Predicting Mobile User’s Future Location

John Doherty (doherty1)

Introduction

One of the most powerful features of modern smartphones is their ability to provide us with realtime, locationally aware information. For example, our smartphones can predict when we are heading to work and give us an estimate of the travel time. While these features can be useful, they typically do not generalize to users with more complicated schedules. If our smartphones could predict our destination, they could give us more relevant information about the places we are going.

Imagine the following situation. You hop in your car to head to the gym after work. You don’t turn on your phone’s navigation because you know the way, but there is an accident on your route and a lot of traffic. If your phone knows where you are going without you telling it, it could warn you before you start driving and help you reroute. Of course all this location prediction sounds a bit creepy, but if it is all happening on device, and never storing your location history in the cloud, maybe people will feel a little more secure. For my project I created a system that can predict a user’s next location using their current location and the current time, learning only from that user’s location history.

This system is trying to answer the question: If I were to leave from my current location right now, what is my most likely destination? This can be reduced to a multi-class classification problem where the set of classes is the set of previously seen locations, and where we try to predict the class (destination) that maximizes the following probability:

$$P\ (destination \mid current\ location,\ date,\ time)$$

While this question seems like a straight forward machine learning question, a number of unique challenges arise when working with real data. Location history datasets are often small and noisy. Even so, the results are encouraging and with some tweaking it could have use in a real product.

Data

Data for this project was collected using the GPS location tracking app, Moves[1]. For this project, I used my personal location data collected over the last two months and a friend’s location data collected over a summer. This location data was obtained in the form of GPX files. GPX files are designed to store information about a user’s movements throughout the day, so it tracks the latitudes and longitudes of locations visited, in addition to the routes taken between those locations.

Since I am trying to solve a classification problem, I need the output to take the form of discrete classes. This means performing some sort of preprocessing to cluster continuous latitude longitude coordinates into a set of discrete locations. I did this by looking at the physical distances between coordinates and clustering points that were less than 50 meters apart. This clustering step is necessary because the actual latitude longitude coordinates of locations do not have much meaning. The resulting discrete locations are more useful than raw latitude longitude coordinates, but there are still far too many unique locations to preform any sort of useful classification. For example, my personal location history contains 78 unique locations, 66 of which were visited just once. To fix this problem I filter out locations visited fewer than 5 times.

With a set of discrete locations on hand I am able to build my training examples. Training examples are built using pairs of consecutive location points from the location history. These pairs of points (A, B) are instances from the dataset where the user traveled from A to B. My input vector consists of features of the date, time, and location of the first point in the pair. The target variable is the location of the second point.

The performance of this system depends heavily on the dataset given to it. Specifically, the performance is going to be much better if we use the data of someone with a consistent schedule rather than that of someone with a more irregular schedule. To make sure I am testing both ends of the spectrum I am using two datasets, my irregular location history from this school year and my friend’s more regular location history collected while working over the summer. A comparison of the distributions of locations can be seen in figure 1.

