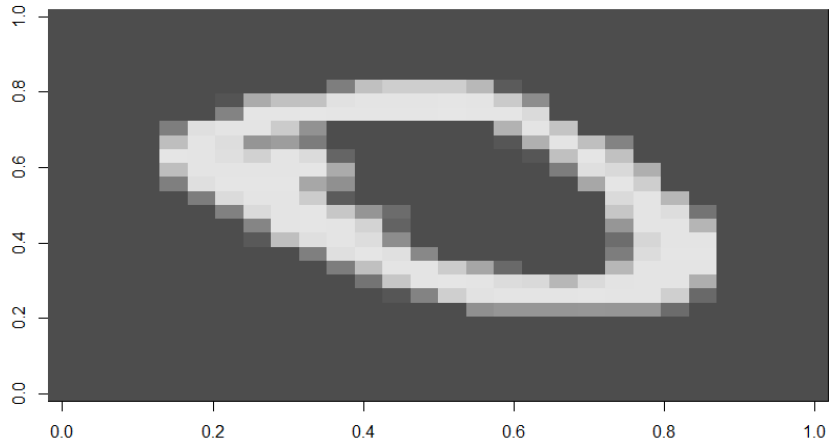
HW3: Logistic Regression

GT account name: xtao41

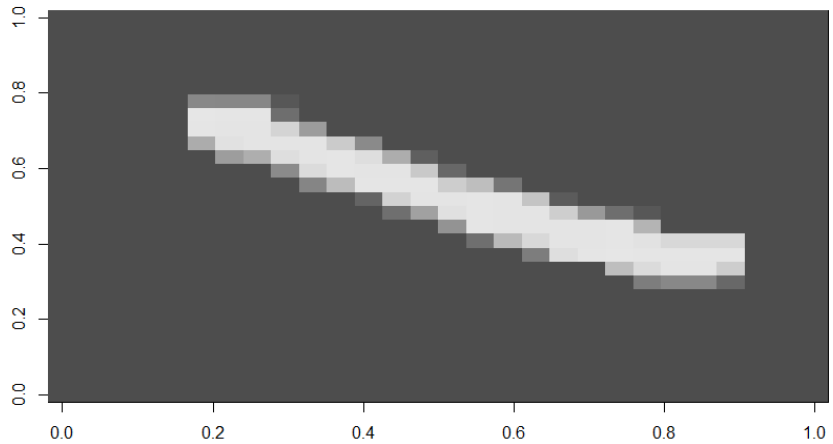
0. Data Preprocessing

   4 sample images:

