Deriving variables: Distance

Don Li

05/06/2020

Data and stuff

Load data. This contains test set, training set, and controls for 7-fold CV.

```
load( "G:/azure_hackathon/datasets2/model_subset/testing_subset.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.

```
head( training_set )
```

```
##    trj_id timediff crow_dist path_dist path_dist2 weekday hour rush_hour
## 1:  65500     1551  19.70113  23.97545   19.26083     Thu   21        No
## 2:  23471     1254  14.40836  17.50876   13.90154     Fri    1        No
## 3:  59010     1277  18.03334  26.49038   16.47832     Fri   22        No
## 4:  79901     1436  20.52864  25.29980   19.74366     Wed   13      Work
## 5:  45899      957  12.71138  15.95230   13.22123     Wed    9   Morning
## 6:  37069     1397  17.07647  21.69056   17.18856     Sun   15      Work
##     start_x  start_y    end_x    end_y sampling_rate sampling_rate_var
## 1: 1.288671 103.8199 1.350731 103.9857      1.307757      46.893391252
## 2: 1.273804 103.8428 1.344331 103.7343      1.009662       0.009576195
## 3: 1.302475 103.8542 1.439841 103.7684      1.039902       0.432799338
## 4: 1.445042 103.8289 1.342071 103.9820      1.124511       3.986836065
## 5: 1.320543 103.7771 1.322991 103.8913      1.174233       0.746018299
## 6: 1.343925 103.7069 1.297568 103.8532      1.348456       8.318074907
##    mean_speed    var_speed azure_dist OSRM_dist trip_start trip_end
## 1: 0.01720031 3.923431e-05     24.686  24.72538       C150     C110
## 2: 0.01398457 5.699054e-05     17.218  16.82265       C150  generic
## 3: 0.02105743 3.200899e-05     20.155  20.24935        C16  generic
```

```
## 4: 0.01956870 2.988724e-05     26.011  26.08958        C17     C110
## 5: 0.01927735 4.963859e-05     15.766  19.34844    generic  generic
## 6: 0.01845827 3.413946e-05     21.588  21.65940    generic      C16
```

Baseline

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

```
azure_baseline = sqrt( mean( (test_set$path_dist - test_set$azure_dist)^2 ) )
azure_baseline
```

