Deriving variables: Distance

Don Li

05/06/2020

Data and stuff

```
load( "G:/azure_hackathon/dataset/trip_summary3_landmarks.RData" )

set.seed(1)
data_subset = trip_summary[ sample.int( .N, 7000 ) ]

save( data_subset,
    file = "G:/azure_hackathon/dataset/deriving_variables/derive_dist.RData" )
```

What are we doing here?

In this document, we want to consider predicting path distance. You will note that in the model inputs, path distance is not a given input:

- latitude_origin
- longitude_origin
- latitude_destination
- longitude_destination
- hour_of_day
- day_of_week

Although we can compute the distance as the crow flies (CF), we will also need the path distance. In our ETA model, we will use path distance, but we will treat it as a missing value. Therefore, we will need some model-based imputation to get some path distance information.

Brief note: The landmarks variables made the models really bad. Probably because the factors force complete pooling of observations and fragments the data too much.

A view of the variables before we start.

```
load( "G:/azure_hackathon/dataset/deriving_variables/derive_dist.RData" )
head( data_subset )
```

```
##    trj_id timediff crow_dist path_dist path_dist2 weekday hour rush_hour
## 1:  68938      949  6.698328  9.564086   5.582699     Mon   19     Night
## 2:  35520     1264 12.277911 20.559480  15.282477     Fri   20        No
## 3:  73124      865 14.105926 16.176197   8.418124     Mon   17        No
## 4:  58947     1225 13.115309 17.615926   9.452248     Mon   13        No
## 5:  67593     1419 15.139523 18.998952  11.931849     Fri    1        No
## 6:  39564      862 10.332575 13.352407  10.445341     Thu   14        No
##     start_x  start_y    end_x    end_y    N sampling_rate sampling_rate_var
## 1: 1.387601 103.8413 1.337177 103.8085  867      1.095843         0.8104567
## 2: 1.361755 103.8951 1.440709 103.8181 1175      1.076661         1.7451837
## 3: 1.441069 103.7715 1.327641 103.8280  810      1.069221         1.7526282
## 4: 1.426336 103.7838 1.308519 103.7835 1123      1.091800         0.1548123
## 5: 1.294732 103.8502 1.410435 103.7787 1144      1.241470         3.8540756
```

```
## 6: 1.314560 103.9376 1.343709 103.8494  761      1.134211         2.4905277
##    mean_speed var_speed azure_dist azure_eta OSRM_dist OSRM_eta trip_start
## 1:   11.98927  32.98629      9.568       867  9.609045    741.1    generic
## 2:   17.24607  15.46363     20.254      1187 16.186235   1238.9    generic
## 3:   19.73807  20.39490     17.788      1572 16.146324    921.6        C41
## 4:   15.81157  35.09184     17.515      1072 17.043101   1133.3        C41
## 5:   15.71189  39.97094     23.718      1293 18.629650   1227.5        C16
## 6:   16.89138  27.62735     16.060       907 13.424947    916.1    generic
##    trip_end
## 1:  generic
## 2:      C17
## 3:  generic
## 4:  generic
## 5:      C41
## 6:  generic
```

Partition the data. Take 25% for an out-of-bag set. Take the 75% for 7-fold CV. If we need to optimise the hyperparameters, they will be done using random search.

```
data_subset[ , c("trip_start", "trip_end", "trj_id", "timediff",
    "N", "sampling_rate", "sampling_rate_var", "mean_speed",
    "var_speed" ) := NULL ]

set.seed(99)
inTrain = createDataPartition( data_subset$path_dist, p = 0.75 )

data_subset1 = copy( data_subset )
data_subset1[ , path_dist2 := NULL ]

training_set = data_subset1[ inTrain$Resample1 ]
test_set = data_subset1[ -inTrain$Resample1 ]

cv_folds = 7
cv_fold_id = createFolds( training_set$path_dist, k = cv_folds, returnTrain = T )
train_control = trainControl(
    method = "cv", number = cv_folds,
    verboseIter = TRUE, search = "grid",
    index = cv_fold_id, savePredictions = "final"
)

save( inTrain, training_set, test_set, train_control,
    file = "G:/azure_hackathon/dataset/deriving_variables/datasets.RData" )
```

