Project 1 Report: Logistic Regression

Si Chen and Yufei Wang

Department of ECE
University of California, San Diego
La Jolla, 92093
{sic046, yuw176}@ucsd.edu

Abstract. In this project, we study learning the Logistic Regression model by gradient ascent and stochastic gradient ascent. Regularization is used to avoid overfitting. Some practical tricks to improve learning are also explored, such as batch-based gradient ascent, data normalization, grid searching, early stopping, and model averaging. We observe the factors that affect the result, and determine these parameters. Finally, we test the algorithm on Gender Recognition [DCT] dataset.[1]. We use L-BFGS method on the same dataset as a comparison. Our model trained by stochastic gradient ascent achieves around 92.89% accuracy.

1 Introduction

Logistic regression is a widely used statistical classification model. In this project, we implement L_2 regularized logistic regression models with two optimization methods, stochastic gradient ascent (SGA) and L-BFGS. Stochastic gradient ascent method is realized by ourselves. We apply some practical tricks to improve the model: we use a batch of examples instead of single example for each round of gradient-based update; we use early stopping to further avoid overfitting; we use model averaging to reduce the noise caused by data randomization. By experiments, we observe each parameter's effect on the model, and decide the value of parameters.

2 Regularized logistic regression model

Maximum likelihood estimation. Maximum likelihood (ML) is a method to estimate parameters of a statistical model. When we have n independent examples drawn from a distribution with parameter θ, the likelihood function is

$$L(\theta; x_1, ..., x_n) = f(x_1, ..., x_n; \theta) = \prod_{i=1}^{n} f_\theta(x_i; \theta) \tag{1}$$

where $f_\theta(x_i; \theta)$ is the probability of ith example.

[1] http://mlcomp.org/datasets/1571

ML gives the estimation of θ that maximizes the likelihood function

$$\hat{\theta} = argmax_{\theta} L(\theta; x_1, ..., x_n) \tag{2}$$

ML can be easily generalized to conditional likelihood condition

$$\hat{\theta} = argmax_{\theta} \prod_{i=1}^{n} f(y_i|x_i; \theta) \tag{3}$$

where $f(y_i|x_i; \theta)$ is the conditional probability of ith example.

Logistic regression model. Logistic regression model is the conditional model

$$p(y|x; \beta) = \frac{1}{1 + exp - \left\{\beta_0 + \sum_{j=1}^{d} \beta_j x_j\right\}} \tag{4}$$

where y is a Bernoulli outcome and x is real-valued vector. β_j are parameters to estimate.

Given training set $\{\langle x_1, y_1\rangle, ..., \langle x_n, y_n\rangle\}$, we estimate the parameters by maximizing the log conditional likelihood

$$\Lambda = LCL = log\left\{\prod_{i=1}^{n} p(y_i|x_i; \beta)\right\} = \sum_{i=1}^{n} logp(y_i|x_i; \beta) \tag{5}$$

Stochastic gradient ascent. This maximization problem can be solved with gradient ascent approach. The partial derivative of the objective function is

$$\frac{\partial}{\partial \beta_j} LCL = \sum_{i=1}^{n} (y_i - p_i) x_{ij} \tag{6}$$

where x_{ij} is the value of jth feature of ith training example. The gradient-based update of parameter β_j is

$$\beta_j := \beta_j + \lambda \frac{\partial}{\partial \beta_j} LCL \tag{7}$$

where λ is step size.

The problem of this approach is that it has high computational cost. Therefore, we use SGA. We randomly choose single example to approximate the true derivative based on all the training data. For each β_j, we define a random variable Z_j such that

$$E[Z_j] = \frac{\partial}{\partial \beta_j} LCL \tag{8}$$

and use Z_j to approximate the partial derivative $\frac{\partial}{\partial \beta_j} LCL$.

