithm

We use Levenberg-Marquardt Algorithm [5, 6] to do the optimization above. Roughly speaking, this is a compromise between Newton method and gradient descent. Like both methods, Levenberg-Marquardt algorithm is an iterative algorithm, with update as:

$$x \leftarrow x - M(x)^{-1}\nabla f(x), \tag{8}$$

where

$$M_{ij} = (1 + \delta[i-j]\lambda)H_{ij}. \tag{9}$$

Here H_{ij} is the Hessian matrix whose elements are $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. We can see that when λ is zero, it is exactly the Newton method; when λ is very large, it is then like the gradient descent. So λ here is in a sense a parameter to "control" which method to apply.

When doing an optimization problem using an iterative method, in early stage, we may be far from solution and the parabolic assumption is sometimes wrong, so steepest descent is more safe. When we are close to the solution, then Newton method is better for speeding up. So, here's what Levenberg-Marquardt algorithm does: whenever $f(x_{k+1})$ gets worse (larger than $f(x_k)$), increase λ by a factor $\lambda \leftarrow \beta_1\lambda$ to make step more like a gradient step. Whenever $f(x_{k+1})$ gets better (less than $f(x_k)$), decrease λ by a factor $\lambda \leftarrow \beta_2\lambda$ to make step more like a Newton step. We use $\beta_1 = 10$, $\beta_2 = 0.1$ in our program.

4 Some Issues to Deal With

There are some issues in the optimization procedure: the choice of the sigmoid training set, and the method to avoid over-fitting this set.

When we are given a training data set, to build a sigmoid model, one may think that we can train an SVM with all training samples, then feed it with the same samples to get each f_i. However, this cause the SVM outputs f_i to be a biased estimate of the sample distribution. For example, for the data at the margin, the SVM outputs are forced to have the same absolute value 1, which are not the common values for test samples. So in our program we use a 5-fold cross validation to generate f_i. That is, the training samples are split into five parts, and each of five SVMs are trained on permutations of four out of five parts, then the f_i are evaluated by the remaining one. By cross-validation we can get a good unbiased f to make the sigmoid model.

Another isuue is over-fitting, which is indicated in [1]. They notice that with unbalanced data, fitting a sigmoid with the SVMs will make A biased. Therefore there can be an infinite number of sigmoids with infinite steep sigmoids when the validation set is perfectly separable. [1] sugeests to use the out-of-sample data. Out-of-sample data is modelled with the same empirical density as the sigmoid training data, but with a finite probability of opposite label. In other words, when a positive example is observed at a value f_i, we do not use $t_i = 1$ but assume that there is a finite chance of opposite label at the same f_i in the out-of-sample data. Therefore, a value of $t_i = 1 - \epsilon_+$ will be used. Similarly, a negative example will ues a target value of $t_i = \epsilon_-$. In [1] they set:

$$t_+ = 1 - \frac{1}{N_+ + 2} \ , \ t_- = \frac{1}{N_- + 2}, \tag{10}$$

where N_+ and N_- are number of positive and negative samples. These targets are then used instead of {0,1} for all data in the sigmoid fit.

5 Implementation

We modified the **svm_predict()** in svm.cpp of libsvm to output the decision value. Then a post-processing program using libsvm python interface is written to find out correct parameters A, B that minimize the likelihood function.

Figure 1 shows the predicted probability against decision value mapping function and ROC curve of **heart_scale** for demostration.

Dataset	#Positive	#Negative
22pos500	400	100
22bln500	250	250
22neg500	100	400
imgpos500	400	100
imgbln500	250	250
imgneg500	100	400

