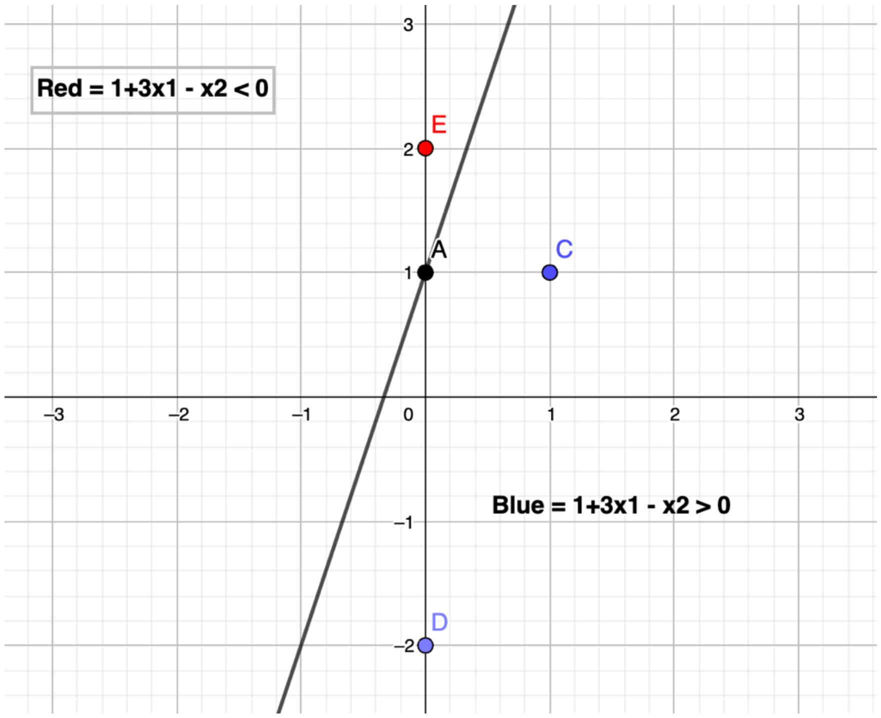
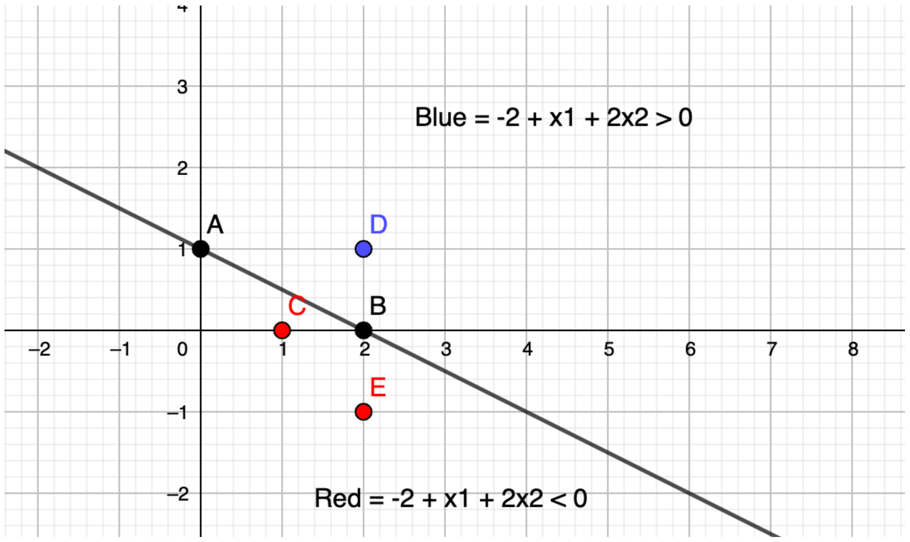
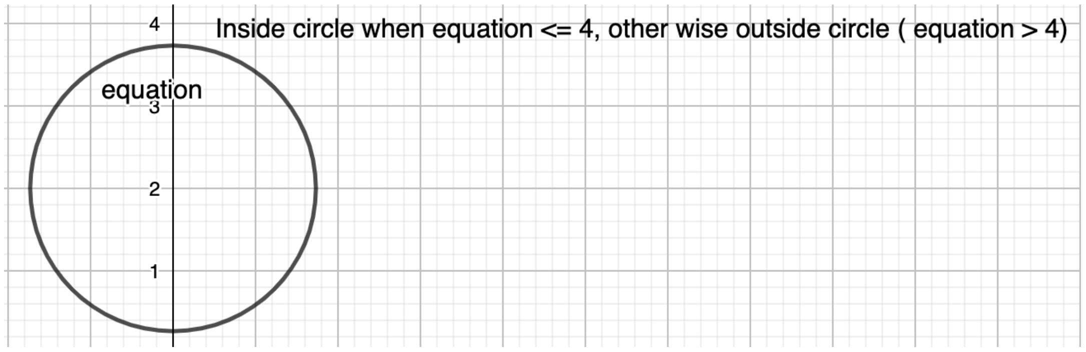
1. Hyperplanes
   a. $1 + 3X1 - X2 = 0$



