# ORF525, Assignment 1, Problem 4

## Data preparation

Let's load the data:

```
train.data <- read.csv("train.data.csv", header=TRUE)
test.data <- read.csv("test.data.csv", header=TRUE)
train.data$zipcode <- as.factor(train.data$zipcode)
test.data$zipcode <- as.factor(test.data$zipcode)
all.data = rbind(train.data, test.data)
```

Now we extract the train and test response - vectors $Y.train$ and $Y.test$:

```
Y.train = as.numeric(train.data[, "price"])
Y.test = as.numeric(test.data[, "price"])
N.train = length(Y.train)
N.test = length(Y.test)
```

Further, we create the features for questions (a) - (e). They will be stored in matrices $X.train$ and $X.test$:

```
X.all = matrix(0L, nrow=N.train+N.test, ncol=21)
colnames(X.all) = list("bedrooms", "bathrooms", "sqft_living", "sqft_lot",
                       "bedrooms_bathrooms", "bedrooms_sqft_living", "bedrooms_sqft_lot",
                       "bathrooms_sqft_living", "bathrooms_sqft_lot", "sqft_living_sqft_lot",
                       "Ind_view", "sqft_living_2",
                       "q10", "q20", "q30", "q40", "q50", "q60", "q70", "q80", "q90")
```

First four basic features:

```
X.all[ , 1:4] = as.numeric(as.matrix(all.data[ , c("bedrooms", "bathrooms",
                                                   "sqft_living", "sqft_lot")]))
```

Interaction terms:

```
X.all[ , 5] = all.data[, "bedrooms"] * all.data[, "bathrooms"]
X.all[ , 6] = all.data[, "bedrooms"] * all.data[, "sqft_living"]
X.all[ , 7] = all.data[, "bedrooms"] * all.data[, "sqft_lot"]
X.all[ , 8] = all.data[, "bathrooms"] * all.data[, "sqft_living"]
X.all[ , 9] = all.data[, "bathrooms"] * all.data[, "sqft_lot"]
X.all[ , 10] = 0.001*all.data[, "sqft_living"]*all.data[, "sqft_lot"] # to avoid overflow
```

Zipcode is treated as feature as well. But since it is a categorical variable (has no numerical meaning), we have to use dummy variables to encode all possible zipcodes (though linear model automatically does it with factors, we just do this as it may be of potential use in other problems):

```
library("fastDummies")
X.all = cbind(X.all, fastDummies::dummy_cols(all.data["zipcode"],
                                             select_columns = "zipcode",
                                             remove_first_dummy = TRUE))
X.all = X.all[ , -22]
```

Finally, we create more sophisticated features:

```
X.all[ , 11] = as.numeric(all.data["view"] == 0)
X.all[ , 12] = all.data[ , "sqft_living"]^2
```

```
q = quantile(train.data[, "sqft_living"], probs=0.1*(1:9))
for (i in 1:9) {
  X.all[ , 12+i] = pmax( all.data[, "sqft_living"] - q[i], 0)^2
}
```

Let's unite it all into train dataframe and test dataframe:

```
X.train = X.all[1:dim(train.data)[1], ]
X.test = X.all[(dim(train.data)[1]+1) : (dim(train.data)[1] + dim(test.data)[1]), ]
data_train = data.frame(price=Y.train, X.train)
data_test = data.frame(X.test)
```

Now we are ready to fit models based on different features.

### (a)

Let's fit the model based on the four basic features only:

```
fitted_a = lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot, data=data_train)
```

Now we predict response on the test data and compute the out-of-sample $R^2$:

```
Y.pred_a = predict(fitted_a, newdata=data_test)
R2_a = 1 - sum((Y.test - Y.pred_a)^2)/sum((Y.test - rep(mean(Y.train), N.test))^2)
sprintf("(a) Out-of-sample R^2 = %f", R2_a)
```

```
## [1] "(a) Out-of-sample R^2 = 0.505149"
```

So, our out-of-sample $R^2 = 0.505149$.

