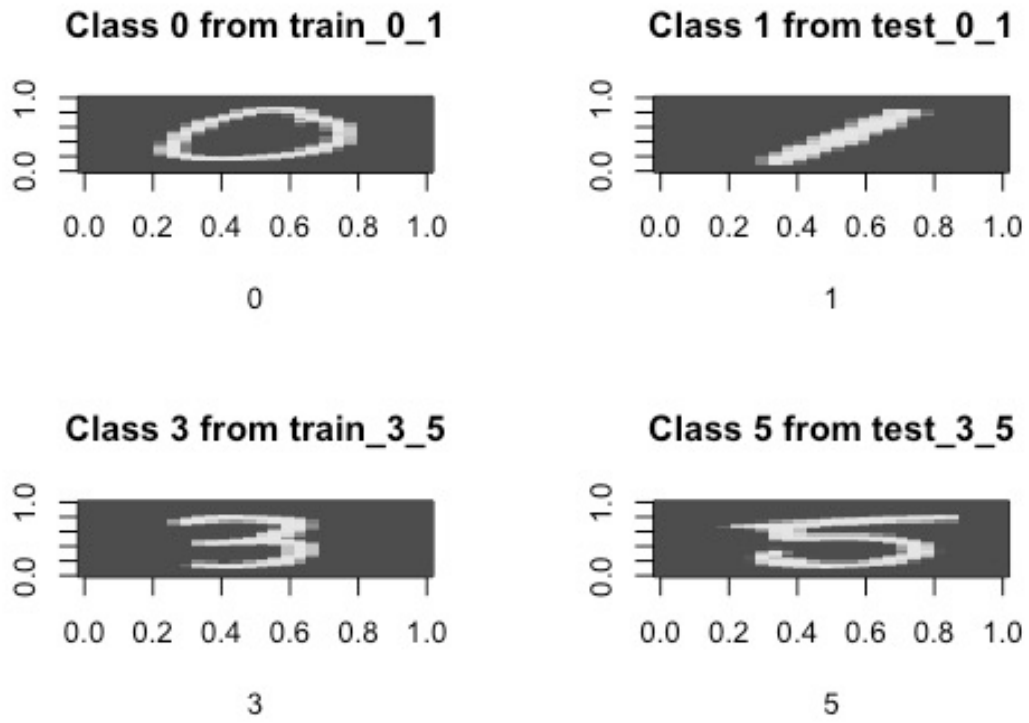
Logistic Regression

0. Data Preprocessing

The picture below shows 4 sampled images for 4 classes. True labels are under the x-axis.



1. Theory

1a. Write down the formula for computing the gradient of the loss function used in Logistic Regression. Specify what each variable represents in the equation.

The logistic function I use here is the same as lecture video:

$p(Y = y|X = x, \theta) = \frac{1}{1+\exp(y\theta^T x)}$

, where y = -1 or +1.

When $\theta^T x$ <= 0, $p(Y = 1|X = x, \theta)$ >= 0.5, then we classify y=1.
And when $\theta^T x$ > 0, $p(Y = 1|X = x, \theta)$<0.5, then we classify y=-1.

Based on max likelihood:

$\hat{\theta}_{mle} = argmax_\theta \sum_{i=1}^{n} \log \frac{1}{1+\exp(y^i \theta^T x^i)} = argmin_\theta \sum_{i=1}^{n} \log(1 + \exp(y^i \theta^T x^i))$

$$= argmin_\theta \ \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp\,(y^i\theta^T x^i))$$

So loss function here I used in logistic regression is:

$$\frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp\,(y^i\theta^T x^i))$$

Then gradient of the loss function for θ_j (j is in range of 1 to d+1) is:

$$\frac{\partial \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp\,(y^i\theta^T x^i))}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \log\left(1+\exp\left(y^i\theta^T x^i\right)\right)}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+\exp\left(y^i\theta^T x^i\right)} \frac{\partial\left(1+\exp\left(y^i\theta^T x^i\right)\right)}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+\exp\left(y^i\theta^T x^i\right)} \frac{\partial \exp\left(y^i\theta^T x^i\right)}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{\exp\left(y^i\theta^T x^i\right)}{1+\exp\left(y^i\theta^T x^i\right)} \frac{\partial\,(y^i\theta^T x^i)}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+\exp\left(-y^i\theta^T x^i\right)} \frac{\partial\,(y^i\theta^T x^i)}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{y^i x_j^i}{1+\exp\left(-y^i\theta^T x^i\right)}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{x_j^i}{y^i\left(1+\exp\left(-y^i\theta^T x^i\right)\right)} \quad \text{\#See note below}$$

#Note: the last step is just to make vectorized computation a bit faster. Since y^i is only -1 or 1, y^i at numerator or denominator is equivalent.

