**Testing Functions:**

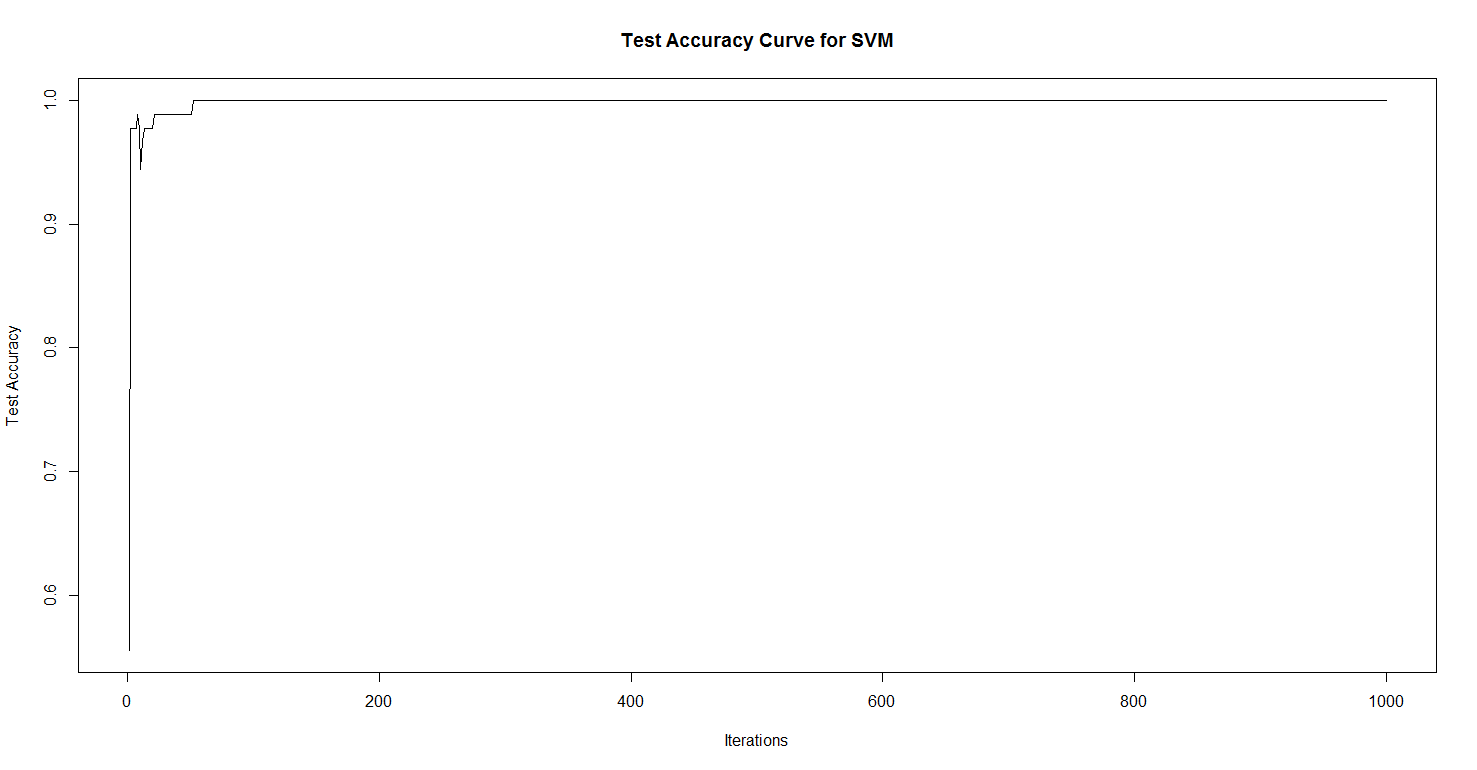
testNN <- function(){  
 final\_test\_acc = rep(0,1000)  
 final\_train\_acc = rep(0,1000)  
 total\_acc = 0.0  
 for(i in 1:10)  
 {  
 nn=my\_NN(X\_train, Y\_train, X\_test, Y\_test)  
 test\_acc = nn$acc\_test  
 train\_acc = nn$acc\_train  
 final\_test\_acc = final\_test\_acc + test\_acc  
 final\_train\_acc = final\_train\_acc + train\_acc  
 total\_acc = total\_acc+test\_acc[length(test\_acc)]  
 }  
 total\_acc = total\_acc/10  
 print(total\_acc)  
 final\_test\_acc = final\_test\_acc/10  
 final\_train\_acc = final\_train\_acc/10  
 plot(final\_test\_acc,type='l',xlab = "Iterations", ylab = "Test Accuracy", main = "Test Accuracy Curve for Neural Network")  
 plot(final\_train\_acc,type='l',xlab = "Iterations", ylab = "Train Accuracy", main = "Train Accuracy Curve for Neural Network")  
}

