This document describes the high level steps to get from the raw input data to a submission file

Preprocessing of raw data

Both the training and test data sets are chopped into equally sized **random location** chunks of at most 30,000 observations in order to manage the processing (time + memory) complexity. The batch size is still large enough to learn complex patterns but small enough to fit into memory. The concept of random batches is crucial. It was verified from the exploratory analysis that there is no obvious spatial pattern in the distribution of the check ins or accuracy of the check-ins. Learning a single overall model of the check-in probability using a complex blend of a large number of overfitted models is my secret sauce. Location based features such as the local density of historical check ins were added at a later stage but did not improve the predictions significantly. This agrees with the important assumption that there is no clear spatial pattern. All major approaches discussed on the forums fit rectangular location based models. This approach was also considered but less advanced patterns can be learnt using this approach since it requires the models to be fit using a small subset of the overall data. My approach likely introduces (limited) bias by using all the data in the overall model generation but has far less variance. It seems like the reduction in variation outweighs the introduction of bias.

The training data is split into a summary and a training period where the training period comes after the summary period. The size of the training period was set at 70% of the train data, this is similar to the ratio of the test data to the train data.

Chronological outline of the chunks:

Summary period	Training period	Test period
	Train batch	Test batch
	Validation batch	Test batch
	Train batch	Test batch
	Validation batch	Test batch

Time

Cutoff points:

- Summary period: 0-587158
- Training period: 587158-786239
- Test period: 786242 - 1006589

Batch count:

- 219 train batches
- 73 validation batches
- 287 test batches

The summary period is used to generate features for train and validation batches in the construction phase of the desired approach.

The preferred classification approach is selected by assessing the predictive performance of the validation set.

The test data is predicted by calculating summary features on the summary + Training (train + validation) period. Features with the same relative interpretation as the learning phase are then used to generate predictions using the preferred classification approach.

Candidate selection 1: from 100,000 to 100 candidates – Top N candidate selection

The complexity of the high number of classes is handled by using a staged approach. Features are only calculated for the potential places that belong to the top N of the staged approach. The reduction of possible classes is referred to as candidate selection.

The number of considered candidates for each new observation in the training and test period is set to 100. An arbitrary choice that includes all serious candidates. Places are ranked based on their 2500 NN count where variation in the Y direction is 2.5 times as important as variation in the x-direction. This can be visualized as an ellipse where the x axis length is 2.5 times the length of the y-axis. The mean time difference since the observations for each place id is used to resolve ties.

Calculating the nearest neighbors efficiently is enforced by considering neighbors block by block where the blocks are overlapping. All blocks are equally sized rectangles and every next block starts at the middle of the previous block. That way the maximum distance to the edge of the block is at least blockDim/4 for each dimension. Assuming (this is approximately true) a constant x-y density, the number of blocks should be at most 2e7/2500*pi/4 ~ **6000** if the ratio of the dimensions is equal to the relative distance measure in order to have at least 2500 neighbors in the considered block. Using a safety margin of 25%, the maximum numbers of blocks was fixed at at 4800

This is enforced using:

- 49(X)*49(Y) = 2401 0.4 by 0.4 blocks when the relative x distance constant is set to **1**
- 49(X)*49(Y) = 2401 0.4 by 0.4 blocks when the relative x distance constant is set to **1.75**
- 39(X)*99(Y) = 3861 0.5 by 0.2 blocks when the relative x distance constant is set to **2.5**
- 19(X)*99(Y) = 1881 1 by 0.2 blocks when the relative x distance constant is set to **4**
- 19(X)*99(Y) = 1881 1 by 0.2 blocks when the relative x distance constant is set to **5.5**
- 24(X)*199(Y) = 4776 0.8 by 0.1 blocks when the relative x distance constant is set to **7**
- 19(X)*249(Y) = 4731 1 by 0.08 blocks when the relative x distance constant is set to **12**
- 9(X)*399(Y) = 3591 2 by 0.05 blocks when the relative x distance constant is set to **30**

The neighbor finding logic is the bottleneck of the prediction process and is optimized using parallel execution and a custom C implementation of the main bottleneck (ordering of a large numeric vector).

