Anurag Pande: 604749647

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")
}
```

Anurag Pande: 604749647

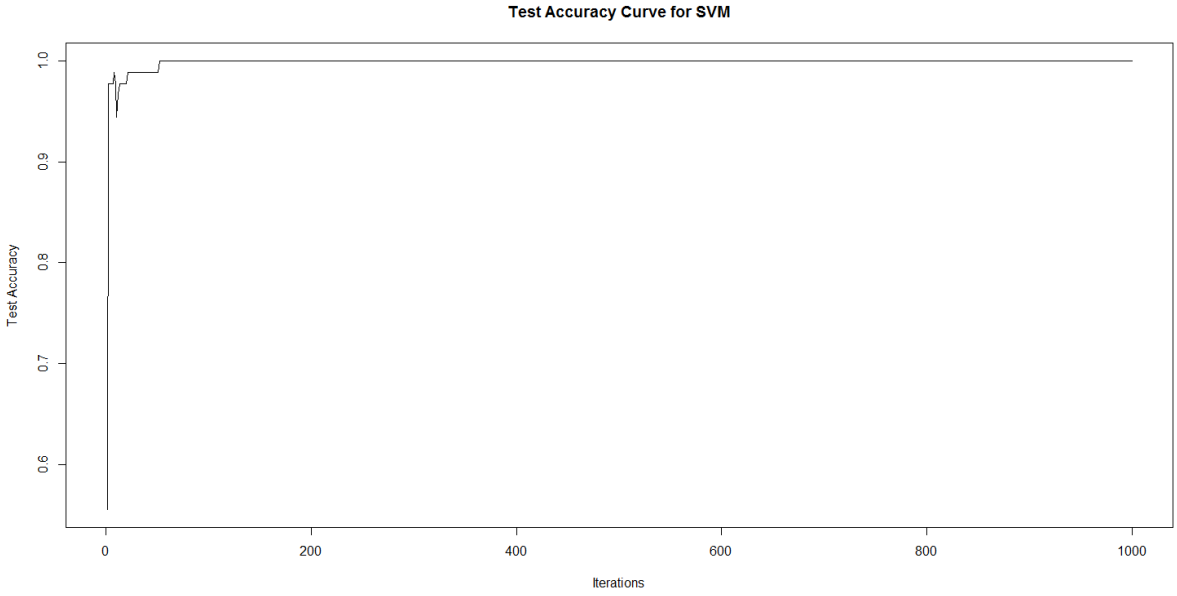
```
testADA <- function(){
  final_test_acc = rep(0,1000)
  final_train_acc = rep(0,1000)