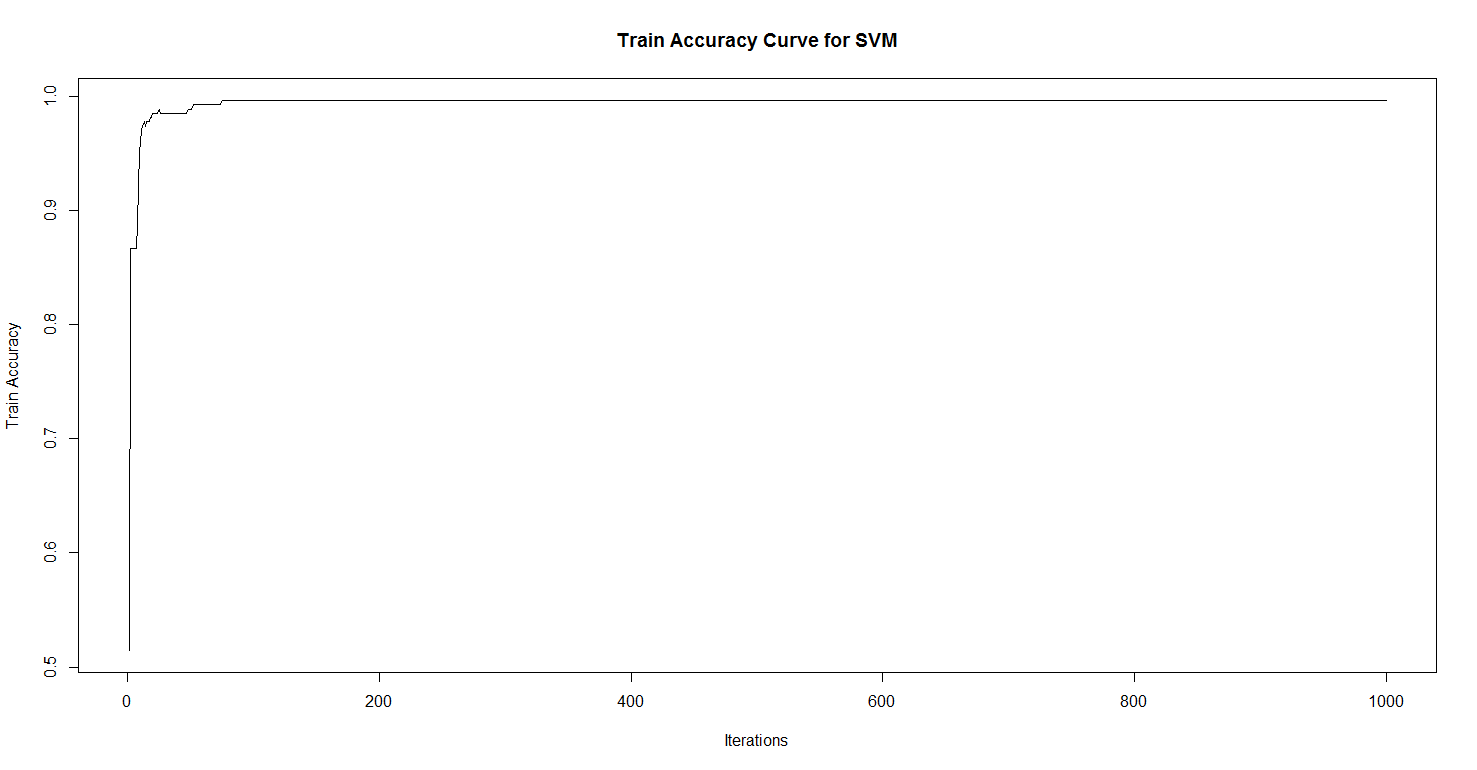
testSVM <- function(){  
 final\_test\_acc = rep(0,1000)  
 final\_train\_acc = rep(0,1000)  
 total\_acc = 0.0  
 for(i in 1:50)  
 {  
 svm=my\_SVM(X\_train, Y\_train, X\_test, Y\_test)  
 test\_acc = svm$acc\_test  
 train\_acc = svm$acc\_train  
 final\_test\_acc = final\_test\_acc + test\_acc  
 final\_train\_acc = final\_train\_acc + train\_acc  
 total\_acc = total\_acc + test\_acc[length(test\_acc)]  
 }  
 total\_acc = total\_acc/50  
 final\_test\_acc = final\_test\_acc/50  
 final\_train\_acc = final\_train\_acc/50  
 print(total\_acc)  
 plot(final\_test\_acc,type='l',xlab = "Iterations", ylab = "Test Accuracy", main = "Test Accuracy Curve for SVM")  
 plot(final\_train\_acc,type='l',xlab = "Iterations", ylab = "Train Accuracy", main = "Train Accuracy Curve for SVM")  
}