When a training example $\langle x_i, y_i \rangle$ is chosen randomly with uniform probability, we can calculate one such Z_j

$$Z_j = n(y_i - p_i)x_{ij} \tag{9}$$

Therefore, the stochastic gradient update of β_j is

$$\beta_j := \beta_j + \lambda'(y_i - p_i)x_{ij} \tag{10}$$

where $\langle x_i, y_i \rangle$ is randomly selected example, and learning rate $\lambda' = n\lambda$ controls the magnitude of changes each time.

λ' is a function of sample size n, which means changes of n cause changes to the choice of λ'. To avoid this, we modify the objective function to

$$\Lambda' = \frac{1}{n} LCL = \frac{1}{n} \sum_{i=1}^{n} logp(y_i | x_i; \beta) \tag{11}$$

Correspondingly, the partial derivative of the objective function becomes

$$\frac{\partial}{\partial \beta_j} \Lambda' = \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i)x_{ij} \tag{12}$$

and the update of parameter β_j becomes

$$\beta_j := \beta_j + \lambda(y_i - p_i)x_{ij} \tag{13}$$

Regularization. This model has the problem of overfitting. Regularization is done by imposing a penalty term μ. The objective function becomes

$$\Lambda'' = \frac{1}{n} LCL - \mu \left\| \beta \right\|_2^2 \tag{14}$$

where $\left\| \beta \right\|_2^2 = \sum_{j=1}^{d} \beta_j^2$ is the squared L_2 norm of vector β.

The partial derivative of objective function Λ'' is

$$\frac{\partial}{\partial \beta_j} \Lambda'' = \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i)x_{ij} - 2\mu\beta_j \tag{15}$$

So the stochastic gradient-based update rule with regularization is

$$\beta_j := \beta_j + \lambda \left[(y_i - p_i)x_{ij} - 2\mu\beta_j \right] \tag{16}$$

where $\langle x_i, y_i \rangle$ is randomly selected example. β_0 is not regularized. The choice of hyperparameters λ and μ is not sensitive to sample number n.

Note that equation 14 corrects an error which occurs on page 11 of the lecture note [1]. With the coefficient $\frac{1}{n}$, objective function in equation 14 matches the update rule in equation 16.

3 Design and analysis of algorithms

Algorithms are realized in python.

3.1 Data preparation

Data structure. Training data, validation data and test data are stored in matrix to accelerate the computing speed, because matrix operations can be calculated quickly in python.

Every data sample x_i has d dimensions. Conditional probability $p(y_i|x_i;\beta)$ in equation 4 can be calculated with matrix

$$p(y_i|x_i;\beta) = \frac{1}{1 + exp - (z_i \cdot \beta)} \tag{17}$$

where $z_i = [1, x_{i1}, x_{i2}, ..., x_{id}]$ is the expansion of x_i with an extra 1, and $\beta = [\beta_0, \beta_1, ..., \beta_d]^t$.

Therefore, training data is an $n_{training} \times (d+1)$ matrix, where $n_{training}$ is the number of training examples. Each row z_i is features of ith example with an extra 1 at first column. Similarly, test data is an $n_{test} \times (d+1)$ matrix, where n_{test} is the number of test examples. Label of training data is an $n_{training} \times 1$ matrix, and label test data is an $n_{test} \times 1$ matrix.

Data randomization. The training examples given may not be in random order, which may produce misleading results. Therefore, we need to randomize the dataset first before dividing it into validation subset and training subset.

The feature matrix and label matrix are combined into one extended matrix, and the extended matrix is shuffled. In this way, the dataset can be randomized efficiently and correctly.

Randomization is done once at the very beginning of the system.

Feature normalization. Hyperparameters λ and μ remain the same for all β_j, as is shown in Equation 16. Therefore, we need to standardize the range of the features $x_j.(j = 1, ..., d)$

We use z-scoring to scale the features. After randomization, training subset is used to normalize features. We make every feature in training data have 0 mean and 1 standard deviation

$$Normalized(x_{ij}) = \frac{x_{ij} - mean(x_j)}{std(x_j)} \tag{18}$$

where $mean(x_j)$ is the mean of jth feature over all training examples, and $std(x_j)$ is the standard deviation of jth feature over all training examples.

