Midterm Project

Florin Andrei

3/23/2021

```r
library(gbm)
```

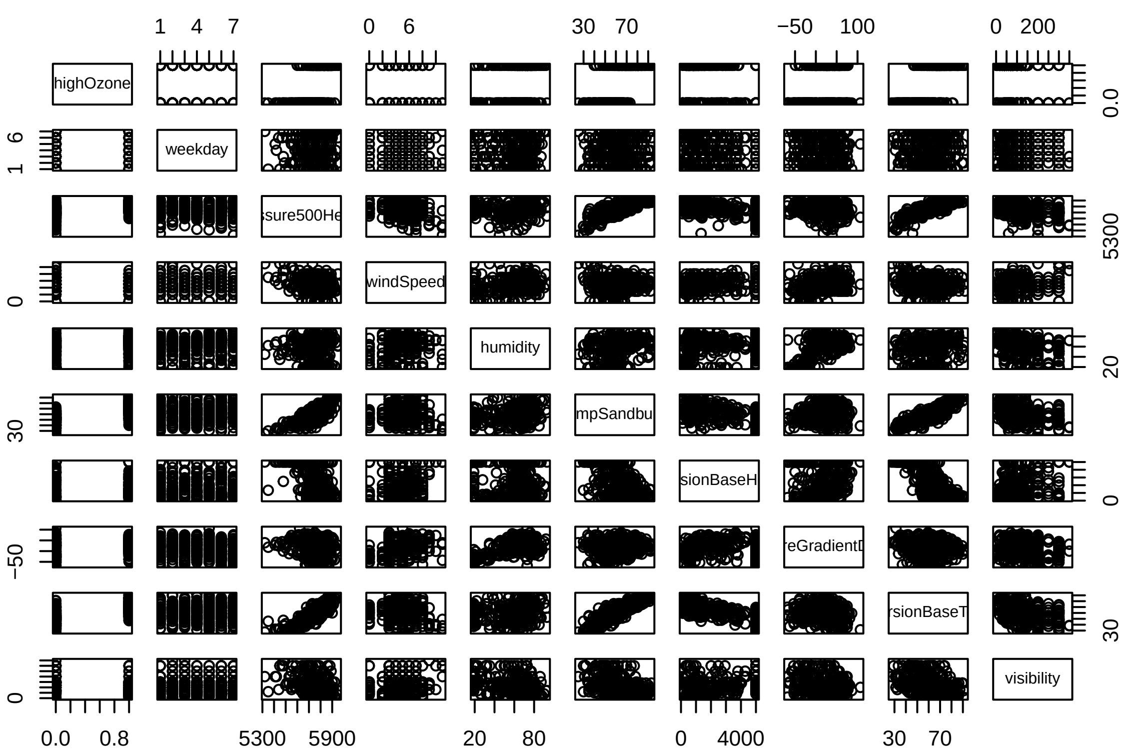
```
## Loaded gbm 2.1.8.1
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 4.1-8
```

Load Data, preprocess, explore

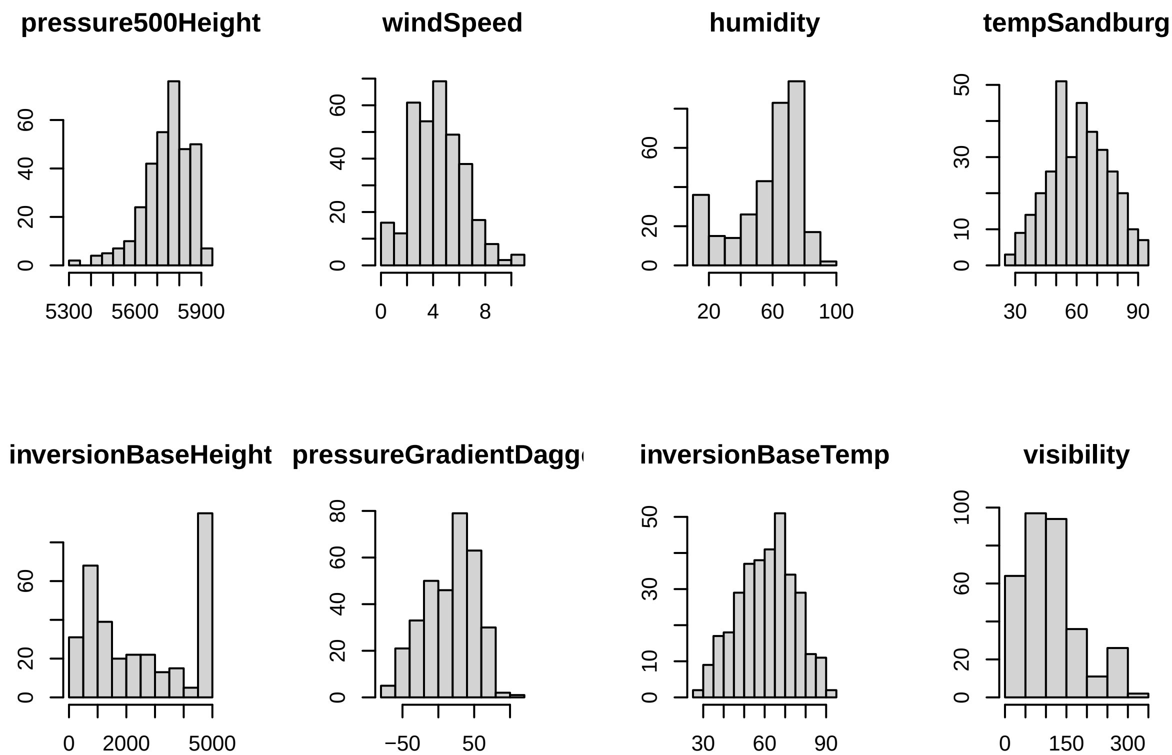
```r
ds = read.csv("ozone.csv")
ds = na.omit(ds)
ds$weekday = as.factor(ds$weekday)
ds = ds[, !(colnames(ds) %in% c("hourAverageMax"))]

pairs(ds)
```