testADA <- function(){  
 final\_test\_acc = rep(0,1000)  
 final\_train\_acc = rep(0,1000)  
 total\_acc = 0.0  
 for(i in 1:50)  
 {  
 ada=myAdaboost(X\_train, Y\_train, X\_test, Y\_test)  
 test\_acc = ada$acc\_test  
 train\_acc = ada$acc\_train  
 final\_test\_acc = final\_test\_acc + test\_acc  
 final\_train\_acc = final\_train\_acc + train\_acc  
 total\_acc = total\_acc+train\_acc[length(test\_acc)]  
 }  
 total\_acc = total\_acc/50  
 print(total\_acc)  
 final\_test\_acc = final\_test\_acc/50  
 final\_train\_acc = final\_train\_acc/50  
 plot(final\_test\_acc,type='l',xlab = "Iterations", ylab = "Test Accuracy", main = "Test Accuracy Curve for ADABoost")  
 plot(final\_train\_acc,type='l',xlab = "Iterations", ylab = "Train Accuracy", main = "Train Accuracy Curve for ADABoost")  
}

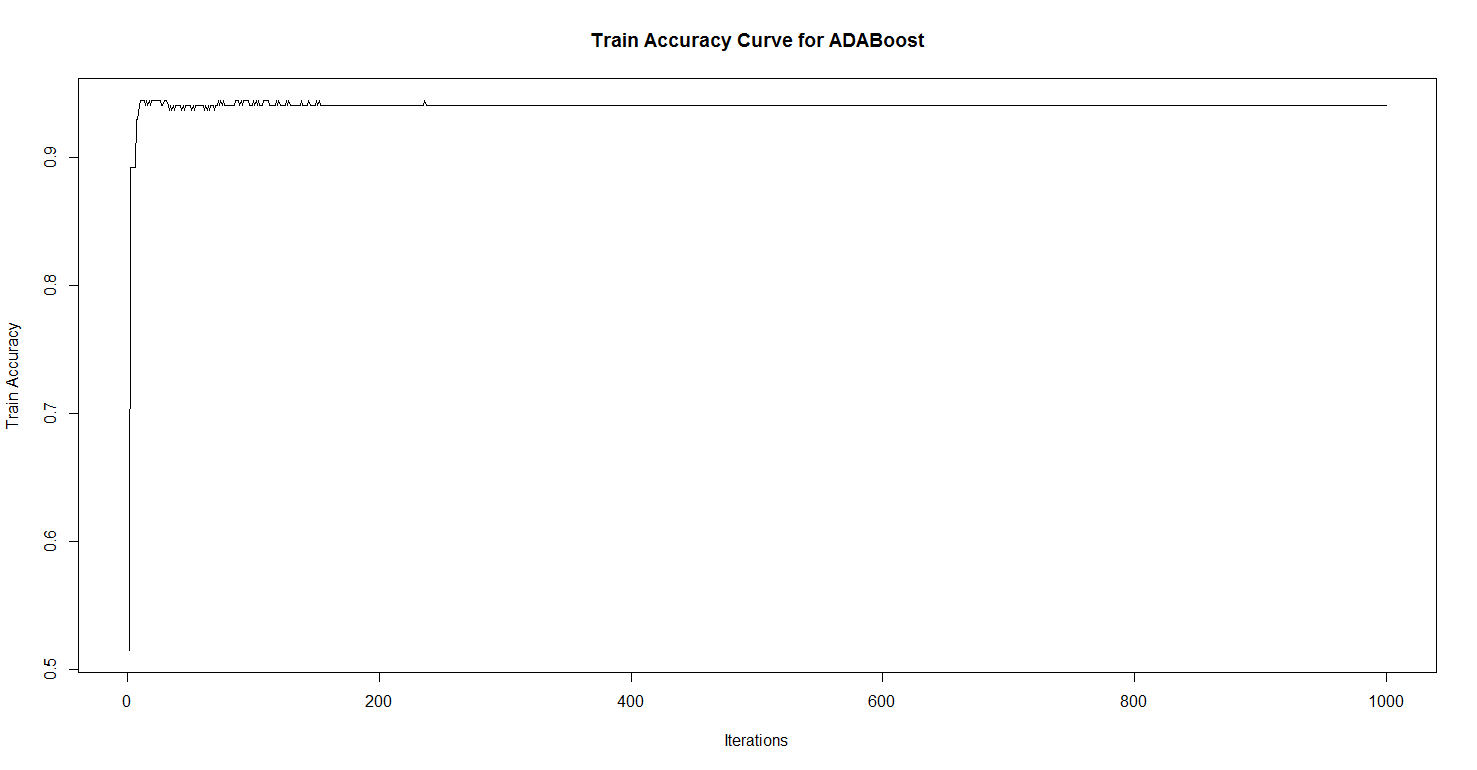
testLogis <- function(){  
 final\_test\_acc = rep(0,1000)  
 final\_train\_acc = rep(0,1000)  
 total\_acc = 0.0  
 for(i in 1:50)  
 {  
 logis=myLogistic(X\_train, Y\_train, X\_test, Y\_test)  
 test\_acc = logis$acc\_test  
 train\_acc = logis$acc\_train  
 final\_test\_acc = final\_test\_acc + test\_acc  
 final\_train\_acc = final\_train\_acc + train\_acc  
 total\_acc = total\_acc+test\_acc[length(test\_acc)]  
 }  
 total\_acc = total\_acc/50  
 print(total\_acc)  
 final\_test\_acc = final\_test\_acc/50  
 final\_train\_acc = final\_train\_acc/50  
 plot(final\_test\_acc,type='l',xlab = "Iterations", ylab = "Test Accuracy", main = "Test Accuracy Curve for Logistic Regression")  
 plot(final\_train\_acc,type='l',xlab = "Iterations", ylab = "Train Accuracy", main = "Train Accuracy Curve for Logistic Regression")  
}