Feature generation for top N (100) candidates

NN features

- The K nearest neighbor counts are used as features where K = c(2500,1000,500,250,100,50,20,10,5,1) and the distance constants are set to 1, 2.5, 4, 5.5, 7, 12 and 30. The distance constant means that the distance in the X direction is distanceConstant times **less** important compared to the distance from a new observation in the Y direction. The rescaled Euclidian distance is used as the distance measure.
  ⇨ 7*10=70 features
- The relative mean time differences (relative wrt the size of the summary period) of all K-neighbor counts are also used as features
  ⇨ 7*10=70 features

- *Features that interpolate the KNN counts using the relaxed madX/relaxed mad Y ratio => 10 features. This does not seem to add much value. This makes sense given the exploratory analysis: it's hard to fit the distribution using a parametric model!*
- *10 features that indicate the distance from a new observations to the Kth closest neighbors. This is again rescaled based on the total raw data count in order to obtain features with similar interpretation in the test set.*
- A predictor that indicates the distance to the x and y borders at both edges.
- Ratio of mads of X/Y. Mad ratio relaxed wrt historical count and log of the relaxed mad ratio.

Time features

- Density of hour and day of week (also relaxation derived features). The smoothed (other days and hours) features are also included. Relaxed densities based on the historical counts were also considered. Going deeper than hour seems to add no value. There might be an interaction between day of week and hour so additional features were also calculated for hourDay (0-167).
- Time (in minutes) since the end of the summary period and Log of 3 + days since the end of the summary period (in days). These predictors were considered dangerous with respect to confounding but resulted in a better public leaderboard score. Time since the first check-in and a logical flag if this time is at least 52 weeks.
- Time between the end of the summary period and the most recent check-in to a location.
- Relative frequency of 27 sequential 2-week period check-ins during the summary period. Relative frequencies are preferred rather than actual counts since the number of check-ins varies with time (two major dips). The total relative count and the relative number of active day counts were also added as features.
- Relative frequency of going 2 week periods back in time (77-1 weeks in steps of two weeks). Missing values are added in this step so these predictors are ignored by some of the downstream learners. Relative frequency of going 52 weeks back in time (both exactly 52 weeks, Gaussian smoothing and uniform smoothing)
- Feature that captures total week density (extrapolate last week)!

Accuracy features

There is less variation in check-in locations for accuracy measurement in the 45-85 range. The accuracy was split into both 3 and 32 groups of similar observation count sizes.

- Three binary features indicating if a new observation belongs to a certain accuracy group
- Modified x and y Z scores, conditional on the accuracy group variation
- Relative density of the accuracy groups (also relaxed features)
- TODO: find pattern in the three accuracy peaks – Scipio post?

Z score features

- Robust Z scores (observation – median of historical distribution of place observations)/ (mad of historical distribution of place observations) for
  - ⇨ Relative accuracy (accuracy varies wrt time), relaxed wrt the historical observation count
  - ⇨ Log of relative relaxed accuracy
  - ⇨ X, log of xZ and log of xZ squared (same for Y)

Dropped predictors

- *Hour seems to interacts with day, maybe a "fortnight" cycle as well? Doesn't seem like it*

