zillow project — tree based model

YueWen

Missing data imputation

In previous work, we have finsihed missing data imputation, so all the data is complete, and here we import the imputated data directly from preveious work.

```
setwd("D:/data_camp/zillow_project")
#this csv is derived by imputation_part2.R
train <- read.csv('train_imputed2.csv', stringsAsFactors = F)
```

Feture engineering

However, regard to the feature engineering part. We made several changes to tailor the need for tree-based methods.

1) Change build year to numerical type, and impute the missing value by randomly sampling. The practice can be validated by two reasons: the first is that built-year is not strong related to other features, so we might don't want to use mice to impute it. The second is that for tree based method, built-year will have more power as numerical value.
2) For interacttion terms, we create it in this part because I don't know how to include them in the formula for tree based methods
3) For other feature engineering part, we use the same collpase techinique and generate the same new features as in what we did in linaer model. However, we will delete all the ananlysis part for generating new features, such as t-test, correlaton analysis.

Handle variable type

```
train <- train[,!names(train) %in%c("X.1","X","trans_year")]

variable_numeric = c("area_firstfloor_finished",
                     "area_base", "area_base_living",
                     "area_garage",
                     "area_live_finished",
                     "area_liveperi_finished",
                     "area_lot",
                     "area_patio",
                     "area_pool",
                     "area_shed",
                     "area_total_calc",
                     "area_total_finished",
                     "area_unknown",
                     "tax_building",
                     "tax_land",
                     "tax_property",
                     "tax_total",
                     "latitude",
                     "longitude")
```

```r
# discrete
variable_discrete = c("num_75_bath",
                      "num_bath",
                      "num_bathroom",
                      "num_bathroom_calc",
                      "num_bedroom",
                      "num_fireplace",
                      "num_garage",
                      "num_pool",
                      "num_room",
                      "num_story",
                      "num_unit")
variable_binary = c("flag_fireplace",
                    "flag_tub",
                    "flag_spa",
                    "flag_pool_spa",
                    "flag_pool_tub",
                    "tax_delinquency")

# categorical variable
variable_nominal = c("aircon",
                     "architectural_style",
                     "county",
                     "deck",
                     "framing",
                     "heating",
                     "id_parcel",
                     "material",
                     "region_city",
                     "region_county",
                     "region_neighbor",
                     "region_zip",
                     "story",
                     "zoning_landuse",
                     "zoning_landuse_county")
variable_ordinal = c("quality")

# date
variable_date = c("tax_year",
                  "build_year",
                  "tax_delinquency_year",
                  "trans_year",
                  "trans_month",
                  "trans_day",
                  "trans_date",
                  "trans_weekday")

# others
variable_unstruct = c("zoning_property")

# don't understand
variable_unknown = c('censustractandblock',
                     'rawcensustractandblock')
```

```
# Conversion
# - convert some binary to 0, 1
# - convert to date to int
# - convert to numeric to double
# - convert to discrete to int
# - convert to categorical to character
train[train$flag_fireplace == "", "flag_fireplace"] = 0
train[train$flag_fireplace == "true", "flag_fireplace"] = 1
train[train$flag_tub == "", "flag_tub"] = 0
train[train$flag_tub == "true", "flag_tub"] = 1
train[train$tax_delinquency == "", "tax_delinquency"] = 0
train[train$tax_delinquency == "Y", "tax_delinquency"] = 1
# convert to date to int
train[,variable_date[variable_date %in% names(train)]] =
  sapply(train[,variable_date[variable_date %in% names(train)]], as.character)
# convert to numeric to double
train[,variable_numeric[variable_numeric %in% names(train)]] =
  sapply(train[,variable_numeric[variable_numeric %in% names(train)]], as.numeric)
# convert to discrete to int
combine_int_col = c(variable_discrete, variable_binary)
train[,combine_int_col[combine_int_col %in% names(train)]] =
  sapply(train[,combine_int_col[combine_int_col %in% names(train)]], as.integer)
# convert to categorical to character
combine_char_col =  c(variable_nominal, variable_ordinal)
train[,combine_char_col[combine_char_col %in% names(train)]] =
  sapply(train[,combine_char_col[combine_char_col %in% names(train)]], as.character)
```

date related feature

roomcnt realted feature

equipment

tax

quality

geo_info

combine all the selected features

tree models

From here, we start to build tree models based on the missing value imputation part as well as the feature engineering part. This part is structured into three sections.

1) Basic tree exploartion
2) Random Forest
3) Boosting tree

Prepare work

This part we split train and test set, and then generate the formula. Let us take a look at the formula we generated.

```r
tree.df <- train[,selected.feature]

# train/test split
set.seed(500)
train.ind <- sample(nrow(tree.df),0.7*nrow(tree.df))
train.set <- tree.df[train.ind,]
test.set <- tree.df[-train.ind,]

##tree method
x_formula <- paste(names(train.set[,-1]),collapse = "+")
formula  <- as.formula(paste("logerror~",x_formula))
print(formula)
```

```
## logerror ~ county6111 + zip.density.revised + new.quality + tax_delinquency +
##     tax_total + tax_property + tax_land + right.tax + tax.per.area +
##     build.to.total + tax.perc + flag_fireplace + heating.and.tub +
##     new.heating + num_room + num_bedroom + num_bathroom + right_room +
##     bed_to_bath + are_per_room + zero_room + area_total_calc +
##     build_year
```

1. Simple tree method

```r
library(rpart)
# step1
# it takes a little bit long to run, save it and import it directly
#tree0 <- rpart(formula, data = train.set, control = rpart.control(cp= 0.0001))
#save("tree0",file = "tree0.RData")
load("tree0.RData")
printcp(tree0)
```

```
##
## Regression tree:
## rpart(formula = formula, data = train.set, control = rpart.control(cp = 1e-04))
##
## Variables actually used in tree construction:
##  [1] are_per_room        area_total_calc     bed_to_bath
##  [4] build.to.total      build_year          county6111
##  [7] heating.and.tub     new.heating         new.quality
## [10] num_bathroom        num_bedroom         num_room
## [13] tax.per.area        tax.perc            tax_delinquency
## [16] tax_land            tax_property        tax_total
## [19] zip.density.revised
##
## Root node error: 1554/63192 = 0.024592
##
## n= 63192
##
##             CP nsplit rel error  xerror     xstd
## 1   0.00303075      0   1.00000 1.00003 0.045200
```

```
## 2   0.00180874      1   0.99697 0.99780 0.045182
## 3   0.00129311      2   0.99516 0.99632 0.045185
## 4   0.00095938      3   0.99387 0.99761 0.045172
## 5   0.00089970      4   0.99291 0.99916 0.045171
## 6   0.00077875      5   0.99201 0.99988 0.045178
## 7   0.00069258      6   0.99123 1.00331 0.045201
## 8   0.00067421      7   0.99054 1.00776 0.045207
## 9   0.00064016      8   0.98986 1.00925 0.045208
## 10  0.00059965     12   0.98730 1.01442 0.045276
## 11  0.00057022     18   0.98338 1.01772 0.045287
## 12  0.00056476     19   0.98281 1.01847 0.045282
## 13  0.00053323     37   0.97221 1.02059 0.045406
## 14  0.00053279     44   0.96848 1.02106 0.045409
## 15  0.00051334     46   0.96741 1.02246 0.045417
## 16  0.00047396     60   0.95963 1.02342 0.045412
## 17  0.00045660     61   0.95916 1.02604 0.045413
## 18  0.00044937     69   0.95535 1.02669 0.045461
## 19  0.00044505     71   0.95445 1.02724 0.045463
## 20  0.00043782     73   0.95356 1.02749 0.045465
## 21  0.00043554     75   0.95268 1.02834 0.045463
## 22  0.00043254     76   0.95225 1.02855 0.045464
## 23  0.00042918     77   0.95182 1.02860 0.045464
## 24  0.00042776     79   0.95096 1.02860 0.045464
## 25  0.00042690     80   0.95053 1.02877 0.045465
## 26  0.00041517     85   0.94840 1.02909 0.045465
## 27  0.00040840     87   0.94756 1.03209 0.045478
## 28  0.00040493     91   0.94584 1.03227 0.045478
## 29  0.00038180     93   0.94503 1.03427 0.045463
## 30  0.00037341     96   0.94389 1.04080 0.045470
## 31  0.00035912    101   0.94202 1.04203 0.045438
## 32  0.00035485    104   0.94094 1.04390 0.045434
## 33  0.00034727    105   0.94059 1.04545 0.045471
## 34  0.00034629    106   0.94024 1.04779 0.045479
## 35  0.00034511    116   0.93627 1.04797 0.045480
## 36  0.00031725    118   0.93557 1.05139 0.045482
## 37  0.00031568    119   0.93526 1.05563 0.045495
## 38  0.00031158    120   0.93494 1.05732 0.045500
## 39  0.00030689    121   0.93463 1.05791 0.045495
## 40  0.00030525    128   0.93248 1.05905 0.045497
## 41  0.00030477    129   0.93218 1.05918 0.045497
## 42  0.00030221    132   0.93126 1.05924 0.045494
## 43  0.00030215    134   0.93066 1.05922 0.045495
## 44  0.00029952    135   0.93036 1.05956 0.045496
## 45  0.00029865    144   0.92766 1.05961 0.045496
## 46  0.00029727    149   0.92614 1.05995 0.045499
## 47  0.00029408    150   0.92584 1.06018 0.045497
## 48  0.00029017    155   0.92437 1.06033 0.045498
## 49  0.00028821    156   0.92408 1.06131 0.045492
## 50  0.00027018    158   0.92351 1.06482 0.045494
## 51  0.00026870    159   0.92324 1.06847 0.045502
## 52  0.00026781    161   0.92270 1.06871 0.045502
## 53  0.00026369    163   0.92216 1.06946 0.045496
## 54  0.00026284    166   0.92137 1.07085 0.045505
## 55  0.00026044    171   0.92001 1.07124 0.045519
```