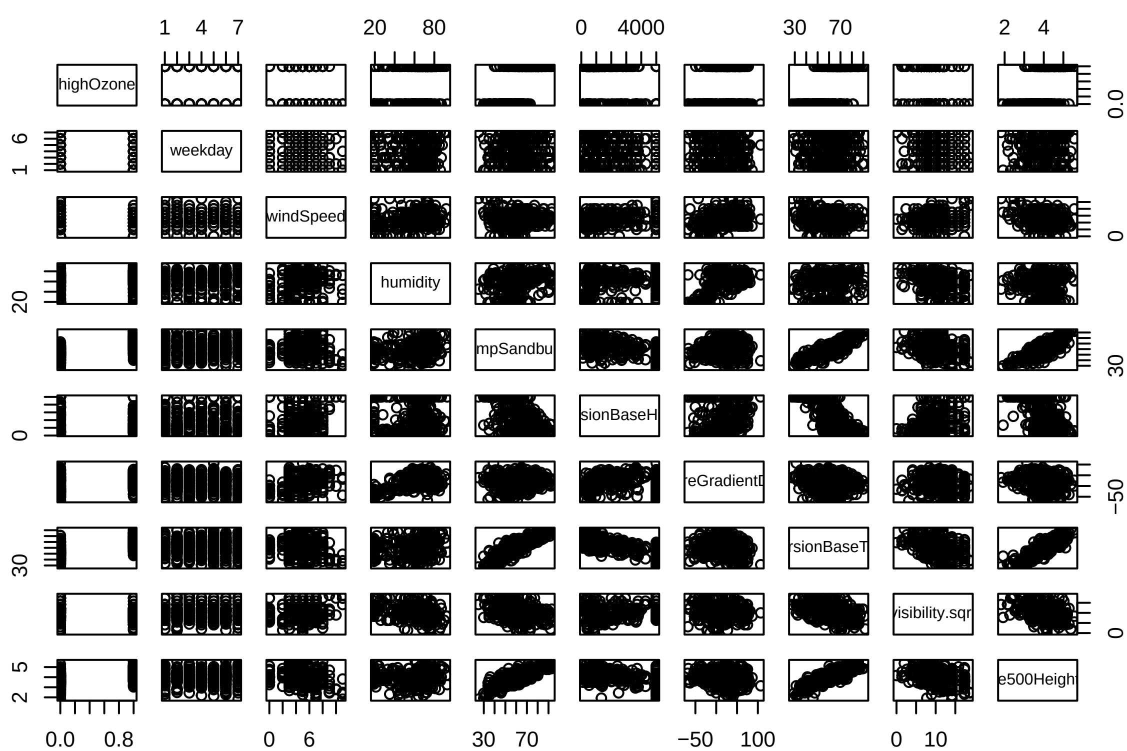
```r
summary(ds)
```

```
##    highOzone          weekday   pressure500Height   windSpeed
##  Min.   :0.0000   Friday   :47   Min.   :5320      Min.   : 0.000
##  1st Qu.:0.0000   Monday   :47   1st Qu.:5690      1st Qu.: 3.000
##  Median :1.0000   Saturday :49   Median :5760      Median : 5.000
##  Mean   :0.5061   Sunday   :48   Mean   :5750      Mean   : 4.848
##  3rd Qu.:1.0000   Thursday :41   3rd Qu.:5830      3rd Qu.: 6.000
##  Max.   :1.0000   Tueday   :50   Max.   :5950      Max.   :11.000
##                   Wednesday:48
##     humidity      tempSandburg   inversionBaseHeight pressureGradientDaggett
##  Min.   :19.00   Min.   :25.00   Min.   : 111.0      Min.   :-69.00
##  1st Qu.:47.00   1st Qu.:51.00   1st Qu.: 877.5      1st Qu.: -9.00
##  Median :64.00   Median :62.00   Median :2112.5      Median : 24.00
##  Mean   :58.13   Mean   :61.75   Mean   :2572.9      Mean   : 17.37
##  3rd Qu.:73.00   3rd Qu.:72.00   3rd Qu.:5000.0      3rd Qu.: 44.75
##  Max.   :93.00   Max.   :93.00   Max.   :5000.0      Max.   :107.00
##
##  inversionBaseTemp   visibility
##  Min.   :27.50     Min.   :  0.0
##  1st Qu.:51.26     1st Qu.: 70.0
##  Median :62.15     Median :120.0
##  Mean   :61.01     Mean   :124.5
##  3rd Qu.:70.52     3rd Qu.:150.0
##  Max.   :91.76     Max.   :350.0
##
```

```r
par(mfrow = c(2, 4))
for (col in c("pressure500Height", "windSpeed", "humidity", "tempSandburg", "inversionBaseHeight", "pres
  hist(ds[[col]], main = col, xlab = NA, ylab = NA)
}
```

