Predicting the availability of bicycles at rental stations using statistical modelling

Frederica Caira, Maria Marinova

December 2015

This paper concerns a city bikeshare system in Valencia, Spain. Our aim was to predict the number of available bicycles in all rental stations 3 hours in advance. These predictions could be used by both 1. a bike user, who plans to rent or return a bike in 3 hours' time and needs a bike station which is not empty or full, and 2. the bike company, who needs to know which stations are more likely to be empty or full soon so they can avoid moving bikes between stations.

The performance of our predictions are assessed using Kaggle test data unavailable to us. There is however a leaderboard to which we may submit a prediction twice a day, which returns the mean absolute error of our result (based on comparison with 66% of the test data set), giving us an indication of our progress.

Contents

N.B. All values are rounded to 2 decimal places.

1 Training Data and Preprocessing

We were provided with datasets detailing 75 rental stations, each set containing the same types of features. We examined this data and performed preprocessing, to determine the best features to use in our model.

1.1 Training Data Fields

The data contains:

- **4 station features**
  - *station* — integer from 1 to 275 representing the station number (also in the filename)
  - *latitude* — real number representing station's geographical latitude
  - *longitude* — real number representing station's geographical longitude
  - *numDocks* — positive integer representing the maximal number of bikes the station can hold
- **8 time features**
  - *timestamp* — integer representing the Unix timestamp (seconds since Unix Epoch)
  - *year* — integer with 4 digits representing year format YYYY
  - *month* — integer from 1 (January) to 12 (December)
  - *day* — integer from 1 to 31 representing day of the month
  - *hour* — integer from 0 to 23 representing hour of the day
  - *weekday* — string representing day of the week (from 'Monday' to 'Sunday')
  - *weekhour* — integer from 1 to 168 representing the hour of the week (Monday 0h is weekhour 1, Sunday 23h is weekhour 168)
  - *isHoliday* — 1 represents a national or local holiday, 0 represents *not* a holiday
- **7 weather features**
  - *windMaxSpeed.m.s* — real number representing maximum wind speed in metres per second
  - *windMeanSpeed.m.s* — real number representing mean average wind speed in metres per second
  - *windDirection.grades* — real number representing wind direction in degrees
  - *temperature.C* — real number representing temperature in degrees celsius
  - *relHumidity.HR* — integer representing relative humidity in
  - *airPressure.mb* — real number representing air pressure in millibars
  - *precipitation.l.m2* — integer representing precipitation in litres per metre^2
- **1 task-specific feature**
  - *bikes_3h_ago* — integer representing the number of bikes in the station 3 hours ago
- **4 profile features**
  The profile variables are calculated from earlier available timepoints on the same station.
  - *full_profile_bikes* — real number representing the arithmetic average of the target variable *bikes* during all past timepoints with the same weekhour, in the same station.
  - *full_profile_3h_diff_bikes* — real number representing the arithmetic average of the difference between *bikes* and *bikes_3h_ago* during all past timepoints with the same weekhour, in the same station.
  - *short_* — the same as the *full_* profiles except they only use the past 4 timepoints within the same weekhour. If there are less than 4 such timepoints then all are used. The missing values are ignored in all profile calculations, i.e. only the timepoints with existing values are averaged.
- **1 target variable**
  - *bikes* — non-negative integer representing the median number of available bikes during the respective hour in the respective rental station

1.2 Preprocessing Methods

We chose to use Matlab for the duration of this project as it provides a straightforward way of importing data from .csv files into matrices.

At first glance, the most obviously useful features seemed to be *bikes_3h_ago*, and the four profile features. We reasoned that the weather features were also potentially useful because weather usually affects one's decision to hire a bike — e.g. one may consider cycling in wind and rain unpleasant or even dangerous and therefore refrain from renting a bike.

A number of preprocessing steps were necessary before we were able to evaluate our predictions. In order to import the .csv files to a Matlab matrix and perform further calculations, we needed to resolve features containing string types. We converted numeric strings to the number type, and replaced non-numeric strings ('N/A' - representing missing data) with NaN (not a number).

1.2.1 Enumerating String Data

We wrote a script to change each string in *weekday* to the corresponding integer from 1 to 7 (Monday is 1, Tuesday is 2, etc.).

1.2.2 Removing 'NaN' Rows

The dataset contained some missing cell values, so in order to operate on it properly once we'd determined which features we wanted to use, we removed any row containing a NaN cell.