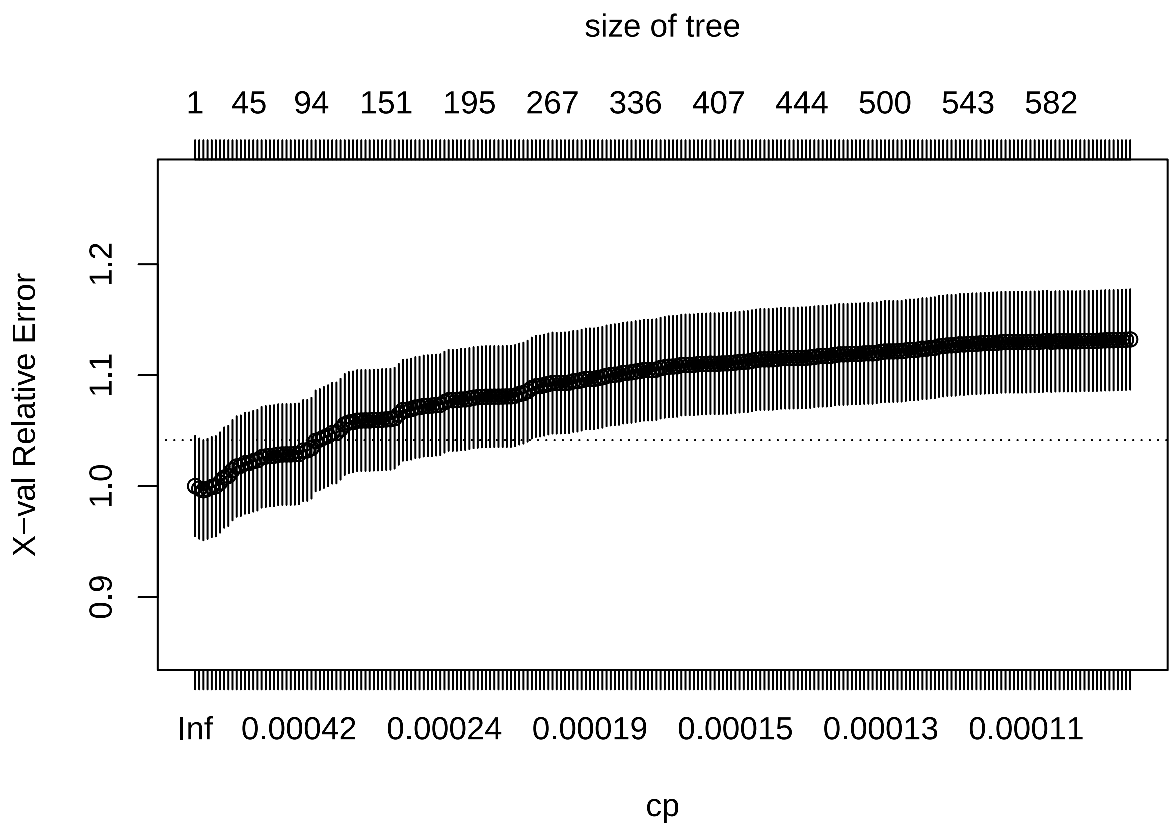
```
## 56  0.00025500    172   0.91975 1.07227 0.045517
## 57  0.00025411    173   0.91950 1.07247 0.045509
## 58  0.00025309    174   0.91924 1.07266 0.045509
## 59  0.00025059    175   0.91899 1.07309 0.045505
## 60  0.00024953    176   0.91874 1.07339 0.045505
## 61  0.00023998    179   0.91799 1.07573 0.045525
## 62  0.00023888    181   0.91751 1.07745 0.045569
## 63  0.00023841    182   0.91727 1.07740 0.045569
## 64  0.00023720    184   0.91679 1.07749 0.045569
## 65  0.00023607    186   0.91632 1.07815 0.045571
## 66  0.00023537    193   0.91466 1.07823 0.045571
## 67  0.00023335    194   0.91443 1.07893 0.045568
## 68  0.00023040    197   0.91373 1.07972 0.045568
## 69  0.00023020    199   0.91327 1.08015 0.045571
## 70  0.00022502    202   0.91258 1.08045 0.045529
## 71  0.00022480    206   0.91168 1.08068 0.045527
## 72  0.00022401    208   0.91123 1.08068 0.045527
## 73  0.00022339    219   0.90876 1.08072 0.045522
## 74  0.00022274    221   0.90832 1.08072 0.045521
## 75  0.00022259    234   0.90542 1.08079 0.045522
## 76  0.00022141    235   0.90520 1.08088 0.045521
## 77  0.00022114    236   0.90498 1.08093 0.045521
## 78  0.00021907    237   0.90475 1.08162 0.045528
## 79  0.00021840    238   0.90454 1.08300 0.045538
## 80  0.00021565    241   0.90388 1.08387 0.045536
## 81  0.00020948    243   0.90345 1.08565 0.045539
## 82  0.00020860    244   0.90324 1.08844 0.045546
## 83  0.00020214    245   0.90303 1.08998 0.045543
## 84  0.00020207    246   0.90283 1.09029 0.045539
## 85  0.00020060    248   0.90242 1.09110 0.045539
## 86  0.00020011    249   0.90222 1.09211 0.045543
## 87  0.00019575    266   0.89849 1.09297 0.045542
## 88  0.00019533    267   0.89830 1.09285 0.045475
## 89  0.00019466    269   0.89791 1.09296 0.045476
## 90  0.00019409    276   0.89653 1.09314 0.045476
## 91  0.00019234    277   0.89634 1.09340 0.045477
## 92  0.00019045    278   0.89614 1.09446 0.045476
## 93  0.00019040    281   0.89557 1.09469 0.045475
## 94  0.00018747    283   0.89519 1.09562 0.045477
## 95  0.00018647    288   0.89425 1.09668 0.045443
## 96  0.00018635    293   0.89332 1.09680 0.045443
## 97  0.00018576    295   0.89295 1.09679 0.045443
## 98  0.00018364    301   0.89179 1.09748 0.045445
## 99  0.00018327    302   0.89160 1.09816 0.045457
## 100 0.00018209    312   0.88974 1.09940 0.045445
## 101 0.00018034    313   0.88955 1.10015 0.045445
## 102 0.00017936    315   0.88919 1.10050 0.045446
## 103 0.00017854    317   0.88883 1.10078 0.045445
## 104 0.00017734    320   0.88830 1.10194 0.045449
## 105 0.00017676    321   0.88812 1.10226 0.045447
## 106 0.00017271    334   0.88551 1.10262 0.045447
## 107 0.00017089    335   0.88534 1.10329 0.045452
## 108 0.00017007    350   0.88245 1.10388 0.045451
## 109 0.00016909    351   0.88228 1.10456 0.045454
```

```
## 110 0.00016833    352   0.88211 1.10466 0.045455
## 111 0.00016743    356   0.88144 1.10471 0.045455
## 112 0.00016529    364   0.88010 1.10512 0.045447
## 113 0.00016473    365   0.87994 1.10646 0.045449
## 114 0.00016346    366   0.87977 1.10714 0.045448
## 115 0.00016271    367   0.87961 1.10773 0.045449
## 116 0.00016227    369   0.87928 1.10786 0.045452
## 117 0.00016115    370   0.87912 1.10805 0.045452
## 118 0.00016034    372   0.87880 1.10916 0.045453
## 119 0.00015963    375   0.87831 1.10931 0.045453
## 120 0.00015952    384   0.87688 1.10939 0.045453
## 121 0.00015934    385   0.87672 1.10939 0.045453
## 122 0.00015728    394   0.87519 1.10990 0.045453
## 123 0.00015663    395   0.87503 1.11016 0.045452
## 124 0.00015605    403   0.87376 1.11017 0.045452
## 125 0.00015504    404   0.87360 1.11047 0.045452
## 126 0.00015450    405   0.87344 1.11036 0.045435
## 127 0.00015409    406   0.87329 1.11040 0.045435
## 128 0.00015395    407   0.87314 1.11068 0.045437
## 129 0.00015379    408   0.87298 1.11068 0.045437
## 130 0.00015352    409   0.87283 1.11102 0.045446
## 131 0.00015248    414   0.87206 1.11145 0.045446
## 132 0.00015247    415   0.87191 1.11180 0.045447
## 133 0.00015144    416   0.87176 1.11216 0.045447
## 134 0.00014951    418   0.87145 1.11232 0.045448
## 135 0.00014649    419   0.87130 1.11312 0.045448
## 136 0.00014539    421   0.87101 1.11379 0.045457
## 137 0.00014521    422   0.87086 1.11428 0.045457
## 138 0.00014515    424   0.87057 1.11428 0.045457
## 139 0.00014496    432   0.86935 1.11428 0.045457
## 140 0.00014360    433   0.86920 1.11435 0.045458
## 141 0.00014227    434   0.86906 1.11481 0.045458
## 142 0.00014211    435   0.86891 1.11530 0.045462
## 143 0.00014180    437   0.86863 1.11540 0.045462
## 144 0.00014171    438   0.86849 1.11539 0.045462
## 145 0.00014165    439   0.86835 1.11550 0.045462
## 146 0.00014106    441   0.86806 1.11552 0.045462
## 147 0.00014066    443   0.86778 1.11572 0.045462
## 148 0.00014055    445   0.86750 1.11580 0.045462
## 149 0.00013827    447   0.86722 1.11608 0.045460
## 150 0.00013791    448   0.86708 1.11659 0.045457
## 151 0.00013678    450   0.86681 1.11709 0.045458
## 152 0.00013672    451   0.86667 1.11722 0.045458
## 153 0.00013598    453   0.86640 1.11723 0.045455
## 154 0.00013493    456   0.86599 1.11759 0.045452
## 155 0.00013345    458   0.86572 1.11833 0.045459
## 156 0.00013332    460   0.86545 1.11879 0.045478
## 157 0.00013319    461   0.86532 1.11879 0.045478
## 158 0.00013259    462   0.86518 1.11886 0.045433
## 159 0.00013225    474   0.86357 1.11921 0.045433
## 160 0.00013173    476   0.86331 1.11925 0.045433
## 161 0.00013138    477   0.86317 1.11934 0.045434
## 162 0.00013133    483   0.86239 1.11969 0.045445
## 163 0.00013082    484   0.86225 1.11976 0.045445
```

```
## 164 0.00013040    486   0.86199 1.11970 0.045450
## 165 0.00013003    487   0.86186 1.12016 0.045449
## 166 0.00012748    495   0.86082 1.12097 0.045454
## 167 0.00012714    499   0.86031 1.12142 0.045454
## 168 0.00012711    500   0.86019 1.12146 0.045454
## 169 0.00012696    501   0.86006 1.12146 0.045454
## 170 0.00012680    505   0.85955 1.12155 0.045455
## 171 0.00012643    509   0.85904 1.12167 0.045455
## 172 0.00012445    510   0.85892 1.12228 0.045461
## 173 0.00012441    514   0.85842 1.12283 0.045458
## 174 0.00012439    515   0.85829 1.12283 0.045458
## 175 0.00012299    516   0.85817 1.12320 0.045457
## 176 0.00012297    519   0.85780 1.12398 0.045461
## 177 0.00012230    520   0.85768 1.12395 0.045461
## 178 0.00012141    523   0.85731 1.12460 0.045462
## 179 0.00012123    524   0.85719 1.12482 0.045462
## 180 0.00011973    525   0.85707 1.12595 0.045464
## 181 0.00011938    526   0.85695 1.12627 0.045465
## 182 0.00011863    527   0.85683 1.12684 0.045462
## 183 0.00011848    529   0.85659 1.12695 0.045461
## 184 0.00011630    531   0.85636 1.12705 0.045458
## 185 0.00011473    532   0.85624 1.12787 0.045470
## 186 0.00011403    538   0.85555 1.12769 0.045440
## 187 0.00011308    542   0.85509 1.12805 0.045440
## 188 0.00011298    543   0.85498 1.12838 0.045440
## 189 0.00011272    544   0.85487 1.12838 0.045440
## 190 0.00011152    545   0.85476 1.12868 0.045446
## 191 0.00011135    546   0.85464 1.12888 0.045450
## 192 0.00011092    547   0.85453 1.12901 0.045450
## 193 0.00011060    548   0.85442 1.12920 0.045452
## 194 0.00011059    549   0.85431 1.12933 0.045452
## 195 0.00010940    550   0.85420 1.12969 0.045450
## 196 0.00010906    552   0.85398 1.12984 0.045451
## 197 0.00010886    556   0.85355 1.12987 0.045451
## 198 0.00010869    563   0.85278 1.12978 0.045450
## 199 0.00010861    564   0.85267 1.12978 0.045450
## 200 0.00010690    565   0.85257 1.12976 0.045450
## 201 0.00010686    567   0.85235 1.12979 0.045447
## 202 0.00010632    568   0.85225 1.13000 0.045447
## 203 0.00010629    569   0.85214 1.13007 0.045446
## 204 0.00010589    573   0.85171 1.13017 0.045446
## 205 0.00010518    574   0.85161 1.13043 0.045446
## 206 0.00010470    575   0.85150 1.13081 0.045444
## 207 0.00010423    581   0.85074 1.13020 0.045194
## 208 0.00010407    582   0.85063 1.13048 0.045195
## 209 0.00010383    583   0.85053 1.13057 0.045195
## 210 0.00010378    584   0.85043 1.13060 0.045195
## 211 0.00010374    585   0.85032 1.13054 0.045195
## 212 0.00010370    592   0.84960 1.13055 0.045195
## 213 0.00010337    597   0.84905 1.13042 0.045195
## 214 0.00010295    601   0.84861 1.13080 0.045197
## 215 0.00010282    602   0.84851 1.13088 0.045197
## 216 0.00010270    608   0.84788 1.13088 0.045197
## 217 0.00010262    613   0.84736 1.13091 0.045197
```

```
## 218 0.00010174    616   0.84705 1.13109 0.045203
## 219 0.00010170    617   0.84695 1.13127 0.045202
## 220 0.00010153    621   0.84653 1.13145 0.045205
## 221 0.00010150    622   0.84643 1.13148 0.045205
## 222 0.00010146    623   0.84633 1.13152 0.045205
## 223 0.00010146    625   0.84613 1.13168 0.045206
## 224 0.00010129    627   0.84592 1.13192 0.045207
## 225 0.00010099    629   0.84572 1.13205 0.045207
## 226 0.00010000    630   0.84562 1.13225 0.045206
```

```
plotcp(tree0)
```