   b. $-2+X1+2X2 = 0$

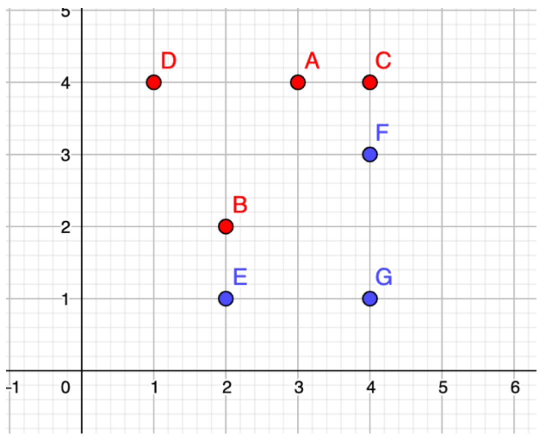


2. non-linear decision boundary.
   a.

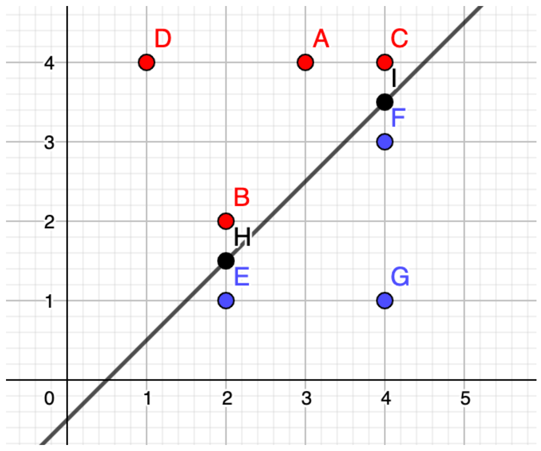


   b. (0,0) = Blue, (-1,1) = Red, (2,2) = Blue, (3,8) =Blue
   c. By expanding the predictor space(X1, X1^2, X2, X2^2) , we can generate a linear decision boundary in the enlarged predictor space but when we return to the original feature space, the decision boundary will be non-linear.

3. Here we explore the maximal margin classifier on a toy data seta.
   a.

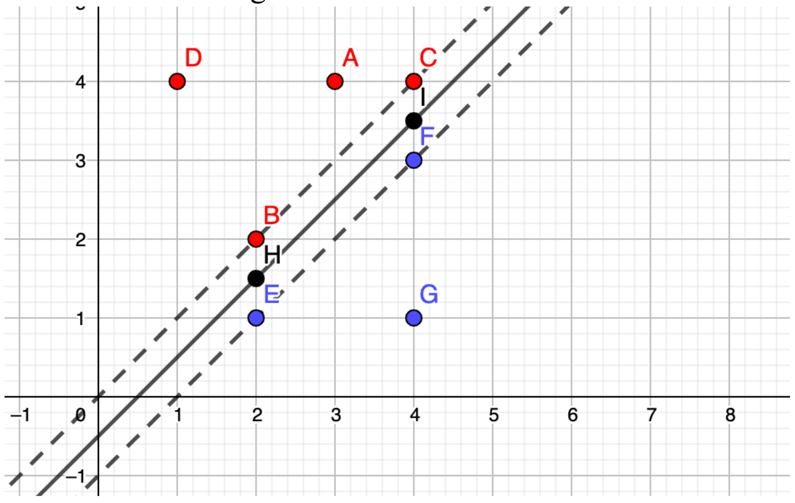


   b. This hyperplane must go through (2,1.5) and (4, 3.5) meaning its equation is X1 – X2 - 0.5 = 0



c. Classify point as Red if X1−X2−0.5 < 0 and classify point as Blue otherwise.