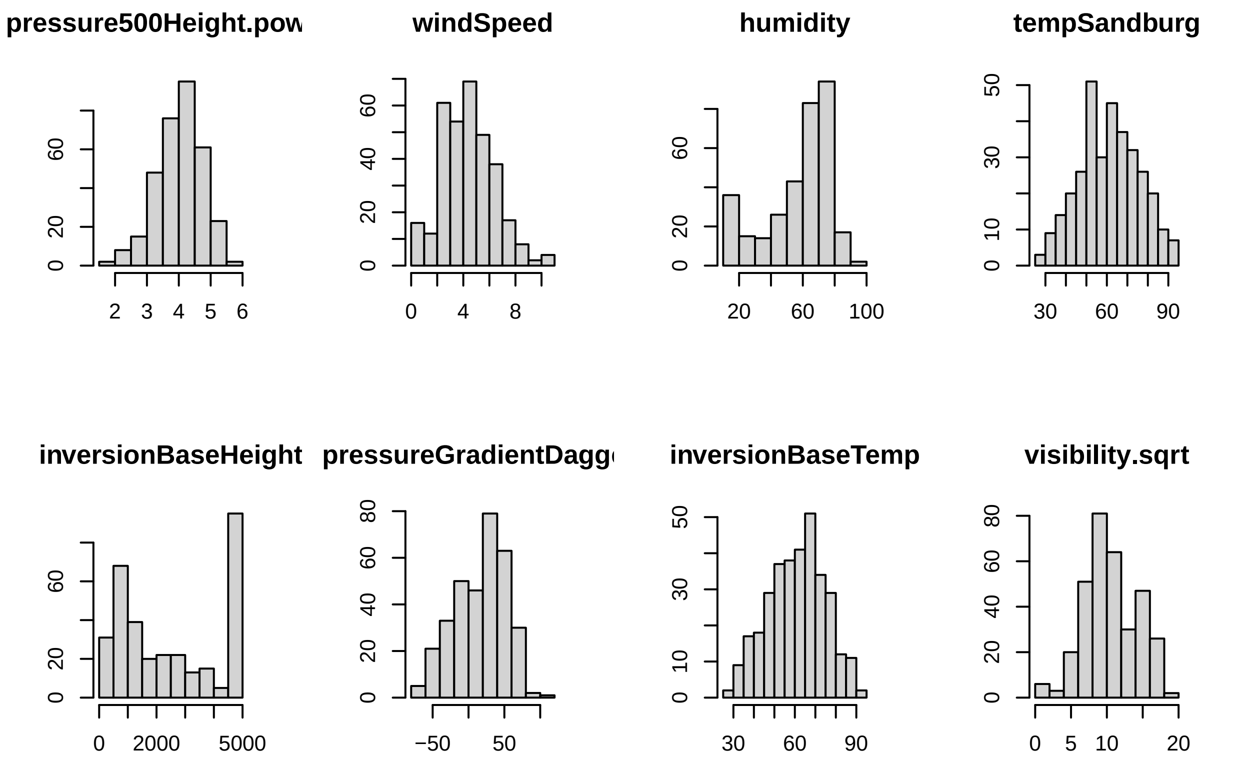



```r
ds["visibility.sqrt"] = sqrt(ds$visibility)
# not sure if pow10 is justified, might be too extreme
# but the histogram sure looks nice
ds["pressure500Height.pow10"] = ds$pressure500Height ^ 10 / 10 ^ 37
ds = ds[, !(colnames(ds) %in% c("visibility", "pressure500Height"))]
pairs(ds)
```