Here, for convenience, I add bias term as x_0=1 to make X as (d+1) by n matrix.
n is the sample size,
i is i^{th} sample from 1 to n,
j is j^{th} feature from 1 to d+1,
y^i is the i^{th} y value in the sample,

x^i is the i^{th} x vector, including
$x_0^i = 1\ to\ represent\ bias\ term, and\ x_1^i \dots \dots x_j^i \dots \dots x_d^i$.

So x_j^i *is* the j^{th} row and i^{th} column in X.

And θ is a (d+1) vector of parameters for defining the classifier.
θ_j is the j^{th} parameter in θ we want to get gradient for.

Then I use BATCH GRADIENT DESCENT method to update θ.

$$\theta_j^{k+1} = \theta_j^k - \alpha \frac{1}{n} \sum_{i=1}^{n} \frac{x_j^i}{y^i(1 + \exp(-y^i \theta^T x^i))}$$

Here, k is the number of iteration, α is step size.

1b. Write pseudocode for training a model using Logistic Regression.

set iteration_limit
set step_size α
set convergence threshold epsilon
initialize d+1 number of θ to random values
transform y labels to -1 and 1
append a row of x_0=1 to x to make x as d+1 by n matrix

start iteration from k=0
repeat{
k=k+1

for i from 1 to n:

innerproduct$_i$ = 0

for j from 1 to d+1:

innerproduct$_i$ = innerproduct$_i$ + $\theta_j * x_{ij}$

denominator$_i$ = $y_i * (1 + \exp(-y_i * innerproduct_i))$

for j from 1 to d+1:

gradient$_j$ = 0

for i from 1 to n:

gradient$_j$ = gradient$_j$ + $\frac{x_{ij}}{denominator_i}$

$$\theta_j = \theta_j - \alpha * \text{gradient}_j/n$$

} until k > iteration_limit or convergence threshold is met

1c. Calculate the number of operations per gradient descent iteration.
(Hint: Use variable n for number of examples and d for dimensionality.)

According to the above pseudo codes, for each gradient descent iteration, we have two tandom two for-loops. In total the number of operations are:

$$n * \big(2 + (d+1)\big) + (d+1) * (2+n)$$
$$= 2n*d+4n+2d+2$$

So in Big O format, each gradient descent iteration, time complexity is O(n*d).