Baseline

For our baselines, we will use the Azure and OSRM distances.

```
load( "G:/azure_hackathon/dataset/deriving_variables/datasets.RData" )
azure_baseline = sqrt( mean( (training_set$path_dist - training_set$azure_dist)^2 ) )
azure_baseline
```

```
## [1] 2.363987
```

```r
osrm_baseline = sqrt( mean( (training_set$path_dist - training_set$OSRM_dist)^2 ) )
osrm_baseline
```

```
## [1] 2.499241
```

Linear regression

Linear regression. All second-order interactions, but some of the weird ones removed. Some quadratics for the start/end locations.

```r
model_str_lm = "path_dist ~ .*. +
    I(start_x^2) + I(start_y^2) + I(end_x^2) +  I(end_y^2) +
    start_y:end_y - start_x:end_x
"
model_formula_lm = as.formula(model_str_lm)

lm_ = train( form = model_formula_lm,
    data = training_set,
    metric = "RMSE", method = "lm", trControl = train_control)


lm_results = data.table( lm_$results )
lm_pred = data.table( lm_$pred )
setorder( lm_pred, rowIndex )

save( lm_, lm_results, lm_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/lm.RData" )
```

```r
load( "G:/azure_hackathon/dataset/deriving_variables/lm.RData" )
lm_results
```

```
##    intercept     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1:      TRUE 2.167828 0.8709461 1.134444 0.1944068 0.02424643 0.0349674
```

CV error is 2.168. Better than out baselines.

Elastic net

Elastic net.

```r
n_enet = 50
enet_tunegrid = data.frame(
    lambda = runif( n_enet, 0, 1e-1 ),
    fraction = runif( n_enet, 0.2, 1 )
)
enet_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "enet", trControl = train_control,
    tuneGrid = enet_tunegrid, standardize = TRUE, intercept = TRUE
    )

enet_results = data.table( enet_$results )
enet_pred = data.table( enet_$pred )
setorder( enet_pred, rowIndex )
```

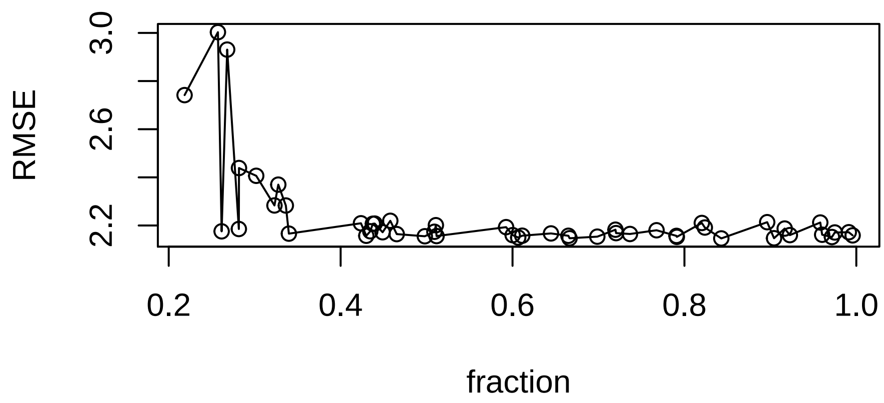
```r
save( enet_, enet_results, enet_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/enet.RData" )
```

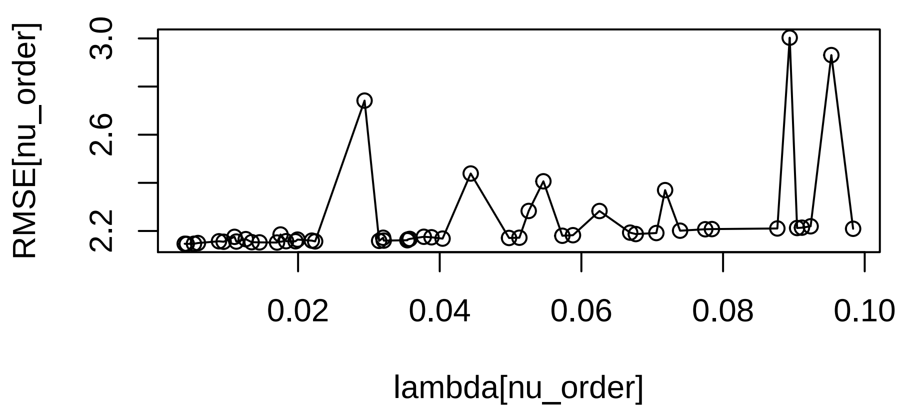
RMSE is around the same as the linear model.

```r
load( "G:/azure_hackathon/dataset/deriving_variables/enet.RData" )
enet_results[ which.min(RMSE) ]
```

```
##         lambda fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD
## 1: 0.004284722 0.842895 2.146332 0.8732509 1.112695 0.2079599 0.02530259
##          MAESD
## 1: 0.05412471
```

```r
par( mfrow = c(2, 1 ) )
enet_results[ , {
    plot( fraction, RMSE, type = "o" )
    nu_order = order(lambda)
    plot( lambda[nu_order], RMSE[nu_order], type = "o" )
} ]
```






```
## NULL
```

