CS534 Project 1
Yaonan Zhong
April 16, 2014

Part 1 Linear Regression with L2 Regularization

1. Experiments: Test the effect of the regularizer on the linear regressor

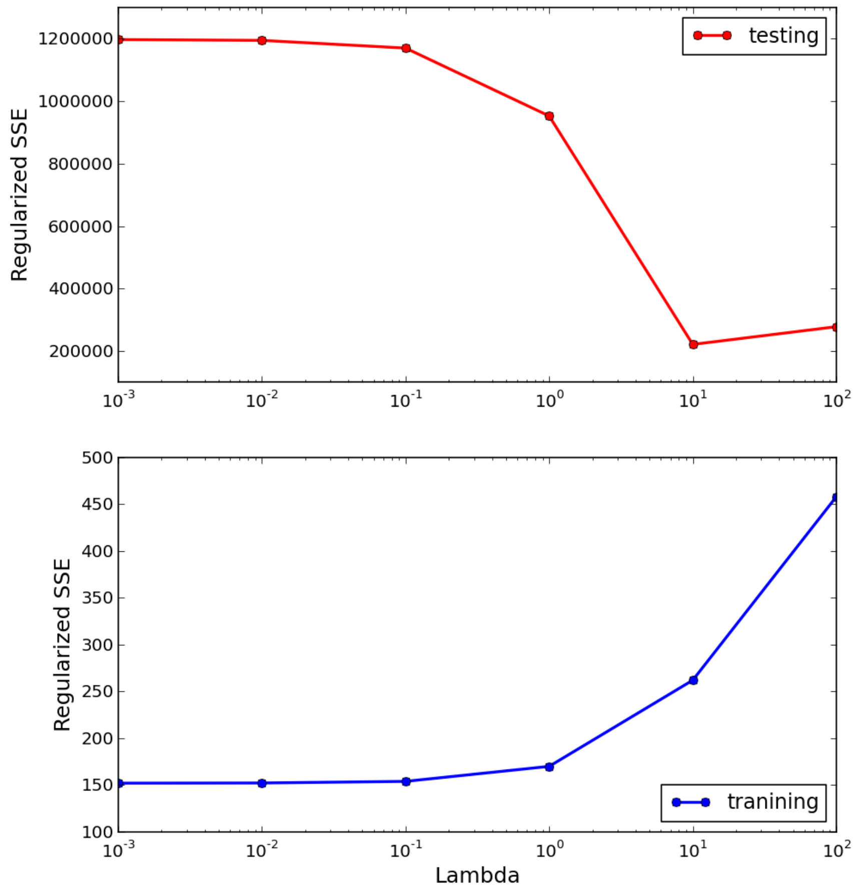

Fig. 1 Regularized SSEs against regularization coefficient Lambda

In this project I use close-form analytical solution to optimize the regularized SSE objective. Figure 1 shows the relationships between regularized SSE and regularization coefficient *lambda* for both training and testing sets. The training SSE increases as lambda increases, while the testing SSE decreases as lambda increases. The reason for the trend from training set is that small lambda encourages better fit of the data, driving the SSE to zero. However, if lambda is too small, it will lead to the over-fitting issue, as shown in the testing set. Another problem we can see here is that the SSEs of testing set are quite large (2e+5 ~ 12e+5). We know that the

SSE come from: $SSE = \frac{1}{2}\sum_{i=1}^{N}(y_i - wx_i)^2 + \frac{1}{2}\lambda|w|^2$ and N=100, so I look into those terms in this equation and find that the large SSEs of testing set actually mainly come from some singular squared errors, which makes the value of $(y_i - wx_i)^2$ pretty large. Figure 2 shows the SSEs after removing those singular points.