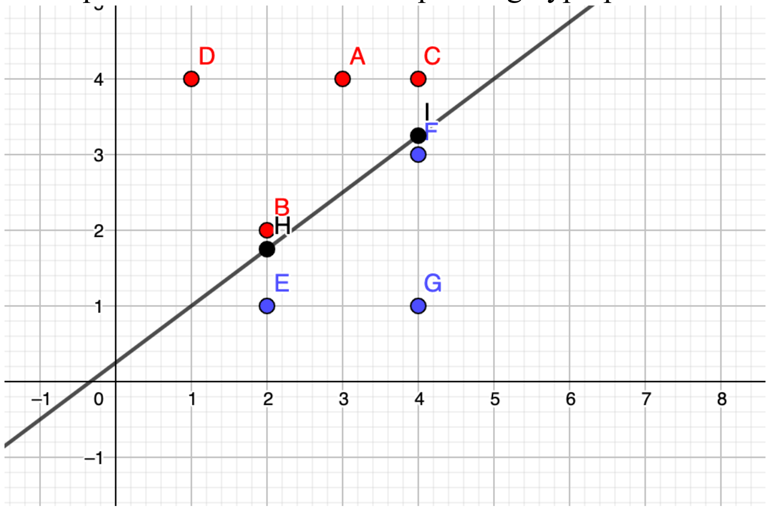
d. Dashed lines = margin



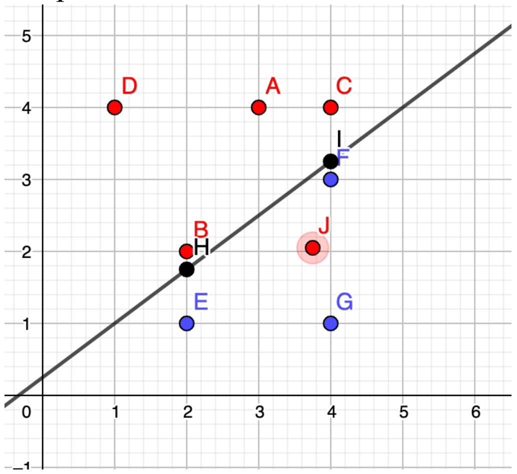
e. Support vectors are points= (2,2), (2,1), (4,3), (4,4)

f. Since the point (4,1) is not a support vector, a slight change in its position will not result in a change of the MM hyperplane.

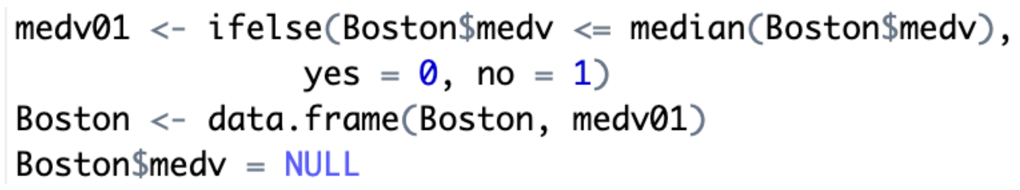
g. The equation to this non ideal separating hyperplane is 0.5 + 1.5X1 – 2X2 = 0