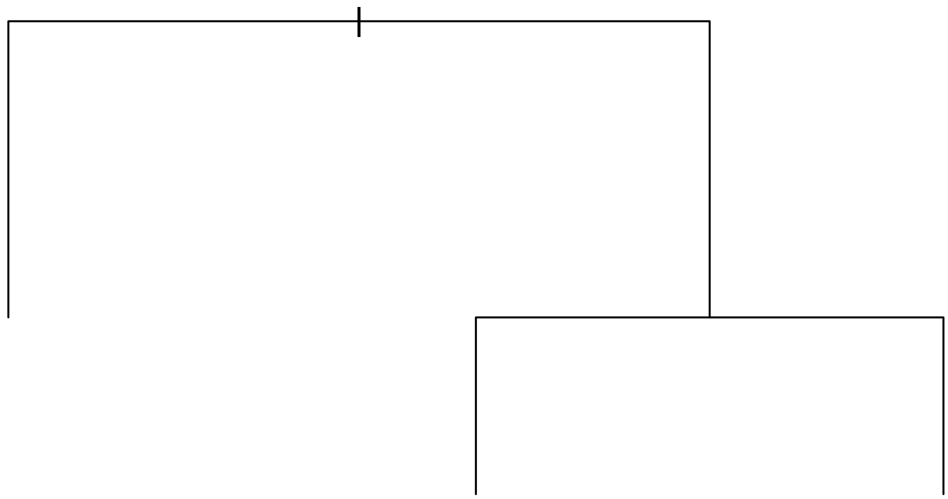



```
#step  2: pick up the tree size that minimizes the c-v error
#we can see cross  validation error increases first and then decreases
# therefore, let us choose the best control paramter
bestcp <- tree0$cptable[which.min(tree0$cptable[,"xerror"]),"CP"]

#step 3: prune the tree with best cp
tree0.pruned <- prune(tree0,cp = bestcp)

#calculate mse
test.pred <- predict(tree0.pruned,newdata = test.set)
tree0.mse <- sum((test.pred - test.set$logerror)^2)/length(test.pred)
print(tree0.mse)
```