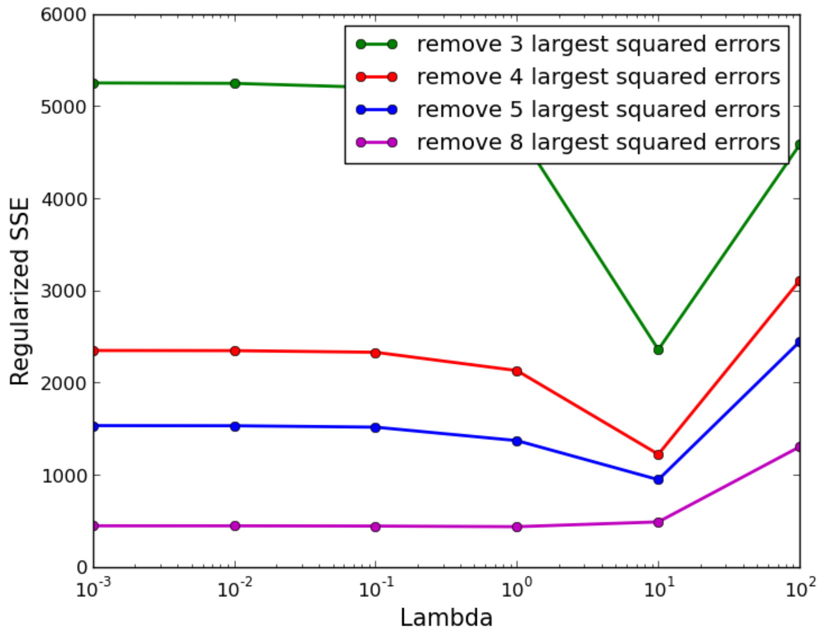

Fig. 2 SSE of testing set after removing some singular squared errors

Figure 2 shows that the SSE of testing set reduce to ~2e+3 (~1e+6 in fig.1) after removing 4 largest squared errors. Note that the total test set has 100 examples.

2. Explorations: Identify a set of features relevant to the target variable

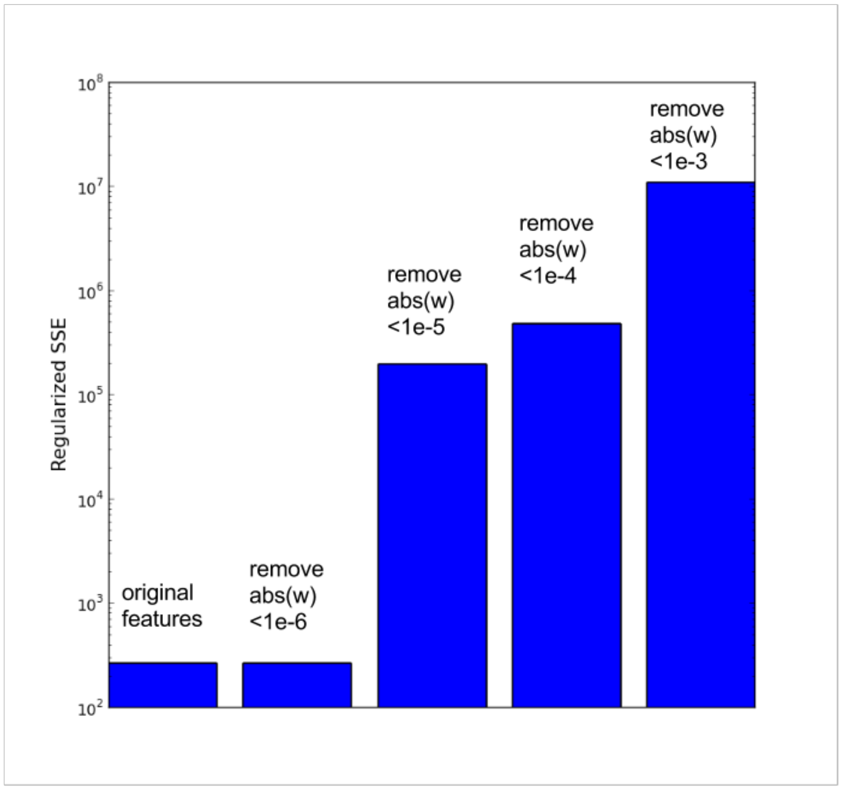

Fig. 2 Regularized SSE after removing some features with different thresholds

In figure 1, we can see that lambda = 10 would be a proper starting point. Figure 2 shows the training SSEs after removing some features based on different thresholds(1e-6, 1e-5, 1e-4, 1e-3). For example, if a absolute weight is smaller than the threshold, its value is set to 0, then the corresponding feature has no effect on the target variable. We can see that after removing abs(w)<=1e-6, the SSE is almost the same as that of original features. The reason is that these features has little contribution to the target variable. However, if we increase the threshold, the SSE increases dramatically, because more elements of W are set to 0 and the model become simpler.