- Going deeper than hour seems to add no value
- Add feature that counts the number of non zero counts in certain time windows
- Block region recent activity feature – maybe some places are more popular when the region is popular?
- *Maybe some places are more popular at popular times - ??*
- *Optional: Calculate quantile feature of accuracy in week?!*
- Try to find a pattern in the 52 or 53 week periods – Not successful
- Investigate extreme daily popularity analysis – spatial relation between 52 and 53 week locations?? Not obviously

```
predictors <- c(paste0("k", 1:(nbKs*nbDifferentKCounts))
            , paste0("kInt", 1:nbKs)
            , paste0("meanTimeDiff", 1:(nbKs*nbDifferentKCounts))
            , paste0("neighborDistance", 1:(nbKs*nbDifferentKCounts))
            , "timeSinceSummary" # DANGEROUS predictor wrt confounding!
            , "logTimeSinceSummaryDays"
            , "noCheckTimeSumEnd"
            , "relativeAccuracyZ", "logRelativeAccuracyZ"
            , "relativeRelaxedAccuracyZ", "logRelativeRelaxedAccuracyZ"
            , "relativeCount", "nonZeroDailyRelativeCount"
            # , "count", "nonZeroDailyCount"
            # , "countLast175D", "countLast70D", "countLast35D"
            , "hourDens", "hourDensR", "smoothHourDens", "smoothHourDensR"
            , "dayDens", "dayDensR", "smoothDayDens", "smoothDayDensR"
            , "hourDayDens", "hourDayDensR", "smoothHourDayDens",
            "smoothHourDayDensR"
            , "xZ", "xZLog", "xZLog2", "yZ", "yZLog", "yZLog2"
            , "relaxedMadRatio", "logRelaxedMadRatio"
            , paste0("periodDensity", 0:27)
            , paste0("weekBackDensity", seq(75,1,-2))
            , "weeklyCount"
            , "distanceXBorder", "distanceYBorder"
            , "distanceXLowBorder", "distanceXHighBorder"
            , "distanceYLowBorder", "distanceYHighBorder"
            , "timeOpen", "isAtLeastOpenYear"
            , "dayDens52WeekBack", "smoothGDayDens52WeekBack"
```

```
    , "smoothUDayDens52WeekBack"

    , "dayDens53WeekBack", "smoothGDayDens53WeekBack"

    # , "smoothUDayDens53WeekBack"

    , "accuracy", "isAcGroup1", "isAcGroup2", "isAcGroup3"

    , "xZAcGroup", "yZAcGroup"

    , "acGroup3Density", "acGroup3DensityR"

    , "acGroup32Density", "acGroup32DensityR"

    , "regionDensity", "smoothRegionDensity"

)
```

Target measure

The problem is approached as a **binary classification problem**: predict the probability that an observation belongs to a true place_id. Optimizing the quality of the model will inherently result in a higher performance of the target measure (mean average precision)

Learning of second candidate selection model using features of top N candidates

At this point, a simple blend of five XGBoost models is used as the learning model.

This will become a smart blend of models in the final submission

Use validation data to perform feature selection and tune model hyperparameters

1) A first set of validation batches (1-4) was used to make an initial assessment of appropriate hyperparameters.

Found to work well:

- ⇨ 100 trees
- ⇨ Learning rate of 10/100
- ⇨ Tree depth of 8
- ⇨ Column and row subsamples of 0.5

2) Feature selection using a second set of validation batches (5-8):
   - ⇨ Start with all features
   - ⇨ Fit xgboost model
   - ⇨ Drop least important feature
   - ⇨ Keep track of all validation scores

Pick the feature set at the highest validation score + add two features since this process is prone to overfitting

It seems like **all retained features** contain valuable information so none were dropped.

3) Grid search on the most important hyperparameters in order to come up with a final model.

Second candidate selection: from Top N to Top M where M<<N

Use a single xgboost model to make this second selection. This greatly reduces the storage space of the features and speeds up further learning stages. Initial value for M: 20

Second round of model building using at most M candidates for each data point

Found to work well: high number of trees (400), fast learning rate (10/400), deep trees (>=10)

Use preferred model (eventually a blend of models) to generate predictions

Take high max depth of trees – blends excellently The overfitting gets compensated by the blending

Calculate the probabilities for all top 100 candidates and use the order of the probabilities in order to come up with top 3 predictions.

TODO

Use accuracy to calculate better location centers and use these better location centers in Z score calculations. Attention: sometimes different location centers for different accuracy groups!? – *Captured by group specific Z scores*! – Done

Other relaxed Z scores were also added as additional features

Decide on 366, 365 or 364 days back (or all ☺)

Create features that indicate the 52 and 53-week periodicity strengths – Correlation between 52 and 53 week counts for places with count periods of > 52/53 weeks, otherwise NA - relaxed according to the number of non-zero weekly counts

Create time series models for extrapolated weekly counts

Create additional time series profile indicators (peakedness/frequency)

Low priority

KNN with distance constant of 1.75

Optional: add a feature that decides between a 52 and 53 weeks pattern or match the right previous year count.