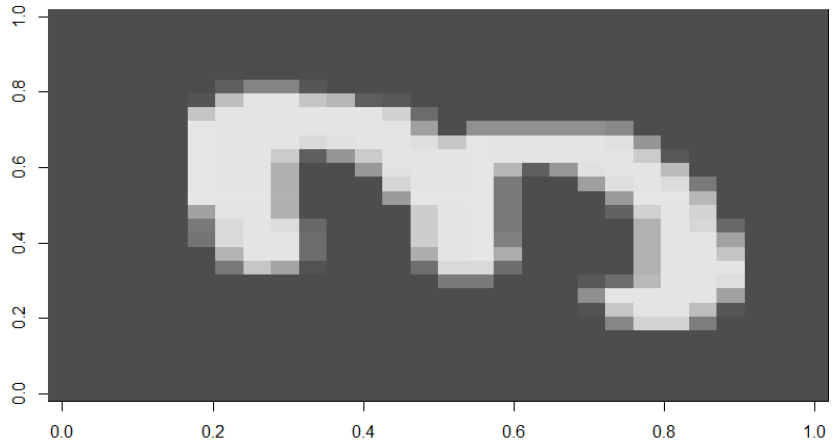
   1) Class label 0 example



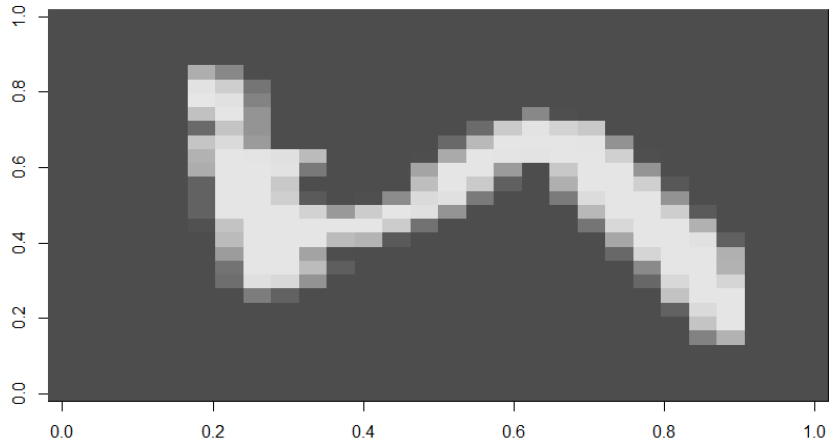
   2) Class label 1 example



   3) Class label 3 example



4) Class label 5 example



1. Theory
   1) Formula for computing the gradient of the loss function

      As per the Udacity lecture and piazza post 946 and 1037, the loss function is the negative of the log likelihood of the training set.

      $$\log(1 + \exp(y^{(i)}\langle\theta, x^{(i)}\rangle))$$

      And we want to minimize it (equal to maximize the log likelihood).

      $$\hat{\theta}_{MLE} = \arg\min_{\theta} \sum_{i=1}^{n} \log(1 + \exp(y^{(i)}\langle\theta, x^{(i)}\rangle))$$

      In order to compute the gradient of it, we need to take the partial derivative of the above function,

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

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

$$= \sum_{i=1}^{n} \{\left(\exp\left(y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right) \frac{\partial}{\partial \theta^j}\left(y^{(i)}\langle\theta, x_j{}^{(i)}\rangle\right) / \left(1 + \exp\left(y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right)\}$$

$$= \sum_{i=1}^{n} \{\left(\exp\left(y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right)\left(y^{(i)} x_j{}^{(i)}\right) / \left(1 + \exp\left(y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right)\}$$

$$= \sum_{i=1}^{n} \{\left(y^{(i)} x_j{}^{(i)}\right) / \left(1 + \exp\left(-y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right)\}$$

Thus, by updating θ_j according to above rule, we get new θ^j is the old θ_j minus α (which is the step size, i.e. learning rate) times the partial derivative of the loss function,

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{n} \{\left(y^{(i)} x_j{}^{(i)}\right) / \left(1 + \exp\left(-y^{(i)}\langle\theta, x^{(i)}\rangle\right)\right)\}$$

In the above formula,

$\hat{\theta}_{MLE}$ is the maximum likelihood Estimator (MLE)

$x^{(i)}$ is measurement vector

$y^{(i)}$ is the label- the vector we are trying to predict

θ is a vector that describes the parameter vector or weights vector, with θ_j for each dimension.

α is the step size, which is some number and may correspond to the learning rate.

2) pseudocode for training a model using Logistic Regression:

```
Initialize the vector θ to random values with dimension d
Set the learning rate α
Set a threshold value
While (update >= threshold) {
    for j in (1, d)     # d is number of dimensions
        for i in (1, n)         # n is number of examples
        {
                Compute and update per the formula
                θ_j = θ_j − α Σ_{i=1}^{n} {(y^(i) x_j^(i)) / (1 + exp(−y^(i) ⟨θ, x^(i)⟩))}
        }
```

}

3) Number of operations per gradient descent iteration.

For each gradient descent iteration, it loops through n examples and d dimensions. So it takes nd operations. If we use vectorized algorithm, the time complexity of O(nd).

2. Implementation (see code in R file)
3. Training
   a) Train and test accuracies of 0/1 and 3/5 classification

Accuracy	0/1 class	3/5 class
Train	0.993288590604027	0.94494459833795
Test	0.997163120567376	0.949001051524711

   b) Used BATCH GRADIENT DESCENT. Train on 10 random 80% divisions of training data and take average.