```
## [1] 2.616091
```

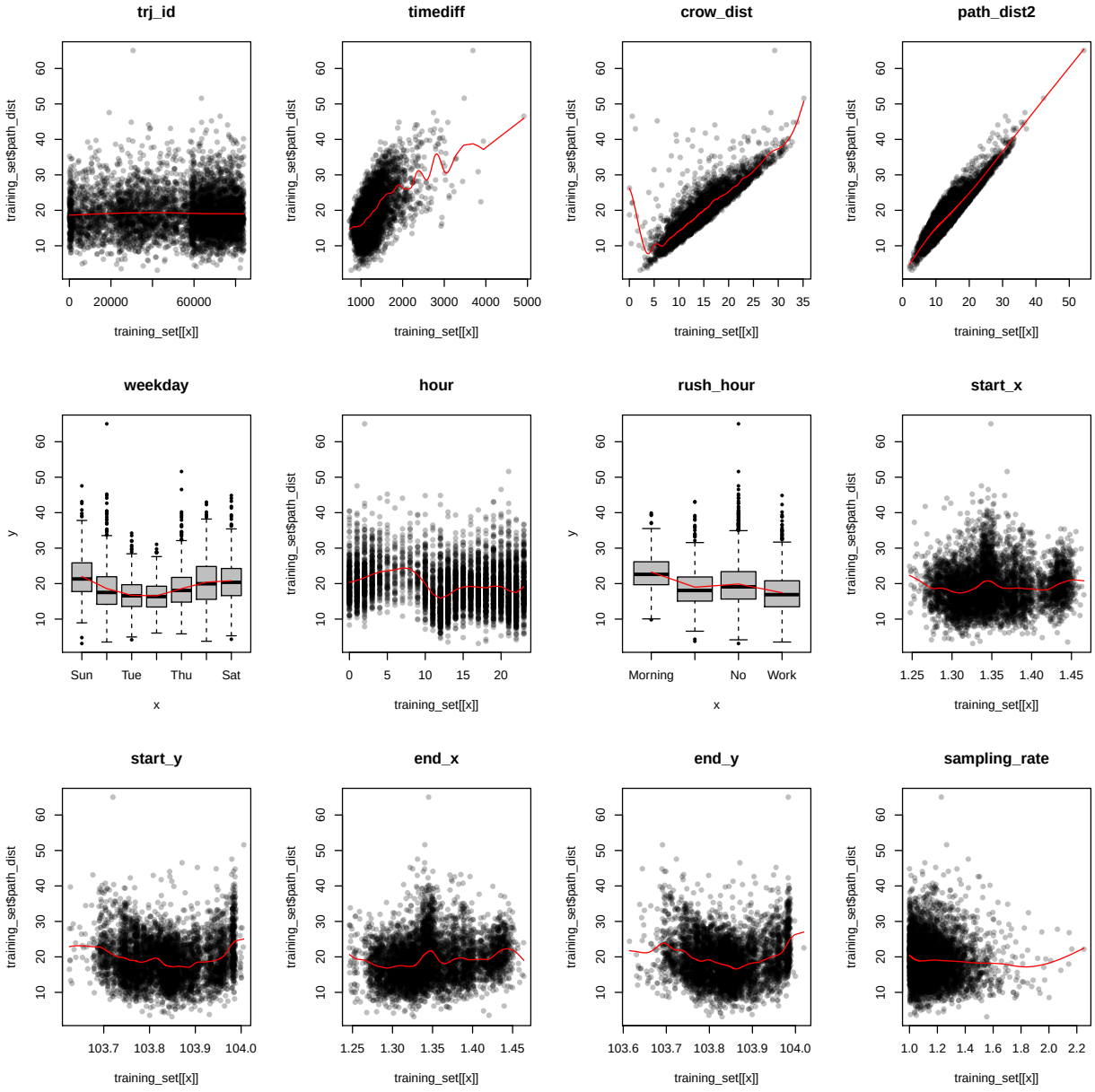
```
osrm_baseline = sqrt( mean( (test_set$path_dist - test_set$OSRM_dist)^2 ) )
osrm_baseline
```

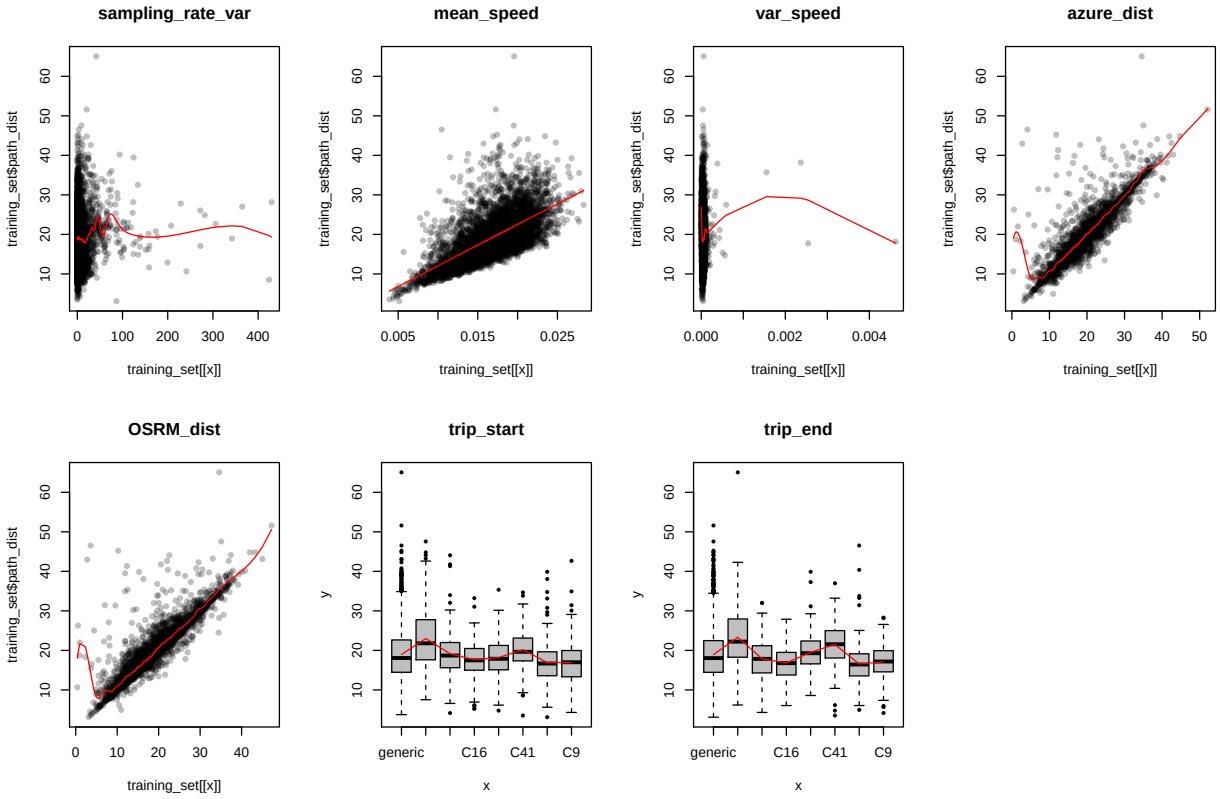
```
## [1] 2.753985
```