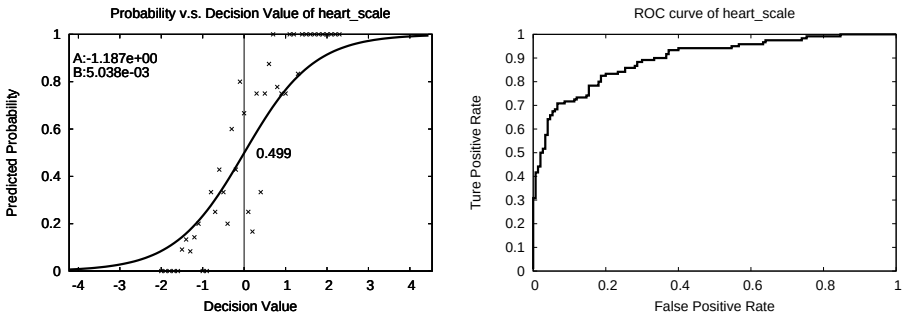

Figure 1: Probability mapping function and ROC curve of heart_scale

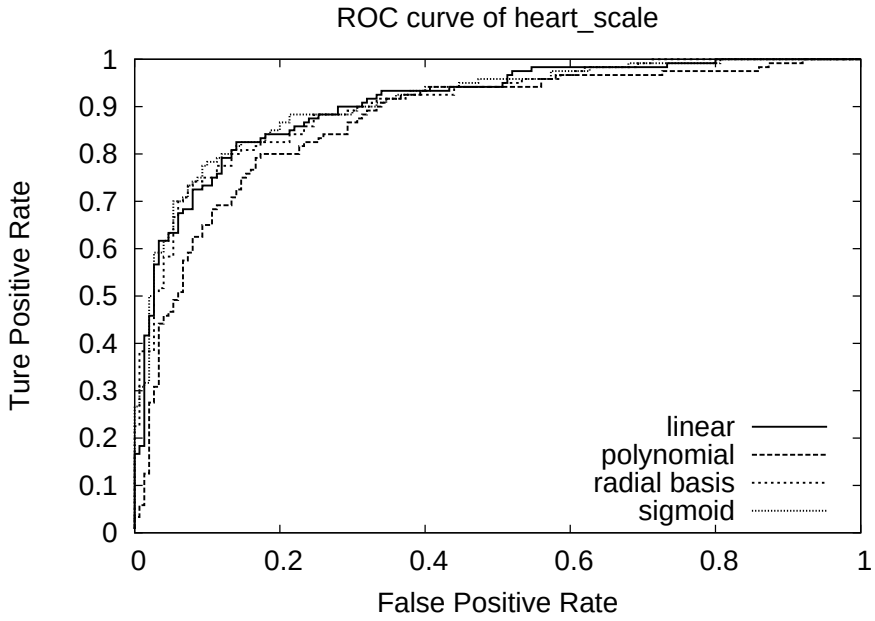

Figure 2: ROC curve with different kernel type

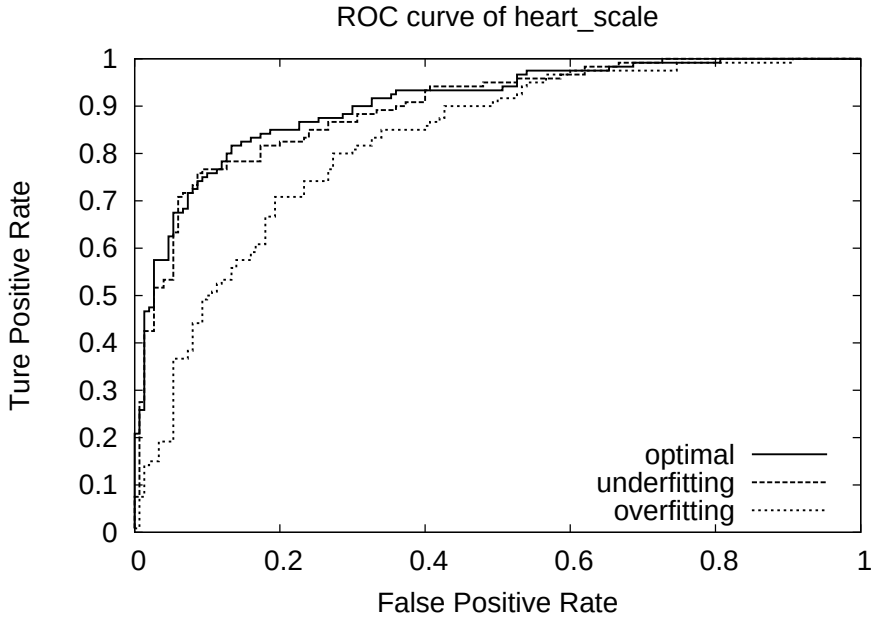

Figure 3: ROC curve with different gamma value

5.1 Characteristics of Probablistic Output SVM: An Empirical View

First we try probability output SVM with different kernel type. In the original paper, the author applied experiment only on linear kernel, Figure 2 shows performance of different kernel with heart_scale. All kernel is fine tuned with their best confiuration. It seems that RBF kernel is better in this case. However, they are closed to each other. Different data such as linear-separable ones may lead to completely different result.

Second, we want to know the influence of inappropriate **cost** and **gamma** to probability output SVM with RBF kernel. Figure 3 shows three ROC curves, each is from heart_scale with same optimized parameters but with different **gammra**. Best **gamma** of this sample is $(C, gamma) = (2^7, 2^{-10})$), the one marked overfitting is plotted with $(C, gamma) = (2^7, 2^0)$, and underfitting is plotted with $(C, gamma) = (2^7, 2^{-20})$. Obviously, SVM with optimal model selection results in best performance under most scenario.

Third, we explored the its ability of learning from unbalanced data. The unbalanced datasets for testing is randomly extracted from 22features and image_scale with specified positive and negative instances(shown in table 1).

