Predict the Destinations of Taxis

CKME136 Capstone
2015 June

Peter Raynham

Contents

Abstract

A taxi dispatch office wants efficiently dispatch taxis to the locations of customers calling for rides. For this, it would be useful to know where taxis engaged by customers will be when they complete their trips and then become available.

We devise a computational method for predicting where an engaged taxi will be when it completes its current trip, using historical data (logs of past trips) and current data (real-time reports of taxi locations). We consider an approach based on k-nearest-neighbour (kNN) modelling to compare the current trips-in-progress against historical trips that had similar characteristics during their progress.

We implement and test a kNN model. We conclude that: (a) predicting a taxi's destination is algorithmically difficult until the final phase of the trip; (b) for most of the trip, kNN tends to yield a moderately accurate but imprecise prediction; (c) it is relatively easy to predict the direction toward the destination, but difficult to predict the distance of the destination from the source location.

Description of Problem

Operators of taxi companies seek to give more efficient service. One idea is to optimize dispatching to reduce the time that a customer has to wait for a taxi to arrive after summoning it and to reduce the time that a taxi driver has to travel to a customer pick-up point. This would increase both the speed of service to the customer and the revenue per hour for the driver.

When a customer calls for a taxi during a busy time, there is often a taxi that is about to make a drop-off near to the caller. If the dispatcher knew which taxi it was, they could assign that taxi to the caller. Unfortunately, drivers usually do not report drop-off locations in advance, so the dispatcher has to guess which taxis might become available close by.

Thanks to telemetry equipment in the taxis, dispatchers have access to the state (free or hired) and location (via GPS) of every taxi. We explore methods of using this real-time data along with historical data of past taxi trips, to predict where taxis will be when they complete their current trips.

(Map from kaggle.com.)

For this we pose the following research problem:

Predict the destinations of taxis that are on route with customers.

Source Dataset

The data for this study comes from the City of Porto, in Portugal. The supplied dataset contains a record of all taxi trips over one year, from 1 July 2013 to 30 June 2014, for 442 cabs that operate in Porto. The data was generated by the dispatch office and by telemetry equipment in the taxis. Each taxi transmits its location (longitude and latitude) every 15 seconds to the dispatch office.

The dataset has one observation per complete taxi trip. Each observation has the following attributes:

1. **Trip Id:** A unique identifier of the trip;
2. **Call type:** The source of the taxi dispatch: (A) central dispatch, (B) taxi stand, (C) on the street;
3. **Origin call:** The customer's phone number, if the customer called central dispatch;
4. **Origin stand:** The id of the taxi stand, if the customer got the taxi at a taxi stand;
5. **Taxi id:** The identifier of the taxi driver.
6. **Timestamp:** The Unix-coded timestamp of the start of the trip.
7. **Day type:** The type of the day of the trip: (A) ordinary day, (B) holiday, (C) day before holiday;
8. **Missing data:** Indicator – *false* if the GPS data is complete; *true* if any GPS data is missing;
9. **Polyline:** A list of GPS coordinates (latitude and longitude) taken every 15 seconds on the trip. The first set show the start of the trip; the last set show the end of the trip.

Highlights of the training data are as follows:

- Total observations: 1,710,617
  - 0.34% of observations have missing trip data.
- Taxi trip distances (distance from pick-up points to drop-off points):
  - Mean distance 4.75 km
  - Percentiles: [5th] 0.35 km, [10th] 0.99 km, [25th] 1.98 km, [50th] 3.64 km, [75th] 6.27 km, [90th] 9.85 km, [95th] 12.4 km.
  - Shortest trip distance 0 km (likely due to missing data); longest trip distance 470 km (almost certainly due to GPS data error).

Data Source

The data is taken from a Kaggle competition, ECML/PKDD 15: Taxi Trajectory Prediction (I). Kaggle hosts competitions for data scientists; at this time fifteen competitions are active. This taxi competition runs from 20 April to 1 July 2015. Incidentally, a $250 prize is offered for the best prediction algorithm.

Sample Observation

Here is a sample observation from the training dataset (a single CSV-format line):

```
"1372636858620000589","C","","","20000589","1372636858","A","False",
"[[-8.618643,41.141412],[-8.618499,41.141376],[-8.620326,41.14251],
[-8.622153,41.143815],[-8.623953,41.144373],[-8.62668,41.144778],
[-8.627373,41.144697],[-8.630226,41.14521],[-8.632746,41.14692],
[-8.631738,41.148225],[-8.629938,41.150385],[-8.62911,41.151213],
[-8.629128,41.15124],[-8.628786,41.152203],[-8.628687,41.152374],
[-8.628759,41.152518],[-8.630838,41.15268],[-8.632323,41.153022],
[-8.631144,41.154489],[-8.630829,41.154507],[-8.630829,41.154516],
[-8.630829,41.154498],[-8.630838,41.154489]]"
```

Here is the same observation with annotations and computed trip totals. The rightmost column of the polyline table is a visualization of the direction and speed of the taxi during the 15-second trip segment.

```
     TRIP_ID: 1372636858620000589
   CALL_TYPE: C  (street hail)
 ORIGIN_CALL:
ORIGIN_STAND:
     TAXI_ID: 20000589
   TIMESTAMP: 1372636858  (Jul-01 00:00:58)
    DAY_TYPE: A  (normal day)
MISSING_DATA: False
    POLYLINE:
             TIME   LONGITUDE  LATITUDE   DIST   DIR
             0.00: -8.618643, 41.141412,    0m, -
             0.25: -8.618499, 41.141376,   22m,  -11°   W<
             0.50: -8.620326, 41.142510,  298m,  155°  NE======>
             0.75: -8.622153, 41.143815,  306m,  152°  NE======>
             1.00: -8.623953, 41.144373,  273m,  167°   E=====>
             1.25: -8.626680, 41.144778,  405m,  174°   E========>
             1.50: -8.627373, 41.144697,  103m, -175°   E==>
             1.75: -8.630226, 41.145210,  425m,  172°   E========>
             2.00: -8.632746, 41.146920,  418m,  153°  NE========>
             2.25: -8.631738, 41.148225,  208m,   44°  NW<====
             2.50: -8.629938, 41.150385,  358m,   42°  NW<=======
             2.75: -8.629110, 41.151213,  153m,   37°  NW<===
             3.00: -8.629128, 41.151240,    4m,  132°  NE>
             3.25: -8.628786, 41.152203,  118m,   65°  NW<==
             3.50: -8.628687, 41.152374,   24m,   52°  NW<
             3.75: -8.628759, 41.152518,   19m,  124°  NE>
             4.00: -8.630838, 41.152680,  307m,  177°   E======>
             4.25: -8.632323, 41.153022,  222m,  170°   E====>
             4.50: -8.631144, 41.154489,  238m,   43°  NW<=====
             4.75: -8.630829, 41.154507,   47m,    2°   W<=
             5.00: -8.630829, 41.154516,    1m,   90°   N>
             5.25: -8.630829, 41.154498,    2m,  -90°   S<
             5.50: -8.630838, 41.154489,    2m, -143°  SE>
   ROUTE_LEN: 3.951km
    TRIP_LEN: 2.313km
```

