STAT 542 / CS 598: Homework 1

Fall 2019, by Ruoqing Zhu (rqzhu)

Due: Monday, Sep 9 by 11:59 PM Pacific Time

Contents

Question 1 [50 Points] KNN

Write an R function to fit a KNN regression model. Complete the following steps

a. [15 Points] Write a function `myknn(xtest, xtrain, ytrain, k)` that fits a KNN model that predict a target point or multiple target points `xtest`. Here `xtrain` is the training dataset covariate value, `ytrain` is the training data outcome, and `k` is the number of nearest neighbors. Use the ℓ_2 norm to evaluate the distance between two points. Please note that you cannot use any additional `R` package within this function.

```
myknn <- function(xtest, xtrain, ytrain, k) {
  apply(xtest, 1, function(a){
    distances <- apply(xtrain, 1, function(b) norm(a-b, type="2"))
    mean(ytrain[order(distances)[1:k]])
  })
}
```

b. [10 Points] Generate 1000 observations from a five-dimensional normally distribution:

$$\mathcal{N}(\mu, \Sigma_{5\times 5})$$

where $\mu = (1, 2, 3, 4, 5)^{\mathrm{T}}$ and $\Sigma_{5\times 5}$ is an autoregressive covariance matrix, with the (i, j)th entry equal to $0.5^{|i-j|}$. Then, generate outcome values Y based on the linear model

$$Y = X_1 + X_2 + (X_3 - 2.5)^2 + \epsilon$$

where ϵ follows i.i.d. standard normal distribution. Use `set.seed(1)` right before you generate this entire data. Print the first 3 entries of your data.

```
covar <- 0.5^abs(sapply(0:4, function(i) i:(i-4), simplify=TRUE))

library("mvtnorm")
set.seed(1)
X <- rmvnorm(1000, 1:5, covar)
colnames(X) <- sprintf("X%d", 1:5)
Y <- X[,1] + X[,2] + (X[,3]-2.5)^2 + rnorm(1000)
```

First 3 lines of X:

```
X[1:3,]
```

```
##              X1       X2       X3       X4       X5
## [1,] 0.4326852 1.972780 2.624989 5.359585 5.632753
## [2,] 0.4186266 2.469279 3.837800 4.651698 4.924032
## [3,] 2.4266353 2.430771 2.229312 2.161263 5.513925
```

First 3 items Y:

```
Y[1:3]
```

```
## [1] 0.904714 5.306756 3.252485
```

c. [10 Points] Use the first 400 observations of your data as the training data and the rest as testing data. Predict the Y values using your KNN function with `k = 5`. Evaluate the prediction accuracy using mean squared error

$$\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2$$

```
train.X <- X[1:400,]
train.Y <- Y[1:400]
test.X <- X[401:1000,]
test.Y <- Y[401:1000]

knn.test <- list()
knn.test[[5]] <- myknn(test.X, train.X, train.Y, 5)
```

MSE when k=5:

```
mean((knn.test[[5]] - test.Y)^2)
```

```
## [1] 1.990502
```

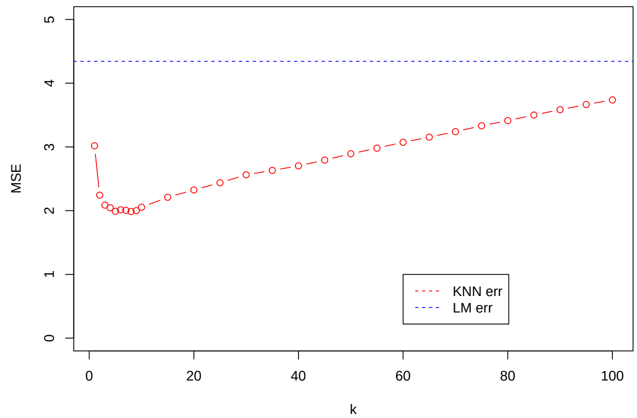
d. [15 Points] Compare the prediction error of a linear model with your KNN model. Consider k being 1, 2, 3, . . ., 9, 10, 15, 20, . . ., 95, 100. Demonstrate all results in a single, easily interpretable figure with proper legends.

```
library("parallel")
knn.mse <- mcmapply(function(k) {
  knn.test[[k]] <- myknn(test.X, train.X, train.Y, k)
  c(k, mean((knn.test[[k]] - test.Y)^2))
}, c(1:9, seq(10,100,5)), mc.cores=8)

train.df <- as.data.frame(cbind(train.Y, train.X))
colnames(train.df)[1] <- "Y"
linearModel <- lm(Y ~ ., data=train.df)

test.df = as.data.frame(test.X)
linearModel.mse <- mean(predict(linearModel, test.df))

plot(knn.mse[1,], knn.mse[2,], xlab="k", ylab="MSE", col="red", ylim=c(0,5), type="b")
abline(h=linearModel.mse, col="blue", type="b", lty=2)
legend(60, 1, legend=c("KNN err", "LM err"), col=c("red", "blue"), lty=2)
```