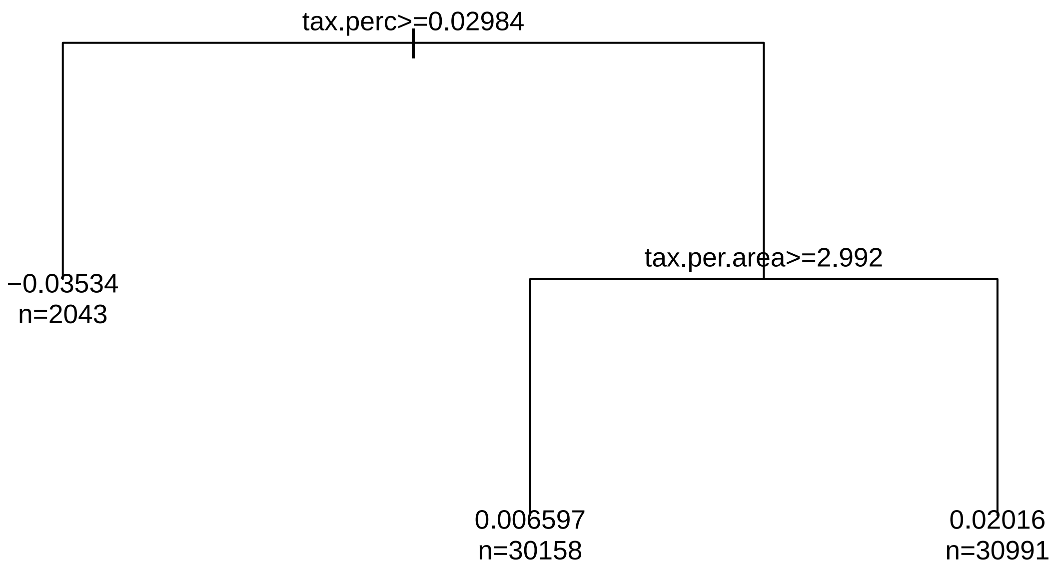
```
## [1] 0.02899413
```

The result is slightly better than random guessing. Let us plot the tree and then try other method to see whether it improved the result.

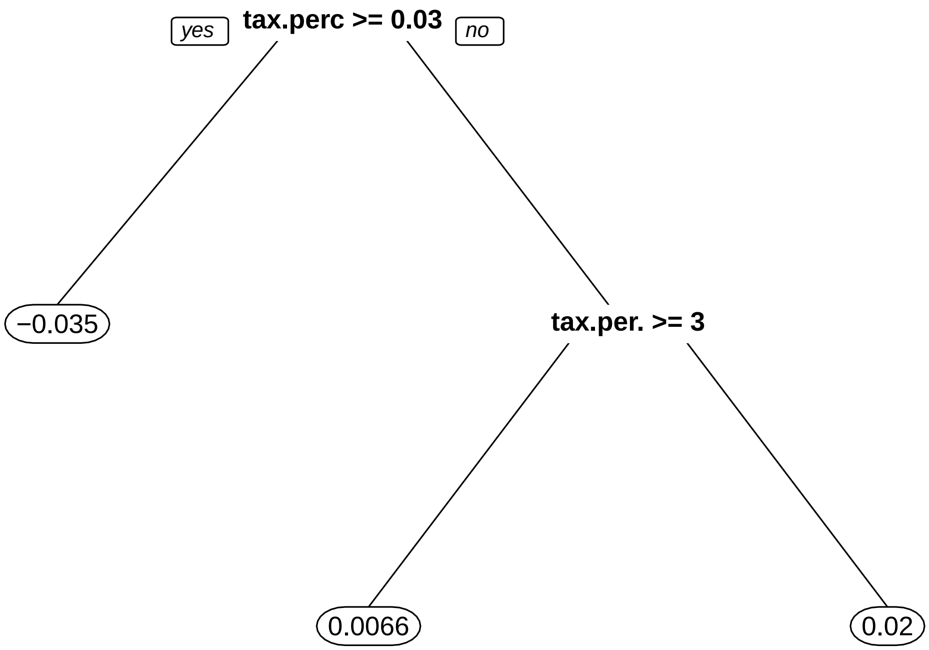
```r
plot(tree0.pruned)
```



```r
plot(tree0.pruned, uniform = T)
text(tree0.pruned, cex = 0.8, use.n = TRUE, xpd = TRUE)
```




```
library(rpart.plot)
prp(tree0.pruned, faclen = 0, cex = 0.8)
```



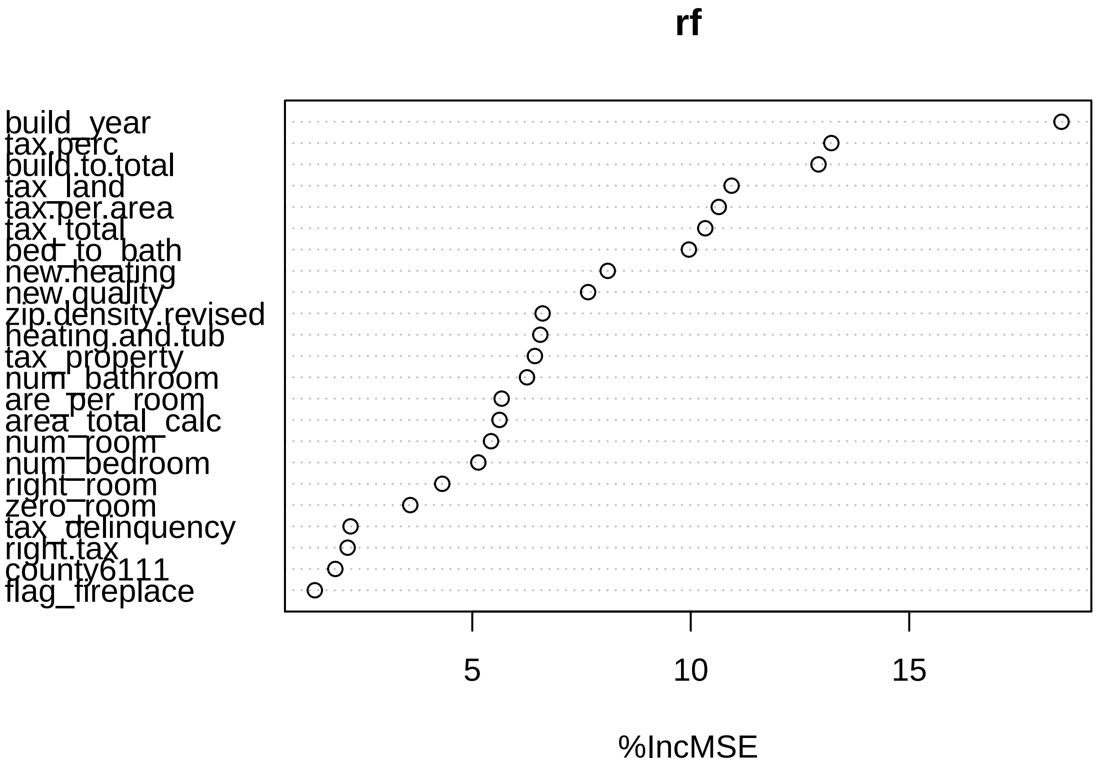
We can see the tree only splited once in this case.

2.Random Forest

```
library(randomForest)
#we can use the same formula we used before
#just a reminder of what we features we selected
factor.feature <- c("new.quality","new.heating")
train.set$new.heating <- as.factor(train.set$new.heating)
train.set$new.quality <- as.factor(train.set$new.quality)
test.set$new.heating <- as.factor(test.set$new.heating)
test.set$new.quality <- as.factor(test.set$new.quality)
```

we use the default ntree = 50 for now to see the initial result and it takes forever to run, we save it as Rdata and import it directly

```
# rf <- randomForest(formula,data = train.set,importance = TRUE, ntree = 50)
# save("rf",file="rf.RData")
load("rf.RData")
varImpPlot(rf,type = 1)
```



Let us interpret the result of random forest, we only use the increase mse plot.

As we can see, build_year is the most important feature here, and tax.percentage is still an important feature, as showned in the pruned tree.

This is different from linear model. Flag_fireplace is still not important features, which is the same as in linear model.