Training and Testing Data

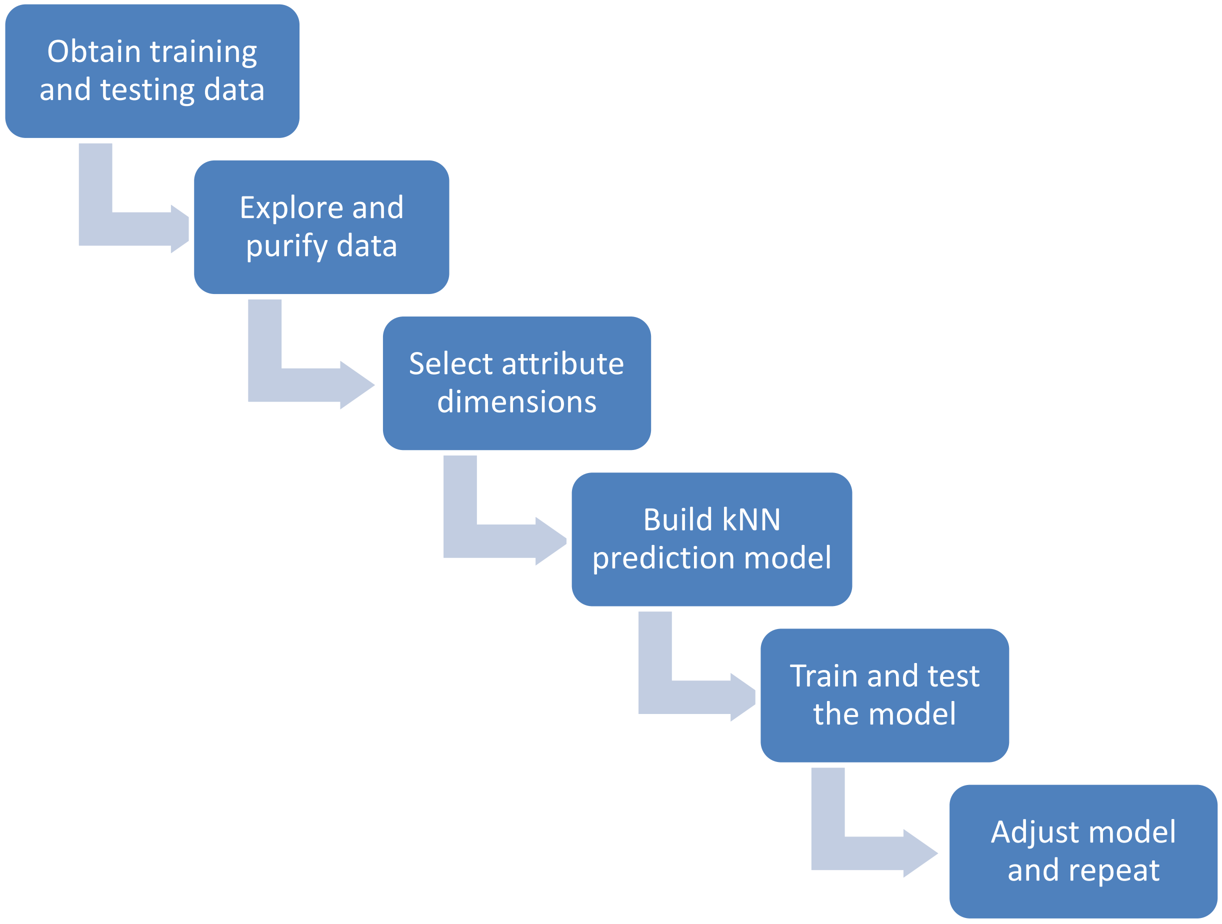
Training and testing data subsets are formed by extracting observations from the dataset. For our project, we selected 100,000 rows, evenly distributed, to use as training data. We extracted smaller subsets, of between 10 and 100 observations, avoiding overlap with any of the training observations, for testing data.

To test the model, we truncate an observed trip at a selected point in time, run the model with the truncated trip to generate a predicted destination location, then compare the predicted destination with the actual destination from the original (un-truncated) observation.

Design of Model

We build a predictor using a kNN (k nearest neighbor) model. The model will attempt to predict the completion of a partial trip by finding those historical trips that had similar initial portions, and then computing the central tendency of the destinations of the historical trips.

The steps of our approach are shown in the following diagram.


Obtain training and testing data
Explore and purify data
Select attribute dimensions
Build kNN prediction model
Train and test the model
Adjust model and repeat

Step 1: Obtain Training and Testing Data

The data are supplied on the Kaggle website [1] for the competition, so it is a simple matter to download the following files from https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data:

1. **train.csv:** Training data – all taxi trips for one year – 1.90 GB
2. **test.csv:** Testing data – snapshots of in-progress trips of taxis at five points in time – 330 KB
3. **metadata_taxistands.csv**: reference table of locations of all taxi stands – 3 KB

For this project, we used only the file *train.csv* with its 1.7 million observations, to supply both testing and training data. We did not use the file *test.csv* as it was missing the destination data we needed for measuring model performance. The file *metadata_taxistands.csv* (locations of all taxi stands in Porto) was determined to be not useful for our model.

Step 2: Explore and Purify the Data

We explore the test dataset to estimate the quality of the data, to discover what problems are in the data, and to implement strategies to handle the errors. The data in general is complete and reasonably clean, but we did find a few problems which were solved as follows:

1. Value *day type* is always "A" (ordinary day).
    - *Fix*: Use a calendar algorithm to determine weekends, and a publicly-available list of holidays in Portugal [2,3] to determine holidays. Set the *day type* accordingly.
2. About 0.3% of observations have empty 'polyline' values, that is, they show zero-length trips.
    - *Fix:* Omit these observations as they are less than 1% of observations and we have no way to re-construct the data.
3. About 3% of observations are for very short trips, less than 30 seconds or less than 30 metres.
    - *Fix:* Omit these observations as they describe trips that are too short for our analytical methods. It is suspected they are due to operational discrepancies, e.g., taxi meter activated by mistake.
4. About 12% of observations have one or more apparently biased GPS readings. They momentarily move locations up to 10 km away from where they should be – sometimes implying a taxi momentarily went faster than the speed of sound (1225km/h)!
    - *Fix:* Implement logic to check reasonableness of coordinates based on inferred speed; replace suspect coordinates with values interpolated from adjacent coordinates.
    - *Alternate fix:* Omit these observations. Although these observations make up a significant percent of the dataset, they appear to be evenly distributed, so that removing them should not introduce bias to the data. For the project, we decided to omit the observations.

Data exploration, extraction, and validation were carried out with programs written in Python. The programs are available in GitHub [4].

Step 3: Select Attribute Dimensions

A kNN model requires us to construct a multi-dimensional "space" with one dimension per observed attribute; into this space we place a point for each observation. When building the model, we prefer a small number of dimensions which represent reasonably independent attributes.

