Machine Learning Nanodegree Capstone

Predicting Facebook Check-In Behavior – Kaggle Competition.

# Project Overview

This project is based on the Kaggle competition begun by Facebook to predict where a person is most likely to check in next. From the competition

The goal of this competition is to predict which place a person would like to check in to. For the purposes of this competition, Facebook created an artificial world consisting of more than 100,000 places located in a 10 km by 10 km square. For a given set of coordinates, your task is to return a ranked list of the most likely places. Data was fabricated to resemble location signals coming from mobile devices, giving you a flavor of what it takes to work with real data complicated by inaccurate and noisy values. Inconsistent and erroneous location data can disrupt experience for services like Facebook Check In.[[1]](#footnote-1)

I chose a Kaggle competition because of the community around each project. It seemed to me to offer much more educational benefit in the developmental stages of a skill to be involved in a larger project with other ideas being floated around instead of creating an entirely unique project.

The dataset provided by Facebook for this competition is completely fabricated and includes a hypothetical “grid” of x,y coordinates, a timestamp, an ID of the check in location, and an undefined “accuracy” measure. In total, the training dataset is 29,118,021 data points. In my estimation, each of these data points is meant to represent an interaction with the Facebook mobile app and returns the equivalent of a GPS coordinate (the x,y value), a timestamp, the location of their next check in (place\_id), and a GPS accuracy measurement of unknown units.

# Problem Statement

The challenge is to predict, given only an x,y coordinate, a timestamp, and an accuracy measurement, where the user will check in next. This is a classification problem of a very large scale as evidenced in table 1:

Table - Unique values of each feature in the feature dataset

|  |  |
| --- | --- |
| Measurement | Number of Unique Values in Dataset |
| X Coordinate | 100001 |
| Y Coordinate | 100001 |
| Timestamp | 786239 |
| Accuracy Measurement | 1025 |

Using the data from Table 1, there are 8.059 x 1018 possible combinations of feature data to relate to 108390 unique places. The sheer size of this dataset makes this a challenging problem and as such I made some decisions and observations about this project before beginning.

## A Note on the Problem:

There are many ways to tackle this problem, however the most effective ways will most likely require the data to be read from a data store and use some in-storage reduction techniques such as MapReduce. That is beyond the scope of this course and while I realize this would most likely be a more realistic approach to solving this problem, for simplicity I will remain working from in-memory solutions. As such, the algorithms will be simplistic, the accuracy will suffer, and the overall usefulness of the solution will be tempered by the fact that simplistic algorithms such as a Nearest-Neighbors analysis will most likely not give highly accurate or reliable results.

All of this being said, my goal is to create a functional, memory efficient approach to solving this problem that will provide *some* useful insights. This approach will most likely not compare well to solutions that have more computational power or a more sophisticated data delivery approach but is an interesting educational experience regardless.

## Hypothetical Solution

This problem is a classic example of a classification problem with the true challenge coming from the vagueness of some of the features, namely timestamp and accuracy, the size of the dataset, and the number of targets. I will approach the problem in four stages:

1. Read in, format, and transform data into useable variables stored in memory, erasing any transient variables used in the process to preserve memory
2. Perform data exploration with basic statistics and graphical measures to understand the distribution of the dataset. This being a geographically related dataset there will also be a “location” graphical aspect to see if there are concentrations of features in specific geographic (x,y) areas.
3. Implement various classification machine learning algorithms with an eye toward efficiency and compare their results to find the best possible algorithm for this problem.
4. Select the appropriate machine learning algorithm based on the tests and tune the algorithm to provide the best possible result.

The problem will be judged based on the accuracy of check-in place\_id predictions. The metric used for these predictions will most likely be the built in SciKit Learn accuracy\_score as the F1 score used in previous projects during this course, to my understanding, is for binary classification problems which this problem most certainly is not.