```
vars_to_rm = c("trj_id", "timediff", "path_dist2", "sampling_rate",
    "sampling_rate_var", "mean_speed", "var_speed" )
test_set_all = copy(test_set)
training_set = copy(training_set)
test_set[ , eval(vars_to_rm) := NULL ]
training_set[ , eval(vars_to_rm) := NULL ]
```

Linear regression

```
xvars = setdiff( names(training_set), "path_dist" )
par( mfrow = c(3, 4) )
col = rgb( 0, 0, 0, 0.25 )
for ( x in xvars ){
    plot( training_set[[x]], training_set$path_dist,
        main = x, col = col, pch = 16 )
    lines( smooth.spline( training_set[[x]], training_set$path_dist ),
        col = "red")
}
```


trj_id
timediff
crow_dist
path_dist2
weekday
hour
rush_hour
start_x
start_y
end_x
end_y
sampling_rate
training_set$path_dist
training_set[[x]]
x
y
Sun
Tue
Thu
Sat
Morning
No
Work



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

```
model_formula_lm = path_dist ~ .*. +
    I(start_x^2) + I(start_y^2) + I(end_x^2) +  I(end_y^2) +
    I(hour^2) + I(hour^3)  -
    start_x:end_x - start_x:start_y - start_x:end_y -
    start_y:end_x - start_y:start_y - start_y:end_y -
    trip_start*. - trip_end*.

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/datasets2/expo/lm.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/lm.RData" )
lm_results
```