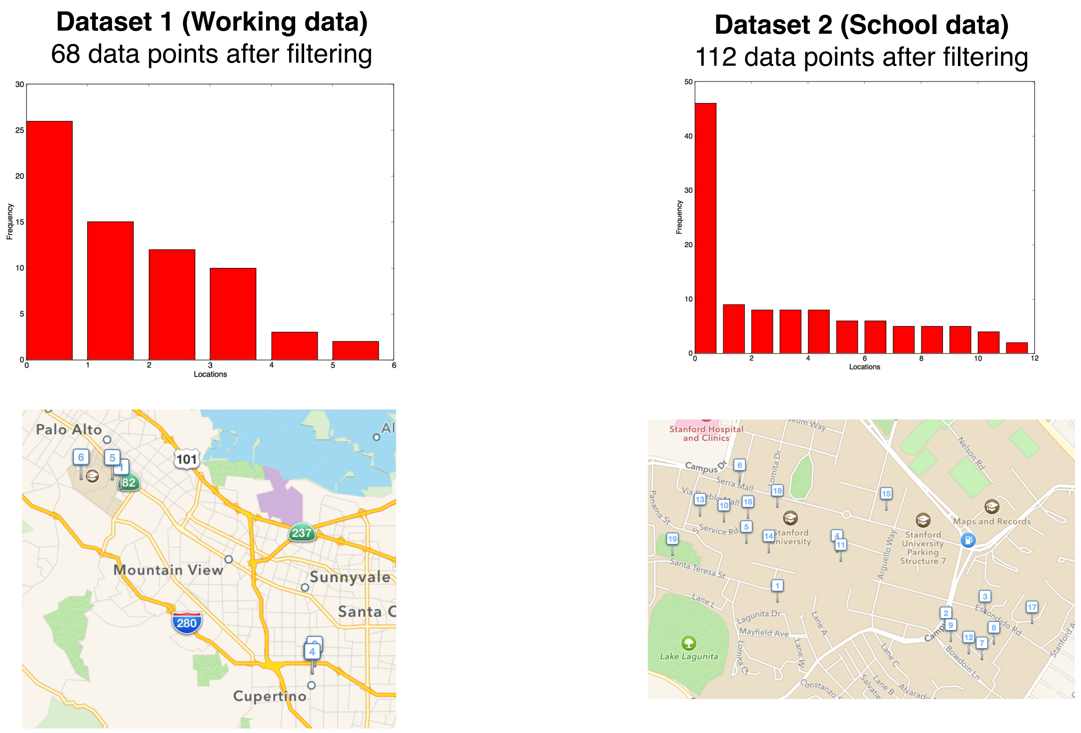


Figure 1: This shows a comparison of the two datasets. On the left (dataset 1) is my friend's location data collected while she was working. On the right (dataset 2) is my location data collected while I was at school. In the graphs, each bar indicates the frequency with which a unique location was visited. The long tail of the graph for dataset 2 shows the large number of locations with just a few visits.

Features

I tested different combinations of binary features on the location, date, and time of input data. My final system has the following features:

- A binarized version of the input location. For example, if there are 5 possible location classes and the current data point is from location 3, then we have the features [0, 0, 1, 0, 0].
- A binarized vector of the day of the week. This is intended to catch the weekly repeating patterns in schedules. Monday is represented as [1, 0, 0, 0, 0, 0, 0].
- A binarized vector representing the current time broken into 6 hour bins. This means that times are grouped into the following bins: 12am to 6am, 6am to 12pm, 12pm to 6pm, 6pm to 12am. For example, 5am would be represented as [1, 0, 0, 0].
- A binary feature indicating if the time is AM or PM.
- A binary feature indicating if the day is a weekend.

Additional features that I tried include: the previous location and different combinations of weekdays (Monday, Wednesday, Friday or Tuesday, Thursday for example). These features did not perform as well.

Model

I tried a number of different classifier models on this problem. All of the models are implemented in Python using the Scikit Learn library [2]. Because I have binary features and a multi-class classification problem, the following models made the most sense:

- One vs rest classification with logistic regression
- Naive Bayes
- Multi-class SVM
- Linear Discriminant Analysis (LDA)
- Random Forest

These are all standard classifier models with multi-class variants. Since I do not know much about the distribution of my datasets, it is worth testing the performance of all of these models.

Results

In this section I will discuss how well these models perform on the two location datasets provided. The first step in evaluating these models is to actually select the data for training and testing. I used two different methods to train and test the models. The first method is k-fold cross validation. It splits the data into k folds, trains on k-1 of them, and evaluates on the remaining one. I average the results for k iterations. This method uses less data for evaluation and saves more for training which is useful given that I have pretty tight data constraints.