h. See point J

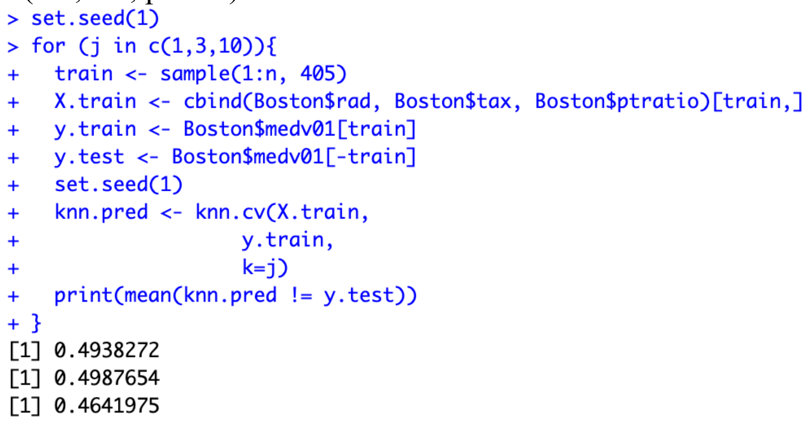


4. Use Boston data:
   a.



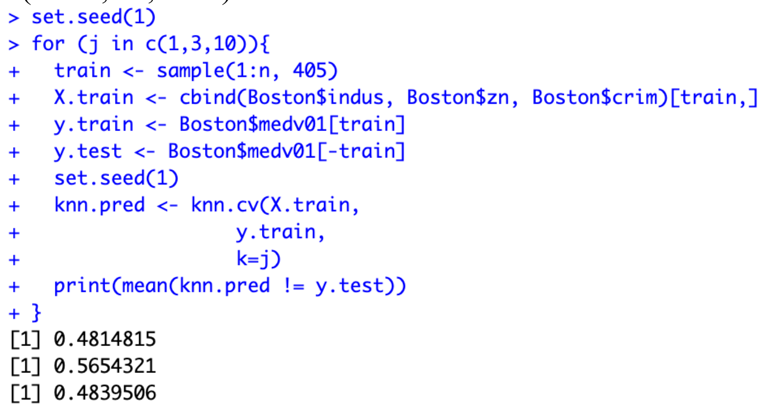
```
medv01 <- ifelse(Boston$medv <= median(Boston$medv),
                 yes = 0, no = 1)
Boston <- data.frame(Boston, medv01)
Boston$medv = NULL
```

   b. We will use k = 1,3,10 and the subsets of (rad, tax, ptratio), (indus, zn, crim), (all predictors except medv01)
   c. We can use LOOCV to get stable results. Applying this gives these results:
      i. (rad, tax, ptratio)



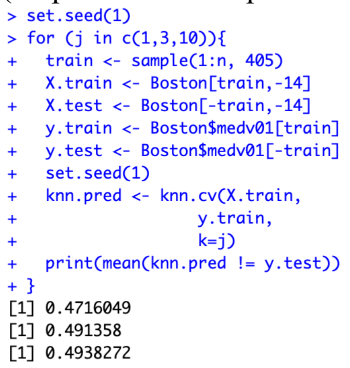
```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- cbind(Boston$rad, Boston$tax, Boston$ptratio)[train,]
+   y.train <- Boston$medv01[train]
+   y.test <- Boston$medv01[-train]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                     y.train,
+                     k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4938272
[1] 0.4987654
[1] 0.4641975
```

      ii. (indus, zn, crim)



```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- cbind(Boston$indus, Boston$zn, Boston$crim)[train,]
+   y.train <- Boston$medv01[train]
+   y.test <- Boston$medv01[-train]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                     y.train,
+                     k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4814815
[1] 0.5654321
[1] 0.4839506
```

      iii. (all predictors except medv01)