Codes: regression.py, linreg.py

Part 2 Batch Perceptron and Voted Perceptron

1. Batch Perceptron

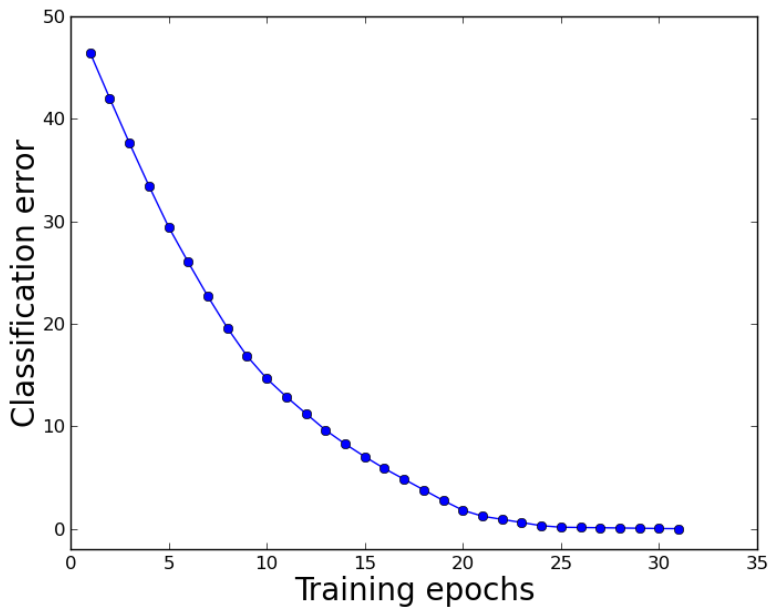

Fig. 3 Classification error against training epochs for batch perceptron

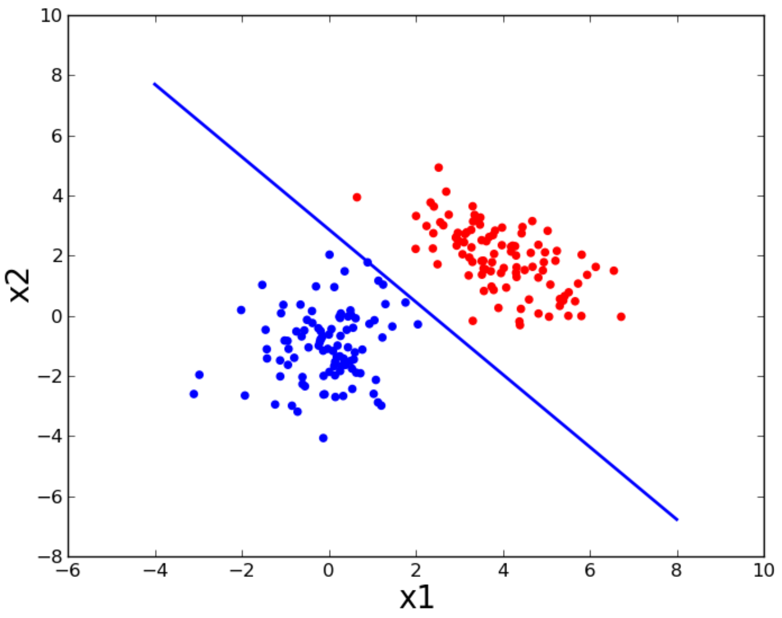

Fig. 4 Scatter plot of training data and decision boundary for batch perceptron
w = [1.5, -0.62886653, -0.52146511]