2 Phase One

2.1 Feature Selection

We first approached feature selection by noting the variation of the features both within stations and across stations (Figure 1).

2.1.1 Excluding Features with No Variance

Features *year*, *month*, and *precipitation.l.m2* are unchanging both within and across stations. While *precipitation.l.m2* initially seemed to be a useful feature, it is '0' throughout the entire training set (i.e. it did not rain throughout October 2014), so we removed it.

The entire training data set is from one month (October) in one year (2014), so we also removed *month* and *year*. A correlation coefficient between two matrices (column vectors in our case) cannot be calculated if either of them have no variance. Therefore we could not calculate a correlation coefficient between *bikes* and any of the above three features.

We considered the features which varied within stations (e.g. *weekday*) for use in our individual models, and those which varied across stations (e.g. *numDocks* for use in our general model. Those that varied both within and across stations (e.g. *bikes_3h_ago*) were considered for use in both models.

All Features	Variation within stations	Variation across stations
station	same	diff
latitude	same	diff
longitude	same	diff
numDocks	same	diff
timestamp	diff	same
year	same	same
month	same	same
day	diff	same
hour	diff	same
weekday	diff	same
weekhour	diff	diff
isHoliday	diff	same
windMaxSpeedms	diff	same
windMeanSpeedms	diff	same
windDirectiongrades	diff	same
temperatureC	diff	same
relHumidityHR	diff	same
airPressuremb	diff	same
precipitation.l.m2	same	same
bikes_3h_ago	diff	diff
full_profile_3h_diff_bikes	diff	diff
full_profile_bikes	diff	diff
short_profile_3h_diff_bikes	diff	diff
short_profile_bikes	diff	diff

Figure 1: *The variation of each feature (excluding* bikes*) both within and across stations.*

2.1.2 Excluding Unhelpful Features

Feature *timestamp* increases infinitely and linearly, meaning it is predictable and will never repeat. It will never be useful for future predictions, so it was removed it from our selection.

While *isHoliday* initially appeared useful, it was ultimately irrelevant so was excluded from our model. The feature does not consider standard weekends to be holiday, despite the fact people are just as likely to be available (i.e. not at work) to rent bikes. An improvement could be made by engineering a more informative binary variable *isBreak* that would include all weekends and holidays.

Feature *day* does differ within stations, but without data from any month other than October 2014, we cannot assume that any particular day of the month is significant. Additionaly, using *day* as a feature on its own would result in having nothing to tie it to — a model including *day* would not distinguish between the 15th October and 15th July, even if they have nothing in common. Therefore we deemed *day* to be unhelpful, and so removed it from our selection.

2.1.3 Excluding Highly Intercorrelated Features

The *short* and *full* profile features are almost identical. We calculated the correlation between the corresponding pairs and found it to be 1.00. We then experimented with including either pair, and found that the *full* features was slightly more correlated with bikes and therefore yielded a better performance, so we removed the two *short* features.

2.1.4 Excluding Features Unsuitable for Linear Regression

We removed *station*, because it does not directly affect the number of bikes rented. Instead, we kept *numDocks* which is specific for the stations and more highly correlated with *bikes*. Similarly, while features *latitude* and *longitude* are specific for the different stations, their values are not related to the value of the target variable *bikes*. Therefore, these three features are better suited to a decision model, e.g. decision trees, in which if-statement style comparisons could be used.

2.1.5 Selection using Correlation Coefficients

Once we had manually excluded all the features we judged to be unsuitable, we were left with the following sets of features:

- Individual Models:
  - *hour*
  - *weekday*
  - *weekhour*
  - *windMaxSpeedms*
  - *windMeanSpeedms*
  - *windDirectiongrades*
  - *temperatureC*
  - *relHumidityHR*
  - *airPressuremb*
  - *bikes_3h_ago*
  - *full_profile_3h_diff_bikes*
  - *full_profile_bikes*
- General Model:
  - *numDocks*
  - *hour*
  - *weekday*
  - *weekhour*
  - *windMaxSpeedms*
  - *windMeanSpeedms*
  - *windDirectiongrades*
  - *temperatureC*
  - *relHumidityHR*
  - *airPressuremb*
  - *bikes_3h_ago*
  - *full_profile_bikes*
  - *full_profile_3h_diff_bikes*

We calculated the correlation coefficients of the according feature set with *bikes* and ranked the absolute values of the results from highest correlated to lowest. We considered two methods of selecting the most appropriate features from this ranking for both the general and individual models:

- **Selecting the top n features**

  We experimented with selecting the n highest absolutely correlated features to include in our linear regression model, concluding that picking the top four (i.e. $n = 4$) produced the best compromise between over-generalising and over-fitting for both the individual models and general model (Figure 2).