Ideally the algorithm will return multiple possible check-in locations. The problem definition provided by Facebook allows for three possible solutions for each possible data point. There are two approaches to this, one would be to return the three locations with the highest probability from a single algorithm, or use multiple algorithms to provide their possible solutions. I believe that using a multi-algorithm approach is most interesting so that is what I will pursue. For purposes of defining the problem completely and easing computational load, I will return three possible place\_id’s and their accuracy score and will check the result against the true place\_id. If any of the three place\_id’s predicted equal the true place ID, that prediction is considered to be 100% accurate.

# Data Exploration

As previously discussed, the dataset is constructed of 4 features and 1 target. The 4 features are x coordinate, y coordinate, timestamp, and accuracy. The target data is a place\_id which is a randomly generated number associated with a hypothetical geographic (or business) location.

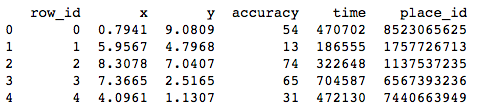


Figure - Sample of dataset

## X Coordinate and Y Coordinate

These can reasonably be correlated to GPS coordinates but have been normalized to a 10 by 10 grid with five significant digits, values on both of these can range from 0.0000 to 10.0000.

## Time

The time feature appears to be a random figure ranging from 1 to 786239, this precludes it from being a machine timestamp such as Epoch time or Unix time. It could be seconds, which would equate to 9.1 days, more likely the timestamp is in minutes which equates to an even 546 days. Considering the volume of interactions, neither of these solutions is entirely satisfactory. If we consider the timestamp to be seconds, that would equate to 3.2 million check-ins per day in this 10km square, where as if it is considered to be minutes that would equate to 53330 check-ins per day. The former case seems like too much volume while the latter seems like it could be too little. Without an understanding of the density of this hypothetical population area, I will have to make an educated guess as to which to choose. Considering how large of a number of check-ins would have to occur for this to be seconds, I will assume that it is minutes.

Why analyze the data beyond the existing timestamps? The time measure is completely linear and does not account for the cyclical nature of how people frequent businesses. Traffic at some businesses will be higher on a Monday, while others will be higher on a Saturday night. By converting the timestamps to a cyclical Day:Hour feature regardless of the year, month, week or even the minute, it reduces the timestamp data significantly from 786239 values to 168 (7 days, 24 hours). This pseudo-feature reduction (not in the strict sense) will make the timestamp a much more useful feature without sacrificing granularity. Most interactions with businesses occur within certain time blocks where, hypothetically, 12:45pm is not much different in interactions than 12:50pm and probably somewhat different than 1:45 pm. Thus we can assume that the 12:00 pm hour and the 1:00 pm hour can be treated as two individual times. If accuracy is lost by this reduction, it could be possible to create time blocks of 15 or 30 minutes to increase the amount of data

## Accuracy

This feature is left entirely up to interpretation. No explanation of the feature was provided at all including units or if a high number was either subjectively “good” or “bad”. With this in mind, to simplify the analysis of this data, I will filter all of the data removing all data points outside of one standard deviation from the mean accuracy measurement. Accuracy will then be removed entirely as a feature. This decision has two justifications, one it will remove noisy data, two it will reduce the number of features significantly, making computation easier.

# Statistics and Visualization

## Visualizing Check Ins by Day, Time, and Location

|  |  |
| --- | --- |
|  |  |

Figure - Check In Volume by Day of the Week and Hour of the Day

After breaking down time into a more manageable, cyclical measurement of day of the week (0-6) and hour of the day (0-23). From this point forward we will add 1 to all measurements to make them easier to talk about, this making our values from 1-7 and 1-24. We can see that although some patterns do arise, suggesting that day 2 has slightly more check-in volume than other days and that between hours 1 and 13 during the day check-in volume is higher, the amount of variation in the data is so small as to suggest that there are no highly significant variations in behavior in the day of the week, or hour of the day.

Next, an analysis of the physical space was performed to see if there was more check-in density from different blocks in the hypothetical test space.