Partial least squares

PLS.

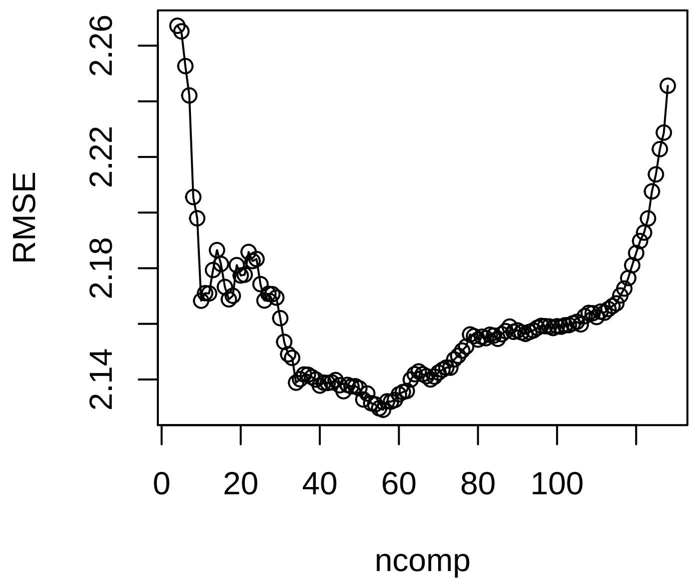
```
full_X = ncol( model.matrix( model_formula_lm, training_set) )
pls_tunegrid = data.frame( ncomp = 1:full_X )
pls_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "pls", trControl = train_control,
    tuneGrid = pls_tunegrid
    )

pls_results = data.table( pls_$results )
pls_pred = data.table( pls_$pred )
setorder( pls_pred, rowIndex )
```

```r
save( pls_, pls_results, pls_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/pls.RData" )
```

```r
load( "G:/azure_hackathon/dataset/deriving_variables/pls.RData" )
pls_results[ which.min(RMSE) ]
```

```
##    ncomp     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1:    56 2.129113 0.8751933 1.098537 0.2048704 0.02487309 0.04829641
```

```r
pls_results[ RMSE < 2.3, {
    plot( ncomp, RMSE, type = "o" )
} ]
```




```
## NULL
```

Principal components regression

PCR.