  total_acc = 0.0
  for(i in 1:50)
  {
    ada=myAdaboost(X_train, Y_train, X_test, Y_test)
    test_acc = ada$acc_test
    train_acc = ada$acc_train
    final_test_acc = final_test_acc + test_acc
    final_train_acc = 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")
}
```

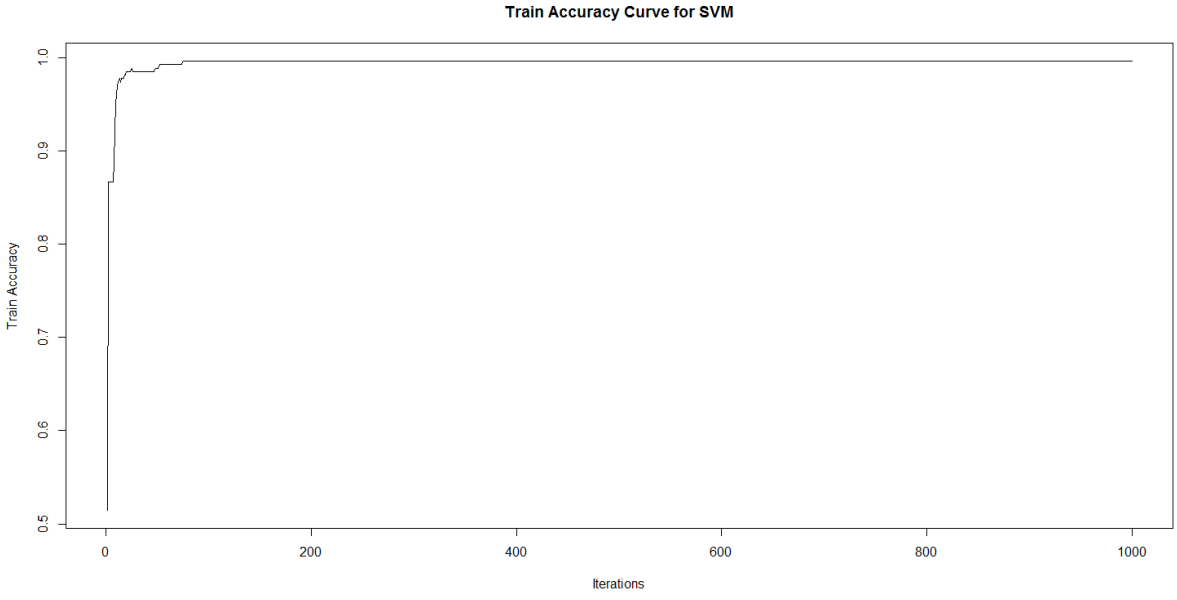
Anurag Pande: 604749647

SVM

(train: 0.992963, test: 0.988889)

Iterations: 200



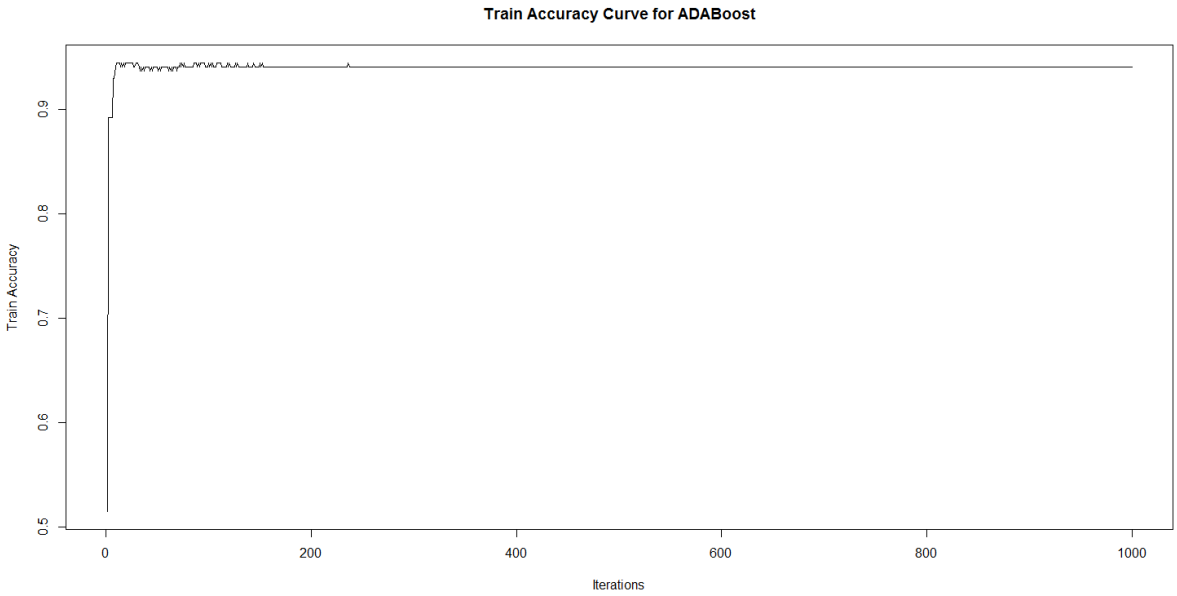


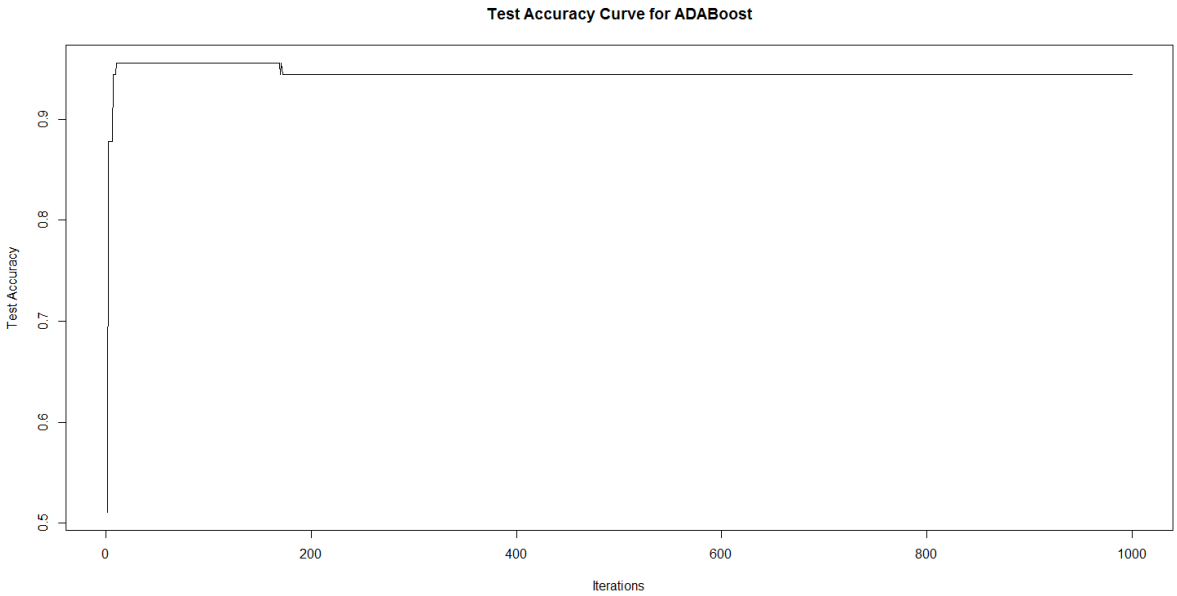
Anurag Pande: 604749647