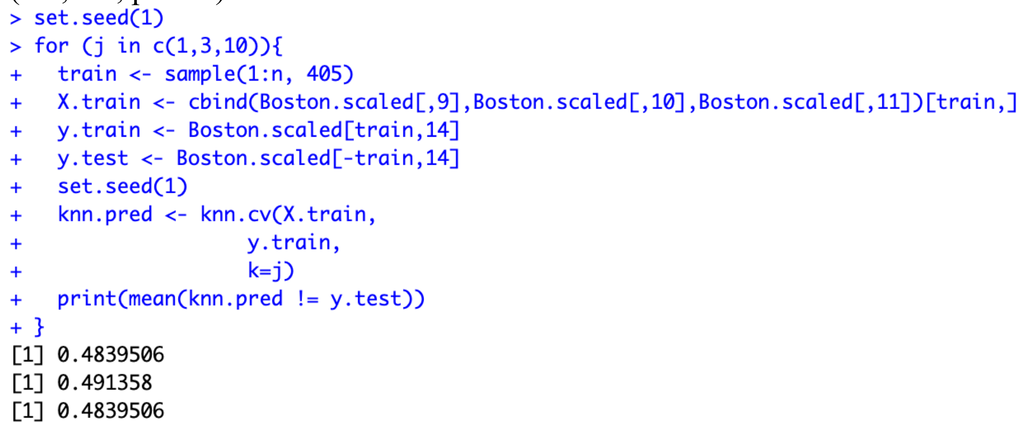
```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- Boston[train,-14]
+   X.test <- Boston[-train,-14]
+   y.train <- Boston$medv01[train]
+   y.test <- Boston$medv01[-train]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                     y.train,
+                     k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4716049
[1] 0.491358
[1] 0.4938272
```

      iv. The subset (rad, tax, ptratio) when k = 10 produced the best test error at .4641

d. No, not all the predictors are on the same scale, we can deal with this by scaling the data using scale().

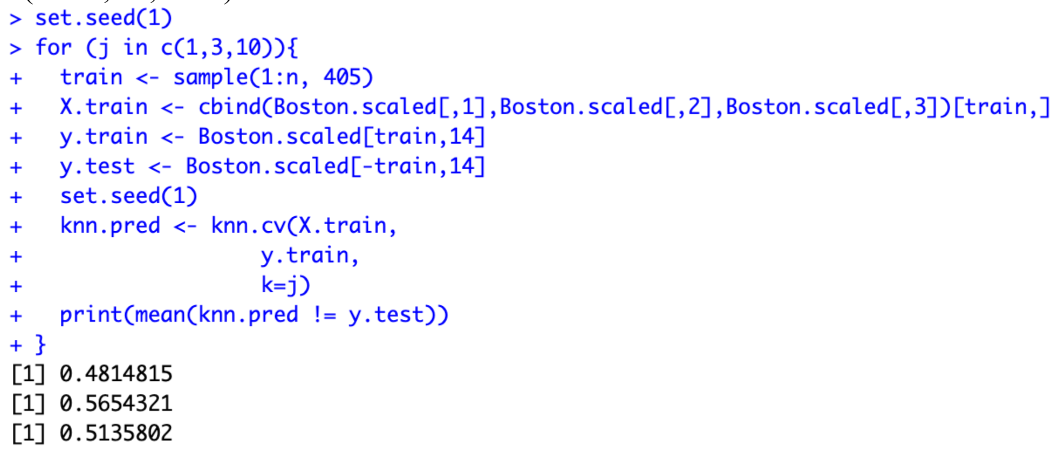
e. Scaled:

i. (rad, tax, ptratio)



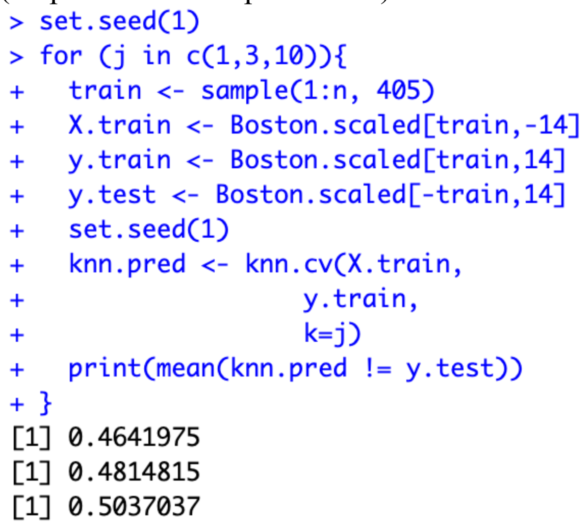
```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- cbind(Boston.scaled[,9],Boston.scaled[,10],Boston.scaled[,11])[train,]
+   y.train <- Boston.scaled[train,14]
+   y.test <- Boston.scaled[-train,14]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                   y.train,
+                   k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4839506
[1] 0.491358
[1] 0.4839506
```