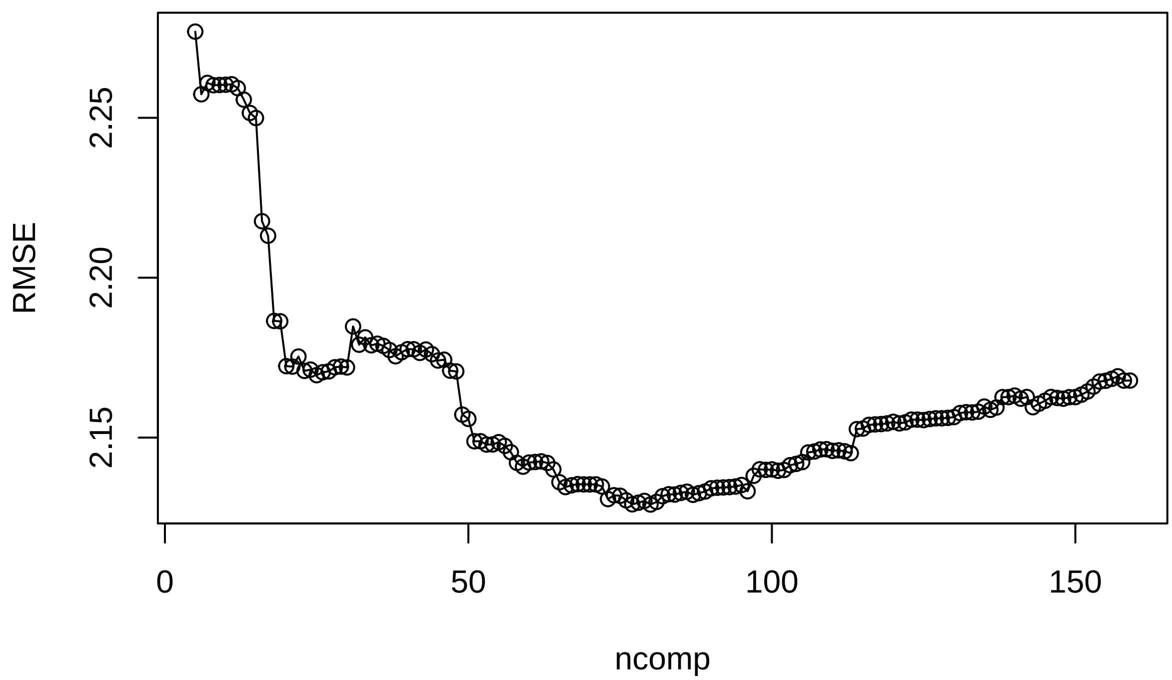
```r
full_X = ncol( model.matrix( model_formula_lm, training_set) )
pcr_tunegrid = data.frame( ncomp = 1:full_X )
pcr_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "pcr", trControl = train_control,
    tuneGrid = pcr_tunegrid
    )

pcr_results = data.table( pcr_$results )
pcr_pred = data.table( pcr_$pred )
```

```r
setorder( pcr_pred, rowIndex )

save( pcr_, pcr_results, pcr_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/pcr.RData" )
```

```r
load( "G:/azure_hackathon/dataset/deriving_variables/pcr.RData" )
pcr_results[ which.min(RMSE) ]
```

```
##    ncomp    RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1:    80 2.12907 0.8752164 1.101311 0.2039446 0.02471663 0.04866422
```

```r
pcr_results[ RMSE < 2.3, {
    plot( ncomp, RMSE, type = "o" )
} ]
```




```
## NULL
```

KNN

KNN.

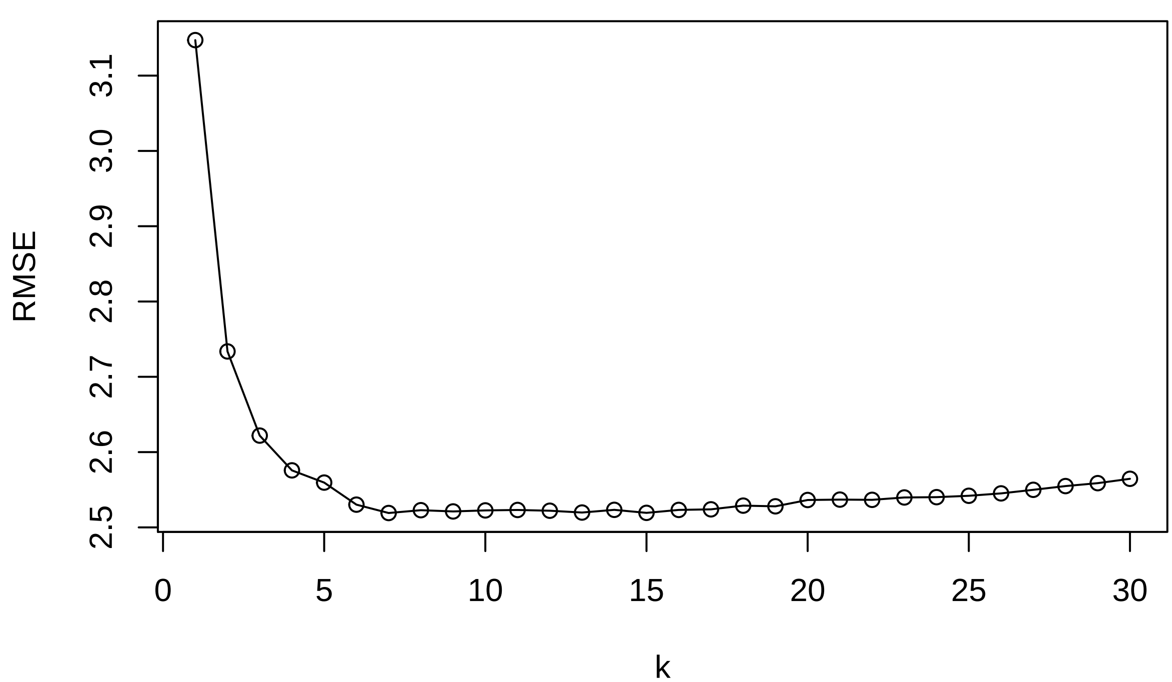
```r
k_grid = data.frame( k = 1:30 )

knn_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "knn", trControl = train_control,
    tuneGrid = k_grid
```

```
    )

knn_results = data.table( knn_$results )
knn_pred = data.table( knn_$pred )
setorder( knn_pred, rowIndex )

save( knn_, knn_results, knn_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/knn.RData" )
```

```
load( "G:/azure_hackathon/dataset/deriving_variables/knn.RData" )
knn_results[ which.min(RMSE) ]
```

```
##    k     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1: 7 2.519113 0.8265916 1.588707 0.1892186 0.02644601 0.07746616
```

```
knn_results[ , plot( k, RMSE, type = "o" ) ]
```




```
## NULL
```