### (b)

Let's fit the model based on the four basic features and the interaction terms:

```
fitted_b = lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot
                        + bedrooms_bathrooms + bedrooms_sqft_living + bedrooms_sqft_lot
                        + bathrooms_sqft_living + bathrooms_sqft_lot
                        + sqft_living_sqft_lot, data=data_train)
```

Now we predict response on the test data and compute the out-of-sample $R^2$:

```
Y.pred_b = predict(fitted_b, newdata=data_test)
R2_b = 1 - sum((Y.test - Y.pred_b)^2)/sum((Y.test - rep(mean(Y.train), N.test))^2)
sprintf("(b) Out-of-sample R^2 = %f", R2_b)
```

```
## [1] "(b) Out-of-sample R^2 = 0.532896"
```

So, our out-of-sample $R^2 = 0.532896$.

### (c)

For the Gaussian Kernel it makes no sense to consider interaction terms, since the feature map of this kernel creates all possible interactions automatically, so we need to focus on the four basic features only. We standardize the data. Note that we compute the mean and the standard deviation of the data based on the train sample only, and then subtract the mean from and divide by the standard deviation both train and test samples:

```
meanX = rep(apply(X.train[ , 1:4], 2 , mean))
stdX = rep(apply(X.train[ , 1:4], 2, sd))

X.train_std = X.train[ , 1:4] - matrix(meanX, ncol=4, nrow=N.train, byrow=T)
X.train_std = as.matrix(X.train_std/matrix(stdX, ncol=4, nrow=N.train, byrow=T))

X.test_std = X.test[ , 1:4] - matrix(meanX, ncol=4, nrow=N.test, byrow=T)
X.test_std = as.matrix(X.test_std/matrix(stdX, ncol=4, nrow=N.test, byrow=T))
```

The following code creates a class for Gaussian Kernel Ridge Regression (GKRR). It has two methods: fit and predict.

```
GKRR = setRefClass("GKRR", fields = list(gamma="numeric",
                                         lambda = "numeric",
                                         X_train = "matrix",
                                         Y_train = "vector",
                                         X_test = "matrix",
                                         Y_pred = "vector",
                                         N_train = "numeric",
                                         N_test = "numeric",
                                         K = "matrix",
                                         alpha = "vector"))

GKRR$methods(
  fit = function(X, Y) {
    X_train <<- X
    Y_train <<- Y
    N_train <<- length(Y_train)
    XX = tcrossprod(X_train)
    X2 = matrix(diag(XX), N_train, N_train)
    K <<- exp(-gamma*(X2 + t(X2) - 2*XX))
    alpha <<- solve(K + diag(lambda, N_train), Y_train)
},
  predict = function(X) {
    X_test <<- X
    N_test <<- dim(X_test)[1]
    K_predict = exp(-gamma*( matrix(diag(tcrossprod(X_test)), N_test, N_train)
                           + t(matrix(diag(tcrossprod(X_train)), N_train, N_test))
                           - 2*tcrossprod(X_test, X_train)))
    Y_pred <<- K_predict %*% alpha
    return(Y_pred)
})
```

