Coursera ML: Multiple linear regression quiz

Varun Boodram

December 11, 2015

This is my first attempt to solve the multiple linear regression programming problems in the second week of the Regression module of the ML course. I intend also to to this in python.

Obtention and cleaning The data were downloaded from the course website in .csv format, and included in the current working directory.

```
train_data<-read.csv("./datasets/kc_house_train_data.csv", sep = ",", quote = " ", stringsAsFactors = F
test_data<- read.csv("./datasets/kc_house_test_data.csv", sep = ",", quote = " ", stringsAsFactors = F)
dim(train_data)
```

```
## [1] 17384    21
```

```
train_data[!complete.cases(train_data)]
```

```
## data frame with 0 columns and 17384 rows
```

The instructions for this assignment require that the each data point have the following classes: str, str, float, float, float, float, int, str, int, int, int, int, int, int, int, int, str, float, float, float, float. Floats are numeric in r

```
classes<-vector()
for (i in 1:ncol(train_data)){
        classes[i]<-class(train_data[,i])
}
classes
```

```
##  [1] "character" "character" "numeric"   "integer"   "numeric"
##  [6] "integer"   "integer"   "character" "integer"   "integer"
## [11] "integer"   "integer"   "integer"   "integer"   "integer"
## [16] "integer"   "character" "numeric"   "numeric"   "integer"
## [21] "integer"
```

There are differences here, which we adjust as follows

```
train_data$bathrooms<-as.numeric(train_data$bathrooms)
```

and save space by not showing every line of code.

```
classes<-vector()
for (i in 1:ncol(train_data)){
        classes[i]<-class(train_data[,i])
}
classes
```

```
##  [1] "character" "character" "numeric"   "numeric"   "numeric"
##  [6] "numeric"   "integer"   "character" "integer"   "integer"
## [11] "integer"   "integer"   "integer"   "integer"   "integer"
## [16] "integer"   "character" "numeric"   "numeric"   "numeric"
## [21] "numeric"
```

Similar adjustments are made to the classes of the `test_data`.

```
classes<-vector()
for (i in 1:ncol(test_data)){
        classes[i]<-class(test_data[,i])
}
classes
```

```
##  [1] "character" "character" "numeric"   "numeric"   "numeric"
##  [6] "numeric"   "integer"   "character" "integer"   "integer"
## [11] "integer"   "integer"   "integer"   "integer"   "integer"
## [16] "integer"   "character" "numeric"   "numeric"   "numeric"
## [21] "numeric"
```

Comparing to str, str, float, float, float, float, int, str, int, int, int, int, int, int, int, int, str, float, float, float, float, shows that our initial data cleaning step is complete

Data engineering *Although we often think of multiple regression as including multiple different features (e.g. # of bedrooms, square feet, and # of bathrooms) but we can also consider transformations of existing variables e.g. the log of the square feet or even "interaction" variables such as the product of bedrooms and bathrooms. Add 4 new variables in both your train_data and test_data.*

- 'bedrooms_squared' = 'bedrooms'*'bedrooms'
- 'bed_bath_rooms' = 'bedrooms'*'bathrooms'
- 'log_sqft_living' = log('sqft_living')
- 'lat_plus_long' = 'lat' + 'long'

```
# create the features for the train set
train_data$bedrooms_squared <- train_data$bedrooms*train_data$bedrooms
train_data$bed_bath_rooms <- train_data$bedrooms*train_data$bathrooms
train_data$log_sqft_living <- log(train_data$sqft_living)
train_data$lat_plus_long <- train_data$lat + train_data$long

# create the features for the test set
test_data$bedrooms_squared <- test_data$bedrooms*test_data$bedrooms
test_data$bed_bath_rooms <- test_data$bedrooms*test_data$bathrooms
test_data$log_sqft_living <- log(test_data$sqft_living)
test_data$lat_plus_long <- test_data$lat + test_data$long
```

```
for (i in (ncol(test_data)-4):ncol(test_data)){
        print(paste("mean of",
                    names(test_data)[i],
```

```
                    "is",
                    round(mean(test_data[,i]),
                          digits = 2),
                    sep = " "
                    )
              )
       }
```

