K-Nearest Neighbors

The following code imports the pre-processed data model matrices and responses.

```
load("train.mat.JV.RData"); load("test.mat.JV.RData")
load("train.JV.RData"); load("test.JV.RData")
```

The `FNN` library has a `knn.reg` function that can be used to do KNN regression. The `Metrics` library has a `rmsle` function that can be used to evaluate error, because this is the same metric that Kaggle uses. The `leaps` library has a `regsubsets` function that will be used to choose 1,2,3,4 "best" variables.

```
library(FNN)
library(Metrics)
library(leaps)
```

KNN

KNN uses averages to estimate a response based on the response of the neighbors of the test point in the training set. Generally, KNN is not good for $p > 4$, but it is worth a shot.

The following code tests different numbers of neighbors to find the one that yields the lowest estimated error using 5-fold cross-validation. It then uses that number of neighbors to make a prediction on the test set. This prediction is then written to a .csv file for submission to Kaggle.

```
K.max <- 100  # maximum possible number of neighbors within reasonable computation time

set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

knn.rmsleki.cv <- matrix(NA, ncol=K.max, nrow=5)  # initialize storage of the k RMSLE's
for (j in 1:5) {
    train.x <- train.mat[fold.index != j,]  # fold training set
    train.y <- train$SalePrice[fold.index != j]  # fold training response
    test.x <- train.mat[fold.index == j,]  # fold test set
    true.y <- train$SalePrice[fold.index == j]  # fold test response

    for (Ki in 1:K.max) {
      knn.predk.cv <- knn.reg(train=train.x, test=test.x, y=train.y, k=Ki)$pred  # make KNN pred
ictions using Ki neighbors
      knn.rmsleki.cv[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.cv)  # calculate estimat
ed RMSLE
    }
}

knn.rmslei.cv <- colMeans(knn.rmsleki.cv)  # calculate the average RMSLE for each possible numbe
r of neighbors

knn.id.cv <- which(knn.rmslei.cv==min(knn.rmslei.cv))  # extract right number of neighbors

knn.pred.cv <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.cv)$pred  # m
ake prediction
knn.pred.cv.df <- data.frame(Id=test$Id, SalePrice=knn.pred.cv); write.csv(knn.pred.cv.df, "knn.
pred.cv.csv", row.names=F)  # write to CSV
```

Best Subset

The following code notes the best predictors for 1, 2, and 3 predictors.

```
train.mat.full <- cbind(train.mat, train$SalePrice)  # make a model matrix with the response
bst.reg <- regsubsets(V277~., data=as.data.frame(train.mat.full), nvmax=3, method="exhaustive",
 really.big=T)  # fit best subsets
```

```
Reordering variables and trying again:
```

```
names(coef(bst.reg, id=3))  # extract variable names
```

```
[1] "(Intercept)"       "`RoofMatlTar&Grv`" "BedroomAbvGr"
[4] "FireplaceQuPo"
```

```
names(coef(bst.reg, id=2))
```

```
[1] "(Intercept)"       "`RoofMatlTar&Grv`" "FireplaceQuPo"
```

```
names(coef(bst.reg, id=1))
```

```
[1] "(Intercept)"   "FireplaceQuPo"
```

The following code tests different numbers of neighbors to find the one that yields the lowest estimated error using 5-fold cross-validation using 1, 2, and 3 best subset predictors.

```
K.max <- 100  # maximum possible number of neighbors within reasonable computation time

set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

knn.rmsleki.1 <- matrix(NA, ncol=K.max, nrow=5); knn.rmsleki.2 <- matrix(NA, ncol=K.max, nrow=5
); knn.rmsleki.3 <- matrix(NA, ncol=K.max, nrow=5)  # initialize storage of the k RMSLE's
for (j in 1:5) {
  train.x <- train.mat[fold.index != j,]  # fold training set
  train.x.1 <- as.data.frame(train.x[,215])
  train.x.2 <- train.x[,c(225,106)]
  train.x.3 <- train.x[,c(225,106,199)]
  train.y <- train$SalePrice[fold.index != j]  # fold training response
  test.x <- train.mat[fold.index == j,]  # fold test set
  test.x.1 <- as.data.frame(test.x[,215])
  test.x.2 <- test.x[,c(225,106)]
  test.x.3 <- test.x[,c(225,106,199)]
  true.y <- train$SalePrice[fold.index == j]  # fold test response

  for (Ki in 1:K.max) {
    knn.predk.1 <- knn.reg(train=train.x.1, test=test.x.1, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.1[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.1)  # calculate estimated R
MSLE
    knn.predk.2 <- knn.reg(train=train.x.2, test=test.x.2, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.2[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.2)  # calculate estimated R
MSLE
    knn.predk.3 <- knn.reg(train=train.x.3, test=test.x.3, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.3[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.3)  # calculate estimated R
MSLE
  }
}

knn.rmslei.1 <- colMeans(knn.rmsleki.1); knn.rmslei.2 <- colMeans(knn.rmsleki.2); knn.rmslei.3 <
- colMeans(knn.rmsleki.3)  # calculate the average RMSLE for each possible number of neighbors

knn.id.1 <- which(knn.rmslei.1==min(knn.rmslei.1)); knn.id.2 <- which(knn.rmslei.2==min(knn.rmsl
ei.2)); knn.id.3 <- which(knn.rmslei.3==min(knn.rmslei.3))  # extract right number of neighbors

knn.pred.1 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.1)$pred  # mak
e prediction
knn.pred.1.df <- data.frame(Id=test$Id, SalePrice=knn.pred.1); write.csv(knn.pred.1.df, "knn.pre
d.1.csv", row.names=F)  # write to CSV
knn.pred.2 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.2)$pred  # mak
e prediction
knn.pred.2.df <- data.frame(Id=test$Id, SalePrice=knn.pred.2); write.csv(knn.pred.2.df, "knn.pre
d.2.csv", row.names=F)  # write to CSV
knn.pred.3 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.3)$pred  # mak
e prediction
knn.pred.3.df <- data.frame(Id=test$Id, SalePrice=knn.pred.3); write.csv(knn.pred.3.df, "knn.pre
d.3.csv", row.names=F)  # write to CSV
```

It turns out this was a poor idea, but it was worth a shot.

Summary of Results

Method	K	Estimated RMSLE	Actual RMSLE	Notes
KNN Regression	6	0.22252	0.38399	Using all predictors
KNN Regression	14	0.39730	0.37080	Using FireplaceQuPo
KNN Regression	16	0.39528	0.36765	Using FireplaceQuPo, RoofMatlTar&Grv
KNN Regression	100	0.39357	0.35547	Using FireplaceQuPo, RoofMatlTar&Grv, BedroomAbvGr