ADABOOST

(train: 0.9666667, test: 0.9518519)

Iterations: 50



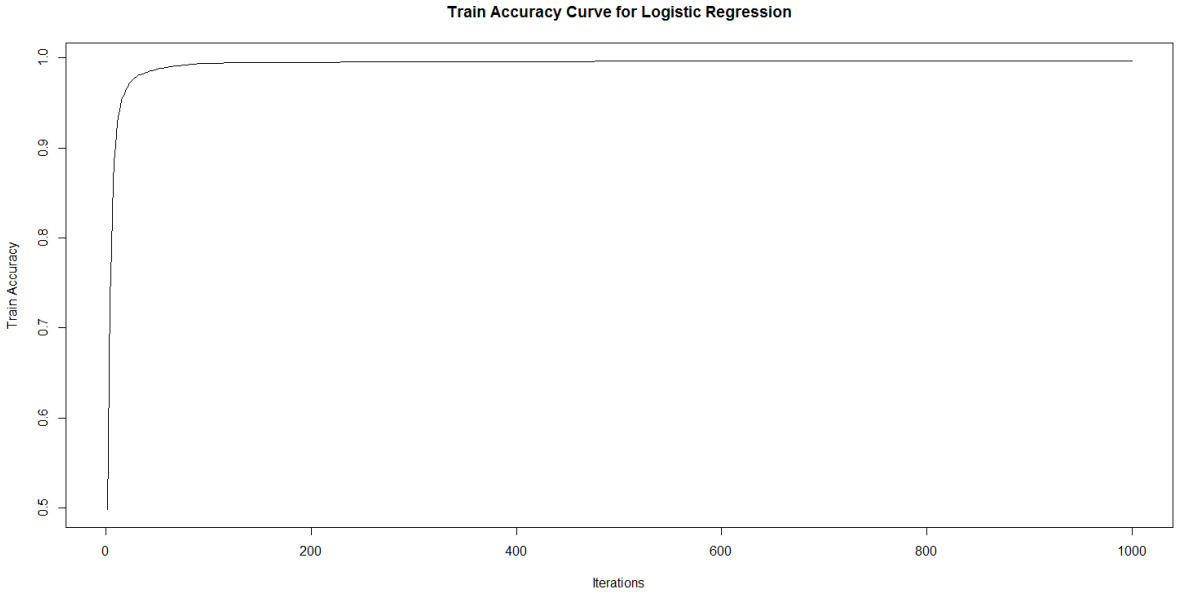


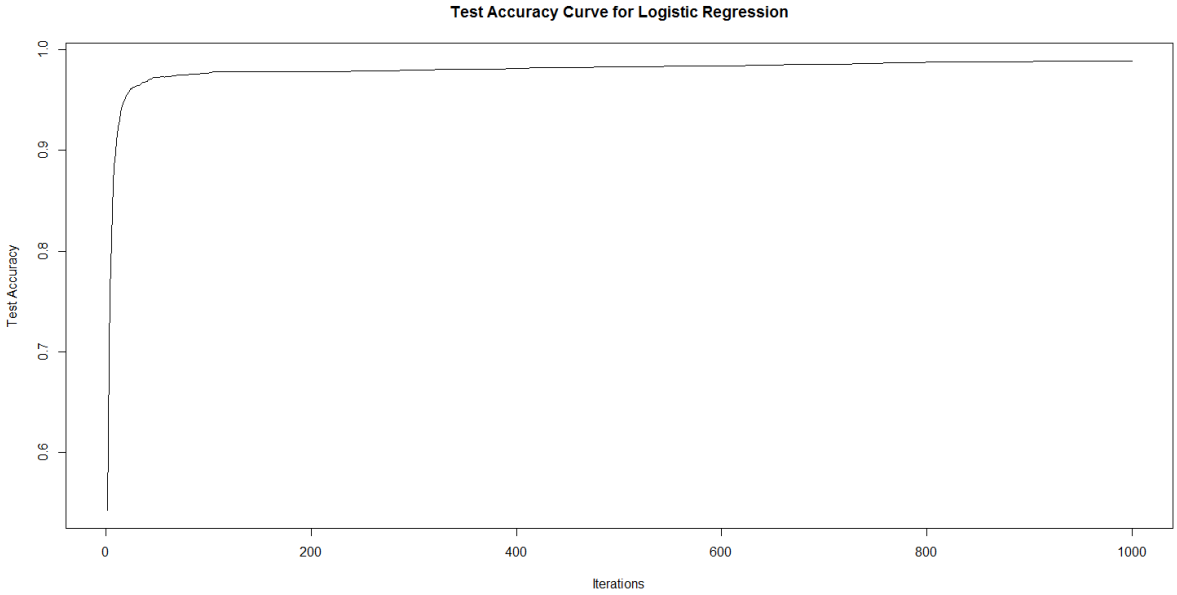
Anurag Pande: 604749647

LOGISTIC REGRESSION

(train: 0.9962963, test: 0.9886667)