3. Training
Train and evaluate the code from question 2 on MNIST dataset.

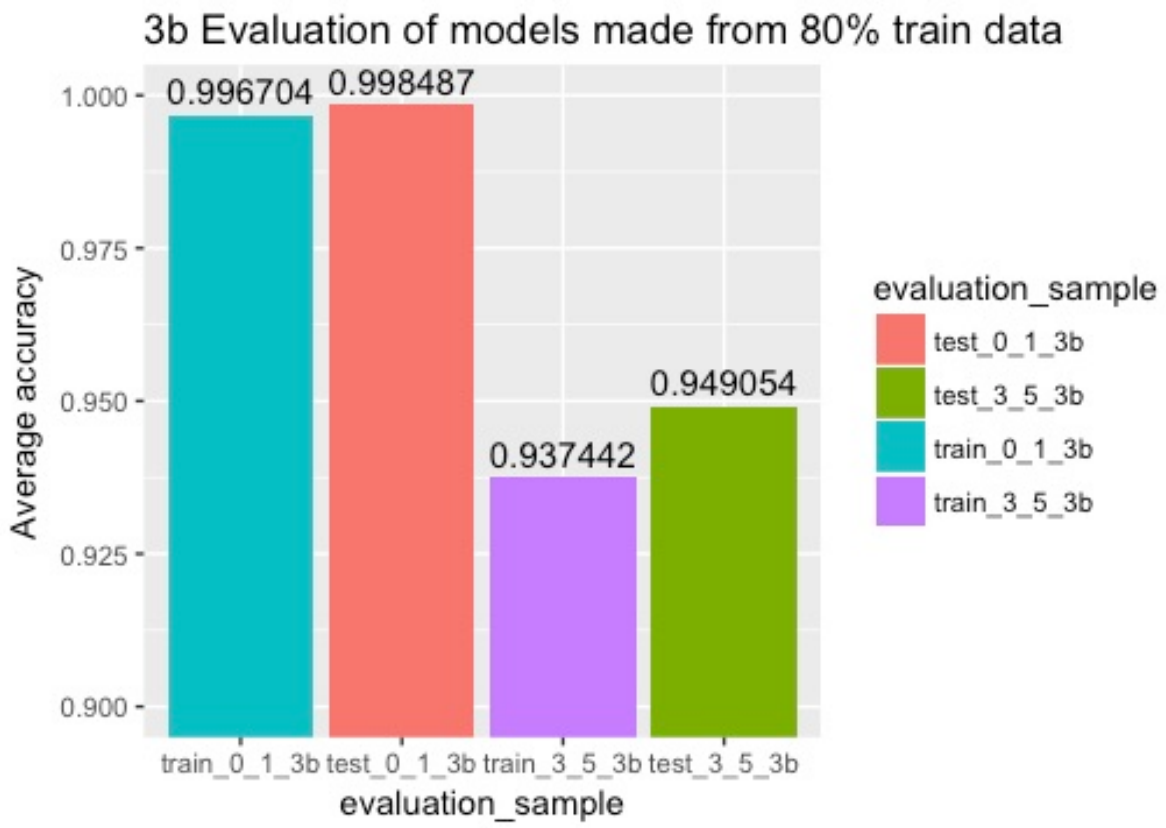
3a. Train 2 models, one on the train_0_1 set and another on train_3_5, and report the training and test accuracies.

Answer:
I made one model model_0_1 from train_0_1 with default settings, and evaluated the training accuracy is 0.996683774180813 and test accuracy is 0.998581560283688.

I made one model model_3_5 from train_3_5 with default settings, and evaluated the training accuracy is 0.9375, and test accuracy is 0.94794952681388.

3b: train on 10 random 80% divisions of training data and report average accuracies.
Here I generated 10 models from 10 random sampled 80% training data for classification of 0 and 1. Evaluated the models on the same training data and the whole test data, and calculated the mean accuracies for train and test.
The average train accuracy for 0 and 1 is 0.996704.
The average test accuracy for 0 and 1 is 0.998487.

Here I generated 10 models from 10 random sampled 80% training data for classification of 3 and 5. Evaluated the models on the same training data and the whole test data, and calculated the mean accuracies for both train and test.
The average train accuracy for 3 and 5 is 0.937442.
The average test accuracy for 3 and 5 is 0.949054.
Detail can be seen in the following plot.



3c. For 0,1 and 3,5 cases, explain if you observe any difference you in accuracy. Also, explain why do you think this difference might be.

We can see in the above plot, overall accuracies for both class 0,1 and 3,5 are higher than 90%, which means these are good logistic regression models.
But accuracies for class 0,1 are higher than 3,5 on both train and test. I think this could be due to the similarity of writings of 3 and 5 is higher than the similarity of writings of 0 and 1. The bottom half part of 3 and 5 are very similar. And top half part of 5 could be miswritten to 3.

I also noticed for both 0,1 and 3,5, test accuracies are higher than train accuracies. This is unusual. In most cases, test accuracy would be lower than train. This expected case could be due to that train and test data were not by random partition.

3d. This assignment deals with binary classification. Explain what you would do if you had more than two classes to classify, using logistic regression.

Let's say we have M classes in total. We randomly select one class as reference class. Train models for binary classification of reference class and one of the other classes. So we get M-1 models. From each model, we can calculate log odds of P(Y=class_i)

and P(Y=class_reference). This is $log\frac{P(Y=class_i)}{P(Y=class_{reference})} = \theta_i^T \% * \% x$. This means we can calculate the ratio of two probabilities. Since we use the same one class as reference, from M-1 models, we should easily tell which class has highest probability when given one sample of X features. And we also know sum of all probabilities equals to 1. So based on this, we also can calculate each probability from M-1 models when given one sample of X features. We just choose the class with biggest probability as the prediction.