We explored a number of combinations of dimensions, but eventually opted for the following set. Note that these attributes are for a *snapshot* (a given trip at a given point in time) instead of for an entire trip.

1. Location of taxi at the start of trip (two dimensions – latitude and longitude);
2. Time interval from the start of the trip, (one dimension – minutes);
3. Location of taxi at *n*–7.5 minutes into the trip (two dimensions), where *n* is the time interval (2nd attribute);

4. Location of taxi at *n*–5 minutes into the trip (two dimensions);
5. Location of taxi at *n*–2.5 minutes into the trip (two dimensions);
6. Location of taxi at *n* minutes into the trip (two dimensions).

Because the source dataset is large (1.7 million observations) we selected a subset of 100,000 observations, distributed evenly from the dataset) for the training dataset.

Step 3A: Flatten the Data

The kNN model requires all observations to have the same attributes (i.e., columns), however the source taxi data has a varying number of attributes in the *polyline* field: the longer the trip, the more location values. To accommodate this, we "flatten" the dataset by expanding each observation from a per-trip form (with key field *trip id):*

```
Trip-id, Loc(0.0), Loc(2.5), Loc(5), Loc(7.5), Loc(10.0), Loc(12.5), …
```

To a per-trip-snapshot form (with key fields *trip id* and *interval from start of trip*) with a fixed number of attributes:

```
Trip-id, 2.5, Loc(0.0), Loc(0.0), Loc(0.0), Loc(2.5)
Trip-id, 5.0, Loc(0.0), Loc(0.0), Loc(2.5), Loc(5.0)
Trip-id, 7.5, Loc(0.0), Loc(2.5), Loc(5.0), Loc(7.5)
Trip-id, 10.0, Loc(2.5), Loc(5.0), Loc(7.5), Loc(10.0)
Trip-id, 12.5, Loc(5.0), Loc(7.5), Loc(10.0), Loc(12.5)
…
```

Taking snapshots every 2.5 minutes into each trip generates on average about 4.2 observations for each trip. For this project, training dataset counts are:

- 100,000 observed trips selected from source data, which become ...
- 85,310 trips kept following data validation (about 15% omitted), which become …
- 371,952 snapshots of trips in progress.

This volume was chosen because it appears large enough to give useful results while being small enough to fit in the available computer resource (4-core processor, 1.5 GHz clock speed, 6 GB of memory).

Step 4: Build kNN Prediction Model

We program a kNN predictor using the R language [7]. We chose to build a predictor, rather than use an existing library such as kknn, for three reasons. (1) existing predictors support only one dependent variable but we have two dependent variables, longitude and latitude; (2) existing predictors are designed for nominal dependent variables but we are predicting continuous variables; (3) writing a kNN predictor is reasonably easy.

The model uses the Euclidean distance formula to calculate distances in the search space, including distances *d* between locations: $d = sqrt((B_{lon} – A_{lon})^2 + (B_{lat} – A_{lat})^2)$. When calculating distances between locations, a correction factor of the cosine of the mean of the latitudes is applied to the longitudes to account for the fact that lines of longitude are closer than lines of latitude.

After the kNN algorithm finds the k training observations closest to the test observation, it calculates the predicted destination using a weighted average of the k training destinations. The destination of each point is weighted by *1 / (1 + d)* where d is the kNN distance, to give more weight to closer points.

The source for our kNN predictor is available in GitHub [5].

Step 5: Train and Test the Model

We trained the model using the training dataset generated from 100,000 taxi trips (generated in steps 3 and 3A).

We tested the model with a selection of testing datasets generated from between 30 to 100 trips. These data volumes were selected to generate meaningful results within acceptable run times.

Results were analyzed by applying statistical tests to the numeric models returned by *taxi.knn.R*, by plotting data from the models using standard R plotting tools, and by plotting data on Google maps using tools programmed in R and built on the *ggmaps2* library [8]. The source for the mapping tools is available in GitHub [6].

Step 6: Adjust Model and Repeat

We ran the model using a number of combinations and weightings of parameters, including:

- Starting location and locations at fixed times into the trip;
- Starting location and locations at snapshot times along the trip;
- Locations supplemented by time-of-day and type-of-day (workday/weekend/holiday);
- Locations supplemented by caller identification when available;
- Locations supplemented by taxi identification.

We discovered that the most accurate predictions are generated from the attribute dimensions listed in step 2, with a slightly heavier weighting applied to the starting location and the location at the instant of the snapshot (with lesser weights on the locations and 2.5, 5, and 7.5 minutes prior to snapshot times).

We also found no positive difference in accuracy due to attributes other than locations. That is, time of day, type of day, and identifying details did not increase accuracy and in some cases reduced accuracy.

Model Performance

Metrics

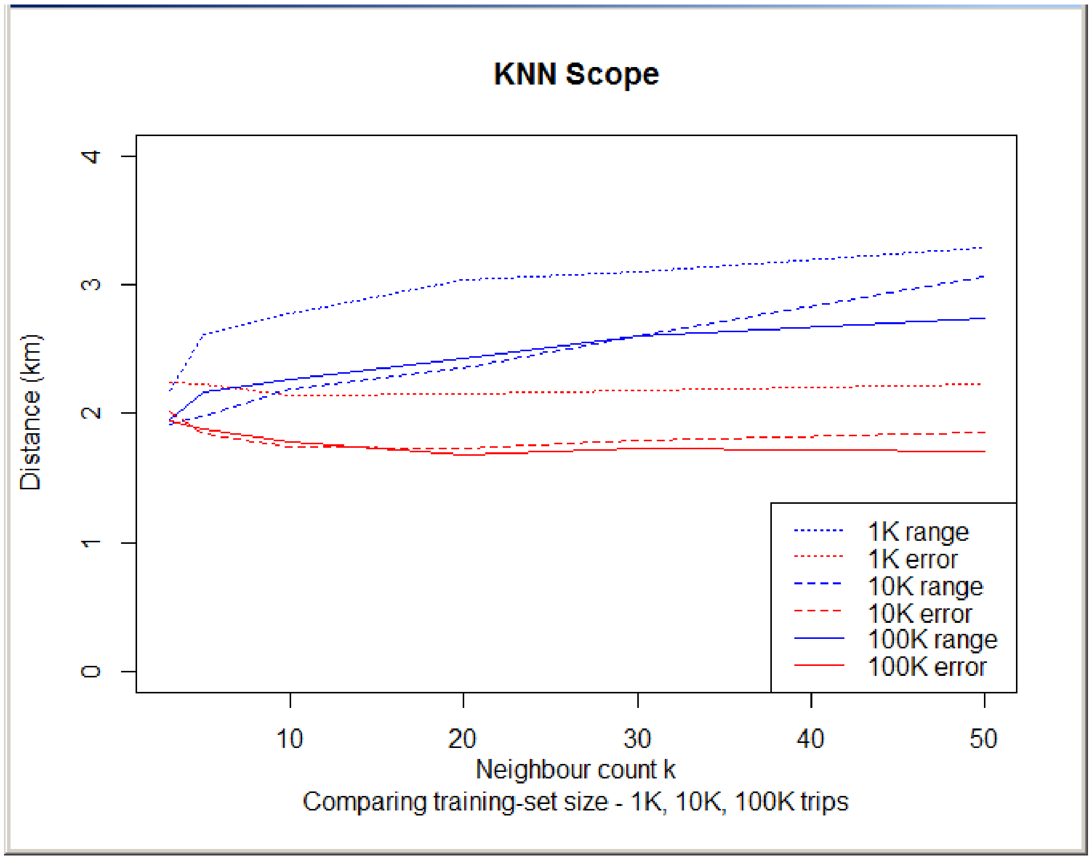
When testing a model we are primarily concerned with two metrics:

- **Accuracy** – How close is the predicted destination to the actual destination? Measured in kilometers.
- **Confidence** – How precise is the predicted destination? That is, how wide-ranging are the k destinations from the training set that were used to calculate the predicted destination? Measure of 1 standard deviation in kilometers.

