Workflow

Aleix

Wednesday, July 02, 2014

Functions

General:

- Be explicit and descriptive with names, other people will read your code!
- Syntax: `myFunction_task` or `my.function_task`. The `_` separates the name from the task.
- `optL <- list()` is a list where you put all the parameters that you need. When put in the header `myFunction(..., optL = list(), ...)` mean that it is optional for the call.
- Each file should start with a small description of what it does
- Include the necessary packages for each function

Data transformations

- Function header: `myFunction_transform <- function(bikeData, optL = list())`
- Folder: 'transform'
- File name: 'myFunction__transform.R'
- Value: data.frame

Modeling functions

- Function header: `myFunction_model <- function(historicBikeData, optL = list())`
- Function header: `myFunction_predict <- function(toPredictionBikeData, historicModel)`
- Folder: 'model'
- File name: 'myFunction__model.R' (both functions in the same file)
- Value of myFunction__model: can be everything, including a list of objects
- Value of myFunction__predict: vector, matrix or data.frame

Cross validation functions and others

- Function header: `myFunction_cv <- function(do_model, do_prediction, do_transform, bikeData, optL = list(kFold = 5))`
- Folder: 'general'
- File name: 'myFunction__cv.R' (both functions in the same file)
- Value: a numeric vector

Example

Transformation : We complete the data set with expliciting time information

```r
require(lubdridate)
```

```
## Loading required package: lubdridate

## Warning: there is no package called 'lubdridate'
```

```
completeTime_transform <- function(bikeData){
  bikeData$datetime <- ymd_hms(bikeData$datetime)
  bikeData$day <- as.Date(bikeData$datetime)
  bikeData$hour <- hour(bikeData$datetime)
  bikeData$month <- month(bikeData$datetime)
  bikeData$weekDay <- wday(bikeData$datetime)

  return(bikeData)
}
```

Model : This model splits by categories in variables and computes the mean

```
require(plyr)
```

```
## Loading required package: plyr
```

```
categoryMeans_model <- function(historicBikeData, optL = list()){
  categoryModel <- ddply(historicBikeData, .(season, hour, workingday), function(splitDataSet){
    mean(splitDataSet$count)
  })
  colnames(categoryModel)[4] <- "mean"

  return(categoryModel)
}

categoryMeans_predict <- function(toPredictionBikeData, historicModel){
  apply(toPredictionBikeData, 1, function(rowData){
    obsSeason <- as.numeric(rowData['season'])
    obsHour <- as.numeric(rowData['hour'])
    obsWorkingDay <- as.numeric(rowData['workingday'])
    selecIndex <- historicModel$season == obsSeason & historicModel$hour == obsHour & historicModel$work
    foundCategory <- ( sum(selecIndex) > 0 )
    if( foundCategory ){
      predictionValue <- historicModel$mean[selecIndex]
    }else{
      predictionValue <- NA
    }
  })
}
```

Loss function : Kaggle's competition loss function

```
kaggle_loss <- function(realValues, predictedValues){
  sqrt( sum( na.omit( ( log( realValues + 1 ) - log( predictedValues + 1 ) )^2 ) ) )
}
```

Cross validation : This cross validation splits the data set randomly

```
random_cv <- function(do_model, do_prediction, do_transform, bikeData, optL = list(kFold = 5)){

  lengthSplit <- floor( nrow(bikeData)/optL$kFold )

  sapply(1:optL$kFold, function(cvIter){
    trainIndex <- sample(1:nrow(bikeData), size = lengthSplit, replace = FALSE)
    trainSet <- bikeData[ trainIndex, ]
    testSet <- bikeData[ -trainIndex, ]

    trainSet <- do_transform( trainSet )
    ## this is important to be done here instead out of the loop to ensure the good working of the cros
    newModel <- do_model( trainSet, optL )

    testSet <- do_transform( testSet )
    newPredictions <- do_prediction( testSet, newModel )
    realValues <- testSet$count

    return( kaggle_loss( realValues, newPredictions ) )
  })
}
```

Execution

```
bikeData <- read.csv('c:/Users/Aleix/Google Drive/Archivos/KaggleBike/train.csv', head = TRUE, stringsA

random_cv(categoryMeans_model, categoryMeans_predict, completeTime_transform, bikeData)
```

```
## [1] 49.44 49.71 49.91 49.66 50.05
```