Quiz Question: what are the mean (arithmetic average) values of your 4 new variables on TEST data? (round to 2 digits)

```
## [1] "mean of sqft_lot15 is 12735.88"
## [1] "mean of bedrooms_squared is 12.45"
## [1] "mean of bed_bath_rooms is 7.5"
## [1] "mean of log_sqft_living is 7.55"
## [1] "mean of lat_plus_long is -74.65"
```

Use graphlab.linear_regression.create (or any other regression library/function) to estimate the regression coefficients/weights for predicting 'price' for the following three models:(In all 3 models include an intercept – most software does this by default).

- Model 1: 'sqft_living', 'bedrooms', 'bathrooms', 'lat', and 'long'

```
library(stats)
model1<-lm(formula = price~sqft_living+bedrooms+bathrooms+lat+long, data = train_data)
```

- Model 2: 'sqft_living', 'bedrooms', 'bathrooms', 'lat','long', and 'bed_bath_rooms'

```
model2 <- lm(formula = price~sqft_living + bedrooms + bathrooms + lat + long + bed_bath_rooms, data = t
```

- Model 3: 'sqft_living', 'bedrooms', 'bathrooms', 'lat','long', 'bed_bath_rooms', 'bedrooms_squared', 'log_sqft_living', and 'lat_plus_long'

```
model3 <- lm(formula = price~sqft_living + bedrooms + bathrooms + lat + long + bed_bath_rooms + bedroom
```

```
summary(model1)$coefficients
```

Quiz Question: What is the sign (positive or negative) for the coefficient/weight for 'bathrooms' in Model 1?

```
##                   Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept) -6.907573e+07 1.647001e+06 -41.940312  0.000000e+00
## sqft_living  3.122586e+02 3.183162e+00  98.096986  0.000000e+00
## bedrooms    -5.958653e+04 2.482861e+03 -23.999140 3.076807e-125
## bathrooms    1.570674e+04 3.587158e+03   4.378603  1.201402e-05
## lat          6.586193e+05 1.309741e+04  50.286230  0.000000e+00
## long        -3.093744e+05 1.326025e+04 -23.330959 1.421639e-118
```

```
summary(model2)$coefficients
```

Quiz Question: What is the sign (positive or negative) for the coefficient/weight for 'bathrooms' in Model 2?

```
##                     Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept)    -6.686797e+07 1.647625e+06 -40.584450  0.000000e+00
## sqft_living     3.066101e+02 3.196886e+00  95.908970  0.000000e+00
## bedrooms       -1.134464e+05 4.797612e+03 -23.646426 1.067118e-121
## bathrooms      -7.146131e+04 7.552563e+03  -9.461863  3.402138e-21
## lat             6.548446e+05 1.303680e+04  50.230461  0.000000e+00
## long           -2.942990e+05 1.324578e+04 -22.218326 7.231375e-108
## bed_bath_rooms  2.557965e+04 1.953134e+03  13.096723  5.260576e-39
```

Is the sign for the coefficient the same in both models? Think about why this might be the case.

The signs are different. This may be the effect of the feature `bed_bath_rooms`, the intercept of which is quite large. Perhaps on their own, the bedroom and bathroom features have less of an impact on housing prices than their product, and so the weight of the effect of the bathroom is reduced in this model.

Now using your three estimated models compute the RSS (Residual Sum of Squares) on the Training data.

```
# resid() returns the residuals of the fit model. The RSS is computed with sum(resid()^2)
RSS1 <- sum(resid(model1)^2)
RSS2 <- sum(resid(model2)^2)
RSS3 <- sum(resid(model3)^2)
all_RSS <- data.frame(RSS1, RSS2, RSS3); all_RSS
```

```
##         RSS1         RSS2         RSS3
## 1 9.6788e+14 9.584196e+14 9.034365e+14
```

Quiz Question: Which model (1, 2 or 3) had the lowest RSS on TRAINING data? This is given by `model3`