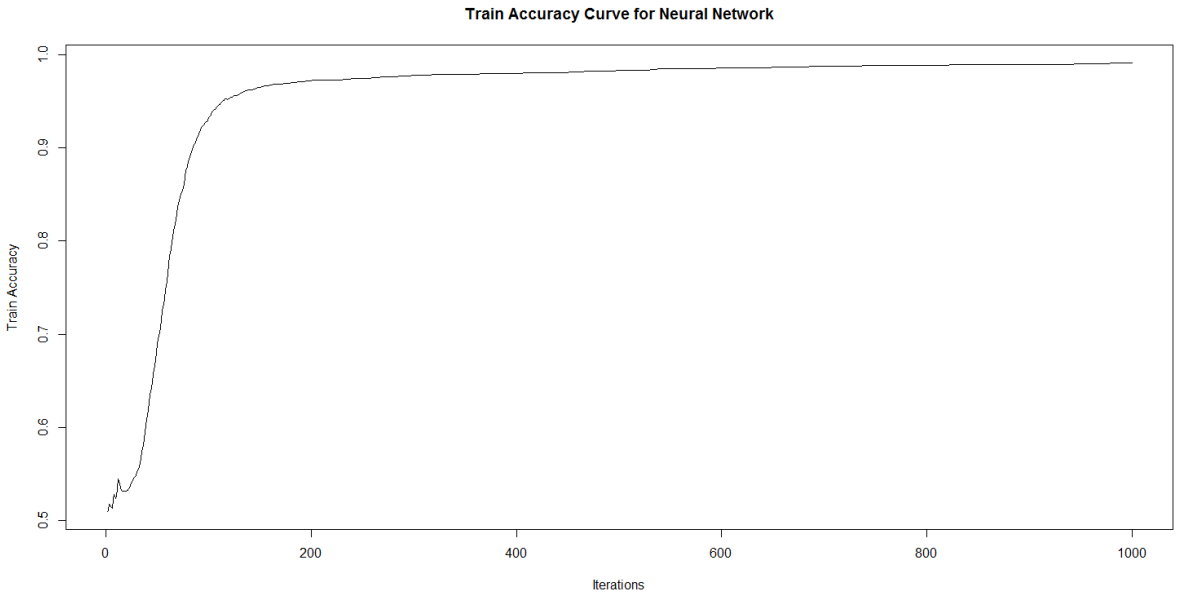
Iterations: 50

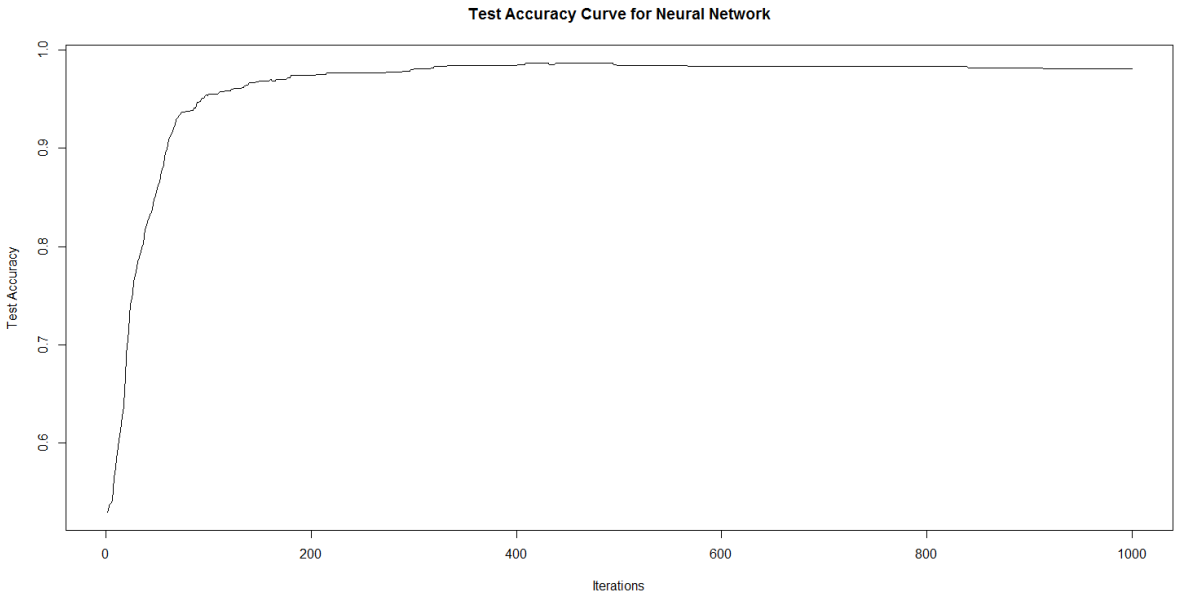




Anurag Pande: 604749647

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





Anurag Pande: 604749647

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.