Average Accuracy	0/1 class	3/5 class
Train	0.991206079747335	0.939558537113179
Test	0.995508274231679	0.948107255520505

   c) 0_1 classification has a much higher accuracy then the 3_5 classification (~0.05). Because the predicting method for 0_1 and 3_5 sets are the same, thus the reason for the difference is more likely to related to the datasets themselves. Maybe for the 3_5 dataset, it is more difficult for the linear hyperplane to divide the positive and negative areas (i.e. the points are more intermingled). That's why we see the classifier failed to predict a portion of the in-sample tests. The fix could be transfer (X1,X2,…) to higher dimensions, which can solve non-linear decision boundary problems.

   d) If I have multi-class classification problems, I can use "one vs. rest" method to make binary classification work for multiclass problems. Each time, chose one class against all the other classes to turn it into multiple separate binary classification problem. Specifically,

      For i in range (1: # of classes)

         Step1. make class i Y=i, all other class Y=-1, train a classifier H^i(Xtrain) to predict the probability that Y=i

         Step2. For a given test data Xtest, pick the class i that maximize the probability that H^i(Xtest)=i

4. Evaluation (only on 3_5 classification for 3b)
   a) Experiment with different initializations of theta. Report the average accuracy

	Baseline(random small positive numbers [0,1])	random large negative numbers [-1,0]	random larger positive numbers [5,10]	random small negative numbers [-10,5]	All 0s
Train	0.9396	0.9338	0.9043	0.9115	0.9297
Test	0.9481	0.9438	0.9117	0.9242	0.9403

With initialization of theta changed to relatively large positive numbers (or small negative numbers), the accuracy decreases. Also with very large theta, it is very difficult to converge, unless changing the learning rate. So the conclusion is it is better to stick with small positive or large negative numbers for theta initialization;

When I set the initial theta to all zeros, the accuracy is slightly smaller than random numbers (in [-1,0] or [0,1] range). Thus, I believe randomization is a little bit better than fixing all the numbers in theta initialization.

b) Experiment with different convergence criteria

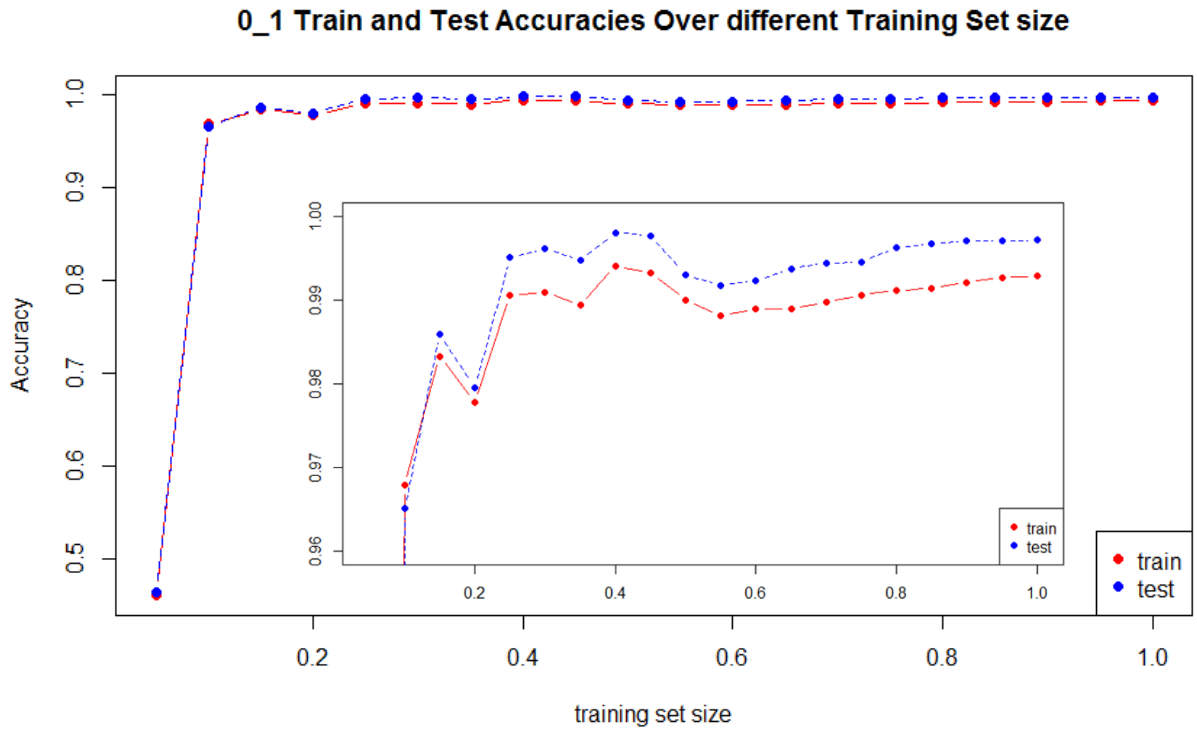
	Threshold (Baseline=0.01)	Change Threshold To 0.1	Change Threshold To 0.001	Using iteration=10 As criteria	Using iteration=100 As criteria
Train	0.9396	0.7990	0.9683	0.9108	0.9550
Test	0.9481	0.8006	0.9595	0.9242	0.9572