```
##    intercept     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1:      TRUE 2.227735 0.8653754 1.171379 0.3524694 0.03559297 0.0680175
```

CV error is 2.228. Better than out baselines.

Elastic net

Elastic net.

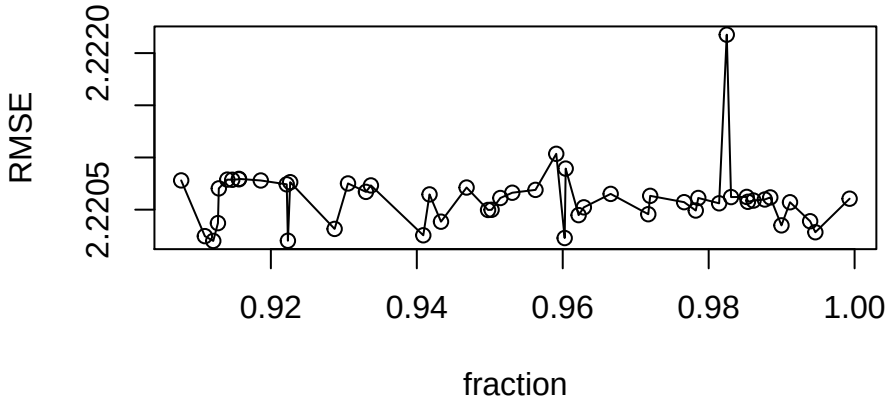
```
n_enet = 50
enet_tunegrid = data.frame(
    lambda = rexp( n_enet, 1/0.0001 ),
    fraction = runif( n_enet, 0.9, 1 )
)
enet_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "enet", trControl = train_control,
    tuneGrid = enet_tunegrid, standardize = TRUE, intercept = FALSE
    )

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

save( enet_, enet_results, enet_pred,
    file = "G:/azure_hackathon/datasets2/expo/enet.RData" )
```

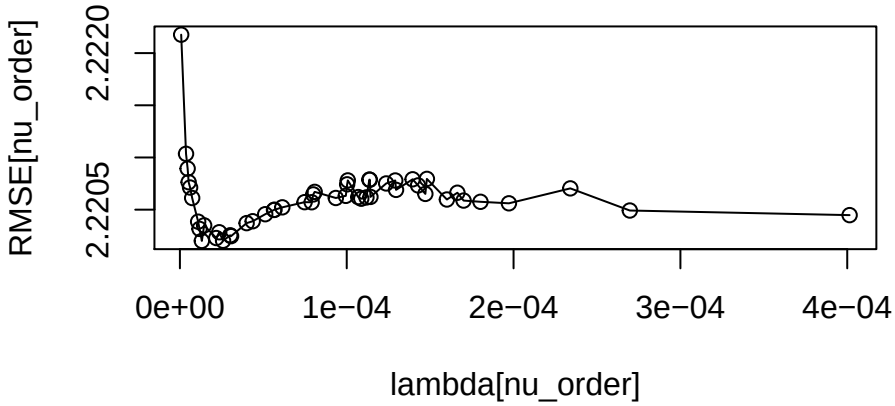
RMSE is around the same as the linear model.

```
load( "G:/azure_hackathon/datasets2/expo/enet.RData" )
enet_results[ which.min(RMSE) ]
```

```
##          lambda  fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD
## 1: 1.314809e-05 0.9121048 2.220201 0.8662129 1.159179 0.3597018 0.03639564
##         MAESD
## 1: 0.07006056
```

```
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.

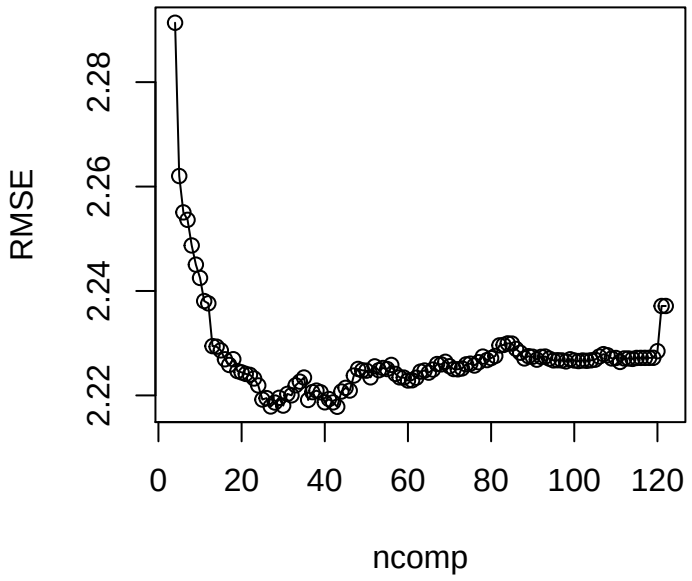
```
full_X = ncol( model.matrix( model_formula_lm, training_set) )
pls_tunegrid = data.frame( ncomp = 1:full_X )
pls_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "pls", trControl = train_control,
    tuneGrid = pls_tunegrid
    )

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