2. Voted Perceptron

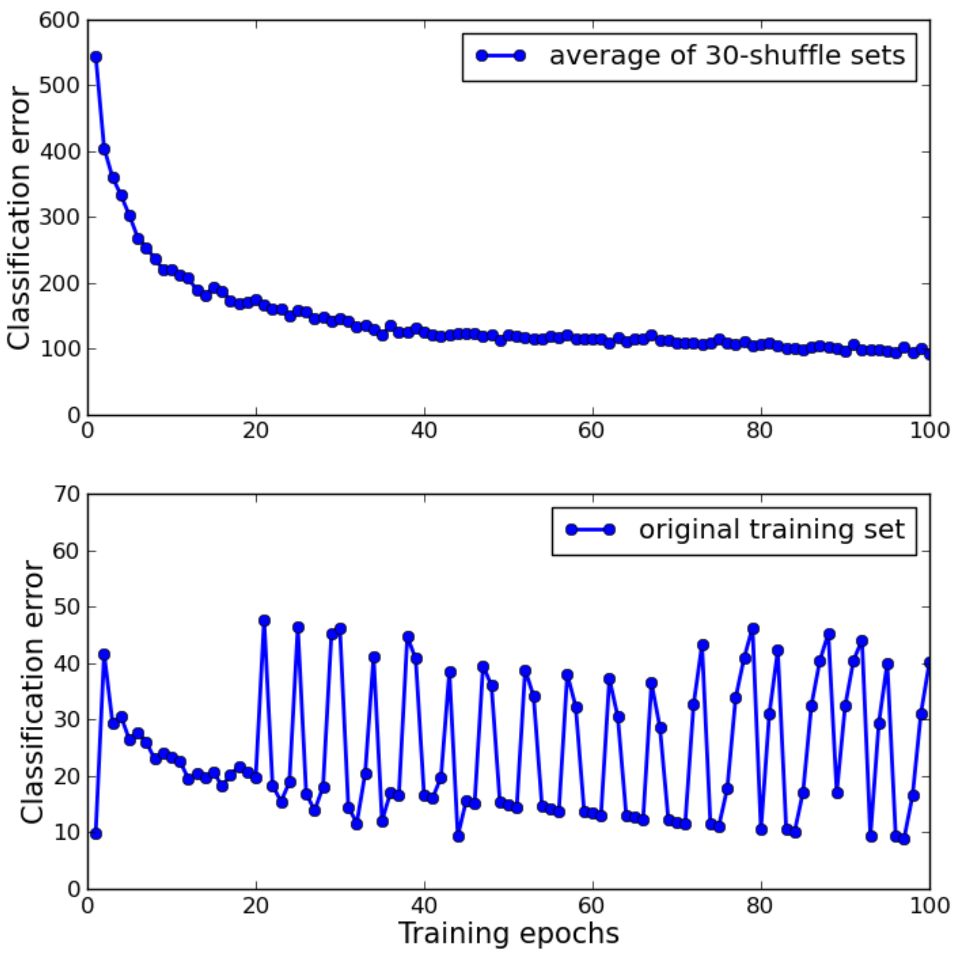

Fig. 5 Classification errors of original training set and shuffled sets for voted perceptron

Figure 5 shows the classification errors of training set against training epochs. The error of original training set does not converge and shows oscillation after 20 epochs, while the average error of shuffled sets converges after 20 epochs. However, the error of the original set is smaller than that of shuffle sets. It is because that our algorithm use stochastic gradient descent which is sensitive to the order of training examples presenting to the algorithm. And our original set is very special: the first half has y=1 and the second half has y=-1. In the shuffled sets, the examples of y=-1 and y=1 are in random order, which makes the classification error larger than that of the original set in special order. However, the random order reduces the sensitivity in some extent and leads to convergence.

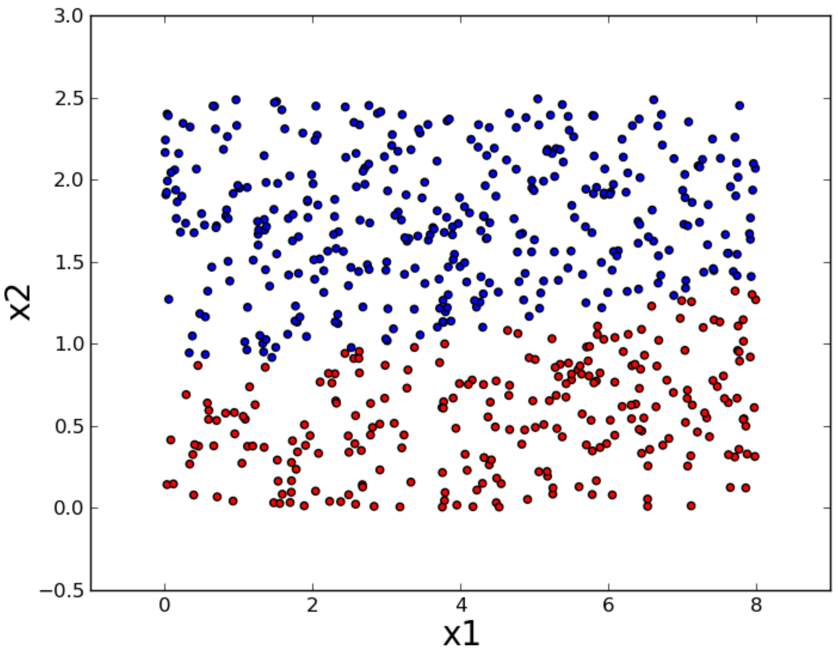

Fig. 6 Visualization of decision boundary for voted perceptron with sample points