We pick up the most important two features, to see how they influence the logerror. It took too long to run as as well, we save it as image and import them directly.

```
#partialPlot(rf, train.set, eval('build_year'), xlab='build_year')
```

The plots are very hard to interpret, but basically we can see there is not linear relationship with the response, it seemes more like a quadratic relationship.

```
plot(rf) # see oob error
```

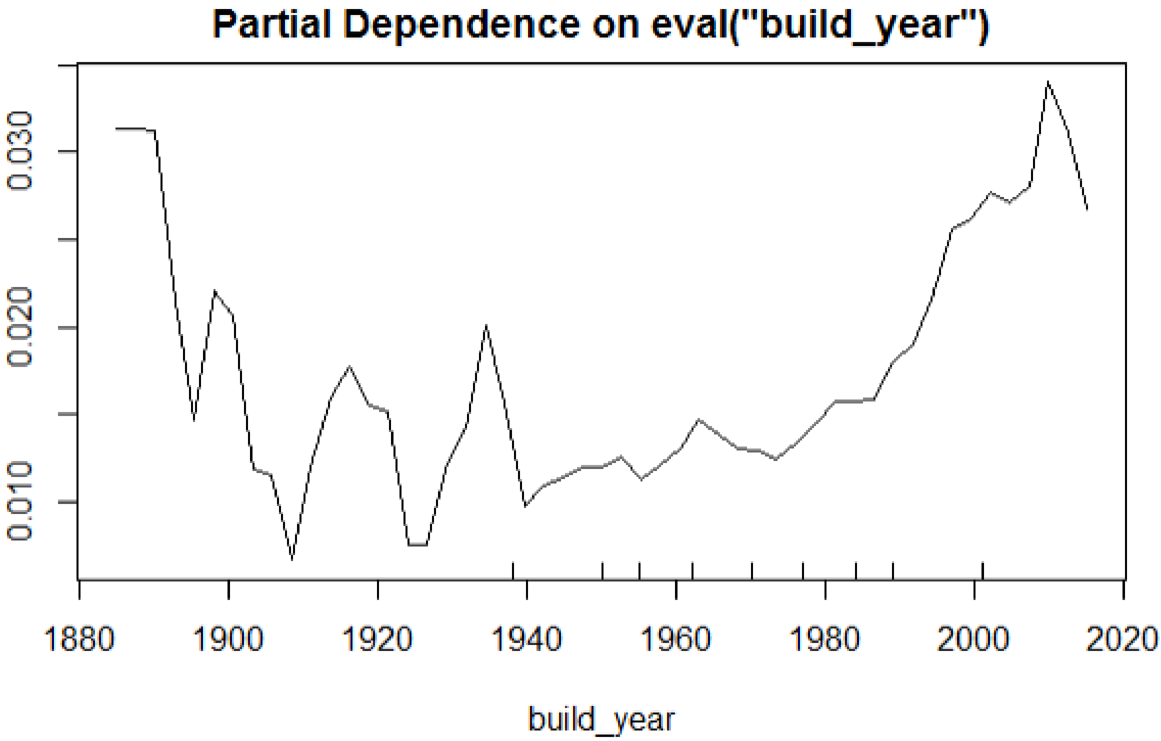
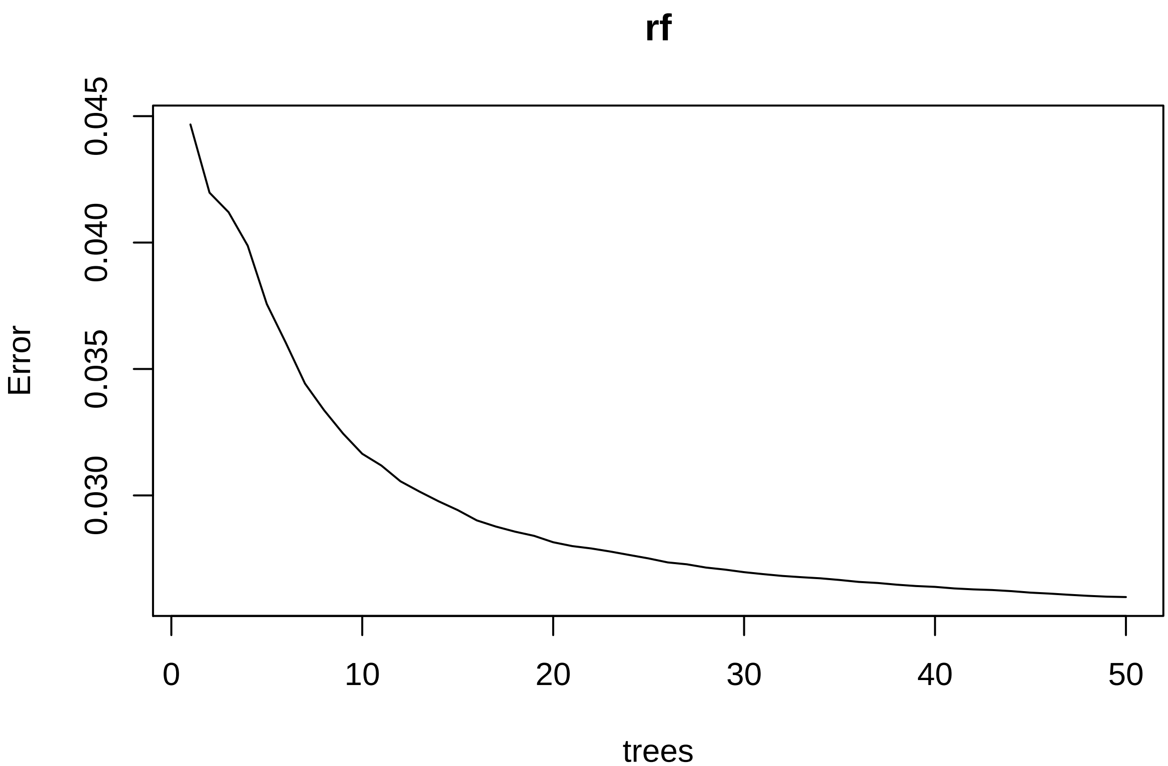


Figure 1: partial plot



We can see the out of bag error decreases as the number of trees increase, and 30 seems to be a knee point for this curve, which make it a reasonable numbe for the number of tree.

Then let us calculate MSE for rfmodel.

```
rf.pred <- predict(rf,test.set)
rf.mse <- mean((rf.pred- test.set$logerror)^2)
print(rf.mse)
```

```
## [1] 0.02934048
```

rf.mse is even worse then the base tree model, which is very disappointing given the fact that it took almost one hour to run. However, let us move on to xgboosting for now.

3.xgboosting

This part, we are trying to build a simple model without selecting paramters first, later we will perform grid search.

```
library(xgboost)
xgboost.train.label <- train.set$logerror
xgboost.test.label <- test.set$logerror

xgboost.feature.matrix <- model.matrix(~.,data = train.set[,-1])
set.seed(1)

# gbt <- xgboost(data = xgboost.feature.matrix,
#                 label = xgboost.train.label,
#                 max_depth = 10,
#                 nround = 50,
#                 objective = "reg:linear",
#                 verbose = 2)
#save("gbt",file = "gbt.RData")
load("gbt.RData")
```

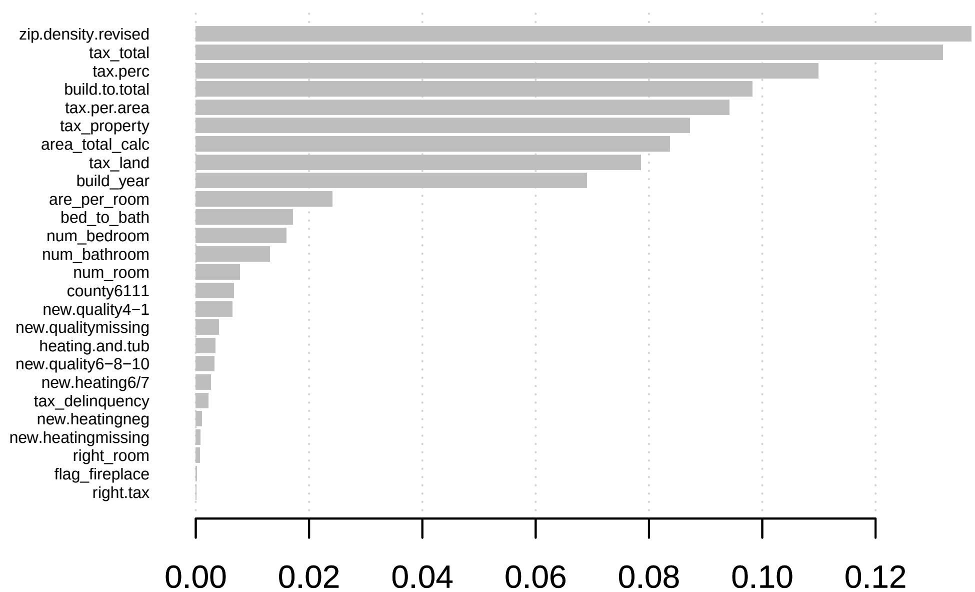
Then let us take a look at importance

```
importance <- xgb.importance(feature_names = colnames(xgboost.feature.matrix), model = gbt)
importance
```

```
##                Feature         Gain        Cover    Frequency
##  1: zip.density.revised 0.1369063619 5.326042e-02 0.1848919099
##  2:           tax_total 0.1318874512 1.261172e-01 0.1317753407
##  3:            tax.perc 0.1099344190 1.145855e-01 0.0907291265
##  4:      build.to.total 0.0983079905 1.193796e-01 0.0886794436
##  5:        tax.per.area 0.0942087446 1.214888e-01 0.0871728391
##  6:        tax_property 0.0872226960 1.162365e-01 0.0773799096
##  7:     area_total_calc 0.0837420771 1.036452e-01 0.0668336779
##  8:            tax_land 0.0785775901 9.589073e-02 0.0720542378
##  9:          build_year 0.0690245139 4.820395e-02 0.0716337900
## 10:        are_per_room 0.0240932972 4.415481e-02 0.0194982657
## 11:         bed_to_bath 0.0171861467 1.275367e-02 0.0229494412
## 12:         num_bedroom 0.0160503911 8.900394e-03 0.0178515119
## 13:        num_bathroom 0.0131046640 9.101827e-03 0.0111944221
## 14:            num_room 0.0077820457 8.516960e-03 0.0067797204
## 15:          county6111 0.0067701544 2.176795e-03 0.0127711012
```

```
## 16:      new.quality4-1 0.0065035936 6.394732e-04 0.0123856908
## 17:  new.qualitymissing 0.0040632521 3.795140e-04 0.0068673137
## 18:     heating.and.tub 0.0034586240 2.797161e-04 0.0047300375
## 19:   new.quality6-8-10 0.0032806133 2.276838e-03 0.0024000561
## 20:      new.heating6/7 0.0026608483 2.326298e-03 0.0048701867
## 21:     tax_delinquency 0.0022489580 6.838806e-03 0.0033285449
## 22:      new.heatingneg 0.0011270970 1.360231e-03 0.0006832276
## 23:  new.heatingmissing 0.0008124042 1.001653e-03 0.0010861568
## 24:          right_room 0.0007478454 1.056339e-04 0.0009635262
## 25:      flag_fireplace 0.0001830044 3.760377e-04 0.0003678918
## 26:           right.tax 0.0001152164 3.420828e-06 0.0001226306
##                Feature         Gain        Cover    Frequency
```

```r
library(Ckmeans.1d.dp)
xgb.plot.importance(importance)
```

