Required sequential steps to generate a submission using the provided files

The following steps should allow you to generate a score which is very similar to the winning submission of the "Facebook V: Predicting Check Ins" competition. The results will not be exactly identical since I did not set my seeds in the xgboost models.

Disclaimer: This stepwise approach was not tested so please inform me of any issues on Kaggle forums. Make sure to select a maximum batch size that your system can handle. I would aim at 3000/48*(available RAM in GB) as a rule of thumb. Generating the features will take a long time so start with a couple of train, validation and test batches!

General

Replace the working directory paths (always at the top of the scripts)

Replace other folder paths. The "targetDate" of the data files refers to the subfolder of the different iterations. I left this at "23-05-2016" but you could add iterations

There are many options in the script at the top of the file, feel free to explore their impact!

Data

Paste the train and test csvs into the Data folder

Run rawToRds to convert the raw data into rds format

Downsampling

This part of the logic splits the train, validation and test data into batches of a given maximum size.

- ⇨ Adjust maxSample at your convenience
- ⇨ Run
  - downsamplerTimeRandomLocTrain
  - downsamplerTimeRandomLocTest

Overlapping grid generation

Nearest neighbors are calculated by considering neighbors in overlapping rectangular x-y grids. Different x-y ratios are used for the different distance constants.

Run blockGenerator in the data folder for train and test and for the relevant distance constant grid constants (distanceConstantX). I would suggest to start with a single distance constant of 2.5 and proceed from there.

Nearest neighbor calculation

Install the fastOrder rcpp package from the fastOrder .tar

Run candidateSel Tier I NN topCalc in the candidate selection folder for the desired batch type and id range with the desired distance constant(s) – **SLOW!**

Run midKnnWrapper in the candidate selection folder for the desired batch type and id range with the desired distance constant(s) – **BOTTLENECK!**

Other feature generation

Run createAccuracySummaryFeatures for train and test

Run createRegionDensityFeatures for train and test

Run createWeeklySummaryFeatures for train and test

Run weeklyDensityExtraction for train and test

Run weeklyDensityForecast for train and test

Create the train and test place summary features by running createFeaturesSummary

Now you can combine the place summary features and the KNN summary features by running combineNeighborSummaryFeatures for the considered train and validation batches. The second candidate selection model is built using train batches with all observations so topM (nb considered candidates after feature generation) should be set >= topN (nb first considered candidates – default of 100).

Second candidate selection generation

Run xgboost under Simple xgboost blend of the first level learners for the desired batches

Validate the model performance using xgboostValidationAvgBlend

Copy the models to the Top M models under the date folder of the considered iteration

Feature generation for first level learners

Set top M to the desired number of candidates after the second candidate selection (I used 20)

Run combineNeighborSummaryFeatures for the considered train, validation and test batches. It is advised to use different train and validation batches compared to the second candidate selection.

Learning of first level learners

Run featureRanking under Simple xgboost blend of the first level learners to generate the feature importance order. Next, paste the ordered features to the top2XX predictors in the manualFeatureImportanceRank script. Adjust XX based on the obtained top predictors count

Run the manualFeatureImportanceRank script from the first level learners folder to order the features after manual corrections.

mainLearnerLogic is used next to generate the base models. Hyperparameters can be set easily. I used fixed values for all hyperparameters except the eta constant, the number of features and the number of considered candidates of the base learners. Setting the number of considered candidates to a value greater than the number of available candidates for that train batch will not have any effect.

Individual learners can be validated using the validate functions of the considered model type (I only used xgboost but the ensembler can be easily extended).

Copy the preferred models for blending to the “Selected models for blending” folders for each considered base learner.

Fit the predictions of the base learners to train, validation and test batches using combineModelPreds under First level learners.

Learning of second level learners

Run xgboost for the considered second level train batches under Second level learners

Validate the models using xgboostValidationAvgBlend

Copy the selected second level models to the test prediction folder

Submission generation

Run xgboost_secondTier_combined under 06-07-2016 of the submission folder.

- Update the predicted test batches
- Set the submission file name “submissionFile”

That’s all, now you can submit your solution to Kaggle ☺