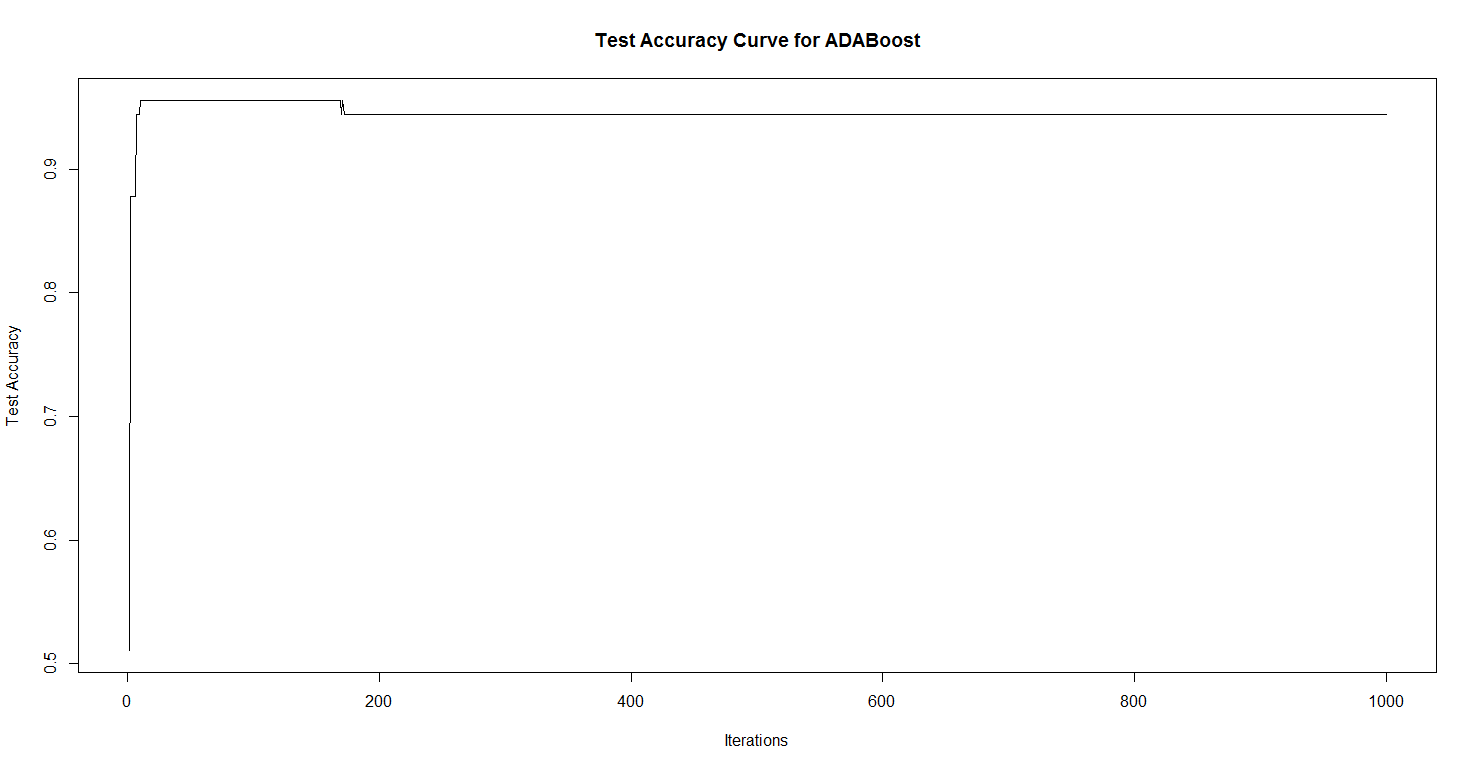
**SVM**(train: 0.992963, test: 0.988889)  
Iterations: 200