Search Space Size

First, we measure the effect of the training-set size on the accuracy and confidence of the predictions. Going from a training set of 1,000 to 10,000 source trips improves prediction accuracy by 20%; going from a training set of 10,000 to 100,000 source trips further improves accuracy by only 3.2%.

Following is a graph of how accuracy and confidence are affected by the size of the training set and the size of k for kNN. Aside from the improvement of accuracy with larger training sets, it shows that k itself should be larger for larger training sets. Here, *error* is the distance from the prediction to the actual location, and *range* is the standard deviation of the k values used to compute the prediction.



Attribute Selection

We select the following attributes as the dimensions of the kNN search space. Each attribute value is normalized by dividing it by its standard deviation, then weighted by multiplying by a factor, then combined with the Euclidean distance formula to calculate distances between points.

In this project we are applying these weights:

Attribute of Trip Snapshot	Weight in kNN Metric Space
Longitude and latitude at start of trip	5.0
Longitude and latitude at 7.5 minutes before snapshot time	2.0
Longitude and latitude at 5 minutes before snapshot time	3.0
Longitude and latitude at 2.5 minutes before snapshot time	4.5
Longitude and latitude at snapshot time	6.75

Note that longitudes are also multiplied by the cosine of a common latitude, to normalize them to the same linear (km) distance as latitudes.

Here is a summary of how these weights compare to other sets of weights that we considered:

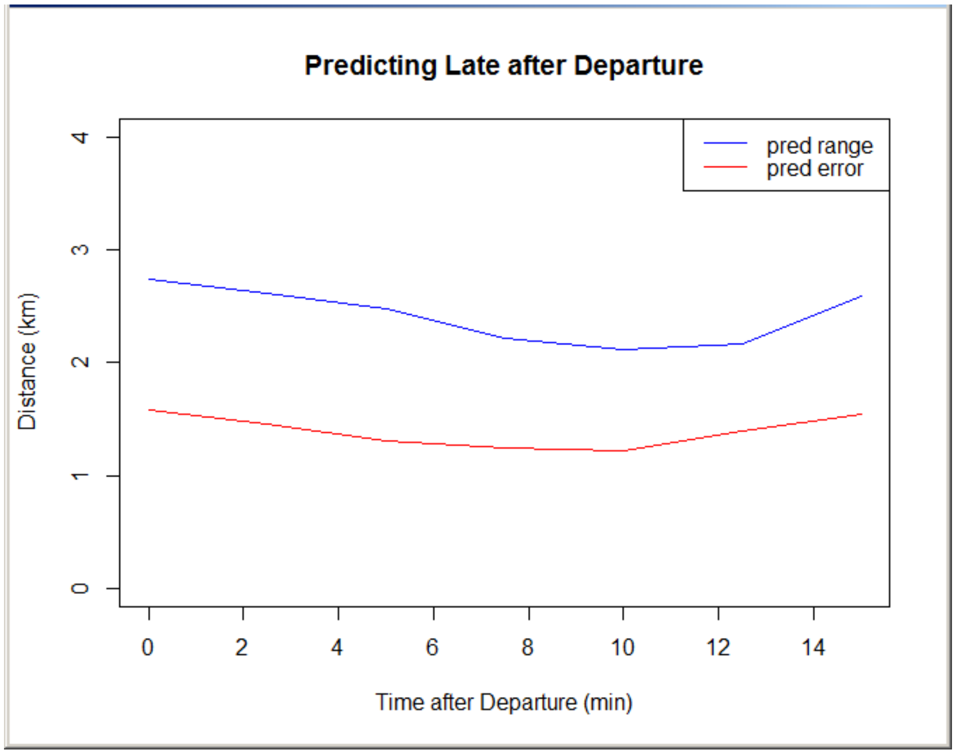
Set of Weights	Motivation	Mean Error of Predictions
Standard weights (above)	Origin and snapshot determined by overall trajectory; weight on most recent snapshots determined by driver's intention	1.89 km
Start time, snapshot times, all weighted the same	All location data are equally important	1.93 km
Start time, snapshot times, and time of day of the trip	Time of day affects destinations, e.g., people tend to go downtown in the morning and outbound in the evening	2.12 km
Start time, snapshot times, and taxi id	Individual cab drivers may prefer to work in different areas of the city	2.36 km

Prediction Performance

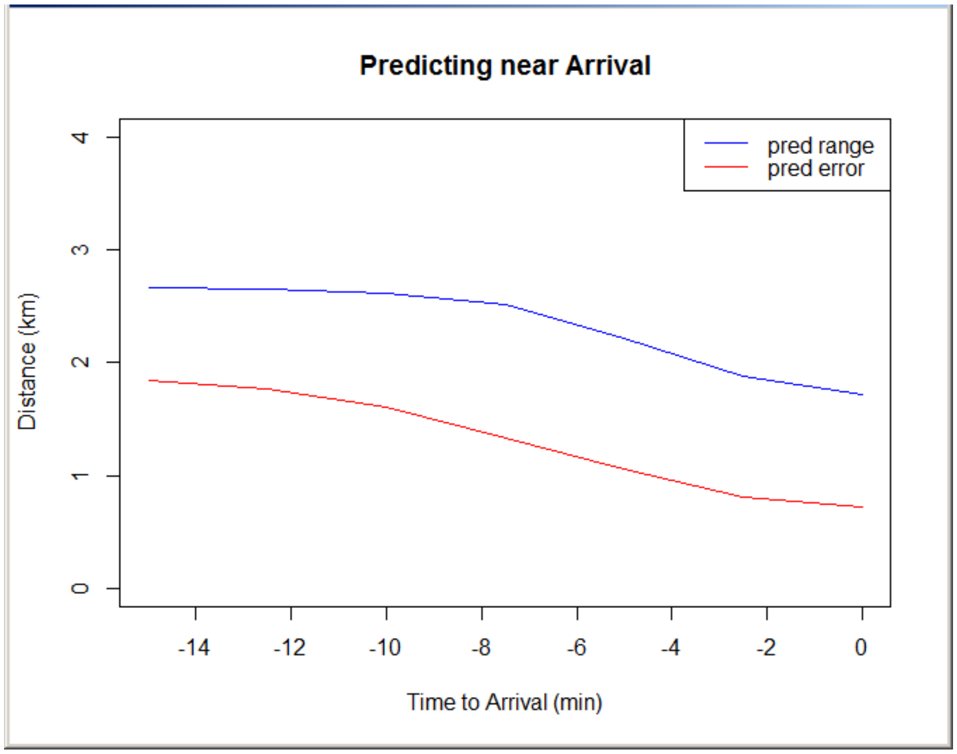
As we might expect, the accuracy of predictions improves as a taxi nears its destination.