Note that validation data should not be used in normalization.

3.2 Parameter tuning

There are two parameters needed to decide: λ and μ.

Grid search is used to find optimal parameters. First, a finite set of reasonable values for each of λ and μ is selected. Then, we train the model with each pair (λ, μ) of these two sets, and the pair which achieves the best classification result in validation set is chosen. The sets consist of exponentially growing sequences of parameters, which provides a large range to find best pairs.

For simplicity, we use a single validation subset instead of using cross-validation. After randomizing the training data, the dataset is divided into training subset and validation subset. The number of examples in validation subset is predefined.

3.3 Modifications in SGA algorithm

Modification of learning rate. Instead of using constant learning rate λ during training process, we modify the learning rate function to

$$\lambda(t) = \frac{2}{t^{1.4}} + \lambda_0 \tag{19}$$

where t is the number of gradient-based update, λ_0 is the hyperparameter we decide in grid search, and $\lambda(t)$ is the learning rate we use in tth gradient-based update.

The advantage of changing learning rate is:

- It accelerates the training process, especially in the early epochs.
- It limits the minimum learning rate to λ_0. This avoids learning rate becoming too small with large epoch number.
- When number of gradient-based update is large enough, the first term $\frac{2}{t^{1.4}}$ is small compared with λ_0, therefore the dominant part in this term becomes λ_0. This ensures the effects of λ_0. $t^{1.4}$ makes $\lambda(t)$ drops with t in appropriate speed.

Batch. The advantage of SGA is it greatly reduces the computing time to evaluate derivatives. However, approximating the gradient by one sample a time may gets bad approximation some time, with examples that are not representative. This will reduce the converging rate.

A compromising method is to use a m-size batch of examples to approximate the gradient. In this way, single outlier will not impact the approximation significantly. The update rule with m-size batch is

$$\beta_j := \beta_j + \lambda_t \left[\frac{1}{m} \sum_{i=1}^{m} (y_{t_i} - p_{t_i}) x_{t_i j} - 2\mu\beta_j \right] \tag{20}$$

where $(t_1, t_2, .., t_m)$ is the chosen numbers from training data to form the batch.

The realization of this approach uses matrix operation. Equation 13 can be rewritten as

$$\beta := \beta + \lambda_t \left[\frac{1}{m} z_{batch}^t \cdot \left[y_{batch} - \frac{1}{1 + exp - (z_{batch} \cdot \beta)} \right] - 2\mu\beta \right] \tag{21}$$

where z_{batch} is the $m \times (d+1)$ matrix, with 1's in the first column, and each batch element's features in each row. y_{batch} is the $m \times 1$ matrix, with each batch element's label in each row. $\beta = [\beta_0, \beta_1, ..., \beta_d]^t$. The fraction $\frac{1}{1+exp-(z_{batch} \cdot \beta)}$ is element wise operation, and is calculated by python. The result of this fraction is a $m \times 1$ matrix.

Early stopping. To further avoid overfitting, we use early stopping. An adaptive parameter $EndNumber$ is set.

The initial value of $EndNumber$ is

$$EndNumber = BatchNumber \times 200 \tag{22}$$

where $BatchNumber$ is number of batches in one epoch. The initial setting is that training process ends within 200 epochs.

For every several epochs, the mean error rate on validation data is calculated. If the current mean validation error rate is best error rate achieved, $EndNumber$ is updated

$$EndNumber = max\{EndNumber, BatchNumber \times EpochNumber \times 2\} \tag{23}$$

where $EpochNumber$ is the current number of epochs. Therefore, $BatchNumber \times EpochNumber \times 2$ is twice the value of current β-update iteration number. This means when the current model gets best performance on validation data, we increase the epoch number for training.

When the iteration number reaches $EndNumber$, the training process is terminated, and the model with best performance on validation set is chosen.