4. Evaluation

In this question, you will experiment with different sets of parameters and observe how your model performs. This should be done ONLY for 3,5 classification.

a. Experiment with different initializations of the parameter used for gradient descent .Clearly mention the initial values of the parameter tried, run the same experiment as 3b using this initialization, report the average test and train accuracies obtained by using this initialization, mention which is set of initializations is the better.

Answer:
My default initial theta value used in 3b is 0, here I try another value 0.5. In case different initial thetas may require different initial learning rate alpha for better performance, I tried initial alphas in a pool of c(0.3, 1,3,10,30,100,300) for both initial thetas. So by combination of initial theta and alpha, we have 2*7 =14 cases of initial settings.
For each case, I run 10 times on different 80% train data of class 3,5. And calculated the average accuracy. Detail can see following plot.

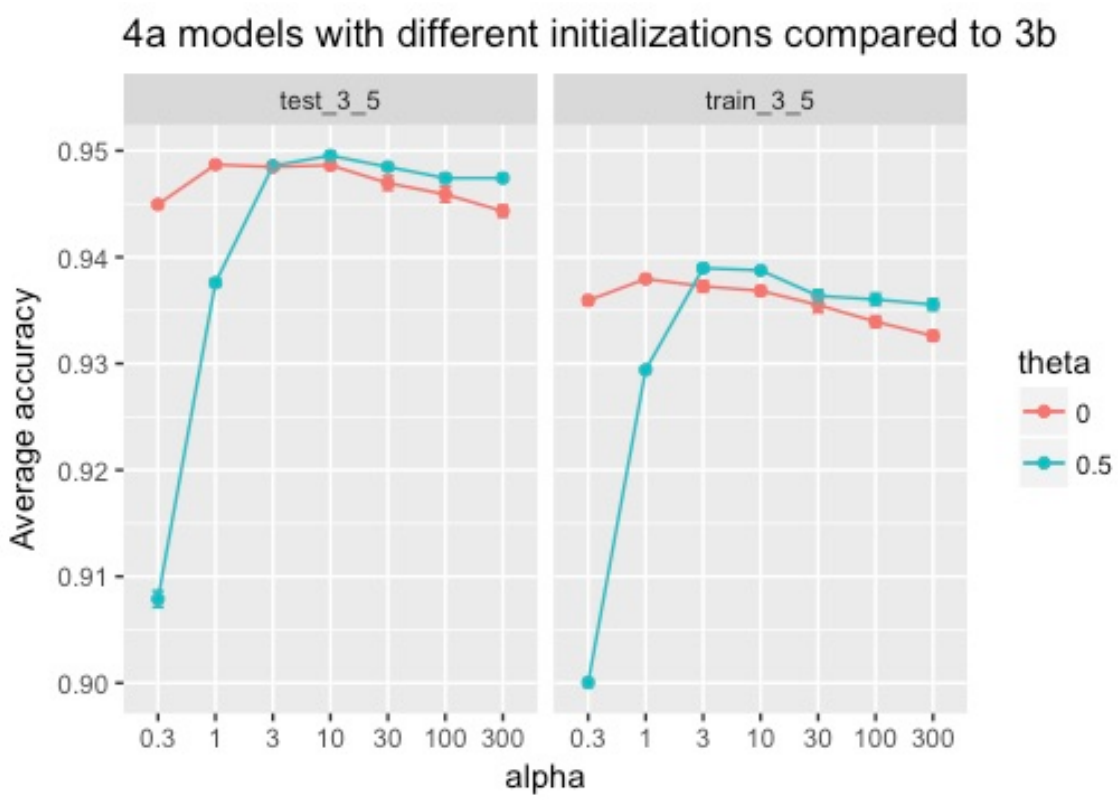
With initial theta 0, average train accuracy has highest value at 0.9379504 with initial alpha = 1.
With initial theta 0, average test accuracy has highest value at 0.9486856 with initial alpha = 1.