```
save( pls_, pls_results, pls_pred,
    file = "G:/azure_hackathon/datasets2/expo/pls.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/pls.RData" )
pls_results[ which.min(RMSE) ]
```

```
##    ncomp     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1:    43 2.217823 0.8663979 1.157298 0.3675327 0.03725146 0.07493254
```

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




```
## NULL
```

Principal components regression

PCR.

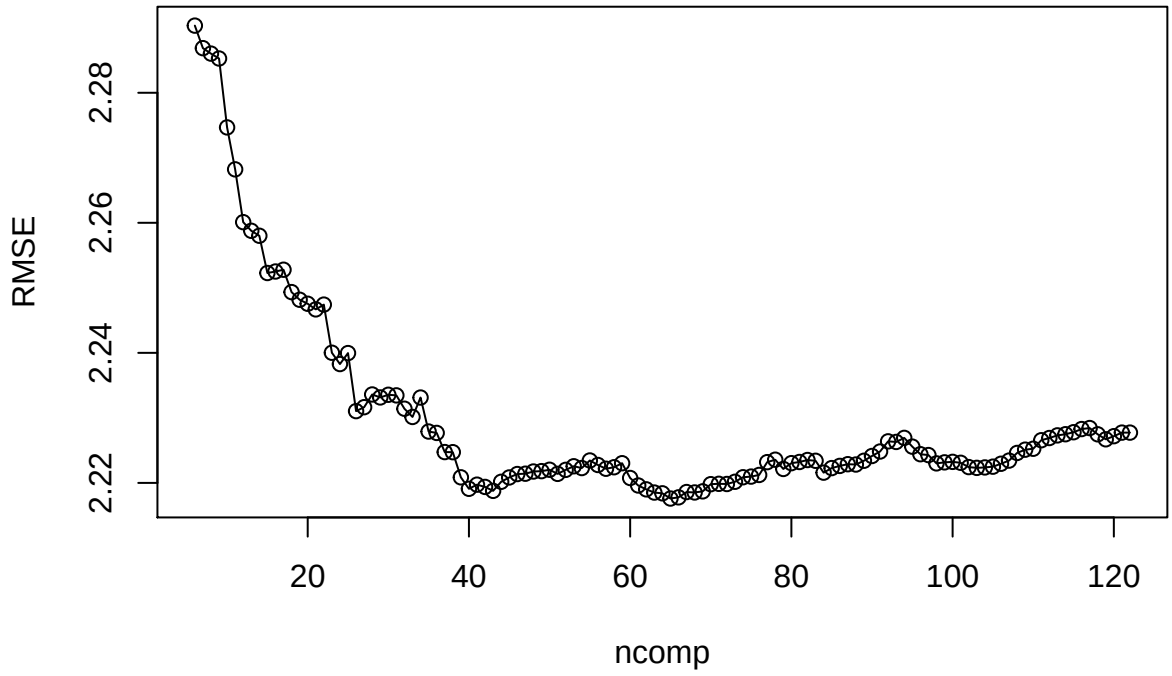
```
full_X = ncol( model.matrix( model_formula_lm, training_set) )
pcr_tunegrid = data.frame( ncomp = 1:full_X )
pcr_ = train( form = model_formula_lm, data = training_set,
    metric = "RMSE", method = "pcr", trControl = train_control,
    tuneGrid = pcr_tunegrid
    )

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

```
setorder( pcr_pred, rowIndex )

save( pcr_, pcr_results, pcr_pred,
    file = "G:/azure_hackathon/datasets2/expo/pcr.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/pcr.RData" )
pcr_results[ which.min(RMSE) ]
```

```
##    ncomp     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1:    65 2.217599 0.8664112 1.156403 0.3693347 0.03748396 0.07471209
```

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




```
## NULL
```

KNN

KNN.