The second method is intended to simulate online learning. In this method, I move through the dataset in chronological order and train on all data up to the current datapoint *n-1* and test on the datapoint *n*. The results of each evaluation are averaged. This method is intended to simulate a real environment where this system would be employed. In the real world, the model will take in more data over time so it is important to evaluate how it will preform as the amount of available data increases.

I use a few different metrics to evaluate the predictions of the models on the test set. Classification accuracy is the most obvious metric and it simply gives the fraction of the test set for which the model predicted the correct output. This can be useful, but is not always the most indicative of true performance. This is especially true in data with an unbalanced output class distribution, like mine.

Since this is a multi-class classification problem precision and recall are important metrics. Precision gives a sense of the number of false positives for each class while recall looks at false negatives. Instead of actually reporting precision and recall for all classes and all models, I will just present the F1 score. This score combines precision and recall for all classes. I will also present confusion matrixes for the best performing models.

All of these models assign a probability to each possible output class for a given input. We can sort the classes by this probability such that the first element of the sorted list is the most probable class and the one we would ultimately return. While most evaluation metrics only look at the one returned output (the class with the highest probability), it is interesting to evaluate the entire list of possible outputs. Specifically, in this problem it might be acceptable to return more than one result. To evaluate the list of possible classes for each model, I give the average position of the correct output in the sorted list of possible classes.

Below are tables of results evaluated on the described metrics. There are two tables for each dataset. One showing the results of k-fold testing and the other showing online learning. In these cases, “Dataset 1” is my friend’s working location data. This is a more consistent dataset and should give better results. “Dataset 2” is my school location data. This is much less consistent and the results reflect that.

	SVM	Logistic Regression	Random Forest	LDA	Naive Bayes
Train Accuracy	0.65	0.8058	0.8647	0.8205	0.7382
Test Accuracy	0.6176	0.6911	0.6323	0.5735	0.5882
F1	0.5429	0.6737	0.6075	0.5643	0.5693
Average position of correct output	0.8235	0.6764	0.7941	0.6764	0.8529

Table 1: Results of k-fold testing for dataset 1

	SVM	Logistic Regression	Random Forest	LDA	Naive Bayes
Test Accuracy	0.5416	0.5833	0.5833	0.5625	0.5833
F1	0.4596	0.5554	0.5870	0.5710	0.5404
Average position of correct output	1.0	0.7291	0.6666	0.8333	0.8125

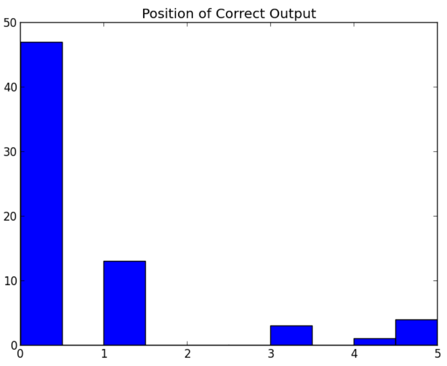
Table 2: Results of online testing for dataset 1

	SVM	Logistic Regression	Random Forest	LDA	Naive Bayes
Train Accuracy	0.4107	0.6535	0.8017	0.7053	0.6142
Test Accuracy	0.4107	0.4553	0.4017	0.3660	0.4375
F1	0.2391	0.3797	0.3566	0.3650	0.3684
Average position of correct output	2.723	2.133	2.794	2.4910	2.169

Table 3: Results of k-fold testing for dataset 2

	SVM	Logistic Regression	Random Forest	LDA	Naive Bayes
Test Accuracy	0.4130	0.4347	0.4130	0.2826	0.4239
F1	0.2433	0.3668	0.3684	0.3017	0.3711
Average position of correct output	3.434	2.684	3.217	3.0978	2.858