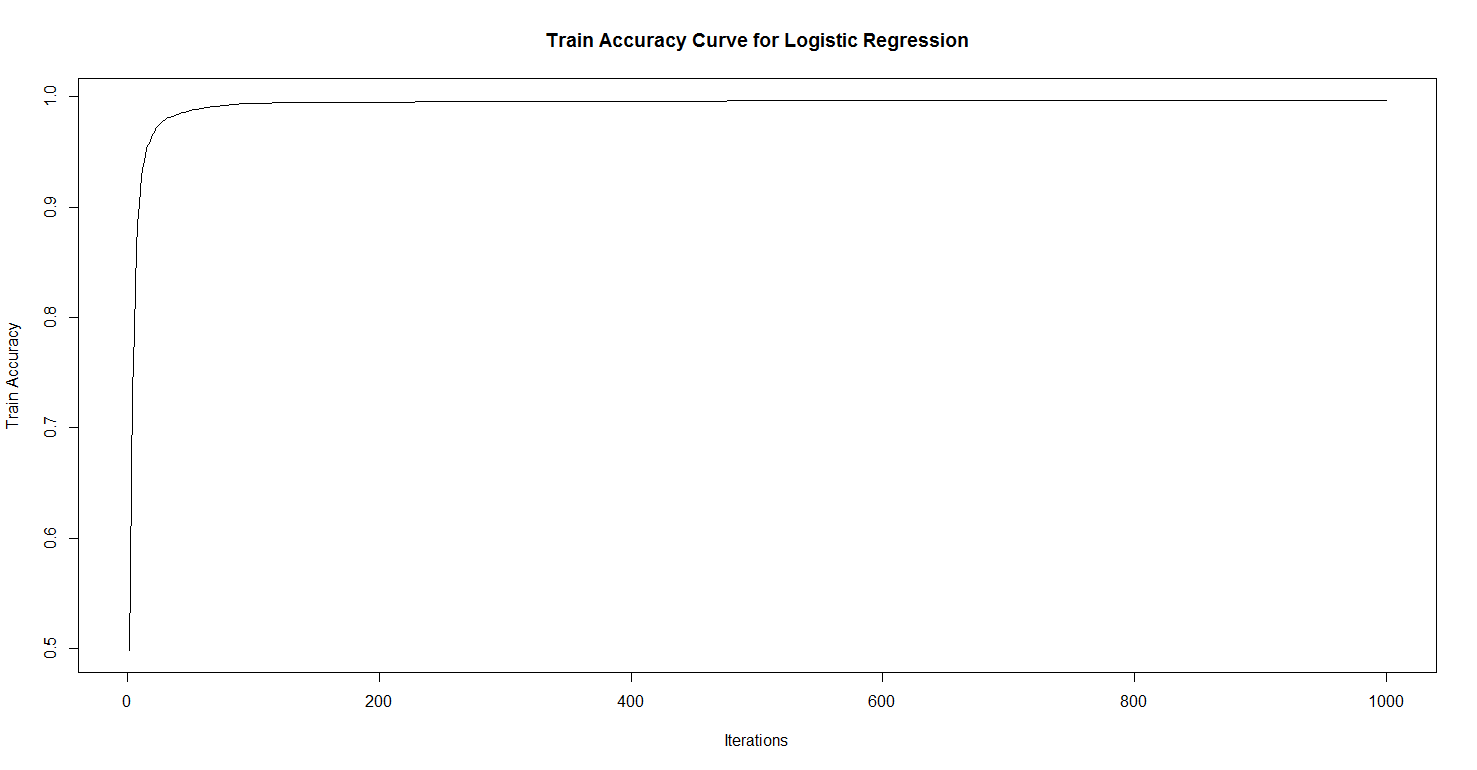


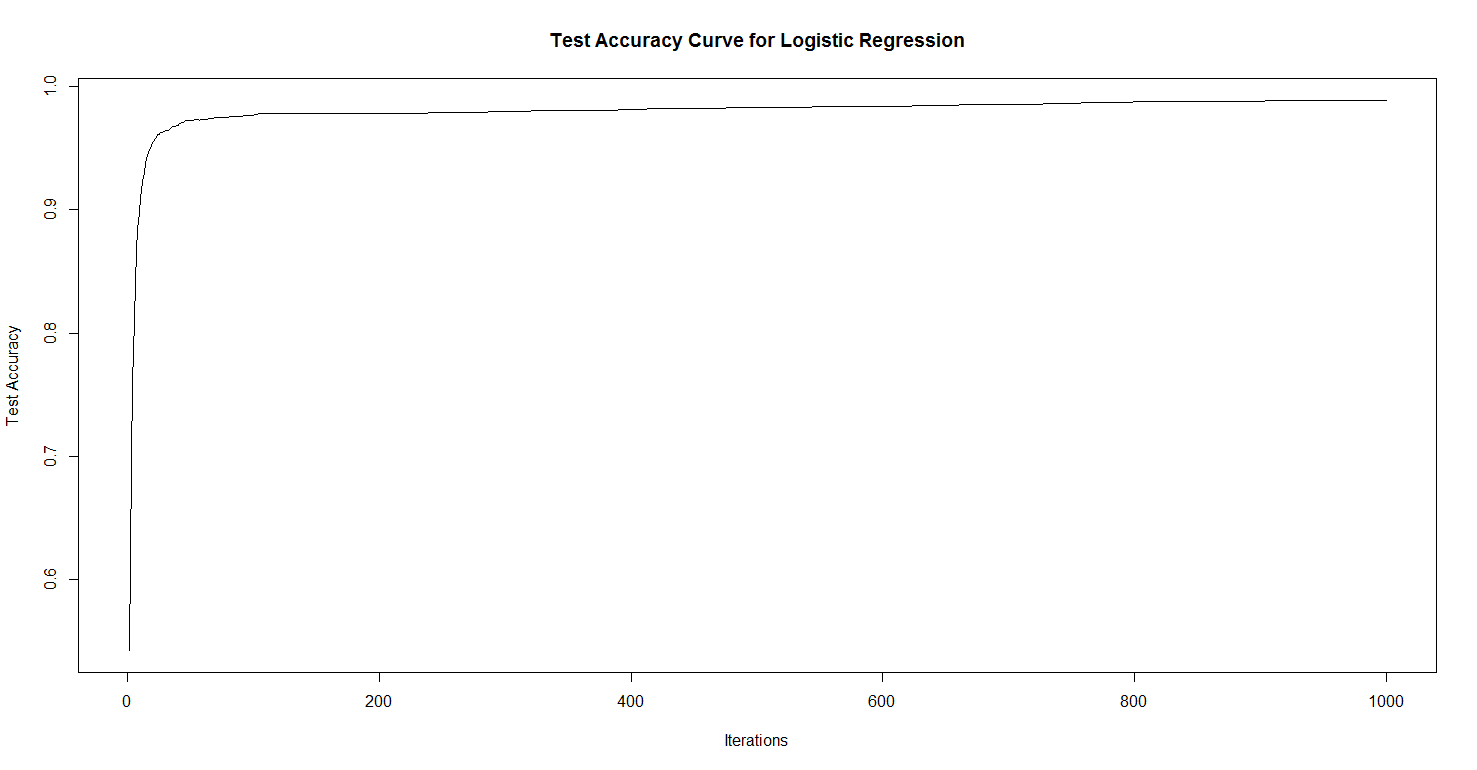
**ADABOOST**(train: 0.9666667, test: 0.9518519)  
Iterations: 50