ii. (indus, zn, crim)



```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- cbind(Boston.scaled[,1],Boston.scaled[,2],Boston.scaled[,3])[train,]
+   y.train <- Boston.scaled[train,14]
+   y.test <- Boston.scaled[-train,14]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                   y.train,
+                   k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4814815
[1] 0.5654321
[1] 0.5135802
```

iii. (all predictors except medv01)



```
> set.seed(1)
> for (j in c(1,3,10)){
+   train <- sample(1:n, 405)
+   X.train <- Boston.scaled[train,-14]
+   y.train <- Boston.scaled[train,14]
+   y.test <- Boston.scaled[-train,14]
+   set.seed(1)
+   knn.pred <- knn.cv(X.train,
+                   y.train,
+                   k=j)
+   print(mean(knn.pred != y.test))
+ }
[1] 0.4641975
[1] 0.4814815
[1] 0.5037037
```

iv. Yes, the results did change and the best model is now all predictors except medv01 and when k = 1 which produced a test error of .4641

5. Support Vectors
    a.
```
library(ISLR)
mpg01.t = ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpg01 = as.factor(mpg01.t)
```
    b. CV errors:
```
- Detailed performance results:
   cost      error dispersion
1 1e-02 0.07403846 0.05471525
2 1e-01 0.03826923 0.05148114
3 1e+00 0.01275641 0.01344780
4 5e+00 0.01782051 0.01229997
5 1e+01 0.02038462 0.01074682
6 1e+02 0.03820513 0.01773427
```
    c. The best classification error is when cost = 1. We can either make false true prediction or false negative prediction and this model make more false negative predictions than false positive predictions.
```
         truth
predict   0   1
      0 196   1
      1   0 195
```
    d. F

6. OJ data set
    a.
```
set.seed(1)

train <- sample(nrow(OJ), 800, replace = FALSE)

x.train <- OJ[train,]
t.test <- OJ[-train,]
```
    b.
```
Call:
svm(formula = Purchase ~ ., data = x.train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  432

 ( 215 217 )


Number of Classes:  2

Levels:
 CH MM
```

c. Training, testing

```
> postResample(predict(svm.res, x.train), x.train$Purchase)
 Accuracy     Kappa 
0.8337500 0.6429578 
> postResample(predict(svm.res, t.test), t.test$Purchase)
 Accuracy     Kappa 
0.8185185 0.6184461 
```

d.

```
svm.res.tune <- train(Purchase ~ ., data = x.train,
                         method = 'svmLinear2',
                         trControl = trainControl(method = 'cv', number = 10),
                         preProcess = c('center', 'scale'),
                         tuneGrid = expand.grid(cost = seq(0.01, 10, length.out = 20)))
```

e.

```
Support Vector Machines with Linear Kernel 

800 samples
 17 predictor
  2 classes: 'CH', 'MM' 

Pre-processing: centered (17), scaled (17) 
Resampling: Cross-Validated (10 fold) 
Summary of sample sizes: 719, 719, 721, 721, 721, 719, ... 
Resampling results across tuning parameters:

  cost        Accuracy   Kappa    
   0.0100000  0.8323551  0.6402131
   0.5357895  0.8336521  0.6455392
   1.0615789  0.8361367  0.6512304
   1.5873684  0.8374025  0.6541524
   2.1131579  0.8386371  0.6569935
   2.6389474  0.8386371  0.6570459
   3.1647368  0.8398871  0.6594124
   3.6905263  0.8386371  0.6564775
   4.2163158  0.8386371  0.6564775
   4.7421053  0.8386683  0.6561369
   5.2678947  0.8361683  0.6509025
   5.7936842  0.8349025  0.6478309
   6.3194737  0.8349025  0.6478309
   6.8452632  0.8336525  0.6449249
   7.3710526  0.8336525  0.6449249
   7.8968421  0.8336525  0.6445955
   8.4226316  0.8311834  0.6389313
   8.9484211  0.8324492  0.6420029
   9.4742105  0.8311992  0.6396677
  10.0000000  0.8299492  0.6372440

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cost = 3.164737.
```