Model averaging. Note that we randomize the data before each model training process, and the validation set is different every time. Therefore, there is much noise in every model we train. A good way to reduce the random noise is to average several models to get the final model. The advantage of this approach can also be explained in terms of utilization efficiency of data. Single model is trained with a part of entire training data: in our implementation 75% of training data is used to train the model. By combining several models, more training data is used, with no impact on the role of validation model.

The modified model is built from K models trained by SGA. The parameters of kth models is $\beta_k, k = 1, 2, ..K$. In the new model, for every test data x_i, the probability of its label being 1 is defined

$$p(y_i|x_i; \beta_1, ..., \beta_K) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{1 + exp - (z_i \cdot \beta_k)} \tag{24}$$

This is the average of probability $p(y_i|x_i;\beta)$ of k models. Decision rule remains the same.

3.4 L-BFGS

Apart from SGA, we also use L-BFGS optimization method for comparison. We use the open-source software SciPy for L-BFGS [2]. We need to input loss function, gradient of the loss function, and penalty term μ.

The loss function is

$$L = -\sum_{i=1}^{n} logp(y_i|x_i;\beta) + \mu \left\|\beta\right\|_2^2 \tag{25}$$

And the corresponding gradient is

$$\frac{\partial}{\partial\beta_j}L = -\sum_{i=1}^{n}(y_i - p_i)x_{ij} + 2\mu\beta_j \tag{26}$$

The best penalty term μ is selected from a finite candidate set. The set consists of exponentially growing sequence.

4 Design of experiments

4.1 Data

We use Gender Recognition dataset to train and test the model. The training dataset has 559 examples. 75% of which are used as training data, and the rest 25% are used as validation data.

4.2 Stochastic gradient ascent.

Choosing batch size Batch size influences computing time largely. Large batch size will increase computing time for gradient, but will make the gradient more precise, thus decreasing β update times to some degree. We fix the pair of hyper-parameter (λ_0, μ) and training/validation dataset, and change batch size from the set $\{1, 2, 4, ..., 256\}$. By computing error rate on validation set and computing time, we choose the best batch size. Experiment is repeated for 5 times, because different randomization of data may cause different result.

Choosing hyperparameters: Grid search. The candidate set of λ_0 is $\lambda_0 = 10^k, k \in \{-5, -4, .., 0\}$. The candidate set of μ is $\mu = 5^l, l \in \{-7, -6, .., 1\}$.

The training data is randomized before grid search, therefore the result of grid search may be different every time. We repeat the grid search process for 5 times and obtain 5 pairs of best (λ_0, μ). We finally choose the pair of (λ_0, μ) from the 5 candidates.

[2] http://scipy.org

Checking convergence. To prove that the model converges at the end of the training process, we train 5 models using the selected pair (λ_0, μ) and draw the curve of objective function v.s. epoch number. The value of objective function should increase at first and then to a certain point remains constant or goes up and down near the constant.

4.3 L-BFGS

Choosing parameter μ. The candidate set of μ is $\mu = 5^l, l \in \{-7, -6, .., 4\}$. We repeat the search for 5 times, and obtain 5 best μ. We finally choose the optimal μ from the 5 candidates.

Comparison of computing time. We compare the efficiency of L-BFGS optimization method and our SGA method.

Randomization of data is done once before training. The order of training data and validation data is the same for the two method. 5 experiments are done and the averages of the two methods are compared.

4.4 Testing model on test data

After deciding all parameters, we finally build the model and test the performance of the model on the test data.

For L-BFGS, one set of β is trained and we use it as the final model.

For SGA, we use a model averaging. The modified model is built from 5 models trained by SGA. The modified model is tested on the test data.

5 Results of experiments and analysis

5.1 Stochastic Gradient ascent

Batch size. To decide the batch size, hyperparameters are fixed to $\lambda_0 = 0.01, \mu = 0.1$. Experiments are repeated for 5 times. Mean values of error rate on validation set and computing time is calculated.