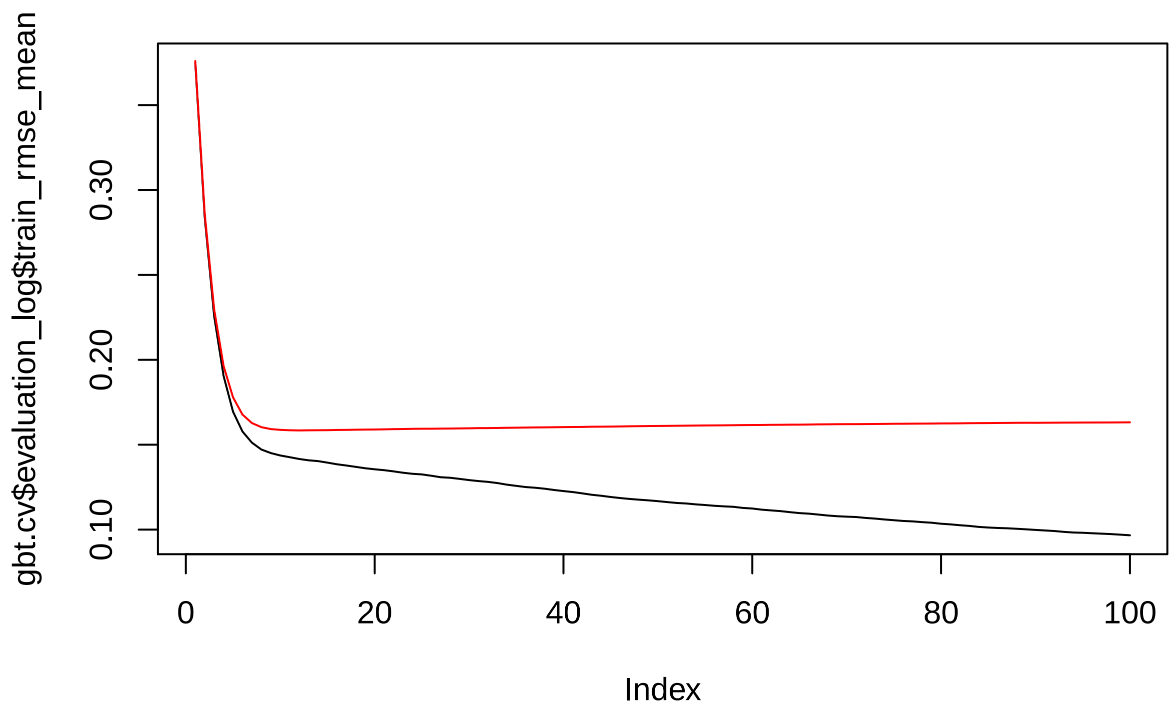


We can see, here, we still use gain to measure the importance feature. tax_toal, zip.density.rivised, tax.perc are important features. And as we recall in the random forest model, they are important features there as well, just the order of importance might change a little bit here.

Then let us choose parameters to find the optimal number of tree.

```r
par <- list( max_depth = 8,
             objective = "reg:linear",
             nthread = 3,
             verbose = 2)
# gbt.cv <- xgb.cv(params = par,
#                  data = xgboost.feature.matrix, label = xgboost.train.label,
```

```r
#                  nfold = 5, nrounds = 100)
# save("gbt.cv",file  =  "gbt.cv.RData")
load("gbt.cv.RData")
plot(gbt.cv$evaluation_log$train_rmse_mean, type = 'l')
lines(gbt.cv$evaluation_log$test_rmse_mean, col = 'red')
```



We can see when the number of tree is bigger than 10, the performance of the model does not improve anymore. Again, we can see the bias-variance trade off here.

```r
nround = which(gbt.cv$evaluation_log$test_rmse_mean == min(gbt.cv$evaluation_log$test_rmse_mean))
print(paste("the best number of  tree is : ",nround))
```

```
## [1] "the best number of  tree is :  12"
```

```r
# then let us fit the model again using the best parameter when n =12
best.tree.gbt <- xgboost(data = xgboost.feature.matrix,
               label = xgboost.train.label,
               nround = nround,
               params = par)
```

```
## [1]  train-rmse:0.375328
## [2]  train-rmse:0.284299
## [3]  train-rmse:0.226429
## [4]  train-rmse:0.191261
## [5]  train-rmse:0.170541
## [6]  train-rmse:0.159522
## [7]  train-rmse:0.152816
## [8]  train-rmse:0.148442
```

```
## [9]  train-rmse:0.146300
## [10] train-rmse:0.144976
## [11] train-rmse:0.144295
## [12] train-rmse:0.143876
```

```r
# to compare with other methods, let us calculate the mse
best.tree.gbt.pred <- predict(best.tree.gbt, model.matrix(~.,data = test.set[,-1]))
best.tree.gbt.mse  <- mean((best.tree.gbt.pred - test.set$logerror)^2)
print(best.tree.gbt.mse)
```

```
## [1] 0.02903037
```

The error is slightly smaller than random forest, bigger than simple tree method, which is very disappointig again.

However, it is not sufficient only to choose only one number of trees, let us use grid search to search the optimal parameters.

grid searching for boosting

Noticed that here rmse is used to choose the best parameter.

```r
# all_param = NULL
# all_test_rmse = NULL
# all_train_rmse = NULL
#
# ######  Takes too long to run, take it out! only save the best paramter
# for (iter in 1:20) {
#   print(paste("-------", iter,"-------"))
#
#   param <- list(objective = "reg:linear",
#                 max_depth = sample(5:12, 1),
#                 subsample = runif(1, .6, .9),
#                 colsample_bytree = runif(1, .5, .8),
#                 eta = runif(1, .01, .3),
#                 gamma = runif(1, 0.0, 0.2),
#                 min_child_weight = sample(1:40, 1),
#                 max_delta_step = sample(1:10, 1)
#   )
#   cv.nround = 30
#   cv.nfold = 5
#   seed.number = sample.int(10000, 1)[[1]]
#   set.seed(seed.number)
#   mdcv <- xgb.cv(data=xgboost.feature.matrix,
#                  label = xgboost.train.label,
#                  params = param,
#                  nfold=cv.nfold,
#                  nrounds=cv.nround,
#                  #metrics = "mae",
#                  early_stopping_rounds = 10,
#                  maximize=FALSE)
#   min_train_rmse = min(mdcv$evaluation_log$train_rmse_mean)
#   min_test_rmse = min(mdcv$evaluation_log$test_rmse_mean)
#
#   all_param <- rbind(all_param, unlist(param)[-1])
#   all_train_rmse <- c(all_train_rmse, min_train_rmse)
```

```
#   all_test_rmse <- c(all_test_rmse, min_test_rmse)
# }
#
# all_param <- as.data.frame(all_param)
# save("all_param", file = "all_param.RData")
# save("all_train_rmse", file = "all_train_rmse.RData")
# save("all_test_rmse", file = "all_test_rmse.RData")

load("all_param.RData")
load("all_train_rmse.RData")
load("all_test_rmse.RData")
best_param <- all_param[which(all_test_rmse == min(all_test_rmse)), ]
# grid.search.gbt <- xgboost(data =  xgboost.feature.matrix,
#                 label = xgboost.train.label,
#                 params = best_param,
#                 nrounds=100,
#                 early_stopping_rounds = 10,
#                 maximize = FALSE)
# save("grid.search.gbt",file = "grid.search.gbt.RData")
load("grid.search.gbt.RData")
# prediction
grid.search.pred <- predict(grid.search.gbt, model.matrix(~.,data = test.set[,-1]))
boost.mse <- mean((grid.search.pred - test.set$logerror)^2)
print(boost.mse)
```

```
## [1] 0.02891494
```