When measuring from the start of the trip, we see that accuracy improves only until about 10 minutes into the trip; later the accuracy decreases. This is likely due to the fact that longer trips are often to suburban or rural places for which there are fewer training observations on which to base a forecast.

In these graphs: *error* measures the distance between predicted and actual locations; *range* is the standard deviation of the *k* destination points that the kNN model used to compute the prediction.



Indeed, if we look at accuracy relative to arrival time, we see that accuracy improves toward the end of the trip, from 1.6 km at 10 minutes before arrival to 0.7 km upon arrival.



Sample Model Predictions

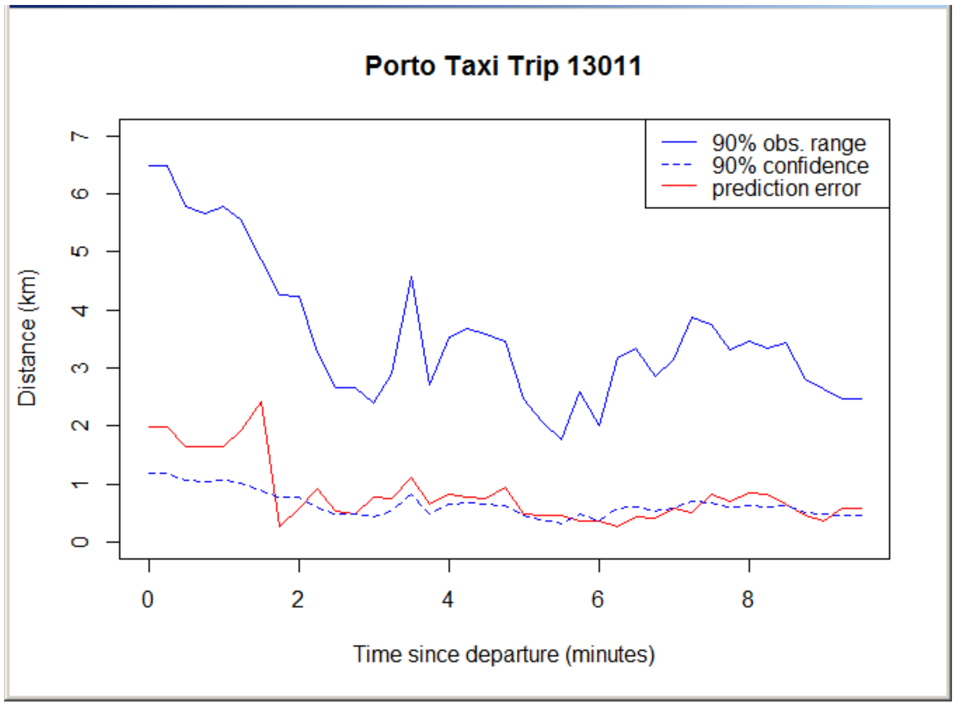
Typical Taxi Trip

Following is a detailed look at the quality of predictions for a typical taxi trip. The trip route is 3.1 km, and takes 9.5 minutes to complete. Following are data collected and computed along the trip:

Interval (min)	**Prediction**				**Actual**	
	Longitude	Latitude	Range (km)	Error (km)	Longitude	Latitude
0	-8.61264	41.16702	4.042	1.978	-8.61252	41.14590
0.25	-8.61264	41.16702	4.042	1.977	-8.61243	41.14584
0.5	-8.60951	41.16477	3.606	1.655	-8.61207	41.14595
0.75	-8.60988	41.16174	3.525	1.640	-8.61188	41.14609
1	-8.60951	41.16477	3.606	1.655	-8.61209	41.14634
1.25	-8.60952	41.15172	3.450	1.926	-8.61124	41.14624
1.5	-8.60985	41.14519	3.032	2.422	-8.61021	41.14640
1.75	-8.59279	41.15982	2.656	0.262	-8.60900	41.14674
2	-8.59337	41.15645	2.632	0.592	-8.60862	41.14723
2.25	-8.59542	41.15388	2.037	0.924	-8.60793	41.14827
2.5	-8.59249	41.15686	1.663	0.521	-8.60693	41.14810
2.75	-8.59200	41.15715	1.658	0.477	-8.60674	41.14840
3	-8.59458	41.15511	1.500	0.770	-8.60585	41.14825
3.25	-8.58841	41.15467	1.813	0.748	-8.60461	41.14783
3.5	-8.59110	41.15115	2.846	1.124	-8.60363	41.14753
3.75	-8.58732	41.15580	1.686	0.655	-8.60238	41.14731
4	-8.58171	41.15778	2.206	0.815	-8.60225	41.14722
4.25	-8.58150	41.15914	2.293	0.772	-8.60224	41.14720
4.5	-8.58276	41.15762	2.230	0.748	-8.60216	41.14719
4.75	-8.58168	41.15588	2.156	0.935	-8.60190	41.14715
5	-8.58819	41.15708	1.540	0.495	-8.60032	41.14758
5.25	-8.58999	41.15711	1.290	0.460	-8.59885	41.14818
5.5	-8.59166	41.15729	1.103	0.455	-8.59866	41.14900
5.75	-8.58779	41.15863	1.616	0.358	-8.59822	41.15009
6	-8.59023	41.15809	1.256	0.351	-8.59731	41.15167
6.25	-8.59184	41.16341	1.970	0.272	-8.59631	41.15338
6.5	-8.59253	41.16476	2.079	0.433	-8.59531	41.15514
6.75	-8.58970	41.16492	1.784	0.410	-8.59469	41.15628
7	-8.58835	41.16630	1.957	0.585	-8.59461	41.15643
7.25	-8.59230	41.16559	2.406	0.511	-8.59399	41.15744
7.5	-8.59502	41.16784	2.342	0.832	-8.59275	41.15955
7.75	-8.59220	41.16743	2.067	0.705	-8.59186	41.16099
8	-8.59037	41.16883	2.155	0.842	-8.59093	41.16078
8.25	-8.59081	41.16863	2.075	0.822	-8.59043	41.16098
8.5	-8.59025	41.16711	2.142	0.651	-8.59032	41.16124
8.75	-8.59009	41.16529	1.742	0.449	-8.59032	41.16125
9	-8.58971	41.16451	1.636	0.365	-8.59031	41.16125
9.25	-8.58823	41.16614	1.541	0.570	-8.59031	41.16125
9.5	-8.58797	41.16623	1.539	0.586	-8.59029	41.16125

The *error* is the distance between the predicted and actual locations. The *range* is the standard deviation of the 30 training destinations used by the kNN model to predict the test destination.