With initial theta 0.5, average train accuracy has highest value at 0.9389676 with initial alpha = 3.
With initial theta 0.5, average test accuracy has highest value at 0.9495268 with initial alpha = 10.

Notice that, the best alpha for theta 0 is 1, these are the settings used in 3b, and also the best accuracy values are similar to the results of 3b.

We see models have a little higher accuracies with intial theta 0.5 and big alphas than initial theta 0. But overall performace of initial theta 0 is more reliable, since in the plot, with setting of initial theta 0.5, models are more sensitive to alphas than initial theta 0.

So I still think we'd better use the default setting of initial theta 0 and initial alpha 1 like experiment in 3b. And also it is common sense to set null hypothesis as the weight of any X feature equals to 0.



4b. Experiment with different convergence criteria for gradient descent. Clearly mention the new criteria tried, run the same experiment as 3b using this new criteria, report average test and train accuracies obtained using this criteria, mention which set of criteria is better.

The default converge_criteria "F", is to calculate the difference of Frobenius norm of old and new gradient, and compare it with epsilon. It means that if gradient change is too small, we think it converges.
Here in 4b, I tried another converge_criteria "L". Since the goal of gradient descent is to find global minimum of loss function. We may think if the change in loss value after one iteration is too small, it converges. So in each iteration, we record the value of loss function by using equation:

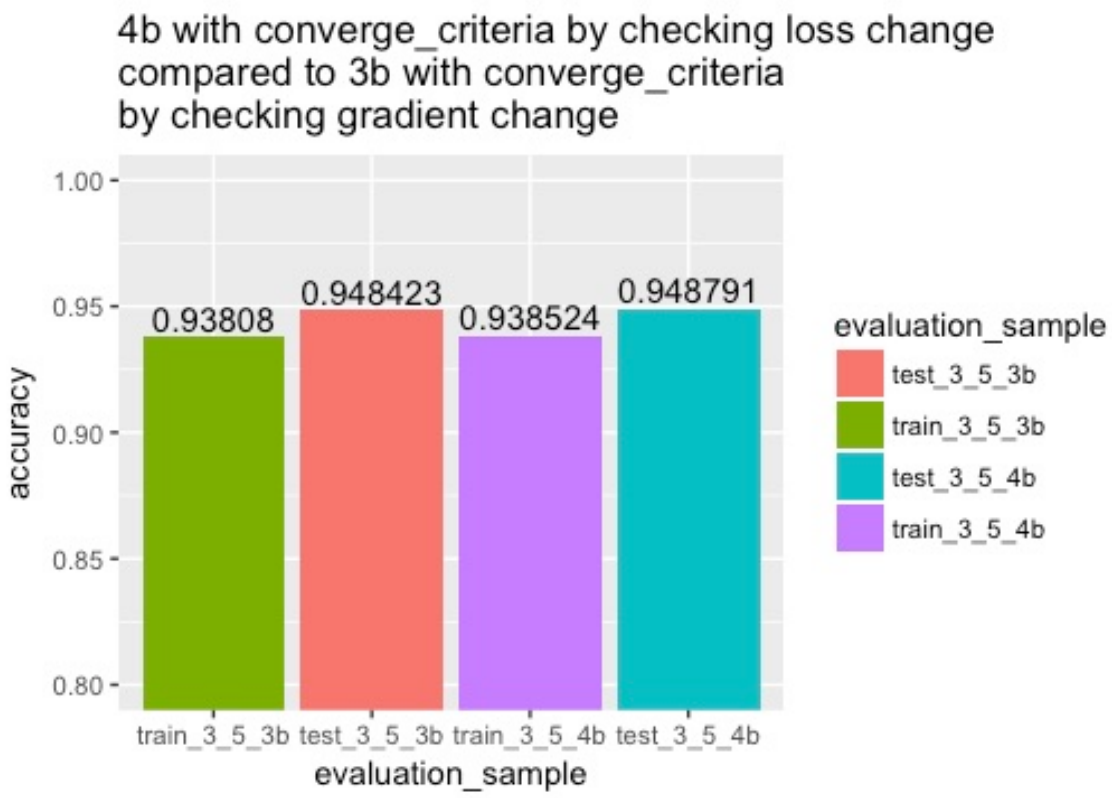
$$\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(y^i\theta^T x^i))$$

We calculate the absolute change of loss values after one iteration, and compare it to epsilon. If the difference is less than epsilon, stop iteration.