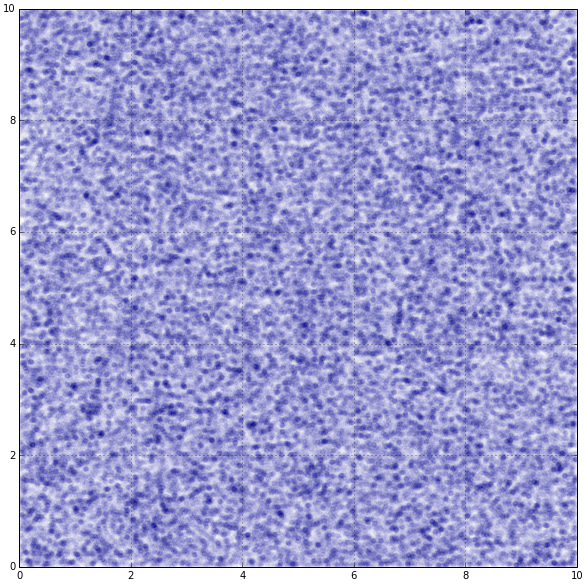


Figure - x,y coordinates of 500,000 randomly sampled check-in data points

Sampling 500,000 check-in data points randomly from the full set and plotting them reveal that there are no large patterns of check-in behavior based on location. Figure 3 appears to be a fairly consistent Gaussian distribution of data points.

## Accuracy analysis

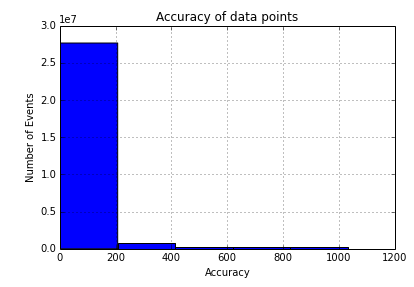


Figure - Accuracy of each event in data set

Accuracy is the least useful feature in this dataset as it is not defined and has no effect on user behavior, simply on the accuracy of correlation between their GPS coordinate and their following check-in. With that in mind, I will remove noisy data and remove the feature entirely from the dataset. Only data within one standard deviation of the mean will be retained for the remainder of this problem.

Table - Statistical analysis of accuracy feature

|  |  |
| --- | --- |
| Mean | 82.85 |
| Standard Deviation | 114.75 |
| Minimum Accuracy | -31.90 (0) |
| Maximum Accuracy | 197.60 |

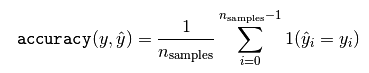
# Algorithms and Techniques

To baseline performance of this modified data, the k nearest neighbors classifier technique built into SK-Learn was used using 1 nearest neighbor and an accuracy score was calculated using the built in *score* function. The data was split into an 80% training 20% testing set and the results were as follows:

Table 3 - Benchmark predictions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Samples | Train Time (s) | Predict Time (s) | Predict/Train Time Ratio | Mean Accuracy Score (%) |
| 10000 | 0.014 | 0.183 | 13.07142857 | 0.90 |
| 100000 | 0.233 | 8.226 | 35.30472103 | 2.11 |
| 1000000 | 11.642 | 345.175 | 29.64911527 | 14.39 |
| 2000000 | 74.217 | 942.868 | 12.70420524 | 21.36 |
| 5000000 | 554.941 | 3184.24 | 5.737979353 | 30.49 |

Here, I am using the score function built into K Nearest Neighbors. This function returns the accuracy of the prediction, the competition on Kaggle uses a metric derived from precision. I have made this decision intentionally to ease computational difficulty. The formula for accuracy is as follows:



Proceeding from this point, I plan to use three algorithms, Decision Tree Classifier, Naïve Bayes Classifier, and Nearest Neighbors, to return three possible results. I also considered support vector machines, but based on recommendations in the documentation, it is not recommended to scale the support vector machine classifier to more than 10,000 samples.

## K-Nearest Neighbors

K-Nearest Neighbors is a simple algorithm that compares a data point that you wish to predict with its nearest K data points in the training set. In the case of any K, a classification is made by vote of the K neighbors, with the majority making the decision. For example given 5-NN to predict a color where 3 of the neighbors are red and 2 are blue, the vote would be red. This can be influenced by various weighting metrics but perhaps the most common is distance meaning those values closest to the prediction point are given more weight than those farther away.