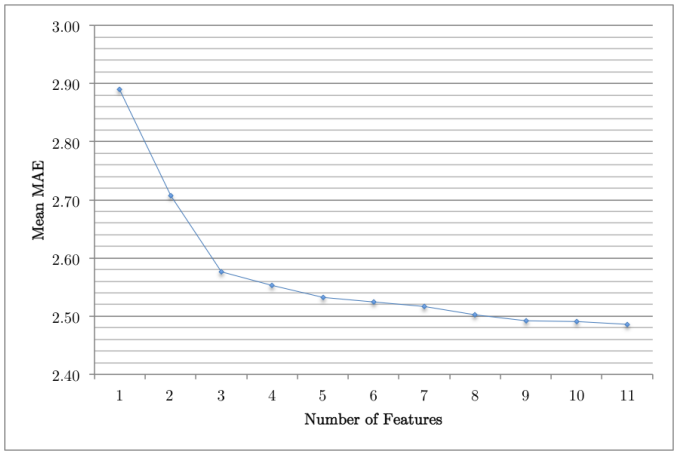


Figure 2: *The mean MAE across all training data 75 stations of using the 'top n' features selection method.*

- **Selecting the highest correlated above a threshold T**

  We also tried selecting all the features with an absolute correlation above a specified threshold T, resulting in models calculated with varying numbers of features. We experimented with different thresholds, but none yielded superior results to selecting the top n features, for either the general or individual models (Figure 3).

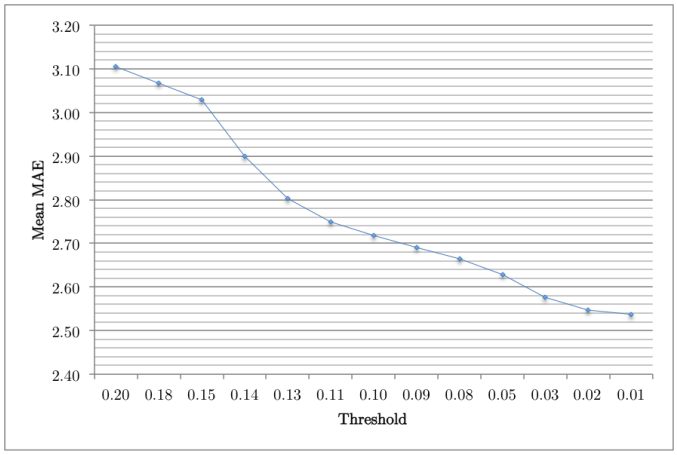


Figure 3: *The mean MAE across all 75 training data stations of using the 'threshold T' features selection method.*

2.2 Model Application

2.2.1 Linear Regression

Our model aims to predict the value of *bikes*, which is a dependent variable, using at least two features, which are independent variables. Therefore, we chose to use multiple linear regression (least squares):

$$a_{LS} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$$

The X matrix comprises of a column vector of ones (the first column) and our chosen features. y contains the values of the *bikes* column of our training data. Once we have found a_{LS}, we can calculate $y_{estimate}$, our predicted bikes vector:

$$y_{estimate} = Xa_{LS}$$

Our Matlab implementation of this mathematics is as follows:

```
X = [ones(size(x,1),1) x]; % add a column of 1s
a_ls = inv(X'X)X'y; % calculate a_ls
y_estimate = Xa_ls; % calculate y_estimate (predicted bikes)
```

2.2.2 Individual Models

Once we had calculated *y_estimate* using the previously mentioned least squares method, we judged the individual models' performance by calculating the mean absolute error (MAE) of our predicted *bikes* vector (*y_estimate*). MAE is the mean average of the absolute values of the least squares residuals, which are given by the difference in values of y and *y_estimate*:

$$MAE = mean(|r|) = mean(|y - y_{estimate}|)$$
$$r = y - y_{estimate} = (1 - X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}})y$$