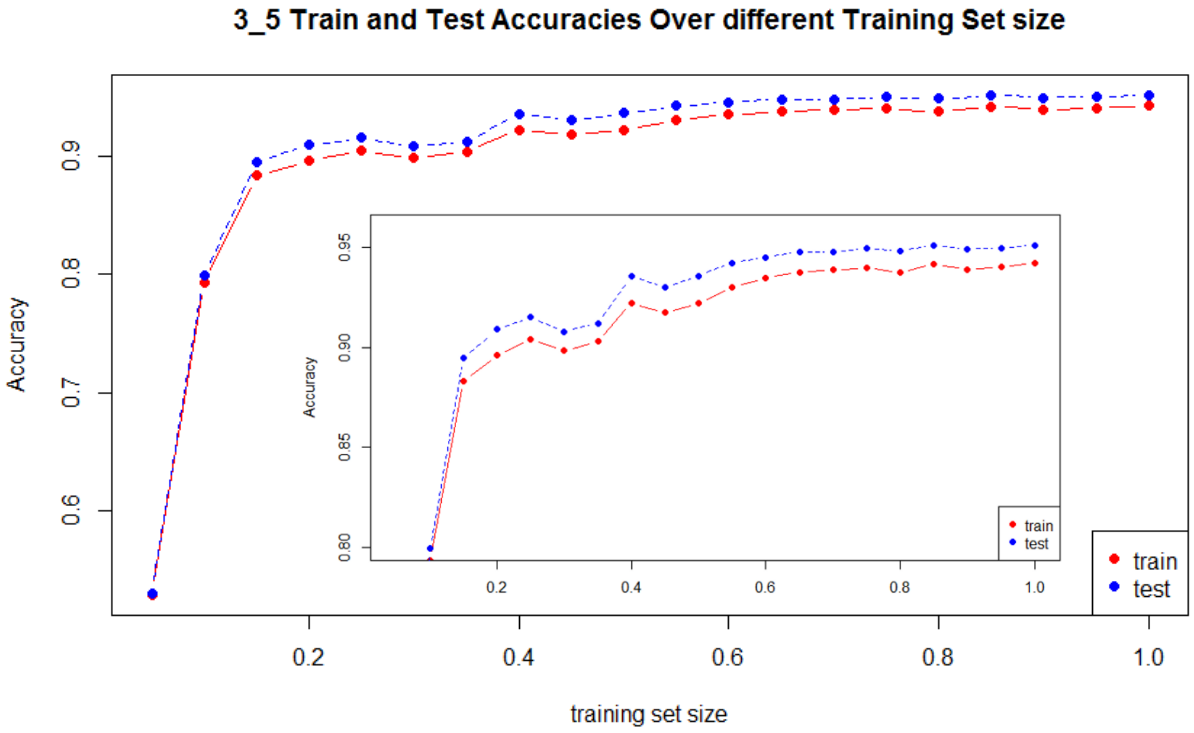
The current implementation of logistic regression is to stop the iteration when all the gradient values in every dimension are less than a threshold value. The baseline of this value is set to inverse of square root of number of samples (i.e. ~0.01). By when the threshold value increases (such as increase to 0.1), the accuracy decrease drastically (from 0.94 to 0.80). On the contrary, decreasing the threshold to 0.001 improves the accuracy, but the computational time also increases a lot. For example, when the threshold value decreases to 0.001, the train accuracy increases by ~0.03 and test accuracy increases by ~0.01. It looks like train accuracy increases at a higher rate than test accuracy. Therefore, though smaller threshold values yield better accuracy, we would also need to balance with computational efficiency.