By using this criteria, I rerun experiment as 3b. The result is shown in the plot below.

The average train accuracy under new criteria is 0.938524, and the average test accuracy is 0.948791. They are both slightly higher than the accuracies from the default criteria in 3b.

Although new criteria is slightly better, it requires calculating loss value in each iteration which is more time consuming, and actually the performances of two criterias are more or less similar. So I insist on the default converge_criteria.



5. Learning Curves

In this question, you will experiment with different training set sizes and see how the accuracy changes with them. This question should be done for both 0,1 and 3,5 classification.

5a. For each set of classes (0,1 and 3,5), choose the following sizes to train on: 5%, 10%,15% … 100% (i.e. 20 training set sizes). For each training set size, sample that many inputs from the respective complete training set (i.e. train_0_1 or train_3_5). Train your model on each subset selected, test it on the corresponding test set (i.e. test_0_1 or test_3_5), and graph the training and test set accuracy over each split. Comment on the trends of accuracy values you observe for each set.

In this experiment, I chose twenty train sizes just as the assignment described, from 5% to 100%. For each train size, run 10 models from random sampling and get average train and test accuracies for both class 0,1 and class 3,5. The results were ploted in the following two figures.

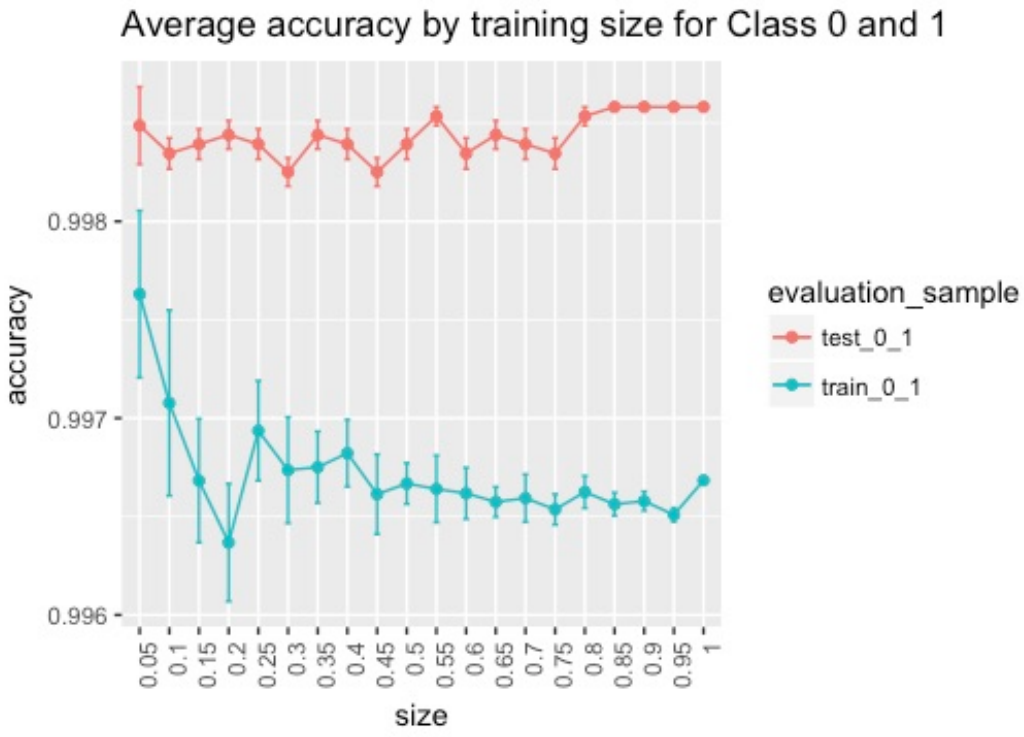
For class 0,1, the train accuracies have trend to decrease from small train size to big train size and steadily about 0.9967 in the end. The test accuracies are not sensitive to train size, and they are all about 0.9985, which is higher than all average train accuracies.