While this algorithm is not memory efficient, 1-Nearest Neighbor on the entire unprocessed dataset was used by the sponsoring organization to create their baseline score. With that in mind, I included K-NN as a baseline and also as one of the three algorithms in my final composite solution. The format of this problem lends itself well to a Nearest Neighbors approach as the number of features is small, only 4 (x coordinate, y coordinate, day, hour), and with the large volume of data it is feasible that there will be very similar data points in the dataset. Thus making the comparative nature of K-NN most likely very reliable.

## Naïve Bayes

Naïve Bayes is an extremely simple, memory efficient algorithm known to be good at classification. The algorithm works purely on probability unlike a decision tree which is a logical progression. In an abstract sense, Naïve Bayes uses Bayes Theorem to assign probabilities to each class given some vector of features. In this way the winning classification is the one with the highest probability. Bayes Theorem is displayed below:



As this is a pure classification problem with a large dataset, I chose this algorithm for all of its strengths in the arena of large classification problems with limited feature sets.

## Decision Tree Classifier

A Decision Tree Classifier is simple algorithm to envision, consisting of a sequence of if-then-else decision nodes that encompass the entire feature set. A perfect (albeit long running) decision tree would have a node for every possible variation in a data set and at each node would pose an If X Then Y Else Z where X, Y, and Z are possible actions to take given the node state. This algorithm can be very costly if the number of data points is very large. The algorithm, however is easy to visualize and can be very accurate. I believed that prior to testing, a decision tree could return very good results when the size of the dataset is confined to smaller x,y coordinate boxes.

My fear is that my desire to run this problem entirely from memory will limit me to using a very small sample of the overall data. This could result in having to limit my data selection to 100,000 data points, randomly sampled, or less to return any solution at all. For this reason I will segment the data by x,y coordinates to refine possible solutions. In experimenting with sampling the data, I found that sampling within a tightly defined x,y coordinate produced a significantly train/predict/score cycle than using the entire data set. So instead of sampling large amounts from the entire dataset, I will sample much smaller amounts from a grid of x,y coordinates.

## Benchmark Score

A benchmark for this problem was provided by the competition organizer based on unprocessed data and 1-NN on the entire data set. Their score which is not accuracy but rather a metric referred to as Mean Average Precision @3 was 0.23967. As I’m using Accuracy in my calculations, I reran a 1-NN on my preprocessed data. The time and computational power required to process the number of data points limited me to using 5,000,000. Therefore my benchmark score will be 0.3049 or 30.49%.

# Preprocessing

Training data for this competition was provided as a single CSV file with 29,118,021 data points. The original features of X, Y, Time, and Accuracy were analyzed and processed. Time, which was a linear timestamp similar to Epoch or Unix time, was processed into a cyclical duo of “day of the week” and “hour of the day”. This was achieved with the function SetDay() in my code. This created a much more cyclical, understandable time structure. Second, the accuracy feature was analyzed and I decided to remove it as a feature altogether. Prior to removing the accuracy, I took the mean and standard deviation of the feature and removed all data points with an accuracy that fell outside of one standard deviation. This processed data set was sampled from directly for all calculations.

Intuitively, I did not view the data as one large 10 x 10 square of x, y coordinates. Instead, I made the assumption that people near to each other would have similar behaviors and people far away from each other would have very different behaviors. For that reason, I didn’t find it useful to analyze behavior at point (10, 10), and point (0, 0) in the same analysis. I created a measure called “side length” which is to say, If I split the data into coordinate “chunks” with origin at (1, 1) and side length of 0.25, I would only be looking at data contained in the square bounded by (1,1) (1.25, 1), (1.25, 1.25), (1, 1.25). This reduces the total area and data points within the training set.