There are some highly correlated variables, mostly pressure and temperature, that very likely reflect actual physical correlations (physical phenomena).

```
par(mfrow = c(2, 4))
for (col in c("pressure500Height.pow10", "windSpeed", "humidity", "tempSandburg", "inversionBaseHeight"
  hist(ds[[col]], main = col, xlab = NA, ylab = NA)
}
```



Histograms look good, with the exception of `humidity` and `inversionBaseHeight` which have outliers and are not very normal.

Main code

```
# number of different model flavors
# same number for trees and enet
n.mod = 8

# generating all combinations of hyperparameters for trees
# like a cube in the hypeparameter space
# this is likely much too small, a bigger grid would work better
# or some kind of search in that space
n.trees = c(2000, 2000, 2000, 2000, 4000, 4000, 4000, 4000)
shrink  = c(0.001, 0.001, 0.0005, 0.0005, 0.001, 0.001, 0.0005, 0.0005)
idepth  = c(3, 4, 3, 4, 3, 4, 3, 4)

# ENet model parameters
# many lambdas - they are all subsumed to the alpha values
lambdalist = exp((-1000:500) / 100)
# 8 alpha values - the "main" model flavors
alphalist = c(0.0, 0.1, 0.2, 0.4, 0.6, 0.8, 0.9, 1.0)
```

```r
fulldata.out = ds
x.out = model.matrix(highOzone ~ ., data = fulldata.out)[, -c(1)]
y.out = fulldata.out[, 1]
k.out = 10
n.out = dim(fulldata.out)[1]

# future predicted points (outer level)
pred.out = rep(NA, n.out)

# outer CV splits
groups.out = c(rep(1:k.out, floor(n.out / k.out)))
if(floor(n.out / k.out) != (n.out / k.out)) {
  groups.out = c(groups.out, 1:(n.out %% k.out))
}
set.seed(10)
cvgroups.out = sample(groups.out, n.out)

# keep everything about best models in one place:
# type of model, hyperparameters, error rates, etc.
# rows: k.out
# cols: one for each thing worth keeping
best.out = data.frame()

# this is very slow, yet CPU utilization is at 25%
# parallelization would be really good here (TBD)
for (j in 1:k.out) {
  cat("\n")
  cat("k.out:", j, "out of", k.out, "\n")

  groupj.out = (cvgroups.out == j)

  traindata.out = fulldata.out[!groupj.out, ]
  trainx.out = model.matrix(highOzone ~ ., data = traindata.out)[, -c(1)]
  trainy.out = traindata.out[, 1]

  validdata.out = fulldata.out[groupj.out, ]
  validx.out = model.matrix(highOzone ~ ., data = validdata.out)[, -c(1)]
  validy.out = validdata.out[, 1]

  ######## begin modeling process ########

  # connect in and out
  fulldata.in = traindata.out

  n.in = dim(fulldata.in)[1]

  # number of inner folds
  k.in = 10

  # do the x/y split for ENet
  x.in = model.matrix(highOzone ~ ., data = fulldata.in)[, -c(1)]
  y.in = fulldata.in[, 1]
```

```r
groups.in = c(rep(1:k.in, floor(n.in / k.in)))
if (floor(n.in / k.in) != (n.in / k.in)) {
  groups.in = c(groups.in, 1:(n.in %% k.in))
}
cvgroups.in = sample(groups.in, n.in)

# the error rate from each tree "flavor"
err.trees = rep(NA, n.mod)

# keep track of the best lambda for each alpha
bestlambda = rep(NA, n.mod)
# error rates from each alpha value
err.enet = rep(NA, n.mod)

# loop through model versions
for (m in 1:n.mod) {
  # prep empty prediction vectors for trees and ENet
  boost.predict = rep(NA, n.in)
  enet.predict = matrix(NA, nrow = n.in, ncol = length(lambdalist))

  # fit both trees and ENet in one fell swoop
  for (i in 1:k.in) {
    groupi = (cvgroups.in == i)

    boost = gbm(highOzone ~ .,
                data = fulldata.in[!groupi, ],
                distribution = "bernoulli",
                n.trees = n.trees[m],
                shrinkage = shrink[m],
                interaction.depth = idepth[m])