Following is a graph of the model's confidence in the predictions and the actual error of the predictions:

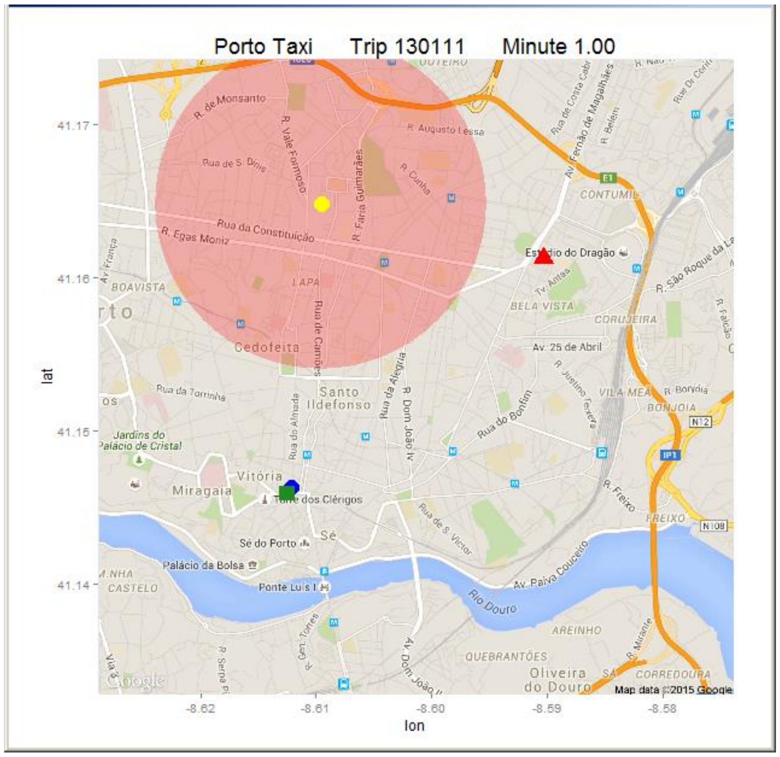


The confidence interval is the radius of a circle around the actual destination, in which the average prediction will fall 90% of the time. The observation range is the radius of a (somewhat larger) circle around the actual destination, inside which individual taxis will be predicted to arrive 90% of the time.

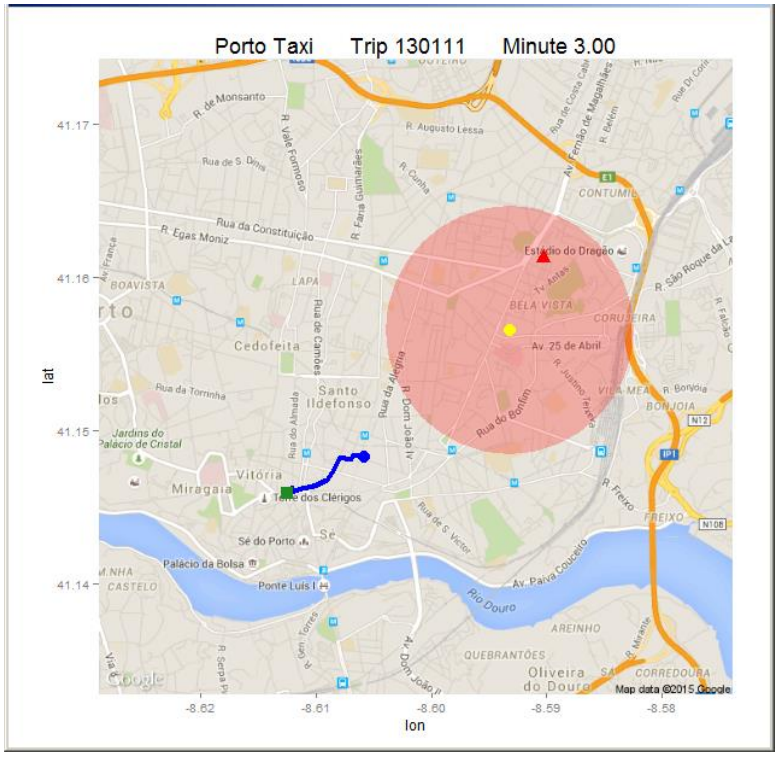
The following maps graphically show the prediction location and confidence as they evolve over the trip. In these maps:

- The green square is the point of departure;
- The red triangle is the actual destination;
- The yellow circle is the predicted destination;
- The blue circle is the current location of the taxi;
- The large red circle indicates confidence: the model is 90% certain that the destination is within the red circle.

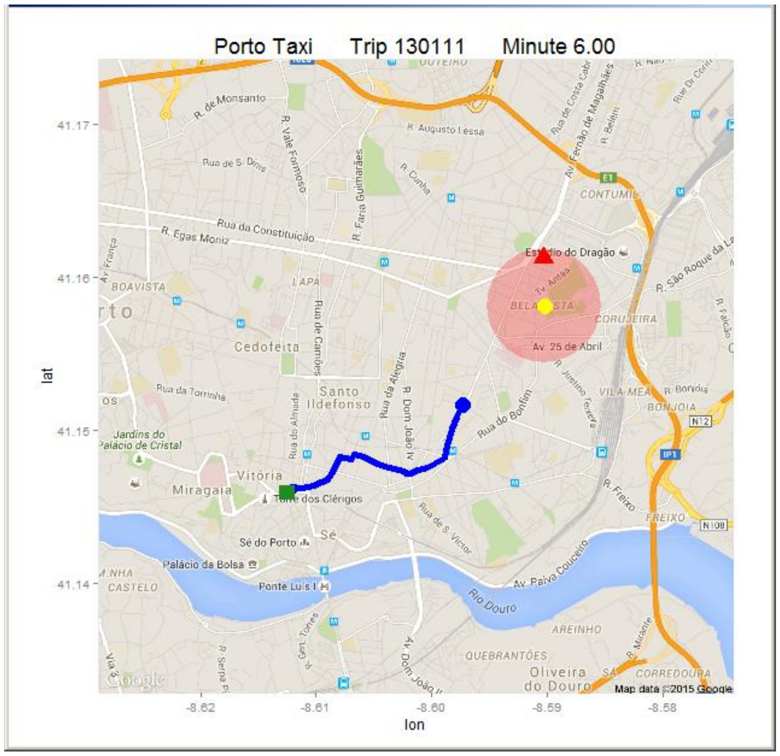
At the outset, just one minute into the trip, the model gives a prediction that is too imprecise to be useful. This is probably due to the fact that the taxi has not made enough route location data.