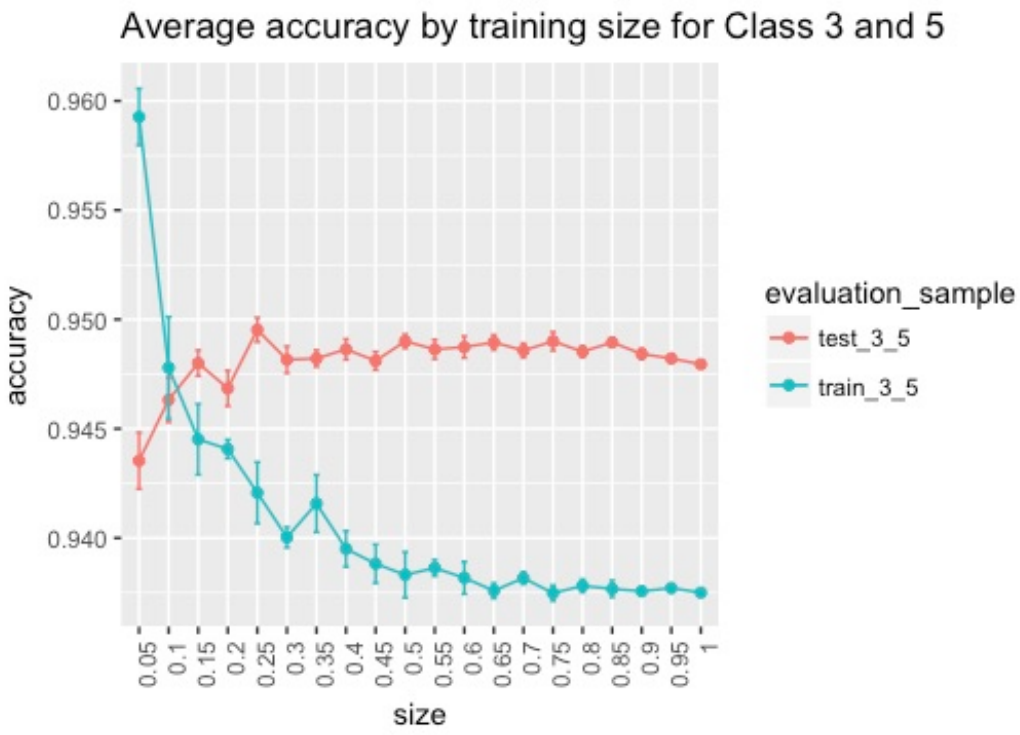
For class 3,5, the train accuracies also have trend to decrease from small train size to big train size and steadily about 0.935 in the end. The test accuracies have trend to increase from small train size to big train size, and steadily about 0.9475 in the end.

So in classification of 3,5, it has better idea that with smaller train size, model may have overfitting issue, that it has a higher train accuracy but lower test accuracy than big train size.

And I also noticed that the error bars which are starndard error of 10 accuracies in each setting, have the trend to get smaller when train size increases. This means the performance is more stable given a bigger train size.

The conclusion is that in order to have overall better performance, we'd better choose big train size.





5b. Repeat 5a, but instead of plotting accuracies, plot the logistic loss/negative log likelihood when training and testing, for each size. Comment on the trends of loss values you observe for each set.

In this experiment, I chose twenty train sizes just like 5a, from 5% to 100%. For each train size, run 10 models from the same random samples as 5a, and get sum negative log likelihood for both class 0,1 and class 3,5. The sum negative log likelihood is:

$$\sum_{i=1}^{n} \log(1 + \exp\left(y^i \theta^T x^i\right))$$

The results were ploted in the following two figures.

From the plots, we see clearly, the sum negative log likelihood is linearly associated with train size for both class 0,1 and class 3,5. And the test size is not changed in each experiment, so the sum negative log likelihood for test is at about same value when train size changes.

This is expected, since with more samples, the maximum log likelihood tends to get smaller. And in the plot the value of sum negative log likelihood is the globle minimum of sum negative log likelihood we can get after gradient descent. This

value is bigger, meaning that the maximum log likelihood is smaller. That's why we see sum negative log likelihood increases as train size increases.

I also noticed the overall sum log likelihood values for class 0,1 are smaller than class 3,5. I think this can explain why we get higher accuracies for class 0,1 than class 3,5 from the models.