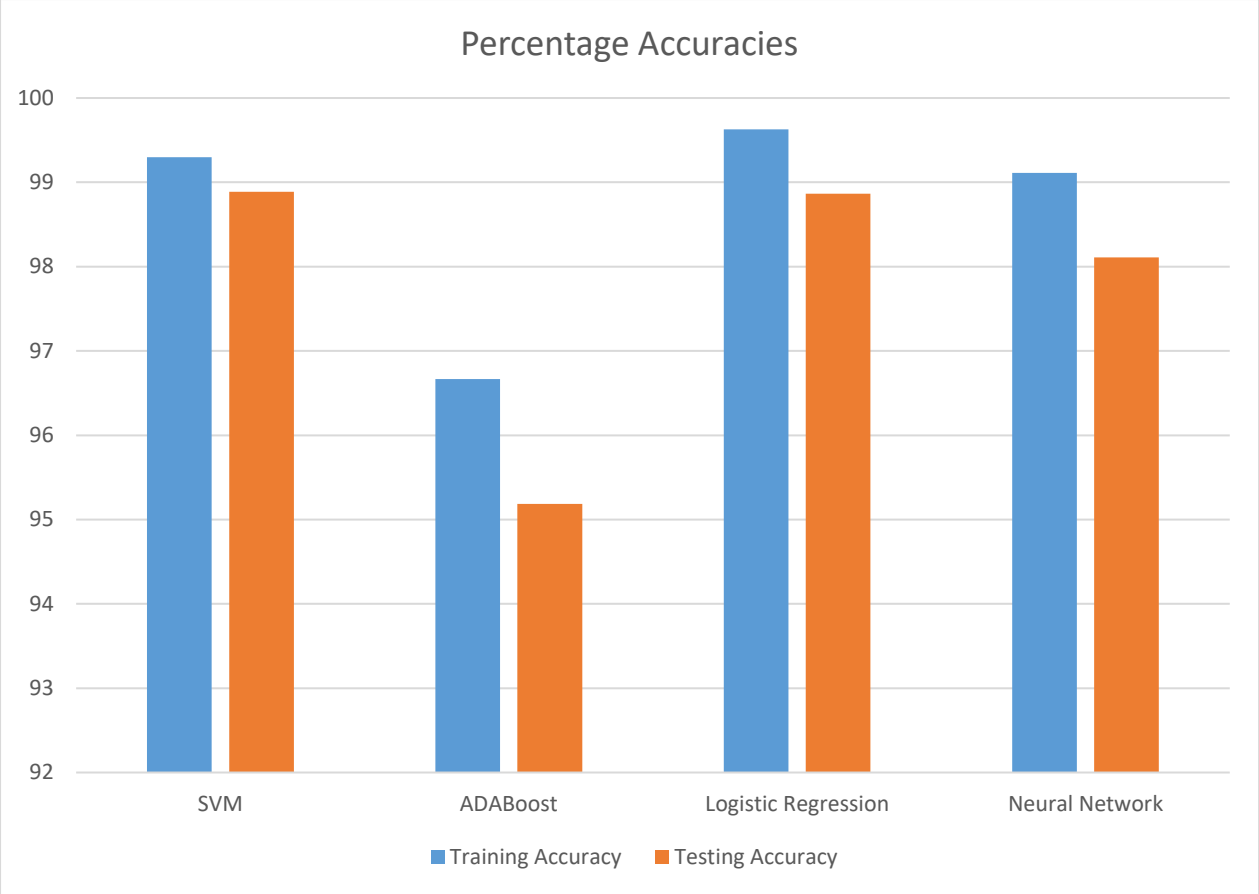
- The testing accuracy is also quite lower than the training accuracy as compared to others.
- ADABoost works well when there is a large class imbalance as well, and should be preferred here.
- Not parametric.
- Reaches effective solution quickly, but later becomes sensitive to noise.
- Works well with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

Anurag Pande: 604749647



Accuracies of the 4 models

(Per-model iterations were equalized to 1000 for all)

Anurag Pande: 604749647

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
```