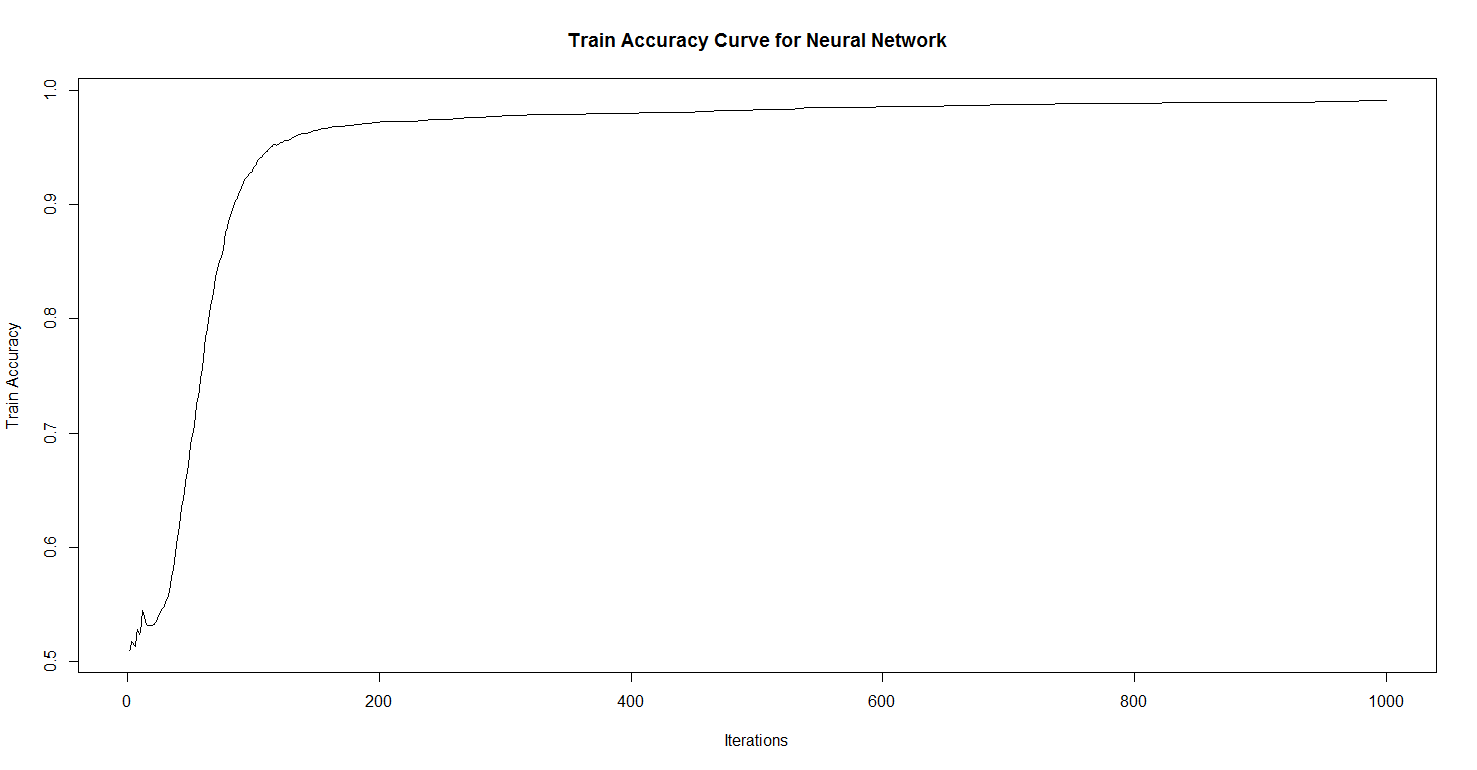


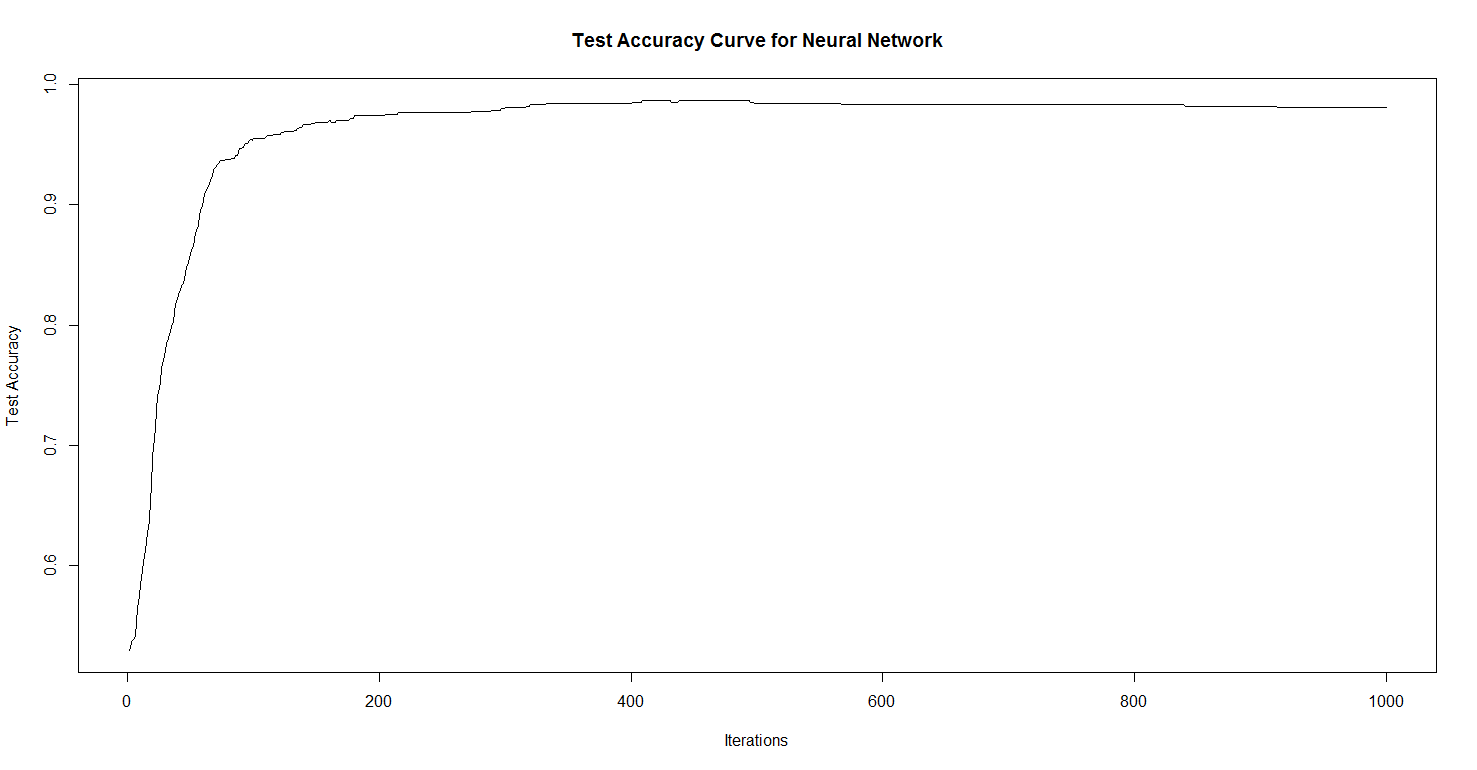
**LOGISTIC REGRESSION**  
(train: 0.9962963, test: 0.9886667)  
Iterations: 50





**NEURAL NETWORK**(train: 0.991111, test: 0.981111)  
Iterations: 20





**Neural Networks:**

An artificial neural network is an interconnected group of nodes arranged in layers. Each node consists of a decision function, usually a sigmoid function. The first layer of an ANN is the input layer, and the last is the output layer. All the layers in the middle make up the hidden layers.

* As the graphs show, ANNs take longer to converge than other models.
* Train time is also very high.
* Artificial neural networks often converge on the local minima, while SVMs do not.
* Also, SVMs do not overfit the data as ANNs often tend to do.
* Not parametric.

**SVM:**

A support vector machine is an algorithm that uses the closest points in different classes as “support vectors” to plot an optimal hyperplane separating the classes.

* The time to convergence is very fast for SVMs.
* SVMs take the least time to train.
* SVMs are more resilient to noise than ANNs.
* Logistic regression performs much better than SVMs in noisy datasets.
* The regularization parameter in SVM avoids overfitting.

**AdaBoost:**

AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm that can be used in conjunction with many other types of learning algorithms to improve their performance.

* The testing accuracy is also quite lower than the training accuracy as compared to others.