CART

Use an Exponential distribution for our random search.

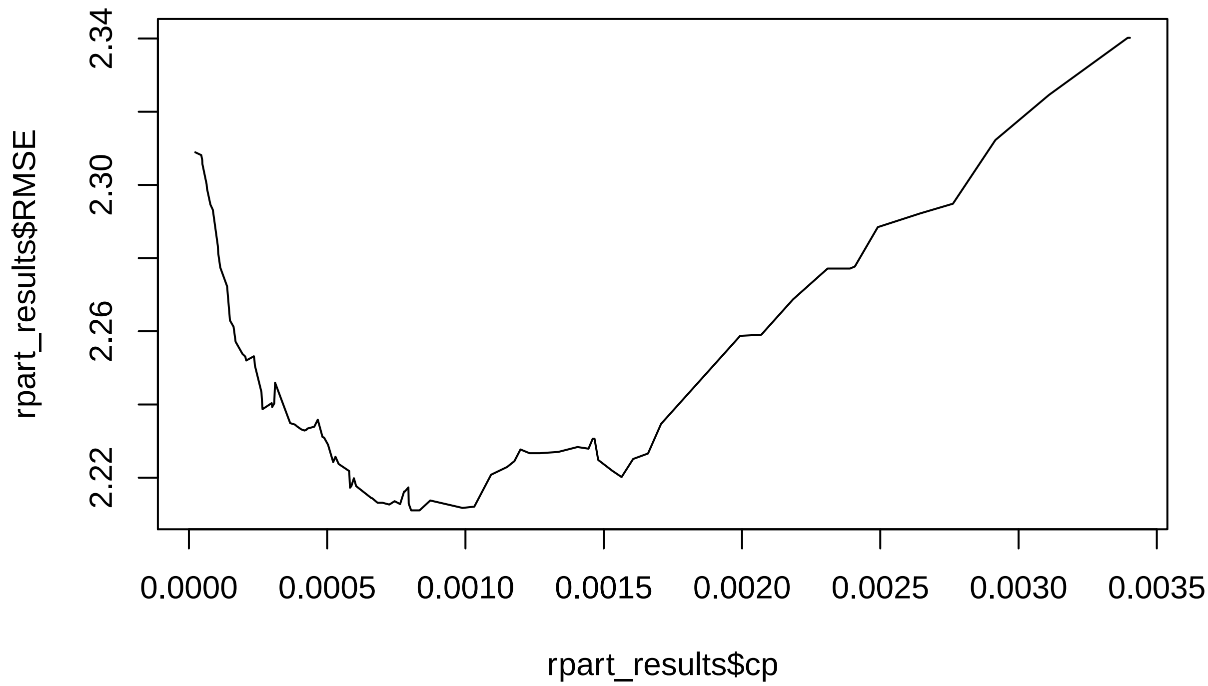
```
cp_grid = data.frame( cp = rexp( 100, 1/0.001 ) )

rpart_ = train( path_dist ~ .,
```

```r
    data = training_set,
    metric = "RMSE", method = "rpart", trControl = train_control,
    tuneGrid = cp_grid
    )

# Feed the values back
cp_grid = data.frame( cp = rexp( 100, 1/rpart_$bestTune$cp ) )

rpart_ = train( path_dist ~ .,
    data = training_set,
    metric = "RMSE", method = "rpart", trControl = train_control,
    tuneGrid = cp_grid
    )

rpart_results = data.table( rpart_$results )
rpart_pred = data.table( rpart_$pred )
setorder( rpart_pred, rowIndex )

save( rpart_, rpart_results, rpart_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/rpart.RData" )
```

```r
load( "G:/azure_hackathon/dataset/deriving_variables/rpart.RData" )
rpart_results[ which.min(RMSE) ]
```

```
##               cp     RMSE  Rsquared     MAE    RMSESD RsquaredSD      MAESD
## 1: 0.0008036135 2.211063 0.8652402 1.17163 0.2345295 0.02863216 0.08064738
```

```r
plot( rpart_results$cp, rpart_results$RMSE, type = "l" )
```