Now using your three estimated models compute the RSS on the Testing data

```
predictions1<-predict(object = model1, newdata = test_data)
predictions2<-predict(object = model2, newdata = test_data)
predictions3<-predict(object = model3, newdata = test_data)
```

```
## Warning in predict.lm(object = model3, newdata = test_data): prediction
## from a rank-deficient fit may be misleading
```

```
test_residuals1 <- predictions1-test_data$price
test_residuals2 <- predictions2-test_data$price
test_residuals3 <- predictions3-test_data$price
test_RSS1 <- sum(test_residuals1^2)
test_RSS2 <- sum(test_residuals2^2)
test_RSS3 <- sum(test_residuals3^2)
all_test_RSS<-data.frame(test_RSS1, test_RSS2, test_RSS3)
all_test_RSS
```

```
##      test_RSS1    test_RSS2    test_RSS3
## 1 2.255005e+14 2.233775e+14 2.592363e+14
```

Quiz Question: Which model (1, 2, or 3) had the lowest RSS on TESTING data? This is given by `model2`

Did you get the same answer for 9 and 11? Think about why this might be the case.

The warning message above for `model3` suggests that it is not full rank: this means that some of the information captured in one feature is also represented in another. It may be the case that `model3` is over-fitting the training data, and so performs less well on the test data.

Write a function that takes a data set, a list of features (e.g. ['sqft_living', 'bedrooms']), to be used as inputs, and a name of the output (e.g. 'price'). This function should return a features_matrix (2D array) consisting of first a column of ones followed by columns containing the values of the input features in the data set in the same order as the input list.

```
# construct_matrix() accepts as input a list of features, a list of outputs, and a data.frame, and retu
construct_features_matrix <- function(features, outputs, data){
        # convert features input to a list
        features <- as.list(features)
        # extract the features data from the data
        subset_data <- get_output(data, features)
        # extract what we want to predict from the data
        subset_outputs <- get_output(data, outputs)
        # append a vector of ones to the features matrix
        features_matrix <- create_matrix(subset_data)
        IO <- list(features_matrix, subset_outputs)
        IO
}

# get_output() subsets the data frame into the inputs provided
get_output <- function(data, features){
        output<-matrix(nrow = nrow(data), ncol = length(features))
        for (i in 1: length(features)){
               output[,i]<-as.numeric(data[[features[[i]]]])
        }
        output
}

# create_matrix appends a column of 1s to the output of get_output()
create_matrix <- function(subset_data){
        length <- nrow(subset_data)
        concatinated <- cbind(rep(1, length), subset_data)
        concatinated
}
```

If the features matrix (including a column of 1s for the constant) is stored as a 2D array (or matrix) and the regression weights are stored as a 1D array then the predicted output is just the dot product between the features matrix and the weights (with the weights on the right). Write a function 'predict_output' which accepts a 2D array 'feature_matrix' and a 1D array 'weights' and returns a 1D array 'predictions'

```
# predict_outputs() takes as inputs a matrix of features, and a weights vector (c()), and returns a vec
predict_output <- function(feature_matrix, weights){
        predictions<-feature_matrix[[1]]%*%weights
        predictions
}
```

If we have a the values of a single input feature in an array 'feature' and the prediction 'errors' (predictions - output) then the derivative of the regression cost function with respect to the weight of 'feature' is just twice the dot product between 'feature' and 'errors'. Write a function that accepts a 'feature' array and 'error' array and returns the 'derivative'

```
compute_errors<-function(predictions, features_matrix){
        predictions-features_matrix[[2]]
}

feature_derivative <-function(features_matrix, errors){
        2*t(features_matrix[[1]])%*%errors
}
```

Now we will use our predict_output and feature_derivative to write a gradient descent function. Although we can compute the derivative for all the features simultaneously (the gradient) we will explicitly loop over the features individually for simplicity. Write a gradient descent function that does the following:

- Accepts a numpy feature_matrix 2D array, a 1D output array, an array of initial weights, a step size and a convergence tolerance.
- While not converged updates each feature weight by subtracting the step size times the derivative for that feature given the current weights
- At each step computes the magnitude/length of the gradient (square root of the sum of squared components)
- When the magnitude of the gradient is smaller than the input tolerance returns the final weight vector

```
regression_gradient_descent<-function(feature_matrix,
                                      initial_weights,
                                      step_size,
                                      tolerance){
        converged = FALSE
        weights = initial_weights
        while (converged==F){
                predictions <- predict_output(feature_matrix = features_matrix, weights = weights)
                errors <-compute_errors(predictions = predictions,
                                        features_matrix = features_matrix)
                derivatives <- feature_derivative(features_matrix = features_matrix, errors = errors)
                 # while not converged, update each weight individually:
                magnitude<-sqrt(t(derivatives)%*%derivatives)
                weights <- weights-step_size*derivatives
                if (magnitude<tolerance){
                converged = T
        }
        }
        weights
}
```

Now we will run the regression_gradient_descent function on some actual data. In particular we will use the gradient descent to estimate the model from Week 1 using just an intercept and slope. Use the following parameters:

- features: 'sqft_living'
- output: 'price'

- initial weights: -47000, 1 (intercept, sqft_living respectively)
- step_size = 7e-12
- tolerance = 2.5e7

```
features_matrix<-construct_features_matrix(features = list("sqft_living"),outputs = list("price"), data
weights<-regression_gradient_descent(feature_matrix = features_matrix, initial_weights = c(-47000, 1), s
round(weights,digits = 2)
```

```
##           [,1]
## [1,] -46999.89
## [2,]    281.91
```

Compare with the results of lm():

```
multiRegFit<-lm(formula = price~sqft_living, data = train_data)
summary(multiRegFit)$coefficients
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) -47116.0791 4923.344373  -9.569934 1.209547e-21
## sqft_living    281.9588    2.164055 130.291922 0.000000e+00
```

They seem close enough.

Quiz Question: What is the value of the weight for sqft_living – the second element of 'simple_weights' (rounded to 1 decimal place)? It is 281.9

Now build a corresponding 'test_simple_feature_matrix' and 'test_output' using test_data. Using 'test_simple_feature_matrix' and 'simple_weights' compute the predicted house prices on all the test data.

```
features_matrix<-construct_features_matrix(features = list("sqft_living"),outputs = list("price"), data
prices<-features_matrix[[1]]%*%weights
```

```
prices[1]
```

Quiz Question: What is the predicted price for the 1st house in the Test data set for model 1 (round to nearest dollar)?

```
## [1] 356134.4
```

Now compute RSS on all test data for this model. Record the value and store it for later

```
epsilon<- features_matrix[[2]]-prices
RSS1<-sum(epsilon^2)
sum(resid(multiRegFit)^2)
```

```
## [1] 1.201918e+15
```

Now we will use the gradient descent to fit a model with more than 1 predictor variable (and an intercept). Use the following parameters:

```
features_matrix<-construct_features_matrix(features = list("sqft_living", "sqft_living15"),outputs = lis
weights<-regression_gradient_descent(feature_matrix = features_matrix, initial_weights = c(-100000, 1, 
```

Use the regression weights from this second model (using sqft_living and sqft_living_15) and predict the outcome of all the house prices on the TEST data.

```
features_matrix<-construct_features_matrix(features = list("sqft_living", "sqft_living15"),outputs = lis
prices<-features_matrix[[1]]%*%weights
```

```
round(prices[1],0)
```

Quiz Question: What is the predicted price for the 1st house in the TEST data set for model 2 (round to nearest dollar)?

```
## [1] 366651
```

What is the actual price for the 1st house in the Test data set?

```
features_matrix[[2]][1,]
```

```
## [1] 310000
```

```
epsilon<- features_matrix[[2]]-prices
RSS2<-sum(epsilon^2)
c(RSS1, RSS2)
```

Quiz question: Which model (1 or 2) has lowest RSS on all of the TEST data?

```
## [1] 2.754000e+14 2.702634e+14
```