Table 4: Results of online testing for dataset 2



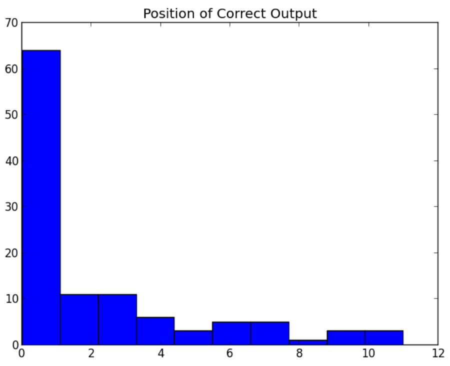


Figure 2: A comparison of results for logistic regression of the two datasets (dataset 1 left, dataset 2 right). These graphs show the distribution of the position of the correct class in the list of predicted classes sorted by probability. The skewed distribution shows that generally the correct class is close to the top of the list. Dataset 2 has a longer tail because it has more possible output classes with similar probability.

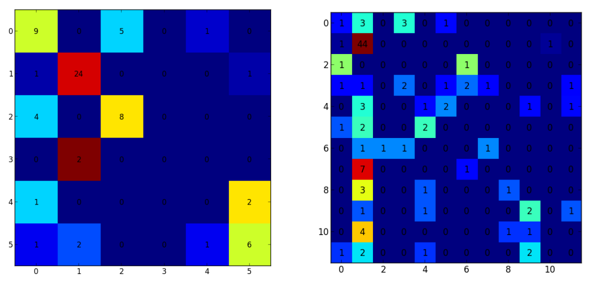


Figure 3: Confusion matrixes for logistic regression of the two datasets (dataset 1 left, dataset 2 right). Rows represent the true class while columns represent predicted class. The number at a position (i, j) is the number of times we predicted class j when the true class was i. The result from dataset 1 looks pretty good with most of the points falling along the diagonal. The model struggled significantly with dataset 2 and resulted to guessing class 2 in most cases.

Discussion

Overall logistic regression performed the best on both datasets. This is most likely because logistic regression is pretty resilient to noise, and there is a lot of noise in both datasets. Logistic regression is not trying to build hard boundaries between classes, like SVM, but instead uses a one vs all classifier system where it just assigns a score to each possible classification. It is interesting that logistic regression, one of the simplest classifiers, can out perform others on small, noisy datasets like these. SVM on the other hand struggled to deal with the fewer number of point. In fact, on dataset 2, the SVM resorted to classifying every test instance as “home.”

There are a number of properties of location data that make it tricky to work with, and the results reflect. First, these datasets are small. For reference, my personal location history collected over two months was reduced to just 112 data points after filtering out rare locations. This is not a lot of data, especially when dealing with more complex schedules that contain a lot of noise. In order to learn more complex schedules, we need more features, but adding features increases the number of parameters that need to be learned which requires more data. As a result, my system uses relatively few features to try to extract the core parts of the user’s schedule.

Another challenge that these location datasets throw at us is the imbalance in the number of times each location is seen. In any location history dataset there are going to be some locations that are visited far more frequently than the rest. I discussed filtering out rare locations, but there will always be places like home and work that you go daily and places like class or the gym that you go a few times a week. While some classifiers can handle this imbalance, others, like SVM, cannot [3]. This is another possible reason for why SVM performs so badly.

Finally, people’s schedules tend to change over time. For example, class schedules change every quarter. Schedules for jobs are more consistent, but people can change jobs and engage in different routines after work. While I did not encounter these properties in my datasets, a real word system would have to be modified such that more recent points were weighted more heavily in training.

Future

While this system did show some encouraging results, clearly there is a lot that still could be done. There are plenty of other features that could be added including ones external to the location history. These include things like weather and category of location (residence, office, classroom, etc.). Even more interesting would be to try modeling this data with latent variables. There might be certain clusters of locations which have similar properties. Additionally, modeling this dataset as a series of state transitions from one location to another could give better results.

References

[1] https://www.moves-app.com/
[2] http://scikit-learn.org/stable/
[3] Wu, G. and Chang E. “Class-boundary alignment for imbalanced dataset learning”