    boost.predict[groupi] = predict(boost,
                                    newdata = fulldata.in[groupi, ],
                                    n.trees = n.trees[m],
                                    type = "response")

    enet = glmnet(x.in[!groupi, ],
                  y.in[!groupi],
                  alpha = alphalist[m],
                  lambda = lambdalist,
                  family = "binomial")

    enet.predict[groupi, ] = predict(enet,
                                     newx = x.in[groupi, ],
                                     type = "response")
  }

  # confusion matrix for trees
  cmtree = table(boost.predict > 0.5, fulldata.in$highOzone)
  # errors for trees
  err.trees[m] = (cmtree[1, 2] + cmtree[2, 1]) / n.in

  # for each alpha, need to figure out the best lambda first
```

```
  err.enet.lambda = rep(NA, length(lambdalist))

  # many confusion matrices, one for each lambda
  for (lindex in 1:length(lambdalist)) {
    # without factor(..., levels = ...) there will be tables with 1 row
    # which breaks the cmenet[x, y] syntax
    cmenet = table(factor(enet.predict[, lindex] > 0.5, levels = c(FALSE, TRUE)),
                   factor(fulldata.in$highOzone == 1, levels = c(FALSE, TRUE)))
    err.enet.lambda[lindex] = (cmenet[1, 2] + cmenet[2, 1]) / n.in
  }

  # best lambda for this particular alpha
  which.min.err.lambda = order(err.enet.lambda)[1]
  bestlambda[m] = lambdalist[which.min.err.lambda]
  # the error for ENet at this particular alpha
  err.enet[m] = err.enet.lambda[which.min.err.lambda]

}

# all errors for all 8 + 8 models
cat("err.trees:", err.trees, "\n")
cat("err.enet :", err.enet, "\n")

# figure out the winner and its hyperparameters
which.best.tree = order(err.trees)[1]
which.best.enet = order(err.enet)[1]

best.tree.err = err.trees[which.best.tree]
best.enet.err = err.enet[which.best.enet]

best.tree.n.trees = n.trees[which.best.tree]
best.tree.shrink = shrink[which.best.tree]
best.tree.idepth = idepth[which.best.tree]

best.enet.alpha = alphalist[which.best.enet]
best.enet.lambda = bestlambda[which.best.enet]

if (min(err.trees) < min(err.enet)) {
  best.model.in = "tree"
  best.err.in = best.tree.err

  # fit on fulldata.in = same as traindata.out
  best.fit.in = gbm(highOzone ~ .,
                    data = fulldata.in,
                    distribution = "bernoulli",
                    n.trees = best.tree.n.trees,
                    shrinkage = best.tree.shrink,
                    interaction.depth = best.tree.idepth)
  pred.out[groupj.out] = predict(best.fit.in,
                                 newdata = validdata.out,
                                 n.trees = best.tree.n.trees,
                                 type = "response")
```

```
    cat("inner best model:", best.model.in,
        "err.in:", best.tree.err,
        "n.trees:", best.tree.n.trees,
        "shrinkage:", best.tree.shrink,
        "i.depth:", best.tree.idepth,
        "\n")
  } else {
    best.model.in = "enet"
    best.err.in = best.enet.err

    # fit on fulldata.in = same as traindata.out
    best.fit.in = glmnet(x.in,
                         y.in,
                         alpha = best.enet.alpha,
                         lambda = best.enet.lambda,
                         family = "binomial")
    pred.out[groupj.out] = predict(best.fit.in,
                                   newx = validx.out,
                                   type = "response")

    cat("inner best model:", best.model.in,
        "err.in:", best.enet.err,