Linear model error is the blue line. KNN result with respect to k in the red line (just do it-nike)

Question 2 [50 Points] Linear Regression through Optimization

Linear regression is most popular statistical model, and the core technique for solving a linear regression is simply inverting a matrix:

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

However, lets consider alternative approaches to solve linear regression through optimization. We use a gradient descent approach. We know that $\widehat{\boldsymbol{\beta}}$ can also be expressed as

$$\widehat{\boldsymbol{\beta}} = \arg\min \ell(\boldsymbol{\beta}) = \arg\min \frac{1}{2n}\sum_{i=1}^{n}(y_i - x_i^{\mathrm{T}}\boldsymbol{\beta})^2.$$

And the gradient can be derived

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^{\mathrm{T}}\boldsymbol{\beta})x_i.$$

To perform the optimization, we will first set an initial beta value, say $\boldsymbol{\beta} = \mathbf{0}$ for all entries, then proceed with the updating

$$\boldsymbol{\beta}^{\text{new}} = \boldsymbol{\beta}^{\text{old}} - \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \times \delta,$$

where δ is some small constant, say 0.1. We will keep updating the beta values by setting $\boldsymbol{\beta}^{\text{new}}$ as the old value and calcuting a new one untill the difference between $\boldsymbol{\beta}^{\text{new}}$ and $\boldsymbol{\beta}^{\text{old}}$ is less than a prespecified threshold ϵ, e.g., $\epsilon = 10^{-6}$. You should also set a maximum number of iterations to prevent excessively long runing time.

a. [35 Points] Based on this description, write your own R function `mylm_g(x, y, delta, epsilon, maxitr)` to implement this optimization version of linear regression. The output of this function should be a vector of the estimated beta value.

```
mylm_g <- function(x, y, delta, epsilon, maxitr) {
  n <<- dim(x)[1]
  p <<- dim(x)[2]
  b_old <<- rep(1, p)
  for (i in 1:maxitr) {
```

```
    intermediate <<- sapply(1:n, function(i){ c(y[i] - x[i,] %*% b_old) * x[i,] })
    gradient <<- -apply(intermediate, 1, mean)
    b_new <<- b_old - gradient * delta
    if (sqrt(sum((b_new - b_old)^2)) < epsilon) {
      break
    }
    b_old <<- b_new
  }
  print(i)
  return (b_old)
}
```

b. [15 Points] Test this function on the Boston Housing data from the `mlbench` package. Documentation is provided here if you need a description of the data. We will remove `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. We will use a scaled and centered version of the data for estimation. Please also note that in this case, you do not need the intercept term. And you should compare your result to the `lm()` function on the same data. Experiment on different `maxitr` values to obtain a good solution. However your function should not run more than a few seconds.

```
library(mlbench)
data(BostonHousing2)
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
X = data.matrix(X)
X = scale(X)
Y = as.vector(scale(BostonHousing2$cmedv))
```

Number of iterations before convergence:

```
beta <- mylm_g(X, Y, 0.1, 10e-6, 1000)
```

```
## [1] 811
```

Beta coefficients:

```
beta
```

```
##          lon          lat         crim           zn        indus         chas
## -0.032352124  0.030214275 -0.097857831  0.118135118  0.010979922  0.071361592
##          nox           rm          age          dis          rad          tax
## -0.199605561  0.287298805  0.007521493 -0.321045579  0.289833911 -0.235399183
##      ptratio            b        lstat
## -0.206749614  0.091230663 -0.417937561
```