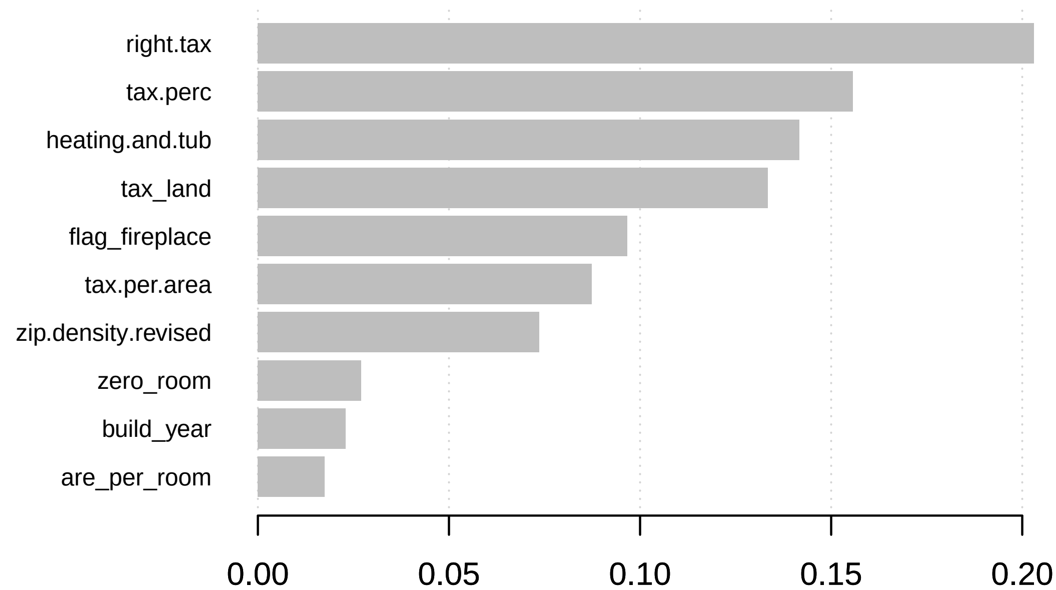
As we can see, our model is now slightlt better than simple tree model, then let us attempt to improve it.

We keep the same parameter as before to save time…

improvement attempt 1 : reduce useless features

Let us take a look at the importance, and then drop the redundant feature to see whether the results are improved.

```
importance <- xgb.importance(feature_names = colnames(train.set), model = grid.search.gbt)
xgb.plot.importance(importance, top_n = 10)
trunc.feature <- xgb.plot.importance(importance, top_n = 10)$Feature
```




```r
mod.data <- tree.df[,trunc.feature]

mod.data.train <- mod.data [train.ind,]
mod.data.test <-  mod.data [-train.ind,]
mod.feature.matrix <- model.matrix(~., data = mod.data.train)
mod.feature.test.matrix <- model.matrix(~., data = mod.data.test)
mod.train.label <- tree.df[train.ind,]$logerror
mod.test.label <- tree.df[-train.ind,]$logerror
mod.gbt <- xgboost(data =  mod.feature.matrix,
               label = mod.train.label,
               params = best_param,
               nrounds=30,
               early_stopping_rounds = 10,
               maximize = FALSE)
```

```
## [1]  train-rmse:0.438664
## Will train until train_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:0.377793
## [3]  train-rmse:0.328202
## [4]  train-rmse:0.288152
## [5]  train-rmse:0.256170
## [6]  train-rmse:0.230891
## [7]  train-rmse:0.211263
## [8]  train-rmse:0.196292
## [9]  train-rmse:0.185017
```

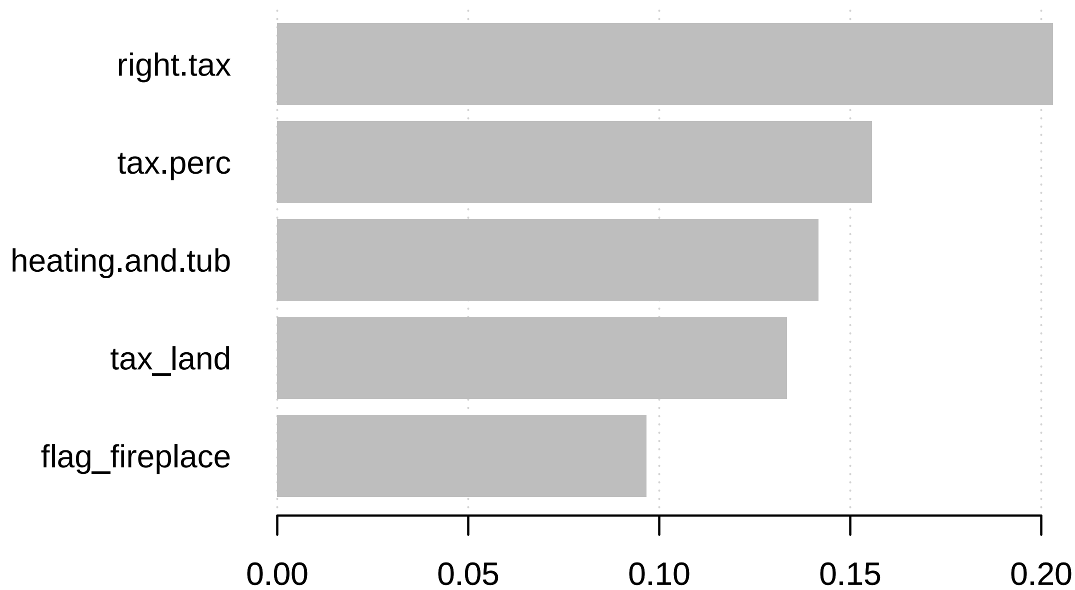
```
## [10] train-rmse:0.176583
## [11] train-rmse:0.170316
## [12] train-rmse:0.165807
## [13] train-rmse:0.162501
## [14] train-rmse:0.160138
## [15] train-rmse:0.158412
## [16] train-rmse:0.157198
## [17] train-rmse:0.156299
## [18] train-rmse:0.155670
## [19] train-rmse:0.155212
## [20] train-rmse:0.154800
## [21] train-rmse:0.154559
## [22] train-rmse:0.154365
## [23] train-rmse:0.154191
## [24] train-rmse:0.154068
## [25] train-rmse:0.153836
## [26] train-rmse:0.153723
## [27] train-rmse:0.153661
## [28] train-rmse:0.153615
## [29] train-rmse:0.153566
## [30] train-rmse:0.153495
```

```
mod.pred <- predict(mod.gbt,mod.feature.test.matrix)
mod.mse <- mean((mod.pred -mod.test.label )^2)
print(paste("reduced  feature(10) mse:",mod.mse,"compared to original mse",boost.mse))
```

```
## [1] "reduced  feature(10) mse: 0.0289382575021307 compared to original mse 0.0289149383886013"
```

Almost the same, then let us reduce the feature to 5 to see the result, note: it is better use cross validation here to choose number of features slected. However, this is just an intuitive attemp.

```
importance <- xgb.importance(feature_names = colnames(train.set), model = grid.search.gbt)
xgb.plot.importance(importance, top_n = 5)
trunc.feature <- xgb.plot.importance(importance, top_n = 5)$Feature
```




```r
mod.data <- tree.df[,trunc.feature]

mod.data.train <- mod.data [train.ind,]
mod.data.test <-  mod.data [-train.ind,]
mod.feature.matrix <- model.matrix(~., data = mod.data.train)
mod.feature.test.matrix <- model.matrix(~., data = mod.data.test)
mod.train.label <- tree.df[train.ind,]$logerror
mod.test.label <- tree.df[-train.ind,]$logerror
mod.gbt <- xgboost(data =  mod.feature.matrix,
               label = mod.train.label,
               params = best_param,
               nrounds=30,
               early_stopping_rounds = 10,
               maximize = FALSE)
```

```
## [1]  train-rmse:0.438639
## Will train until train_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:0.377874
## [3]  train-rmse:0.328330
## [4]  train-rmse:0.288418
## [5]  train-rmse:0.256578
## [6]  train-rmse:0.231411
## [7]  train-rmse:0.211900
## [8]  train-rmse:0.197035
## [9]  train-rmse:0.185809
```

```
## [10] train-rmse:0.177482
## [11] train-rmse:0.171291
## [12] train-rmse:0.166794
## [13] train-rmse:0.163562
## [14] train-rmse:0.161236
## [15] train-rmse:0.159541
## [16] train-rmse:0.158331
## [17] train-rmse:0.157515
## [18] train-rmse:0.156888
## [19] train-rmse:0.156475
## [20] train-rmse:0.156155
## [21] train-rmse:0.155903
## [22] train-rmse:0.155752
## [23] train-rmse:0.155628
## [24] train-rmse:0.155532
## [25] train-rmse:0.155434
## [26] train-rmse:0.155392
## [27] train-rmse:0.155340
## [28] train-rmse:0.155315
## [29] train-rmse:0.155265
## [30] train-rmse:0.155197
```