        "alpha:", best.enet.alpha,
        "lambda:", best.enet.lambda,
        "\n")
  }

  ######## end modeling process ########

  best.out[j, "type"] = best.model.in
  # this is just for the record
  # (the error on the inner level)
  best.out[j, "err.in"] = best.err.in
  if (best.model.in == "enet") {
    best.out[j, "alpha"] = best.enet.alpha
    best.out[j, "lambda"] = best.enet.lambda
  }
  if (best.model.in == "tree") {
    best.out[j, "n.trees"] = best.tree.n.trees
    best.out[j, "shrinkage"] = best.tree.shrink
    best.out[j, "i.depth"] = best.tree.idepth
  }
}
```

```
##
## k.out: 1 out of 10
## err.trees: 0.1447811 0.1481481 0.1548822 0.1582492 0.1515152 0.1582492 0.1481481 0.1481481
## err.enet : 0.1649832 0.1582492 0.1616162 0.1616162 0.1649832 0.1548822 0.1582492 0.1582492
## inner best model: tree err.in: 0.1447811 n.trees: 2000 shrinkage: 0.001 i.depth: 3
##
## k.out: 2 out of 10
## err.trees: 0.1481481 0.1414141 0.1481481 0.1447811 0.1447811 0.1515152 0.1414141 0.1447811
## err.enet : 0.1346801 0.1414141 0.1414141 0.1414141 0.1447811 0.1447811 0.1414141 0.1414141
## inner best model: enet err.in: 0.1346801 alpha: 0 lambda: 0.002009237
```

```
##
## k.out: 3 out of 10
## err.trees: 0.1313131 0.1346801 0.1346801 0.1279461 0.1279461 0.1279461 0.1279461 0.1346801
## err.enet : 0.1346801 0.1380471 0.1313131 0.1313131 0.1380471 0.1447811 0.1414141 0.1414141
## inner best model: tree err.in: 0.1279461 n.trees: 2000 shrinkage: 5e-04 i.depth: 4
##
## k.out: 4 out of 10
## err.trees: 0.1548822 0.1548822 0.1447811 0.1548822 0.1582492 0.1616162 0.1515152 0.1548822
## err.enet : 0.1481481 0.1447811 0.1481481 0.1481481 0.1481481 0.1515152 0.1515152 0.1515152
## inner best model: enet err.in: 0.1447811 alpha: 0.1 lambda: 0.2231302
##
## k.out: 5 out of 10
## err.trees: 0.1548822 0.1548822 0.1548822 0.1548822 0.1582492 0.1683502 0.1515152 0.1515152
## err.enet : 0.1447811 0.1481481 0.1515152 0.1515152 0.1548822 0.1548822 0.1548822 0.1548822
## inner best model: enet err.in: 0.1447811 alpha: 0 lambda: 0.1480804
##
## k.out: 6 out of 10
## err.trees: 0.1380471 0.1447811 0.1414141 0.1447811 0.1447811 0.1515152 0.1447811 0.1447811
## err.enet : 0.1582492 0.1481481 0.1414141 0.1481481 0.1548822 0.1582492 0.1548822 0.1548822
## inner best model: tree err.in: 0.1380471 n.trees: 2000 shrinkage: 0.001 i.depth: 3
##
## k.out: 7 out of 10
## err.trees: 0.1380471 0.1380471 0.1414141 0.1380471 0.1447811 0.1447811 0.1380471 0.1380471
## err.enet : 0.1481481 0.1346801 0.1346801 0.1279461 0.1279461 0.1346801 0.1313131 0.1346801
## inner best model: enet err.in: 0.1279461 alpha: 0.4 lambda: 0.06457035
##
## k.out: 8 out of 10
## err.trees: 0.1245791 0.1346801 0.1313131 0.1313131 0.1279461 0.1313131 0.1245791 0.1346801
## err.enet : 0.1447811 0.1414141 0.1414141 0.1447811 0.1481481 0.1447811 0.1515152 0.1481481
## inner best model: tree err.in: 0.1245791 n.trees: 2000 shrinkage: 0.001 i.depth: 3
##
## k.out: 9 out of 10
## err.trees: 0.1548822 0.1616162 0.1616162 0.1582492 0.1649832 0.1683502 0.1548822 0.1616162
## err.enet : 0.1515152 0.1447811 0.1414141 0.1346801 0.1447811 0.1481481 0.1481481 0.1447811
## inner best model: enet err.in: 0.1346801 alpha: 0.4 lambda: 0.05233971
##
## k.out: 10 out of 10
## err.trees: 0.1313131 0.1279461 0.1279461 0.1313131 0.1313131 0.1279461 0.1245791 0.1279461
## err.enet : 0.1313131 0.1279461 0.1279461 0.1245791 0.1212121 0.1178451 0.1178451 0.1178451
## inner best model: enet err.in: 0.1178451 alpha: 0.8 lambda: 1.750673
```