Figure 1 and Figure 2 are error bars of computing time and error rates with standard deviation. Figure 1 shows that computing time first drops to certain point and then rises with the increase of batch size. This is because when batch size is too large, calculation of gradient is time consuming. Figure 2 shows that error rate on validation set doesn't change significantly with batch size.

We choose batch size with low computing time and low standard deviation. Therefore, we choose $batchsize = 2$. Note that although computing time reaches minimum value when $batchsize = 2^5$, it is with the highest error rate. This probably means when $batchsize = 2^5$, the model doesn't converge to a good point in some examples and cannot be used to calculate computing time. Therefore we don't choose 2^5 as batch size.

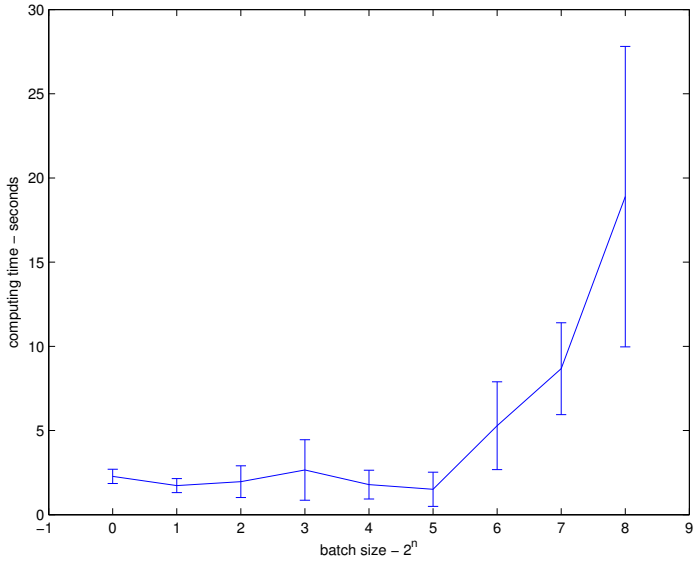

Fig. 1. Error bar: Mean computing time v.s. batch size

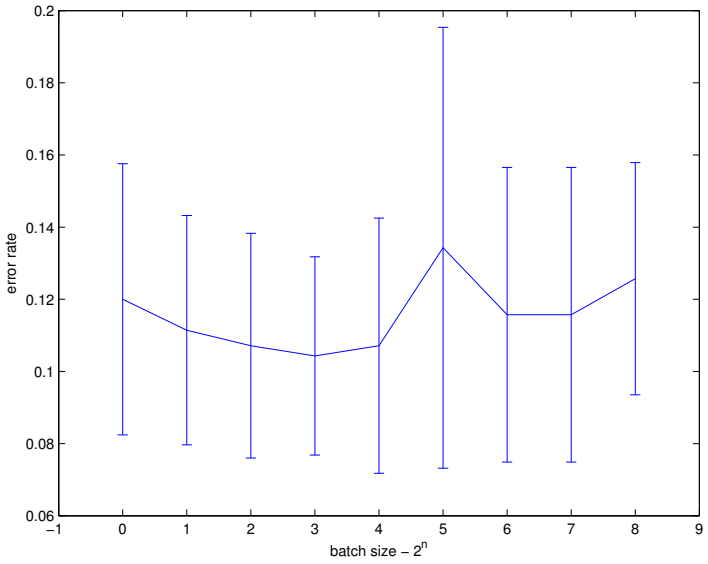

Fig. 2. Error bar: Mean error rate on validation set v.s. batch size

Grid search. For the candidate sets $\lambda_0 = 10^k, k \in \{-5, -4, .., 0\}$ and $\mu = 5^l, l \in \{-7, -6, .., 1\}$, we train models for each pair and repeat the whole process for 5 times. The average error rate for each pair of parameters is calculated.

For each experiment, the tendency of error rate's changes is very similar. Here, we only show the average error rate in Figure 3.

Best results on validation set are obtained when (λ_0, μ) are in a range of values. We choose $(\lambda_0 = 0.001, \mu = 0.04)$, since it is in the middle of the desired range. In this way, we expect to get more stable results with random training set.