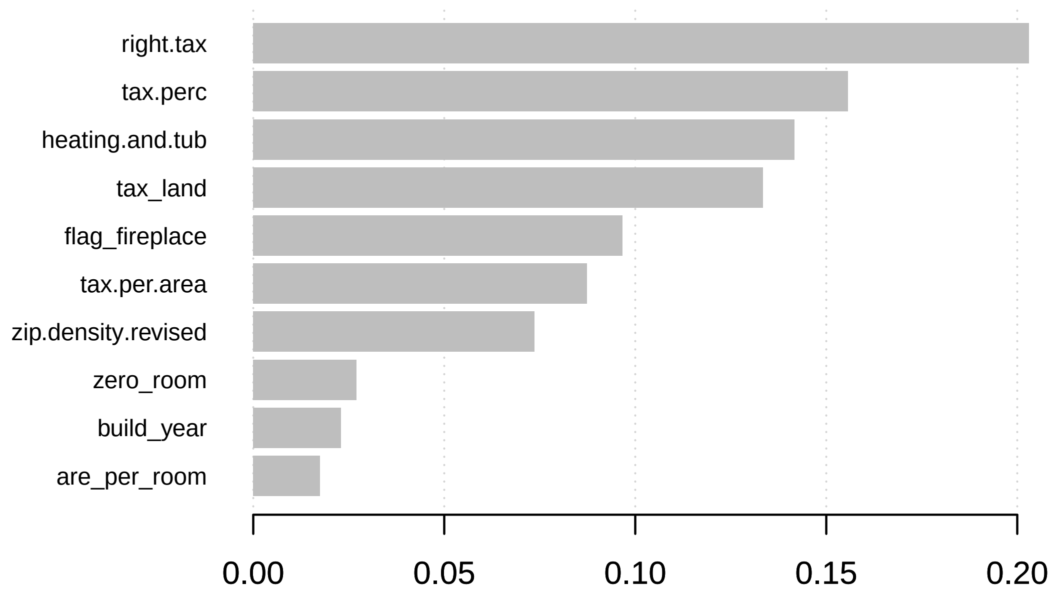
```
mod.pred <- predict(mod.gbt,mod.feature.test.matrix)
mod.mse <- mean((mod.pred -mod.test.label )^2)
print(paste("reduced  feature(5) mse:",mod.mse,"compared to original mse",boost.mse))
```

```
## [1] "reduced  feature(5) mse: 0.0290491961916325 compared to original mse 0.0289149383886013"
```

This is worse, it seems like 10 is better than 5.

improvement attempt 2 : reduce logerror outlier

```
up.bound <- quantile(train.set$logerror, 0.9)
low.bound <- quantile(train.set$logerror,0.1)
trunc.df <- subset(tree.df,  tree.df$logerror < up.bound
                              & tree.df$logerror >low.bound)
trunc.feature <- xgb.plot.importance(importance, top_n = 10)$Feature
```




```
mod.data <- trunc.df[,trunc.feature]
set.seed(1)
train.ind.trunc <- sample(1:nrow(mod.data),0.7*nrow(mod.data))

mod.data.train <- mod.data [train.ind.trunc ,]
mod.data.test <-  tree.df [-train.ind.trunc,trunc.feature]
mod.feature.matrix <- model.matrix(~., data = mod.data.train)
mod.feature.test.matrix <- model.matrix(~., data = mod.data.test)
mod.train.label <- trunc.df[train.ind.trunc,]$logerror
mod.test.label <- tree.df[-train.ind.trunc,]$logerror
mod.gbt <- xgboost(data =  mod.feature.matrix,
               label = mod.train.label,
               params = best_param,
               nrounds=30,
               early_stopping_rounds = 10,
               maximize = FALSE)
```

```
## [1]  train-rmse:0.415660
## Will train until train_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:0.349440
## [3]  train-rmse:0.293942
## [4]  train-rmse:0.247496
## [5]  train-rmse:0.208664
## [6]  train-rmse:0.176230
## [7]  train-rmse:0.149200
```

```
## [8]  train-rmse:0.126733
## [9]  train-rmse:0.108134
## [10] train-rmse:0.092843
## [11] train-rmse:0.080331
## [12] train-rmse:0.070170
## [13] train-rmse:0.062028
## [14] train-rmse:0.055570
## [15] train-rmse:0.050528
## [16] train-rmse:0.046673
## [17] train-rmse:0.043740
## [18] train-rmse:0.041544
## [19] train-rmse:0.039923
## [20] train-rmse:0.038749
## [21] train-rmse:0.037899
## [22] train-rmse:0.037291
## [23] train-rmse:0.036849
## [24] train-rmse:0.036526
## [25] train-rmse:0.036305
## [26] train-rmse:0.036150
## [27] train-rmse:0.036040
## [28] train-rmse:0.035964
## [29] train-rmse:0.035909
## [30] train-rmse:0.035869
```

```
mod.pred <- predict(mod.gbt,mod.feature.test.matrix)
mod.mse <- mean((mod.pred -mod.test.label )^2)
print(paste("reduced outlier mse:",mod.mse,"compared to original mse",boost.mse))
```

```
## [1] "reduced outlier mse: 0.0221497447591616 compared to original mse 0.0289149383886013"
```

An important note here: we remove the outlier when building our model, however, when we test our reulst on test data set, we did mot remove the outlier because we can not use the infomation of response in the test set. So the result can be trusted in this case, which is better than before.

Comparision with linear model

Let us take a look at the whether our tree model is better than our linear model by calculating the MSE for linear model

```
geo.feature<- c("county6111","zip.density.revised")
quality.feature <- c("new.quality")
geo.feature<- c("county6111","zip.density.revised")
tax.feature <- c("tax_delinquency","tax_total","tax_property","tax_land","right.tax","tax.per.area","bu
facility.feature <- c("flag_tub","flag_fireplace","missing.aircon","new.heating")
room.area.feature <- c("num_room","num_bedroom","num_bathroom","right_room","bed_to_bath",
                       "are_per_room","zero_room","area_total_calc")

selected.feature <- c("logerror",geo.feature,quality.feature,tax.feature,facility.feature,room.area.fea
reg.df <- train[,selected.feature]
reg.df.train <- reg.df[train.ind,]
reg.df.test <- reg.df[-train.ind,]

mod <- lm(logerror~.-zip.density.revised +I((zip.density.revised)^(1/18))
           -are_per_room + I(are_per_room^(1/15))
           -county6111-new.quality
```

```r
           #-tax_land +I(tax_land^(1/2))
           -tax_property + I(tax_property^(1/60))
           -num_room
           -bed_to_bath + I((bed_to_bath)^(1/2))
           -tax.per.area + I(tax.per.area^(1/50))
           -flag_fireplace
           -missing.aircon
           -tax.perc + I(tax.perc^(1/12))
           -right.tax
           + right.tax:I(tax_land^(1/3))
           +I(build.to.total^(1/60))
          + flag_tub : new.heating
          +right_room : zero_room
          -flag_tub

          ,data = reg.df.train)
linear.model.pred <- predict( mod,data = reg.df[reg.df.test,])
linear.model.mse <- mean((linear.model.pred - reg.df.test$logerror)^2)
print(paste("Linear model mse is:",linear.model.mse))
```

```
## [1] "Linear model mse is: 0.0300317763166295"
```

```r
print(paste("compared  to our best tree model, of which mse is",mod.mse))
```

```
## [1] "compared  to our best tree model, of which mse is 0.0221497447591616"
```

Conclusion

Overall, tree methods perform better than linear model. However, it takes forever to run random forest and xgboosting(grid search). Compared to linear regression, these two methods are more complicated, so we expected it to have a better performance. However, linear model is simple and easy to interpret compared to xgboost.

In the end

This project contains an end to end project from data exploration, data cleaning, feature engineering to build model. In general, it can be super frustraring project espeically when seeing the result is super bad at 4am in the morning. However, I got hands-on experience in missing data imputation, feature engineering as well as building tree-based model, which is very valuable for me.