* ADABoost works well when there is a large class imbalance as well, and should be preferred here.
* Not parametric.
* Reaches effective solution quickly, but later becomes sensitive to noise.
* Works well with sparse datasets, can augment weak classifiers well.

**Logistic Regression:**

Logistic regression is a regression model where the dependent variable is categorical, and can be trained while the features are expected to be more or less linear.

* The training speed of logistic regression is quite fast.
* It is parametric.
* Unlike the others, it does not automatically learn feature interactions.

**Accuracies of the 4 models**(Per-model iterations were equalized to 1000 for all)

**GIBBS**

Testing Code:

testGibbs <- function(){  
 sim = simulate\_data()  
 X\_gibb = sim$X  
 Y\_gibb = sim$Y  
 beta\_true = sim$beta\_true  
 beta\_out = myGibbs(X\_gibb, Y\_gibb, 10000)  
 errors = beta\_true-beta\_out  
 percentage\_errors = errors/beta\_true\*100  
 print(errors[1:10])  
 print(beta\_out[1:10])  
 print(beta\_true[1:10])  
 print(percentage\_errors[1:10])  
 avg\_percent\_error = sum(abs(percentage\_errors[1:10]))/10  
 print(avg\_percent\_error)  
}

For Gibbs, the percentage errors between beta\_true and beta\_out were (for the first 10 values):

[1] -0.43973644 2.09849595 1.92793609 3.26062930 -1.40516222

[6] 7.73942443 0.08897213 -1.09227031 -0.17340333 -9.08847722

The absolute average percentage accuracy is:

[1] 2.731451