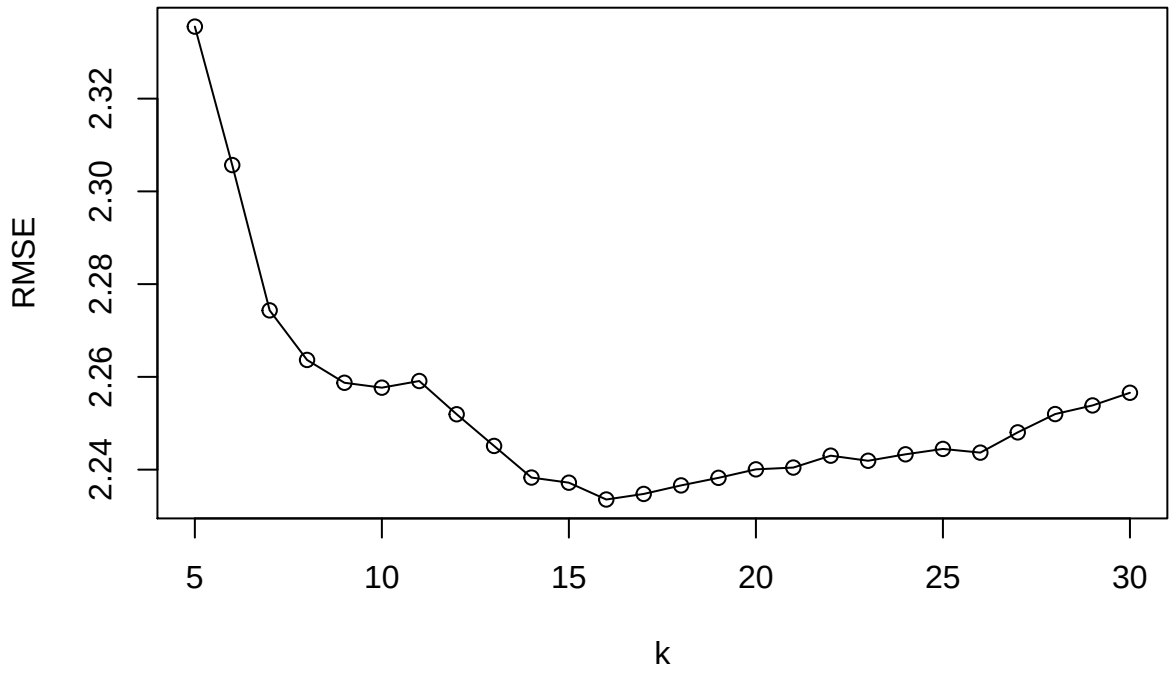
```
k_grid = data.frame( k = 5:30 )

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/datasets2/expo/knn.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/knn.RData" )
knn_results[ which.min(RMSE) ]
```

```
##     k    RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1: 16 2.23356 0.8651132 1.129654 0.3492335 0.03454983 0.07882709
```

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




```
## NULL
```

CART

Use an Exponential distribution for our random search.