## Implementation

### A note on scoring

The way this competition is set up, the organizer allows up to three predicted place\_ids for each data point. With that in mind, I had two possible approaches. The first would have been to use an algorithm which produces the probability of a prediction being correct and returning the three highest probabilities. The second, which I thought would be an interesting study, would be to use three total different algorithms and return their results. Therefore the final result is actually up to three unique place\_ids. Those ids are each considered unique and scored individually, if one or more of the three place\_ids are correct, the prediction is scored as 1, if all three are incorrect the prediction is scored as 0. This is not an average score, rather it is more like having three chances to be correct. From a user experience perspective this can be thought of as showing a user three possibilities from which to “check in”, if one of those three are correct, then the final experience would most likely be positive and the user would check in. If all three are incorrect, the user must search deeper to find the place they are checking in to.

### Testing results

To test the implementation, I will predict accuracy scores from five different x, y coordinate slices that do not overlap and average the accuracy scores to offer a prediction on the accuracy and robustness of the model as a whole. These are each independent training/testing operations with a completely new “training” operation on each coordinate set. These operations are blind to one another and training results from one are not used in another. This process will be done using the default settings for each algorithm. This will act as proof of concept in using my three algorithm approach as well as a baseline for implementing a Grid Search parameter search.

Table - Test Aggregate score of KNN, Naive Bayes, and Decision Trees with Default Settings

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | Y | Side Length | Samples (10,000 or max) | Testing % | Score |
| 1.0 | 1.0 | 0.2 | 10,000 | 20 | 0.591 |
| 2.0 | 1.0 | 0.2 | 10,000 | 20 | 0.5965 |
| 4.0 | 3.0 | 0.2 | 10,000 | 20 | 0.574 |
| 8.0 | 4.0 | 0.2 | 10,000 | 20 | 0.563938 |
| 5.0 | 9.0 | 0.2 | 10,000 | 20 | 0.5485 |
| Average Score | | | | | 0.575 |

This is a vast improvement over the baseline of 1-Nearest Neighbor which, when using the entire data set and sampling to 10,000, returned an accuracy score of 0.009. Compared to the 5,000,000 point baseline this is almost a 30% improvement.

Using this average score as a baseline for my “improved method” of reducing the data to x,y coordinate sets and running the data through three algorithms, I then ran an exhaustive Grid Search on my estimators to further refine the tunable parameters.

Using x, y coordinate side lengths of 0.25 and a sample of 10,000 data points, the Grid Search returned the following optimal parameters:

Table 5 - Grid Search Results

|  |  |
| --- | --- |
| K Neighbors | Decision Tree Classifier |
| {'algorithm': 'ball\_tree',  'leaf\_size': 10,  'n\_neighbors': 5,  'weights': 'distance'} | {'criterion': 'entropy',  'max\_features': None,  'min\_samples\_leaf': 10,  'min\_samples\_split': 4,  'splitter': 'best'} |

Naïve Bayes was not run through a grid search as it does not have any directly tunable parameters.

## Final Model Parameters

## K Neighbors

After performing a grid search on k-neighbors, the optimal parameters were as shown in table 6. Increasing the number of neighbors from 1 to 5, implementing the Ball Tree algorithm instead of Brute Force, and limiting leaf size to 10, and weighing results by distance.

The Ball Tree algorithm greatly increases efficiency over brute force methods as sample size grows so this choice makes sense. Leaf Size is a parameter used by the K-NN tree algorithms do deal with very small sample sizes. When sample sizes are small, brute force methods are more efficient and the leaf size parameter sets that threshold where the tree algorithm should switch to brute force to take advantage of those efficiencies.

## Decision Tree

Grid Search chose the entropy criterion, which judges a split in the tree based on the information gain. The minimum samples required to split a node was set to 4 and the minimum number of samples at each node to 10.

## Testing the Optimized Model

The side length was increased as a result of observing more accurate predictions with a slightly larger data set. Experiments were run at 0.2, 0.25, 0.3, 0.4, 0.5, and 1.0 with the best results in an identical test happening at a side length of 0.25.