Main parameters for the best model in each outer loop:

```r
print(best.out)
```

```
##    type    err.in n.trees shrinkage i.depth alpha      lambda
## 1  tree 0.1447811    2000     1e-03       3    NA          NA
## 2  enet 0.1346801      NA        NA      NA   0.0 0.002009237
## 3  tree 0.1279461    2000     5e-04       4    NA          NA
## 4  enet 0.1447811      NA        NA      NA   0.1 0.223130160
## 5  enet 0.1447811      NA        NA      NA   0.0 0.148080387
## 6  tree 0.1380471    2000     1e-03       3    NA          NA
## 7  enet 0.1279461      NA        NA      NA   0.4 0.064570347
## 8  tree 0.1245791    2000     1e-03       3    NA          NA
## 9  enet 0.1346801      NA        NA      NA   0.4 0.052339706
```

```
## 10 enet 0.1178451      NA        NA      NA   0.8 1.750672500
```

Overall double-cross-validated performance:

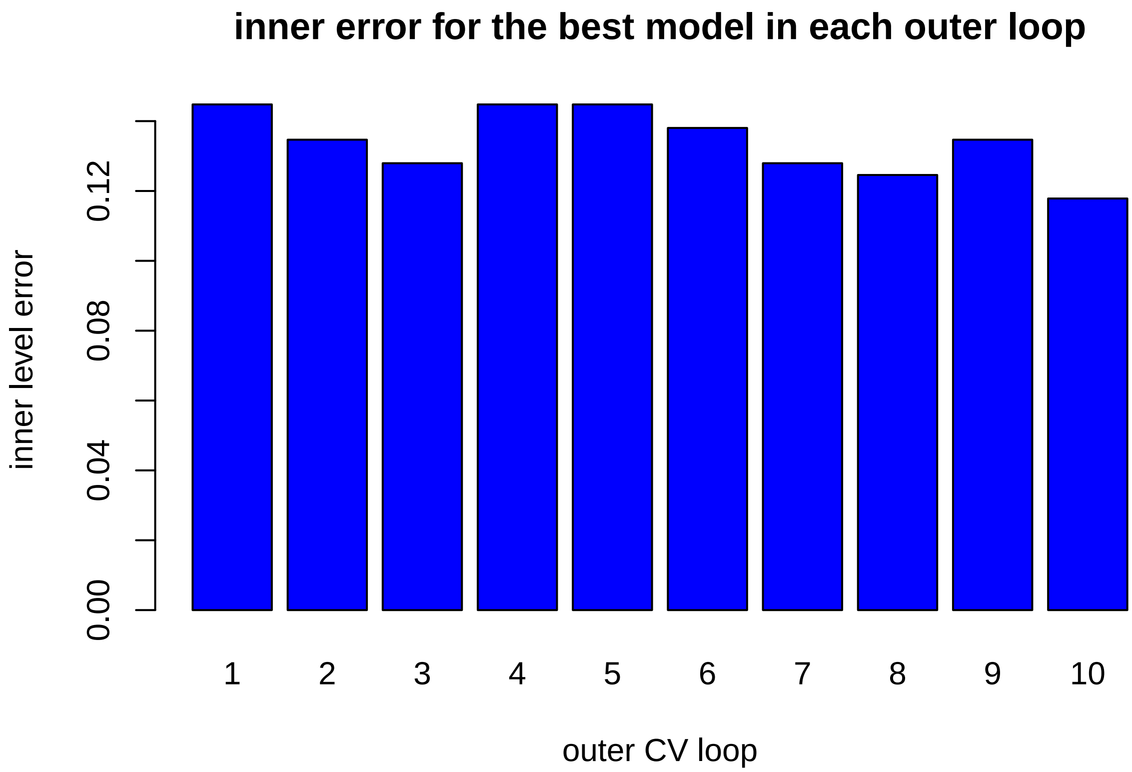
```r
# outer confusion matrix
cm.out = table(pred.out > 0.5, y.out)
# overall error
err.out = (cm.out[1, 2] + cm.out[2, 1]) / n.out
cat("overall error:", err.out)
```

```
## overall error: 0.1878788
```

Conclusions

Well, this is hard. There is no clear winner.

```r
barplot(best.out$err.in,
        names.arg = row.names(best.out),
        xlab = "outer CV loop",
        ylab = "inner level error",
        col = "blue",
        main = "inner error for the best model in each outer loop")
```



Out of 10 winners, 6 are ENet, 4 are boosted trees. I am going to pick ENet.

The best ENet, based on err.in, is in loop #10. However, it has a few issues:

- its parameters (alpha and lambda) are outliers, compared to the other ENet models
- fitted on the whole data, if I apply coef(best.enet, ...) it returns all coefficients equal to zero

That should be investigated, but I don't have time. Regardless, for the purpose of ranking the variables by importance, all models agree on the top 3 ... 4 variables.

So I am going to pick the next best ENet, from outer loop #7. Its performance is decent and it's not an outlier.