As a performance guideline, we calculated two baseline mean absolute error values, *baselineMAE* — using *bikes_3h_ago* — and — using *bikes_3h_ago* and *full_profile_3h_diff_bikes* compared it with our results.

```
y_baseline = dataTable.bikes_3h_ago;
baselineMAE = mean(abs(y - y_baseline));
```

```
y_baseline = dataTable.bikes_3h_ago + dataTable.full_profile_3h_diff_bikes;
baselineMAE = mean(abs(y - y_baseline));
```

We knew that if our MAE was considerably higher than our custom baseline it was not performing well enough on our training data to consider submitting to Kaggle for test data assessment.
In addition to calculating MAE over our training data set to assess performance before Kaggle submission, we also made use of Matlab's Machine Learning Toolkit to perform cross validation. The data sets we were working with were sufficiently large for use with 10-fold cross validation. This method returns mean squared error, which we square rooted to obtain root mean square error (RMSE).

2.2.3 General Model

Our approach to creating a general model did not differ greatly from that to the individual models. We loaded all 75 training data .csv files holding the training data into a single Matlab table (instead of separate tables) and performed the same preprocessing. We again tested our two correlation coefficient based feature selection methods. We then applied least squares regression, but using our newly selected features catered to a general model.

2.3 Analysis and Discussion

Both the general model and individual models displayed good performance. Our general model produced a MAE of 2.44 on submission to Kaggle. However, the MAE on the test data did not perform better than the Kaggle baseline. For example, the individual model created using the training data of station 201 gave a MAE of 3.03 on the test data (from Kaggle submission). This indicated our models had room for improvement.

2.3.1 Removing Outliers

To improve the results of our least squares regression, we wrote a script to remove outliers from both the general and individual models. It calculates the standard deviation of the difference between *bikes* (training data) and our estimate. Any estimated value for which the corresponding standard deviation is over a certain threshold is deleted.

On experimentation with different threshold values, we ultimately found 4.00 to yield the best performance for the general model, and 2.40 for the individual models. Figure 4 plots the result of various threshold values on *bikes* plotted against our (general model) predicted *bikes*. The proximity of its trend's gradient to 1.00 represents our general model's performance.

This significantly reduced the MAE of both the individual and general models, increasing the number of stations whose MAE was better than both our custom baseline, as well as the Kaggle test data baseline.

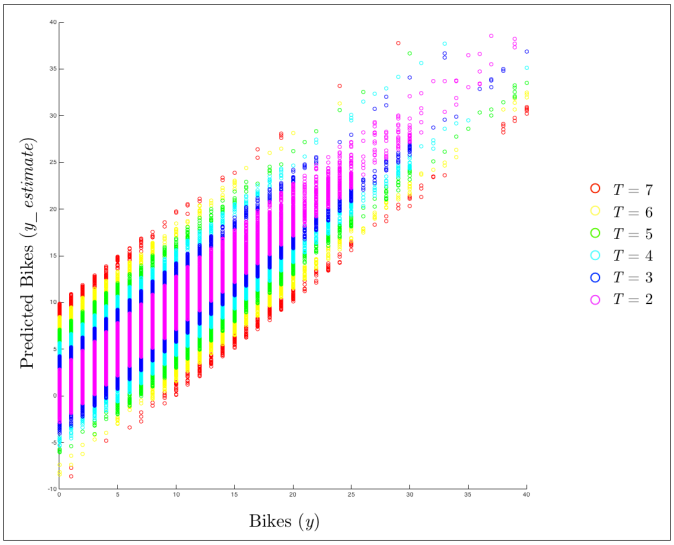
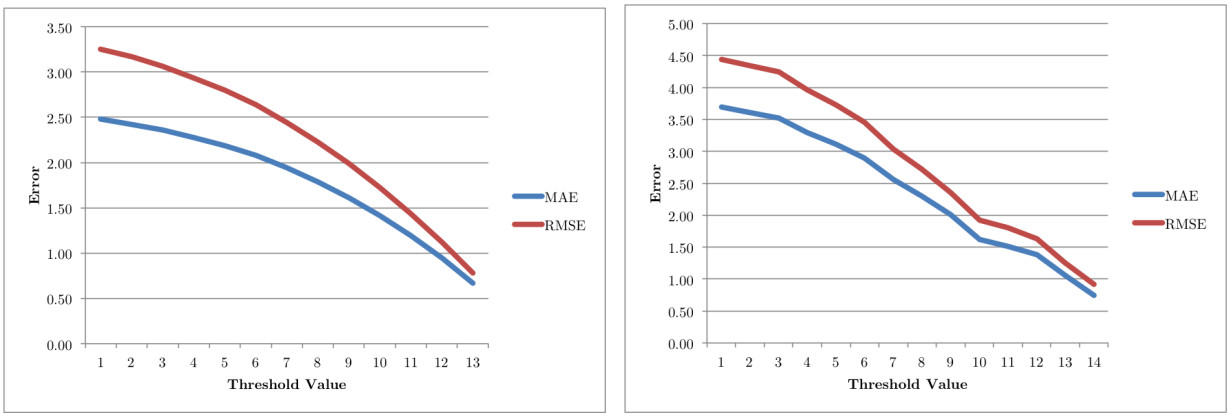