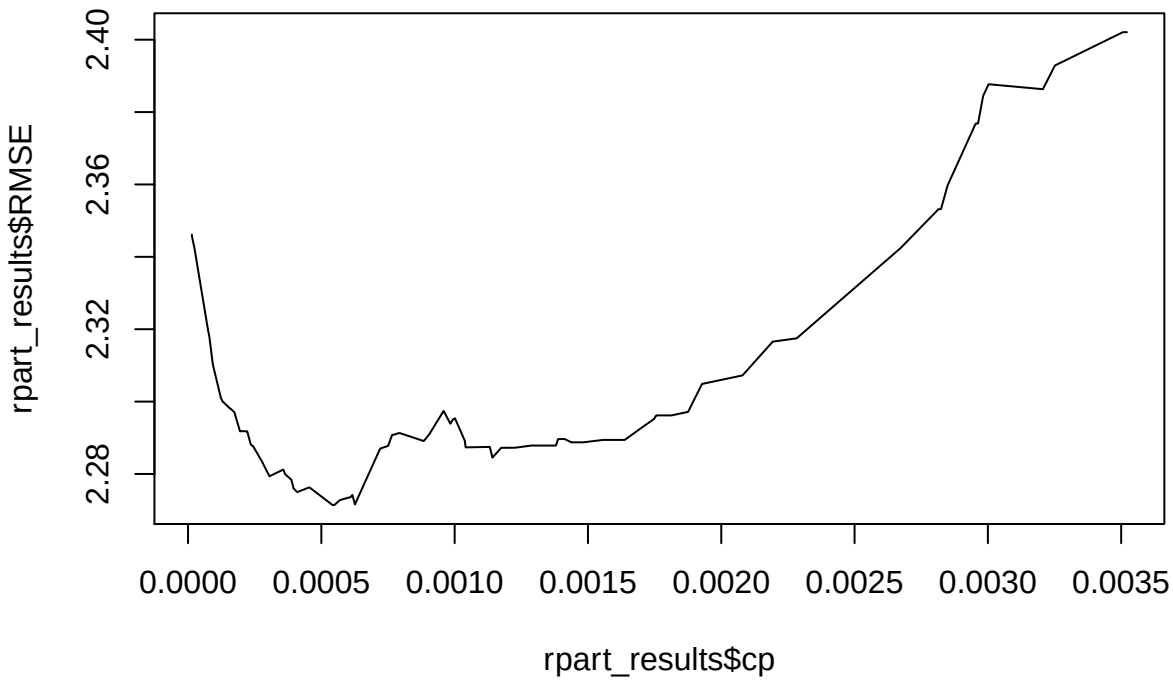
```
cp_grid = data.frame( cp = rexp( 100, 1/0.001 ) )

rpart_ = train( path_dist ~ .,
```

```
    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/datasets2/expo/rpart.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/rpart.RData" )
rpart_results[ which.min(RMSE) ]
```

```
##              cp     RMSE  Rsquared      MAE   RMSESD RsquaredSD      MAESD
## 1: 0.0005422336 2.271404 0.8599859 1.175333 0.333959 0.03272627 0.08490925
```

```
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 ~ . +
            I(start_x^2) + I(start_y^2) + I(end_x^2) +  I(end_y^2) +
    I(hour^2) + I(hour^3)
    , 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/datasets2/expo/gam_spline.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/gam_spline.RData" )
gam_results[ which.min(RMSE) ]
```

```
##    select method     RMSE Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1:  FALSE     ML 2.152747 0.873746 1.089102 0.3488632 0.03389655 0.07369293
```

Stacking

```
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
    # training_set
)

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
    # test_set
)
```

```
stacking_model_formula = path_dist ~ .

stacked_tunegrid = data.frame( ncomp = 1:(ncol(training_OOF)-1) )
stacking = train( stacking_model_formula, data = training_OOF,
    metric = "RMSE", method = "pcr", 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/datasets2/expo/stack.RData" )
```

```
load( "G:/azure_hackathon/datasets2/expo/stack.RData" )


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
)

(all_rmse[ order(all_rmse), ][-1])
```

```
##      gam    stack      pcr      pls      knn     enet       lm    rpart
## 2.280753 2.305106 2.387824 2.391231 2.398938 2.403394 2.410252 2.514303
```

Stacking does a pretty good job. But GAMs are pretty good, but I think this is just a strange sample. I tried with other samples in the dataset and the stack was better for them.

Conclusion

Stack a bunch of models for imputing distance.

Recall that I have a bad distance where I swap the longitude and the latitudes in the calculation. I don't know why, but it ends up being quite good in the model. I suspect it is like a PCA kind of effect.

To impute the proper Haversine, I will impute the terrible Haversine first:

- Impute the Haversine2 distance (longs and lats swapped).
- Use the Haversine2 with the other variables to impute the Haversine (longs and lats right way).