GAM splines

Generalised additive model using splines

```
gam_grid = expand.grid(
    select = F,
    method = c( "GACV.Cp", "REML", "ML" )
)

gam_ = train( path_dist ~ ., data = training_set,
    metric = "RMSE", method = "gam", trControl = train_control,
    tuneGrid = gam_grid
)

gam_results = data.table( gam_$results )
gam_pred = data.table( gam_$pred )
setorder( gam_pred, rowIndex )

save( gam_, gam_results, gam_pred,
    file = "G:/azure_hackathon/dataset/deriving_variables/gam_spline.RData" )
```

```
load( "G:/azure_hackathon/dataset/deriving_variables/gam_spline.RData" )
gam_results[ which.min(RMSE) ]
```

```
##    select method     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
```

```
## 1:  FALSE   REML 2.105365 0.8781029 1.088954 0.1846084 0.02130665 0.04584384
```

Stacking

```r
training_OOF = data.table(
    lm = lm_pred$pred,
    enet = enet_pred$pred,
    pls = pls_pred$pred,
    pcr = pcr_pred$pred,
    knn = knn_pred$pred,
    rpart = rpart_pred$pred,
    gam = gam_pred$pred,
    path_dist = training_set$path_dist
)

test_OOF = data.table(
    lm = predict( lm_, test_set ),
    enet = predict( enet_, test_set ),
    pls = predict( pls_, test_set ),
    pcr = predict( pcr_, test_set ),
    knn = predict( knn_, test_set ),
    rpart = predict( rpart_, test_set ),
    gam = predict( gam_, test_set ),
    path_dist = test_set$path_dist
)

stacking_model_formula = as.formula( "path_dist ~ ." )

stacked_tunegrid = data.frame( ncomp = 1:(ncol(training_OOF)-1) )
stacking = train( stacking_model_formula, data = training_OOF,
    metric = "RMSE", method = "pls", trControl = train_control,
    tuneGrid = stacked_tunegrid
    )

stacking_results = data.table( stacking$results )
stack_pred_OOF = predict( stacking, test_OOF )

save( stacking_results, stack_pred_OOF, test_OOF,
    file = "G:/azure_hackathon/dataset/deriving_variables/stack.RData" )
```

```r
load( "G:/azure_hackathon/dataset/deriving_variables/stack.RData" )

rmse_ = function( m ){
    p = predict( m, test_set )
    sqrt( mean( ( p - test_set$path_dist )^2 ))
}

stacked_rmse = sqrt( mean( ( stack_pred_OOF - test_OOF$path_dist )^2 ) )

all_rmse = rbind(
    as.matrix(sqrt( colMeans( ( test_OOF - test_OOF$path_dist )^2 ) )),
    stack = stacked_rmse
```

```r
)

all_rmse[ order(all_rmse), ]
```

```
## path_dist     stack       gam        lm       pcr       pls      enet     rpart
##  0.000000  2.279358  2.284920  2.400180  2.413419  2.415046  2.433728  2.562077
##       knn
##  2.695836
```

Stacking does a pretty good job.

Conclusion

Stack a bunch of models for imputing distance.