The algorithm was re-run using the optimal parameters with a side length of 0.25 and produced the following results:

Table 6 - Optimized Solution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | Y | Side Length | Samples (12,500 or max) | Testing % | Score |
| 1.0 | 1.0 | 0.25 | 12,500 | 20 | 0.6176 |
| 2.0 | 1.0 | 0.25 | 12,500 | 20 | 0.6056 |
| 4.0 | 3.0 | 0.25 | 12,500 | 20 | 0.594 |
| 8.0 | 4.0 | 0.25 | 12,500 | 20 | 0.58 |
| 5.0 | 9.0 | 0.25 | 12,500 | 20 | 0.5868 |
| Average Score | | | | | 0.5968 |

The optimized solution has improved prediction accuracy by an additional 2.2% over the initial run. Thus making a model that is on average 29.19% more accurate at prediction check-ins than the 1-NN baseline using 5,000,000 data points. More impressively, this has been done using only 12,500 data-points meaning that a local prediction can be made very quickly with an average training time of 0.17 seconds. Figure 5 visualizes these advances made in this process from the small data set 1-NN baseline using 10,000 points, to a larger data set of 5,000,000 to finally the composite solution using only 12,500 data points.

Figure - Final Results Comparison of Training Time and Accuracy.

## Conclusion

The goal of this problem was to predict future behavior of a user, namely, where the user might “check-in” given some basic orienting data such as what time it is, and where they are using a corollary of GPS coordinates, in this case simple X,Y Cartesian coordinates. Users were given 29 million historical data points from which to train an algorithm to predict future behavior.

Beginning with over 29 million data points, this was a challenging problem. The sheer size of the dataset presented challenges to running computations, and multiple attempts failed or completely crashed my machine. In the end however, I believe that this is a realistic and complete solution that shows massive improvement over the provided baseline from the competition organizer and my baseline.

The first steps to arriving at this solution involved completely understanding a somewhat under-defined data set. Adjusting the time scale from linear to cyclical was critical in relating location to behavior. The accuracy measurement was interesting but complete under-defined and was discarded after using statistics to select only data points with an accuracy measurement within one standard deviation from the mean.

After reformatting and reducing data, I then chose to only sample data from small squares of data within the x, y coordinate system. Each one of these squares is treated as its own unique problem. This increased accuracy substantially while lower computational load and decreasing training and testing time. This was done primarily on the assumption that people who behave similarly will be geographically related. Considering the only other metrics in the dataset are related to time, it seems that this was a valid assumption.

The guidelines of the competition said that up to three places could be predicted for each data point and if any of the three are correct then the prediction of the data point will be considered correct. With that in mind I chose to run each data point through three classification algorithms, Decision Tree, K Neighbors, and Naïve Bayes, and score all three results. This approach yielded drastically improved results over the 1-Nearest Neighbor baseline.

The three algorithm composite approach was further refined using a comprehensive grid search of their applicable parameters which yielded a minor improvement in accuracy. In summary overall accuracy was improved by nearly 30% while training and prediction time was significantly reduced.

In summary, the solution presented here is a vast improvement over both a random selection or a basic 1 nearest neighbor algorithmic approach. The accuracy is vastly improved while using a much smaller data set. This method of using a composite approach provided an improvement in accuracy of nearly 30% while using 99.8% less data, thus dramatically improving prediction times. I believe that this is a successful project given the constraints I outlined, namely running this as a single process, in memory, on a single machine.

## Further Refinement

The time necessary to process much larger chunks of the data prevented me from testing it, however I can foresee improvement being made by using larger data sets on a larger coordinate slice and running those larger sets through a grid search to tune parameters. I believe I chose algorithms that were robust enough to handle large sets but running the search and validating the data was so computationally intensive that the processes would run for hours, which added no educational value in my mind, and was thus scrapped.

## Next Steps

I plan to use my algorithm to predict all 8+ million data points presented by Facebook as their test set and submit my results to the competition. This will be a time consuming computation which I am estimating to take 20+ hours.

1. Kaggle, <https://www.kaggle.com/c/facebook-v-predicting-check-ins>, (May 25, 2016) [↑](#footnote-ref-1)