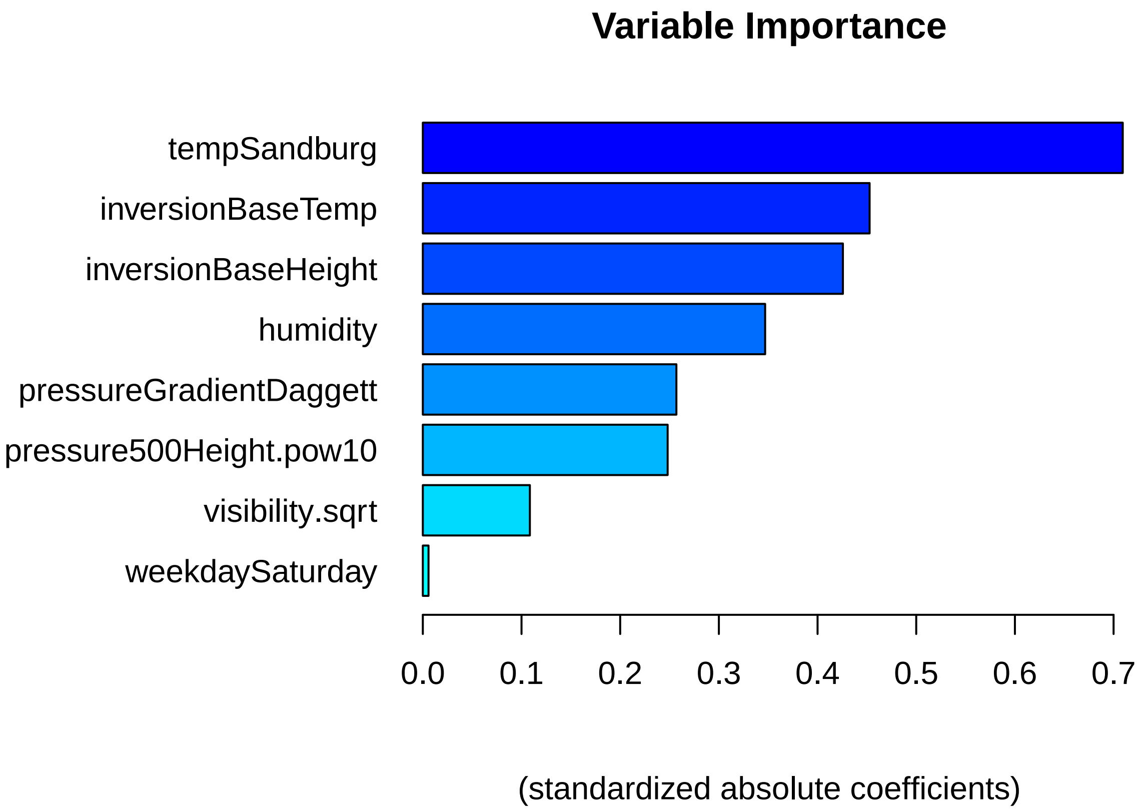
```r
# fit the "best model" on the whole data
best.enet = glmnet(x.out,
                   y.out,
                   alpha = best.out[7, "alpha"],
                   lambda = best.out[7, "lambda"],
                   family = "binomial")

# extract coefficients
enet.coef = coef(best.enet, s = 0.05)
# standardize and keep the absolute value
enet.coef = cbind(enet.coef, rep(0, dim(enet.coef)[1]))
colnames(enet.coef) = c("coef", "abs.std.coef")
for (c in colnames(x.out)) {
  enet.coef[c, "abs.std.coef"] = abs(enet.coef[c, "coef"] * sd(x.out[, c]))
}
enet.coef = enet.coef[order(enet.coef[, "abs.std.coef"], decreasing = T), ]

# save coefficients to CSV
write.csv(as.matrix(enet.coef), file = "enet_coef.csv")

enet.coef
```

```
## 15 x 2 sparse Matrix of class "dgCMatrix"
##                                    coef abs.std.coef
## tempSandburg               0.0490563377  0.709292675
## inversionBaseTemp          0.0328029826  0.452756484
## inversionBaseHeight       -0.0002359874  0.425694395
## humidity                   0.0174631061  0.346904604
## pressureGradientDaggett    0.0071944182  0.256964335
## pressure500Height.pow10    0.3614007841  0.248091027
## visibility.sqrt           -0.0295606834  0.108479645
## weekdaySaturday            0.0161890101  0.005765229
## (Intercept)               -6.6880862904  .
## weekdayMonday              .             .
## weekdaySunday              .             .
## weekdayThursday            .             .
## weekdayTueday              .             .
## weekdayWednesday           .             .
## windSpeed                  .             .
```

```r
# remove zeroes (variables that were deemed irrelevant by the model)
enet.coef = enet.coef[enet.coef[, 2] > 0, ]

cpal = colorRampPalette(colors = c("cyan", "blue"))(dim(enet.coef)[1])
par(mar = c(5.1, 12.0, 4.1, 2.1))
img.out = "var-rank.png"
if (file.exists(img.out)) {
  file.remove(img.out)
}
```

```
## [1] TRUE
```

```r
barplot(sort(enet.coef[, 2], decreasing = F),
        horiz = T,
        las = 1,
        col = cpal,
        main = "Variable Importance",
        sub = "(standardized absolute coefficients)")
```




```r
dev.copy(png, img.out, width = 800, height = 600)
```

```
## png
##   3
```

```r
dev.off()
```

```
## pdf
##   2
```