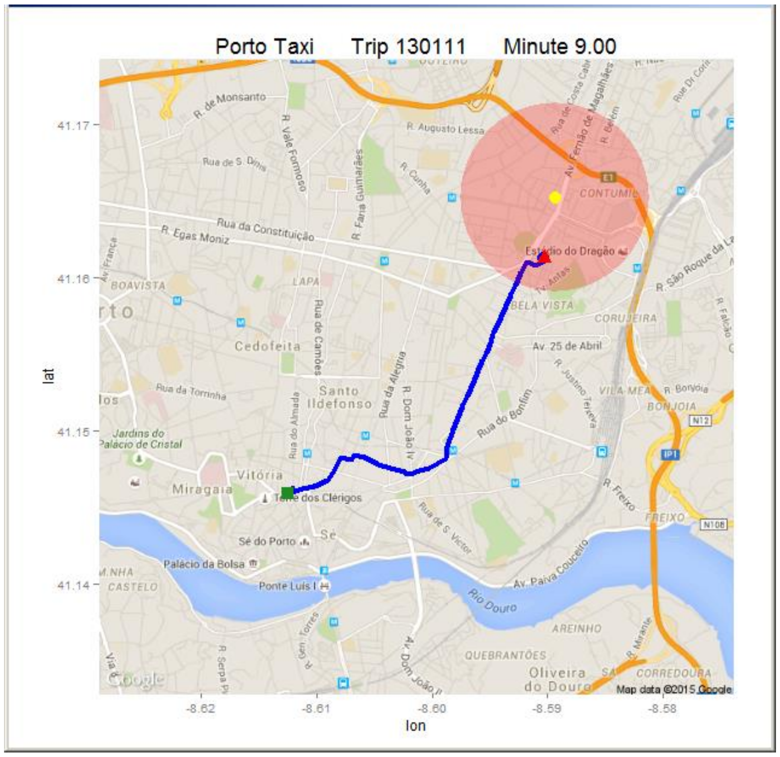
After three minutes, with the taxi well on its route, the prediction becomes more accurate and with a stronger confidence interval:



At six minutes into the trip, as the taxi travels along a main street, the model gives what turns out to be the best prediction of the trip:



Near the end, confidence drops when the taxi heads off the main street into a neighbourhood:



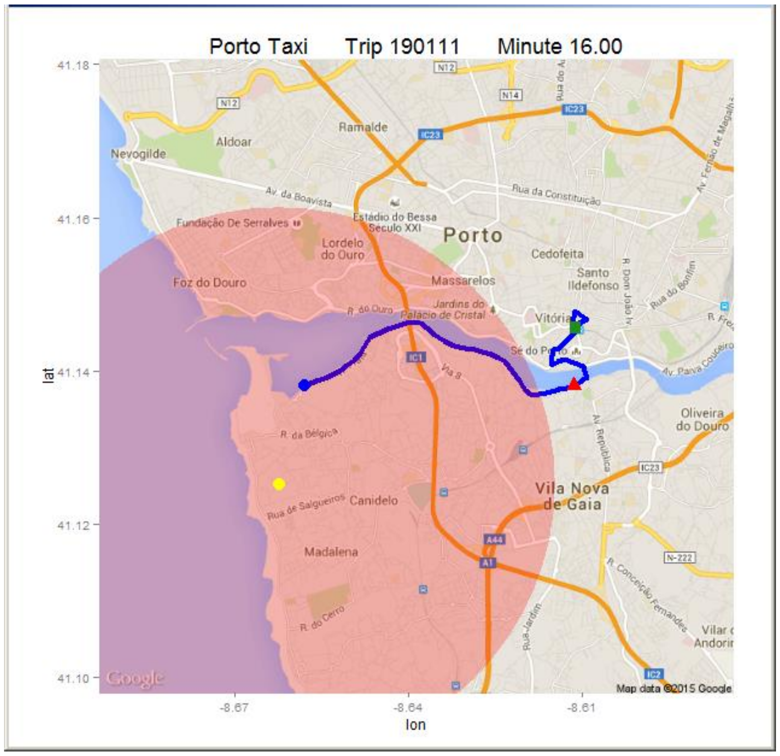
We observe that when the taxi goes off a well-travelled route, and so has fewer historical taxi trips to correlate with, that the confidence interval increases.

One thing we notice is that the model can predict well the direction of the taxi, but not so well how far the taxi will travel in that direction. Early in the trip, it estimates destinations that are too near; late in the trip it estimates a destinations that are too far.

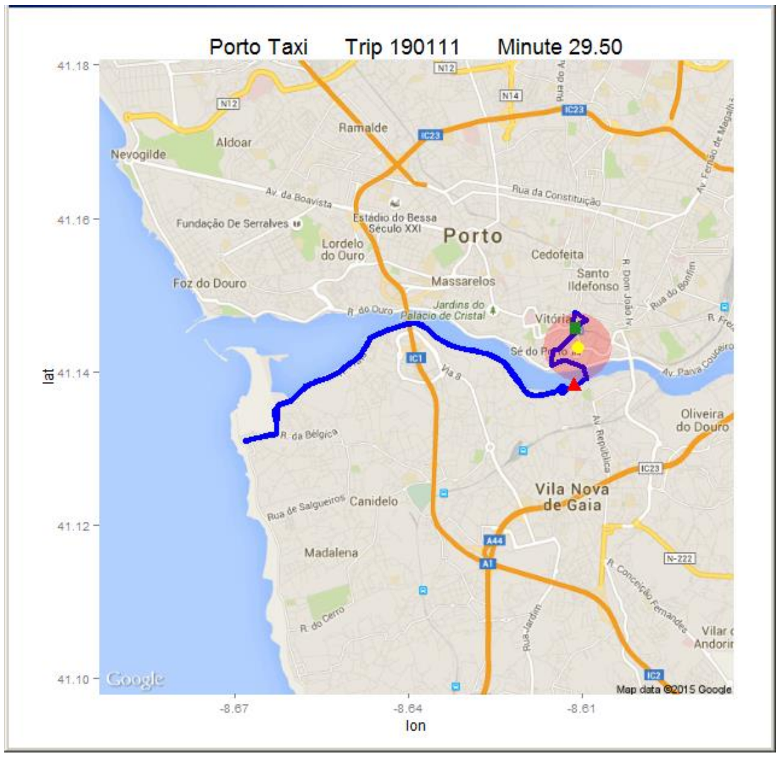
Outlying Taxi Trips

Taxi trips that are to locations far from the city center, or in which the taxi takes an unusual route, cause the model to make very inaccurate or imprecise predictions.