Figure 4: *The general model results of various threshold values for outlier removal. All the points within and including the cyan layer represent our chosen threshold of 4.*

As mentioned above, we used both the baseline MAE values, and the RMSE value returned by the 10-fold cross validation to assess model performance. These led the selection of the most promising model for Kaggle submission. Additionally, we also used those when finding the optimal threshold value for outlier removal (Figure 5).



(a) General Model (b) Individual Model: Station 234

Figure 5: *The MAE and RMSE of various threshold values for outlier removal.*

3 Phase Two

There are six types of model provided for each of 200 stations (station 1 to 200), the training data for which is not given. The types of model are *full_temp*, *full*, *short_full_temp*, *short_full*, *short* and *short_temp*.

All 6 include *(Intercept)* (the y intercept of the least squares regression line) and *bikes_3h_ago*. Those models whose name contains *full* and *short* include *full_profile_bikes* and *full_profile_3h_diff_bikes*, and *short_profile_bikes* and *short_profile_3h_diff_bikes* respectively. Those models whose name contains *temp* includes *temperature.C*.

3.1 Model Application

We first wrote a script to evaluate the models independently of one another, using our training data (station 201 to 275) from Phase One as test data. Calculating the mean absolute error of the provided models enabled us to observe performance.

As expected, different types of models trained on different stations gave varying results on the training data sets. Had we been able to pinpoint the optimal model for the test set (by submitting to Kaggle 1200 times), it would likely have been over-fitted and would not have performed as successfully on an alternative test set.

Therefore, we decided a combination of the given models would yield a better compromise between performance and over-fitting. We tested different permutations on the training data of 75 stations, averaging the weights across the 200 sample stations. The four combinations we tried were:

- **All**
  A combination of the features from all six types of model.
- **Full**
  A combination of the features from the four types of model whose name contains *full*.
- **Short**
  A combination of the features from the four types of model whose name contains *short*.
- **Temp**
  A combination of the features from the three types of model whose name contains *temp*.

To assess each combination, we calculated the mean absolute error in predicted against the training data, and plotted the results (Figure 6).

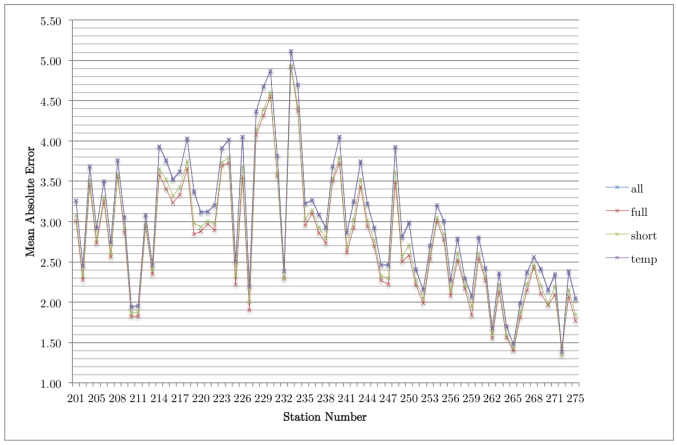


Figure 6: *The mean absolute error of our four combinatorial models for each of the training data stations.*

Observing the graph, we noticed using *all* models gave a significantly worse performance than the other three combinations — *short*, *full* and *temp*. Therefore, the latter were chosen for Kaggle submittion — the three models are represented by the lowest lines on the graph (indicating the lowest MAE). The combination yielding the best result was *short*, which combined the given models *short_full_temp*, *short_full*,

short_temp and *short*.