Let's perform the cross-validation (though not required, we just provide a code here: the ranges of 'gamm' and 'lambd' in the for loops below can be chosen to be some candidate sets).

```
N_folds=5
permutation = sample(N.train, replace=FALSE)

R2best = -10^9
lambda_best = NA
gamma_best = NA


for (gamm in c(0.1^2/2)) {
```

```
  for (lambd in 10^c(-1)) {
    model_GKRR = GKRR(gamma = gamm, lambda=lambd)
    R2cv = rep(0, N_folds)
    for (k in 1:N_folds) {
      index_test = permutation[(round((k-1)/N_folds*N.train) + 1) :
                                 (min(round(k/N_folds*N.train), N.train))]
      index_train = setdiff(1:N.train, index_test)

      N_index = length(index_train)
      Y.pred_cv = rep(0, length(index_test))

      for (i in 1:5) {
        sub_index = index_train[(round((i-1)/5*N_index) + 1) :
                                  min(round(i/5*N_index), N_index)]
        model_GKRR$fit(X.train_std[sub_index, ], Y.train[sub_index])

        Y.pred_cv = Y.pred_cv + model_GKRR$predict(X.train_std[index_test, ])/5
      }
      RSS = sum((Y.train[index_test] - Y.pred_cv)^2)
      TSS = sum((Y.train[index_test] - rep(mean(Y.train[-index_test]),
                                           length(index_test)))^2)
      R2cv[k] = 1 - RSS/TSS
    }
    if (mean(R2cv) > R2best) {
      R2best = mean(R2cv)
      gamma_best = gamm
      lambda_best = lambd
    }
  }
}
sprintf("Best out-of-sample R^2 achieved is %s", R2best)
```

```
## [1] "Best out-of-sample R^2 achieved is 0.559891419342661"
```

```
sprintf("Best gamma is %f", gamma_best)
```

```
## [1] "Best gamma is 0.005000"
```

```
sprintf("Best lambda is %f", lambda_best)
```

```
## [1] "Best lambda is 0.100000"
```

Now let's check the results of the optimal model on the test data:

```
model_GKRR = GKRR(gamma = gamma_best, lambda=lambda_best)

Y.pred_test = rep(0, N.test)
index_train = 1:N.train
for (i in 1:5) {
  sub_index = index_train[(round((i-1)/5*N.train) + 1) :
                            min(round(i/5*N.train), N.train)]
  model_GKRR$fit(X.train_std[sub_index, ], Y.train[sub_index])

  Y.pred_test = Y.pred_test + model_GKRR$predict(X.test_std)/5
}
```

```
rMSE = sqrt(mean((Y.test-Y.pred_test)^2))
sprintf("(c) rMSE = %f", rMSE)
```

```
## [1] "(c) rMSE = 246521.987500"
```

```
MADE = mean(abs(Y.test-Y.pred_test))
sprintf("(c) MADE = %f", MADE)
```

```
## [1] "(c) MADE = 164568.332421"
```

```
RSS = sum((Y.test - Y.pred_test)^2)
TSS = sum((Y.test - rep(mean(Y.train), N.test))^2)
R2_c = 1 - RSS/TSS
sprintf("(c) Out-of-sample R^2 = %f", R2_c)
```

```
## [1] "(c) Out-of-sample R^2 = 0.547292"
```

The Gaussian Kernel Ridge Regression outperforms the linear model from (b).

### (d)

Let's fit the linear model based on the four basic features, the interaction terms and the zipcode:

```
fitted_d = lm(price ~ ., data=data_train[ , -(12:22)])
```

Now we predict response on the test data and compute the out-of-sample $R^2$:

```
Y.pred_d = predict(fitted_d, newdata=data_test[ , -(11:21)])
R2_d = 1 - sum((Y.test - Y.pred_d)^2)/sum((Y.test - rep(mean(Y.train), N.test))^2)
sprintf("(d) Out-of-sample R^2 = %f", R2_d)
```

```
## [1] "(d) Out-of-sample R^2 = 0.761971"
```

So, our out-of-sample $R^2 = 0.761971$.

### (e)

Let's fit the model based on the four basic features, the interaction terms, the zipcode and additional features described in the problem statement:

```
fitted_e = lm(price ~ ., data=data_train)
```

Now we predict response on the test data and compute the out-of-sample $R^2$:

```
Y.pred_e = predict(fitted_e, newdata=data_test)
R2_e = 1 - sum((Y.test - Y.pred_e)^2)/sum((Y.test - rep(mean(Y.train), N.test))^2)
sprintf("(e) Out-of-sample R^2 = %f", R2_e)
```

```
## [1] "(e) Out-of-sample R^2 = 0.780476"
```

So, our out-of-sample $R^2 = 0.780476$.