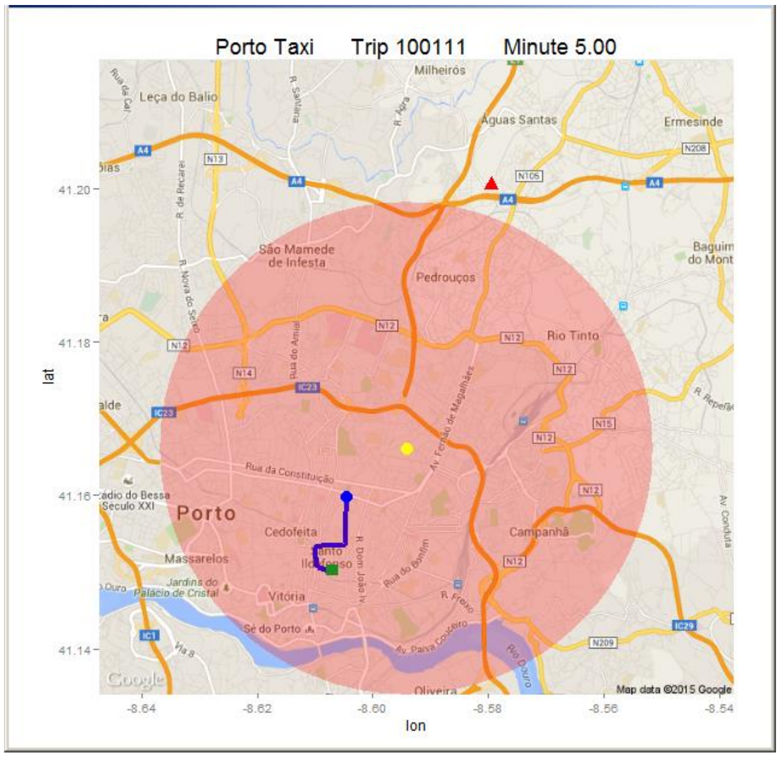
In the following example, a taxi travels to the Atlantic coast. The model makes a passably accurate prediction but with great uncertainty. (Following maps are to smaller scale than above maps).



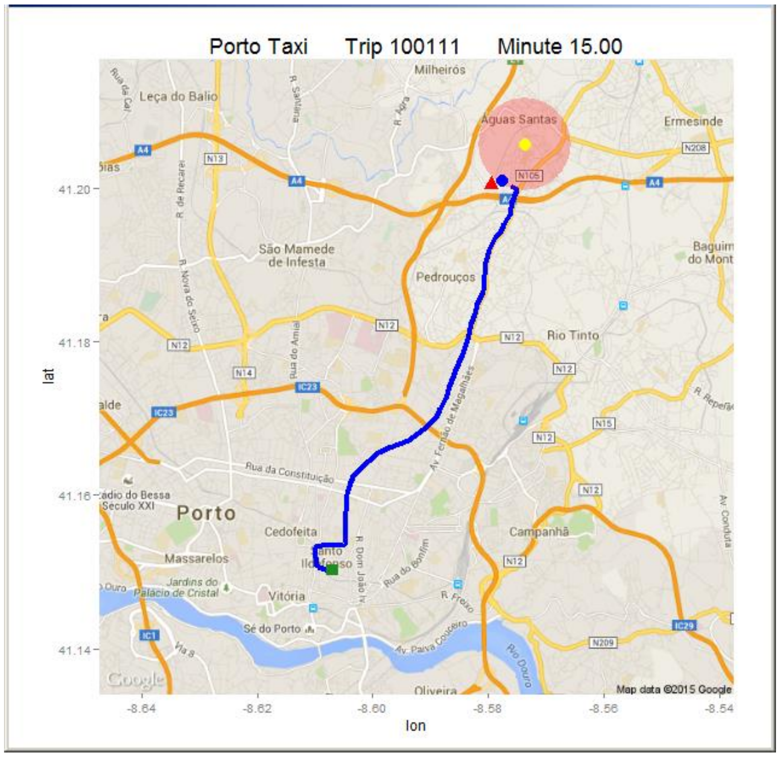
In fact this trip confounds the predictor, because after the taxi visits a location on the coast, it returns to the city to complete the trip. It is only after the taxi returns close to the city centre that the prediction becomes more accurate and precise:



Finally, a trip to a suburb of Porto causes problems. The trip takes 16 minutes and covers 7.4 km. Early in the trip, the model severely undershoots the actual destination despite a very wide confidence circle:



However, for the second half of the trip, the predictions improve both in confidence and accuracy:


Porto Taxi Trip 100111 Minute 15.00
lat
lon
41.20
41.18
41.16
41.14
-8.64
-8.62
-8.60
-8.58
-8.56
-8.54
Porto
Map data ©2015 Google

Conclusions

From our research we draw the major conclusion:

Predicting future taxi destinations based on past trips and on current taxi data is possible with a kNN model, but not with reliable accuracy or preciseness.

We also draw some other conclusions:

- The direction of a taxi on route can be predicted accurately.
- The distance that a taxi on route will travel is difficult to predict.
- Prediction with a kNN model gives the best results for taxis well-travelled areas (such as city cores) for which there exists a large quantity of historical data. Prediction with a kNN model is difficult in sparsely-travelled areas such as suburbs and rural zones.

Future Research

From the results of this project, we believe the following topics are worthy of future research:

- **Fix GPS data errors.** In the section *Explore and Purify Data*, we propose a fix to correct erroneous GPS readings. Implement this fix and rerun tests including the repaired observations.
- **Tune kNN for sparse model data.** When using a kNN predictor, collect the predictions over the duration of a taxi trip, and monitor for unusual changes in the predicted location or the confidence circle. For example, a decrease in the number of near neighbours in the model, or a sudden increase in the confidence circle may indicate that the taxi has left a main street and gone into a neighbourhood, which might imply the taxi is very close to its destination.
- **Use a Markov Model.** Consider using an adaptive Markov model to predict taxi destinations. For example, divide the city into small zones, and assign probabilities that a taxi in any given zone, travelling in a given direction, will have any other given zone as its destination. For example a taxi in a central zone travelling west has higher probabilities of arriving in zones in the west side of the city. As the taxi travels along its route, track its progress and update the probability grid. We believe this model could do better than kNN in avoiding occasional highly inaccurate predictions, but will do worse in predicting the distance to the destination.
- **Handle Special Cases.** Consider detailed research on attributes such as time-of-day, day-of-week, customer phone number, etc. These may be able to improve predictions for small subsets of cases (for example, a given customer may have a small set of destinations they travel to).

References

1. Kaggle. www.kaggle.com/competitions, accessed 2015 June 13.

2. TimeAndDate.com. *Holidays in Portugal 2013,* www.timeanddate.com/holidays/portugal/2013, accessed 2015 June 13.

3. TimeAndDate.com. *Holidays in Portugal 2014*, www.timeanddate.com/holidays/portugal/2014, accessed 2015 June 13.

4. GitHub, *Illume.* github.com/praynham/taxi-predict/blob/master/src/illume.py, updated 2015 June 13.

5. GitHub, *Taxi kNN*. github.com/praynham/taxi-predict/blob/master/src/taxi.knn.R, updated 2015 June 13.

6. GitHub, *Taxi Mapping*. github.com/praynham/taxi-predict/blob/master/src/taxi.mapping.R, updated 2015 June 13.

7. R Core Team (2014). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. www.R-project.org, accessed 2015 June 12.

8. D. Kahle and H. Wickham. *ggmap: Spatial Visualization with ggplot2*. The R Journal, 5(1), 144-161, journal.r-project.org/archive/2013-1/kahle-wickham.pdf, accessed 2015 June 13.