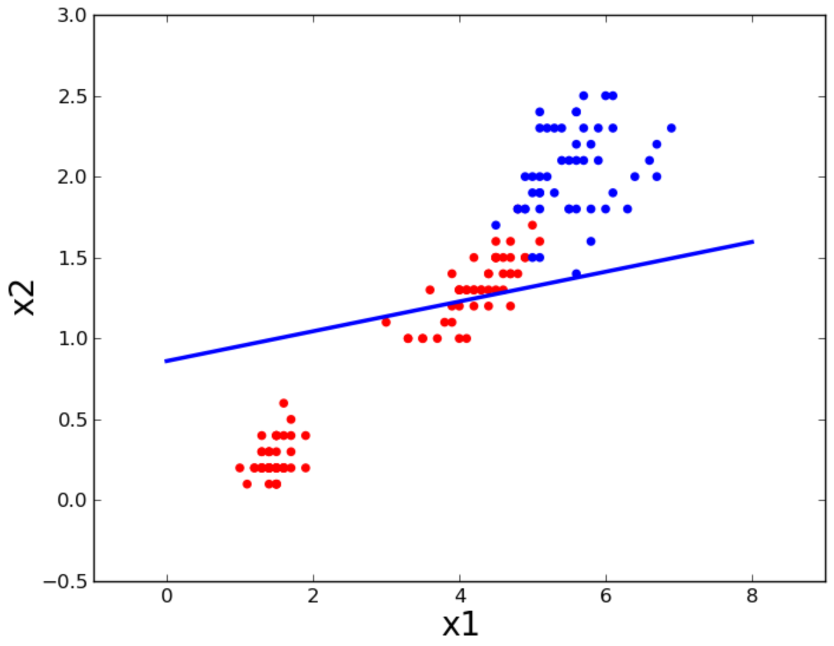

Fig. 7 Scatter plot of training data and average decision boundary for voted perceptron

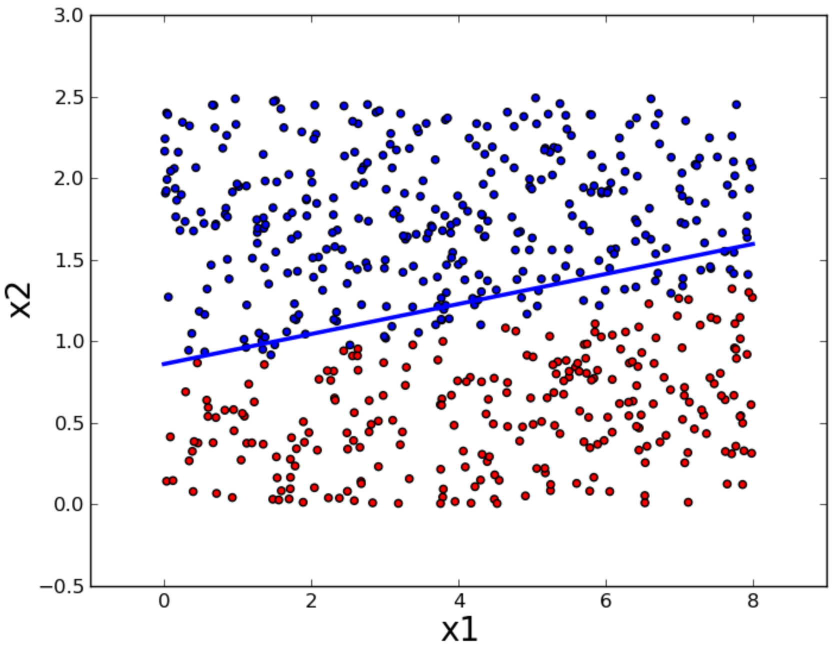

Fig. 8 Scatter plot of sample points and average decision boundary

Figure 8 shows that the average decision boundary is identical to the boundary visualized with sample points except some nonlinearly separable points. The reason is that the average perceptron still maintains a collection of weight vectors and survival times. The only difference is the presence of the interior sign operator. And it is more efficient.

Codes: batch.py, voted.py, perceptron.py

Computation tools: Python, Numpy, Scipy, Matplotlib