Documentation for Regularized LR and Regularized SVM code

Kamalika Chaudhuri Anand Sarwate Claire Monteleoni

July 8, 2013

1 Code

The code for the differentially private regularized LR and regularized SVM algorithms in the paper [1] are in the files `lrsimple.c` and `svmsimple.c` respectively.

Compiling. We implement the convex optimization procedures using the LBFGS optimization procedure. Our code uses the LBFGS library from:

`http://www.chokkan.org/software/liblbfgs/`

This library is needed to compile the code. After installing the library, the following works on Linux or the Mac for compiling the LR code:

```
gcc -lm -llbfgs lrsimple.c
```

To run the resulting program, do:

```
./a.out <data-file>
```

The SVM code can be similarly compiled and run.

```
gcc -lm -llbfgs lrsimple.c
```

Input Format. The LR code expects as input an ASCII text file which has the following format. The first four floating point numbers in the file are: n, d, λ, and ϵ_p. n is the number of training points, d is the dimensionality of the training data, λ is the regularization parameter and ϵ_p is the privacy parameter. Next we have the n training data points: each of the next n lines correspond to a feature vector of length d, and thus has d floating point numbers. Finally we have the n training labels – another n floating point numbers which can be -1 or 1. An example of an input file which works correctly with the code is provided in `bloodlr.txt`.

The input format for the SVM code is exactly the same, except that there are five instead of four floating point numbers in the beginning: n, d, λ, ϵ_p and h. Here h is the Huber constant, which lies between 0 and 0.5; setting h too close to zero leads to numerical instability, and we usually set it to be $h = 0.5$. An example of an input file for the SVM code is `bloodsvm.txt`.

For more details on the input format expected, see Section 2.

Output Format. The code outputs three lines, each with d floating point numbers followed by an integer. The first line corresponds to a non-private classifier trained on the training data using the parameter λ; the second line to an ϵ_p-differentially private classifier obtained by output perturbation and the third to an ϵ_p-differentially private classifier obtained by objective perturbation.

The first d floating point numbers in each line correspond to the d feature values in the classifier w; the last integer is the value returned by the optimization procedure. If the procedure converged correctly, then this value should be zero; any other value indicates an error in convergence. Since LBFGS is a second-order optimization algorithm, it is sometimes numerically unstable for low values of λ and ϵ_p. In some of our experiments, we found that the optimization algorithm had trouble converging for objective perturbation for $\lambda \approx 10^{-4}$, and even for non-private logistic regression for $\lambda \approx 10^{-6}$ or so.

For more details on the output, see Section 2.

2 Algorithm

Preprocessing. The code in `lrsimple.c` and `svmsimple.c` assumes that the data is preprocessed in the sense that the length of each of the training data vectors $\|x_i\| \leq 1$ and the labels are -1 and 1 (**not** 0 and 1); you may need to do some initial preprocessing to ensure this. LR or SVM will not give you the right results unless the labels are -1 and 1, and the privacy guarantees of the differentially private algorithms (output perturbation and objective perturbation) will not hold unless each $\|x_i\| \leq 1$.

Algorithm Version. This code implements a slightly different version of the objective perturbation algorithm than what is provided in [1]. The algorithm implemented here is:

1: Calculate: $\epsilon'_p = \epsilon_p - 2\log(1 + \frac{c}{n\lambda})$.
2: If $\epsilon'_p < 10^{-4}$, output an error message.
3: Otherwise, draw b from the density: $\rho(z) \propto e^{-\frac{1}{2}\epsilon'_p z}$, and solve the optimization problem:

$$\frac{1}{2}\lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(y_i w^\top x_i) + \frac{b^\top w}{n}$$

Here ℓ is the loss function, where $\ell(z) = \log(1+e^{-z})$ for logistic regression and the Huber loss with Huber constant h for SVM. The main difference between this implementation and the algorithm as stated [1] is that in the paper, if ϵ'_p is too small, we do some additional regularization by adding a term $\frac{1}{2}\Delta\|w\|^2$, which is not done in the code. The code is written this way to (a) avoid numerical instability that may arise for very small ϵ'_p, and (b) provide some additional flexibility in how to handle the case when ϵ'_p is too small by overregularizing.

References

[1] Kamalika Chaudhuri, Claire Monteleoni and Anand Sarwate, *Differentially Private Empirical Risk Minimization*, Journal of Machine Learning Research, 2011.