The other method I tried is to fix the iteration time. As iteration count increases, the accuracy increases, but the computational time also increase. It also looks like the accuracy for train data increases at a higher rate than test data. Therefore, with an acceptable computational time, higher iteration the better. Here iteration =100 provides a better criterion than iteration=10.

5. Learning Curve

a. Accuracy: In general, when the size of training set increases, training accuracy increases. This is clearer for 0_3 dataset than 0_1 set, with the latter shows more fluctuations from 30% to 50% of training data. For both 0_1 and 0_3 training set, 5% of the data yields very low (<60% accuracy), but increase sharply until 15% training set size. Afterwards, the rate of increment decreases or becomes steady. This is expected because with more training data, the prediction becomes more accurate and finally the accuracy will be converaged to a certain value.



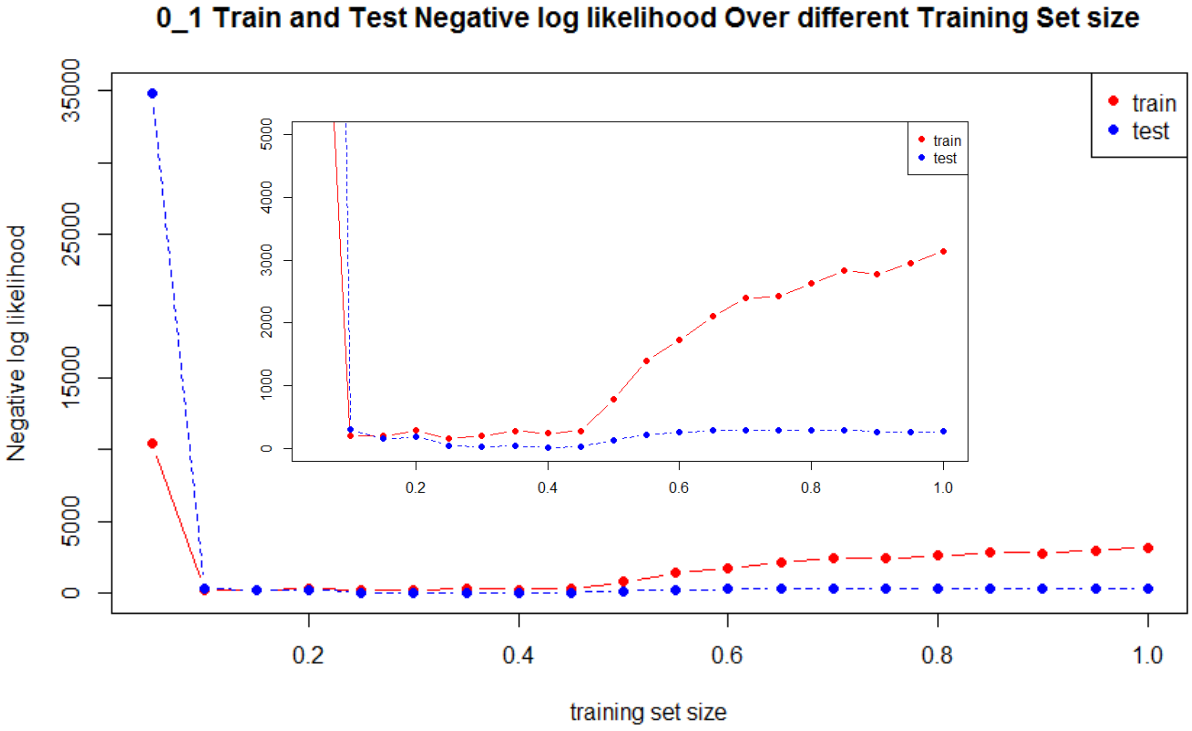


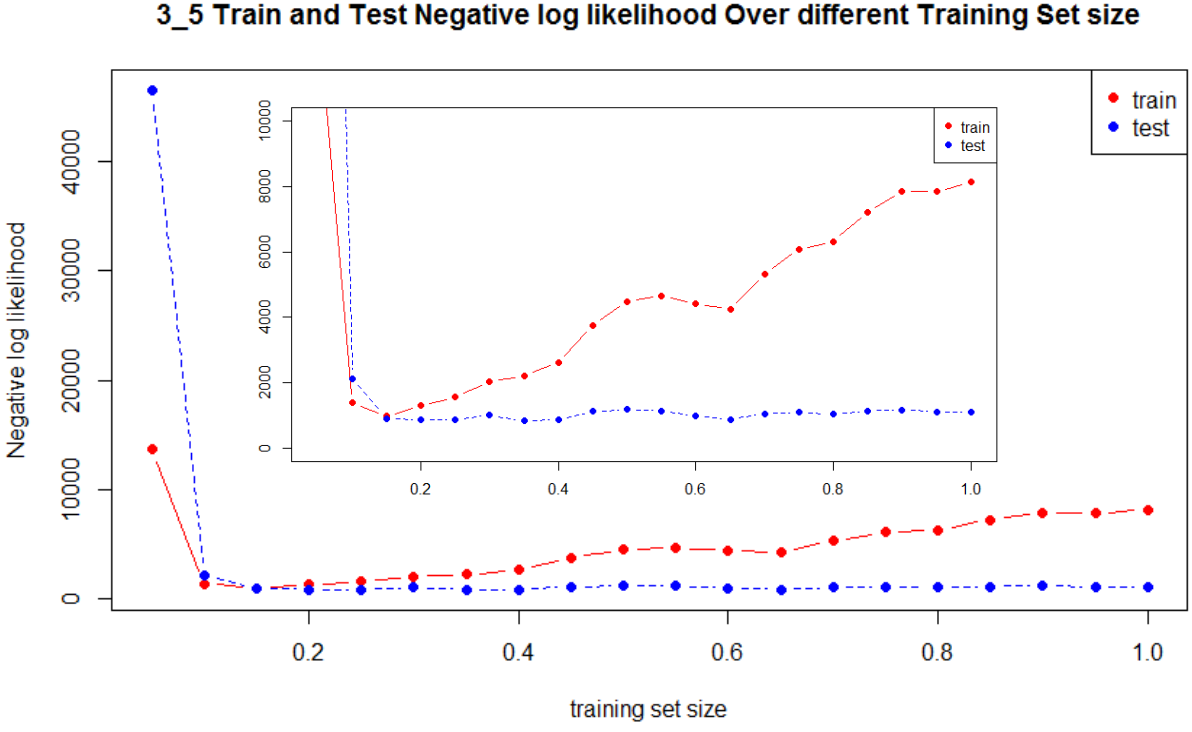
b. Trends of train and test negative log-likelihood over different training set sizes
Negative log-likelihood:

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

Theta is theta value that is optimized using Logistic Regression.

From the negative log-likelihood vs training set size plot shown below, we can see that as training set size increases, the negative log-likelihood of training set increases, while that of the testing set doesn't change much or even decreases slightly, which could because of more training data leading to more accurate prediction (smaller loss function values).
In addition, for the first few partition (5% to 15%) the likelihood dropped from very high values to reasonable range, corresponding to very low accuracies from small size of training data.


0_1 Train and Test Negative log likelihood Over different Training Set size
Negative log likelihood
training set size
train
test
0
5000
15000
25000
35000
0.2
0.4
0.6
0.8
1.0
1000
2000
3000
4000
5000


3_5 Train and Test Negative log likelihood Over different Training Set size
Negative log likelihood
training set size
train
test
0
10000
20000
30000
40000
0.2
0.4
0.6
0.8
1.0
2000
4000
6000
8000
10000