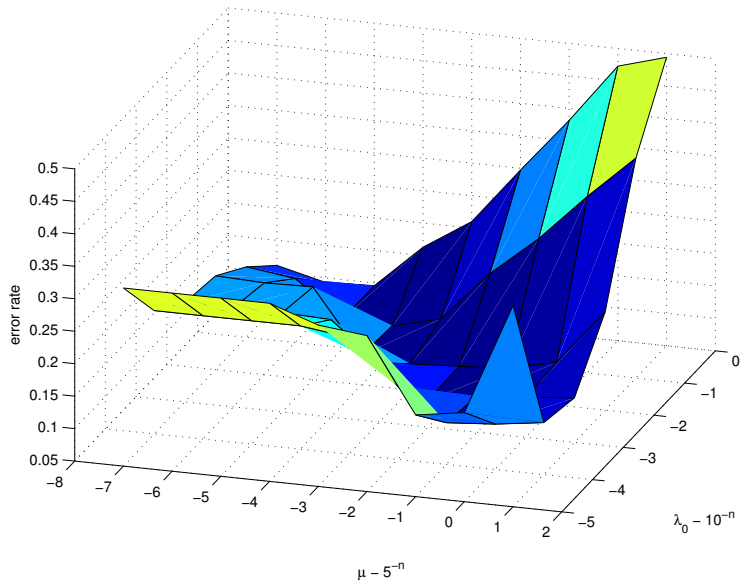

Fig. 3. Mean error rate on validation set v.s. hyperparameters (λ_0, μ)

Checking convergence. Objective function in Equation 14 is calculated at every epoch, and the curve (Figure 4) shows the trend of objective function during the training process. 5 experiments are done, and all of them are shown in Figure 4. As is shown, every curve starts from different value when $epochnumber = 1$, but they all converge to 0 when $epochnumber > 300$.

5.2 L-BFGS

Penalty term μ. Figure 5 shows the change of error rate on validation set with penalty term μ. As is shown, error rate doesn't change much when μ is smaller

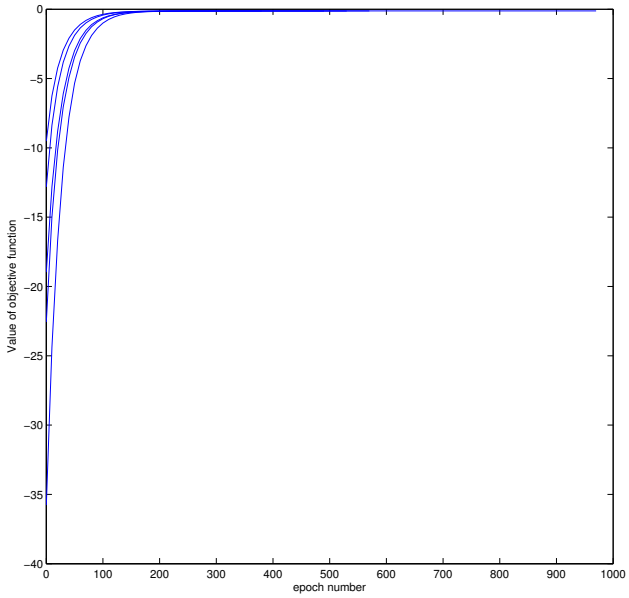

Fig. 4. Convergence of objective function (5 experiments)

than 1, and then goes up when μ becomes too large. The reason is that when μ is too large, the restriction it gives to the model is too strong that β trained is too small.

We choose $\mu = 5^{-2} = 0.04$, because it is in the middle of the desired region.