Coefficients from a linear model:

```
linearModel <- lm(Y~X)
linearModel
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
## (Intercept)         Xlon         Xlat        Xcrim          Xzn       Xindus
##  -1.243e-15   -3.232e-02    3.025e-02   -9.794e-02    1.183e-01    1.139e-02
##       Xchas         Xnox          Xrm         Xage         Xdis         Xrad
##   7.131e-02   -1.997e-01    2.872e-01    7.565e-03   -3.210e-01    2.909e-01
##        Xtax     Xptratio           Xb       Xlstat
```

```
##   -2.365e-01   -2.068e-01    9.124e-02   -4.180e-01
```

Mean squared error of beta:

```r
mean((X %*% beta - Y)^2)
```

```
## [1] 0.2537382
```

Mean squared error of linear model:

```r
mean((predict(linearModel, as.data.frame(X))-Y)^2)
```

```
## [1] 0.253738
```

They are almost identical

Bonus Question [5 Points] The Non-scaled Version

When we do not scale and center the data matrix (both X and Y), it could be challenging to obtain a good solution. Try this with your code, and comment on what you observed and explain why. Can you think of a way to calculate the beta parameters on the original scale using the solution from the previous question? To earn a full 5 point bonus, you must provide a rigors mathematical derivation and also validate that by comparing it to the `lm()` function on the original data.

Linear model:

```r
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
X = data.matrix(X)
Y = as.vector(BostonHousing2$cmedv)

linearModel <- lm(Y~X)
linearModel
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
## (Intercept)         Xlon         Xlat        Xcrim          Xzn       Xindus
##  -4.376e+02   -3.935e+00    4.495e+00   -1.045e-01    4.656e-02    1.525e-02
##       Xchas         Xnox          Xrm         Xage         Xdis         Xrad
##   2.578e+00   -1.582e+01    3.754e+00    2.468e-03   -1.400e+00    3.067e-01
##        Xtax     Xptratio           Xb       Xlstat
##  -1.289e-02   -8.771e-01    9.176e-03   -5.374e-01
```

For my beta model, unfortunately the gradients exploded when values are not centered/scaled. To account for the center, an intercept value "u" will need to be added:

$$\widehat{\boldsymbol{\beta}} = \arg\min \ell(\boldsymbol{\beta}, u) = \arg\min \frac{1}{2n} \sum_{i=1}^{n} (y_i - x_i^{\mathrm{T}} \boldsymbol{\beta} - u)^2.$$

To perform gradient descent on this new estimator equation, I need to do a partial derivate like before:

$$\frac{\partial \ell(\boldsymbol{\beta}, u)}{\partial \boldsymbol{\beta}} = -\frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^{\mathrm{T}} \boldsymbol{\beta} - u) x_i.$$

$$\frac{\partial \ell(\boldsymbol{\beta}, u)}{\partial u} = -\frac{1}{n} \sum_{i=1}^{n} y_i - x_i^{\mathrm{T}} \boldsymbol{\beta} - u.$$

```
mylm_g2 <- function(x, y, delta, epsilon, maxitr) {
  n <<- dim(x)[1]
  p <<- dim(x)[2]
  b_old <<- rep(0, p)
  u_old <<- 0
  for (i in 1:maxitr) {
    intermediate <<- sapply(1:n, function(i){ c(y[i] - x[i,] %*% b_old - u_old) * x[i,] })
    gradient <<- -apply(intermediate, 1, mean)
    b_new <<- b_old - gradient * delta
    u_new <<- u_old - -mean(sapply(1:n, function(i){ y[i] - c(x[i,] %*% b_old) - u_old })) * delta
    if (sqrt(sum((b_new - b_old)^2)) < epsilon && u_new - u_old < epsilon) {
      break
    }
    b_old <<- b_new
    u_old <<- u_new
  }
  print(i)
  return(list("beta"=b_old, "u"=u_old))
}
```

Testing:

```
#betaModel <- mylm_g2(X, Y, 10e-4, 10e-6, 10000)
#betaModel$beta
#betaModel$u
```

Unfortunately my solution does not converge no matter what settings I try