3.2 Analysis and Discussion

Our Phase One general model had performance with an approximate MAE of 2.32, falling well below the Kaggle baseline. Our 75 individual models performed with varying results; many better than the general model, and some worse. Our Phase Two general model showed a similar performance to that in Phase One with a MAE of 2.35.

It is important to note that the general model in Phase One and the one we created in Phase Two used different feature sets. The difference in the selected features and the training data encouraged us to combine those two models in Phase 3.

4 Phase Three

4.1 Method

Phase 3 entailed the challenging task of combining two models for one of which we did not possess the training data. In order to achieve an improved model, we chose the best performing model from each Phase One and Phase Two, which were our general model (Phase One) and our *short* model combination (Phase Two). These displayed similarly good performances. The Phase One general model performed better than any of the individual models because its composition over all 75 stations meant it was more generalised, and therefore successful on test data.

The Phase One model only captured a single month's data, in contrast with the year's worth of data from Phase Two. To allow for this, we weighted our averaging of the two models. Phase One's coefficients were assigned a weight of $1/13$, and Phase Two's coefficients were assigned a weight of $12/13$.

We also tried alternative weights for the a_{LS} coefficients. Phase One's coefficients were scaled down to a weight of $1/12$, and Phase Two's coefficients were unchanged. These weights produced significantly different a_{LS} coefficients — hence the increased MAE was expected. We did not submit this model to Kaggle and discarded the approach.

Once we had calculated the new linear regression coefficient, the two most apparent approaches were to include: 1. all the features used in either model or 2. the features only used in both models: *bikes_3h_ago*, *full_profile_bikes*, and *full_profile_3h_diff_bikes* In both cases we still combined and included the intercept coefficients.

4.2 Discussion

Our use of the Phase One general model, meant we used the combined training data set (all 75 station sets in one table) to compute the *baselineMAE*s, *y_estimate* (predicted bikes), and model MAE. The first approach of using the features used in either model resulted in a MAE much higher than the baseline, unlike the approach using solely mutual features. We therefore submitted the latter to Kaggle, resulting in a very similar performance on the test data to that of the Phase One and Two models. As mentioned above, the both intercept coefficients were still included. Again, we tried two approaches with that — we tried weighting them in a similar fashion to the rest of the coefficients, or leaving them as they were in the original models. The weighted intercept coefficients performed slightly better than the latter approach. Unfortunately, as our phase one and phase two models already showed good performance, it was challenging to beat it in phase three.

5 Conclusion

This Kaggle task challenged us to apply machine learning techniques to an everyday problem. It led us to experiment with different feature selection methods, as well as creation, evaluation and combination of linear regression models. Ultimately we think more training data would be required for an improved model.

6 Appendix A: Script Details

We produced a number of scripts with various functionality in order to complete the tasks given. All scripts should be run from the same folder. This folder should also contain a folder named 'Train' containing the training data on stations 201 – 275, and a folder named 'Models' containing all the various models on stations 1 – 200.

6.1 Import Scripts

Our model scripts depend on four custom import scripts which also perform the preprocessing mentioned in Section 1.2 — *importfile.m*, *importmodel.m*, *importleaderboard.m* and *importtest.m*. These respectively import a training data file, a model file, the leaderboard template (for creating our own submission file), and the example test file provided.

6.2 Phase One Model Scripts

For phase one we wrote two scripts, *individual_models.m* and *general_model.m*. The former takes a station number — between 201 and 275 — as an argument and performs all the steps mentioned in Section 2 on the specific station's training set. The latter reads in the training sets for all stations, combines them into one matrix, and performs all the steps mentioned in Section 2. We also have scripts *individual_models_threshold.m* and *general_model_threshold.m* which explore the alternative feature selection approach.

6.3 Phase Two Model Scripts

For phase two we wrote scripts following the models' names — *model_station_rlm_full.m* etc — which take a station number and a model number as arguments to evaluate the six models independently on each of the 75 training data stations. We then combined them in (*phase_two.m*) to evaluate different combinations of the six models types (specified in Section 3), averaged across the 200 sample stations, on all 75 stations.

6.4 Phase Three Model Scripts

For phase three we have a single script — *phase_three.m* — which combines the chosen models from phase one and phase two. It takes no parameters, and uses the previous models' coefficients to compute new ones — which are then used on the test data in a similar fashion to the rest of the scripts.