Figure 4 shows three curves of probability against decision value mapping function with different bias of data. The curve varies with different bias. The

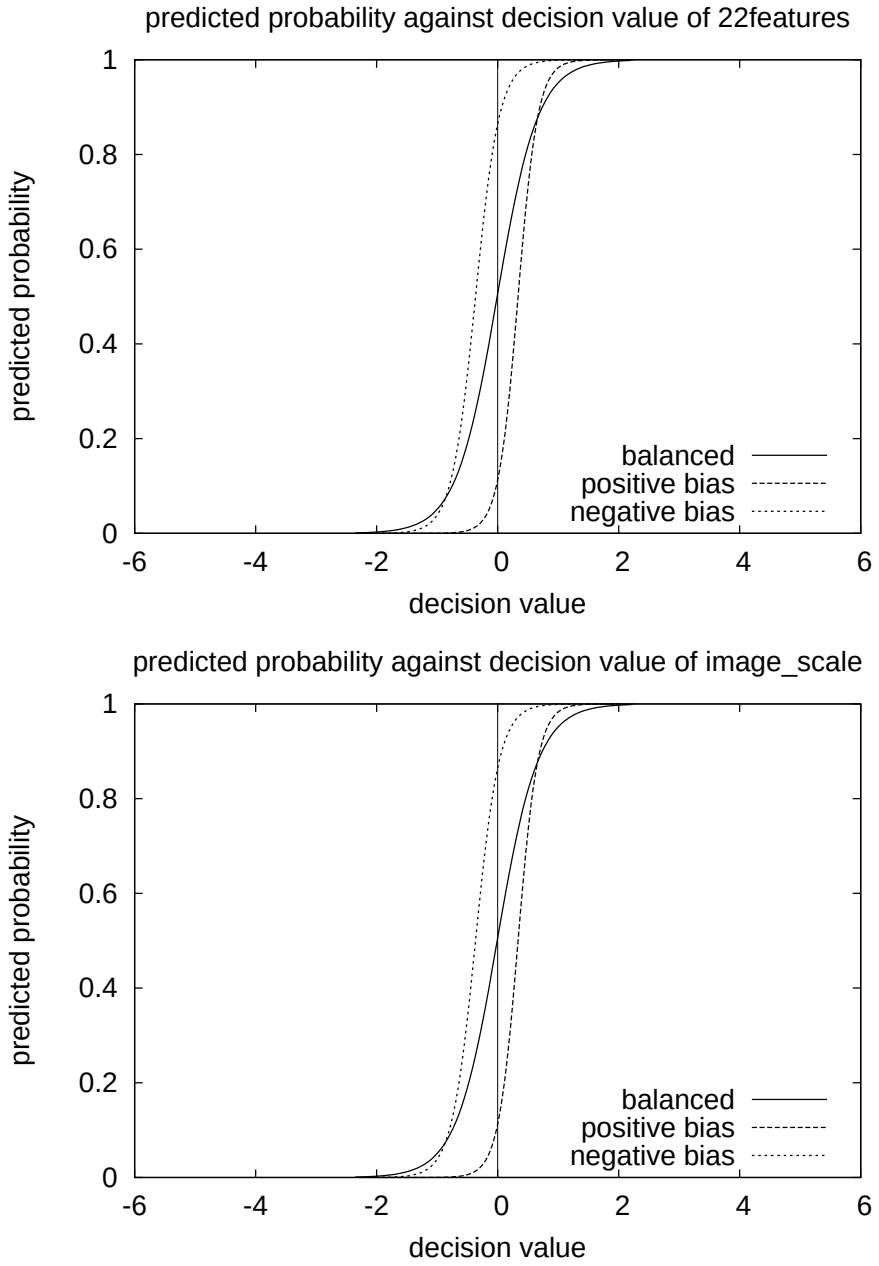

Figure 4: Probability against decision value with different data bias

reason it very simple, because different data distribution constructs different scenario for approximation. Positive bias may provide more positive instance when training, but its distribution, say range of decision value, also have great influence on curve fitting.

To sum up, this implementation enables gerneric svm to output approximated probability instantly with high accuracy. But we suggent everyone using this this implementation to fine tune SVM in standard way before any further processing to get more accurate result.

References

[1] J. C. Platt (1999) *Probabilistic Outputs for Support Vector Machine and Comparisons to Regularized Likelihood Methods*

[2] http://www.python.org

[3] C. C. Chang and C. J. Lin (2001) *LIBSVM: A library for support vector machines* Software available at http://www.csie.ntu.edu.tw/čjlin/libsvm

[4] Vapnik, V. (1998) *Statistical Learning Theory* New York, NY: Wiley

[5] P. E. Gill, W. Murray, and M. H. Wright (1981) *Practical Optimization* Academic Press

[6] http://mayaweb.upr.clu.edu/ jechauz/vg-lm.pdf

[7] http://gim.unmc.edu/dxtests/roc3.htm *The Area Under an ROC Curve*

[8] Gary M. Weiss and Foster Provost (2001) *The Effect of Class Disribution on Classifier Learning: An Empirical Study*

[9] Foster Provost and Tom Fawcett (2001) *Robust Classification for Imprecise Environments*