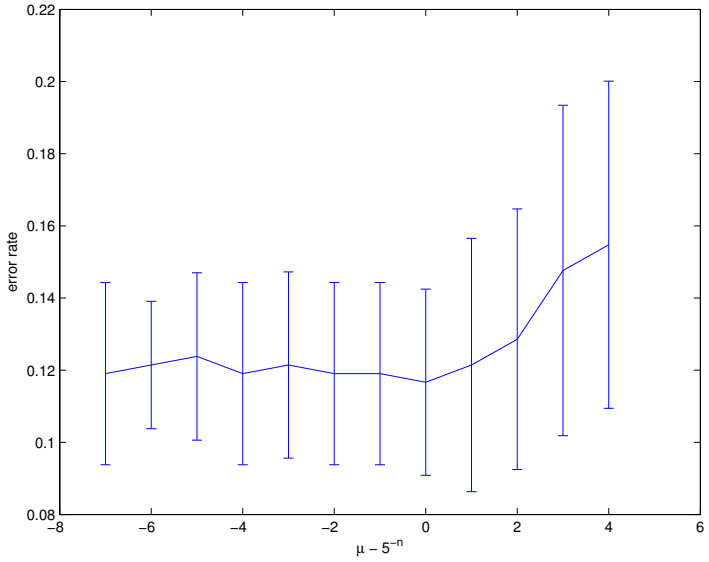

Fig. 5. Error bar: Mean error rate on validation set v.s. μ (L-BFGS)

Comparison of computing time Table 1 shows the computing time of the two optimization method: SGA and L-BFGS. The average computing time is calculated. Although much modification has been done to reduce the computing time of SGA, it is still far slower than L-BFGS.

In SGA, we use small λ_0 to get better convergence, but this means more epochs are needed. This is a trade off in precision and computing time.

Table 1. Computing time of SGA and L-BFGS (ms)

Experiment number	1	2	3	4	5	Average time
SGA	25706402	3491370	24138288	12908189	19416487	23252147
L-BFGS	60333	178217	59457	61117	62767	84378

5.3 Big-O time and space complexity of SGA

Big-O time The training with SGA consists of update within e epochs. In every epoch, $m = n/b$ batches are used, where n is the number of training examples, and b is batch size. For every batch, the parameter update requires $O(bd)$ operations, because feature matrix and parameter vector. For the entire SGA process, the time complexity is $O(e \times n \times d)$.

Space complexity. Training data and test data are stored in matrix. It requires $(N + n) \times (d + 2)$ units of space, where N stands for then number of training examples, and n is the number of test examples. The algorithm of normalization requires $(2 \times d)$ units of space to store mean and standard deviation. The algorithm of randomization requires $N \times (d + 2)$ units to store the temporary shuffled data. The calculation of gradient requires $(d+1)$ units to store gradient. The classification process requires n units to store the classification results.

Therefore, the space complexity is $O((N + n) \times d)$.

5.4 Result on test data

With all the hyperparameters defined, the model is built.

For L-BFGS, $\mu = 0.04$. The two models are built once, and the models are used on test data.

For SGA, $\lambda_0 = 0.001, \mu = 0.04, batchsize = 2$, early stopping is used, and validation subset is 25% of the training data. 5 models are trained, and the modified model is built upon them.

SGA method gets 7.112971% error rate on test data, and L-BFGS gets 8.786611% error rate on test data.

For the purpose of comparison, we provide error rate of 5 models and the average model on test data, as is shown in Table 2. Average model outperforms every single model.

Table 2. Error rate of different models on test data

	Model 1	Model 2	Model 3	Model 4	Model 5	Average model
Error rate	7.949791 %	10.041841 %	9.623431 %	7.531381 %	8.786611 %	7.112971%

6 Findings and lessons learned

In this project, we learned much about how to train logistic regression models, especially on how to use practical tricks to improve learning. Specifically, how to deal with overflow, how to overcome overfitting, how to choose hyper parameters, etc. It turns out that the learning procedure is quite sensitive to the learning rate and the regularization term. Early stopping can considerably reduce training

time. The choice of validation set may contain noise, which can be alleviated by model averaging. Batch-based learning may be even faster than SGA, which is a trade off between gradient accuracy and computation complexity.

References

1. Elkan, C.: Maximum likelihood, logistic regression, and stochastic gradient training. (January 17, 2013)