Final Report

Chi,Calvin & Adams, Cameron & Konagaya, Eugene & Lee, Daniel

December 9, 2016

1 Abstract

Introduction: Our final project was a prediction problem using data from a data science competition hosted by Kaggle and Facebook (https://www.kaggle.com/c/facebook-v-predicting-check-ins). For the competition, Facebook created an artificial world consisting of more than 100,000 places located on a 10 km by 10 km geographic square. The data simulates noisy location signals from mobile devices Facebook collects from its users. Real world signal data varies in signal quality and accurate prediction therefore requires incorporating location error into the prediction. The data consists of check-in observations with the following variables:

1. row-id: a unique id for the observations (training: 29,118,021 observations; test: 8,607,230 observations)
2. x,y: map coordinates where check-in occurred
3. signal accuracy: how accurate the location data is, i.e. signal quality
4. time: time the check-in occurred
5. place-id: id of the place (108,390 unique places in training data)

Our group conducted exploratory data analysis learned this semester to examine the data, and discovered cyclic relationships between check-ins and time. We used K-nearest Neighbors (KNN) and Neural Networks (NN) algorithms to predict check-in place-ids in the test dataset.

Methods: We first implemented KNN and NN on a dataset with known places. The Kaggle training data set was divided into a validation training (90% of the data) and test (10% of the data) sets. After tuning our KNN and NN parameters on the validation set we implemented KNN and NN on the full Kaggle training and test data and submitted the predicted place-id for each check-in event in the test set.

Results: After tuning and parameter optimization, KNN and NN successfully predicted the place-ids of the check-in events in the validation test set with 56.51 and 44.4 mean average precision, respectively. On the test set, KNN performed substantially better than NN, with mean average prediction precision of 47.6 and 4, respectively.

2 Introduction

Facebook and Kaggle launched a data science competition in which competitors were asked to predict what place a person would check-in to based on the following variables:

1. row-id: a unique id for the observations (training: 29,118,021 observations; test: 8,607,230 observations)
2. x,y: map coordinates where check-in occurred
3. signal accuracy: how accurate the location data is, i.e. signal quality
4. time: time the check-in occurred
5. place-id: id of the place (108,390 unique places in training data)

For the purposes of this competition, Facebook created an artificial world consisting of more than 108,390 unique places located in a 10 km by 10 km square. This data was divided by time into a training set with known place-ids (the past) and a test set with unknown place-ids (the future). For a given set of features in the test set, our task was to return a ranked list of the three most likely place-ids for a check-in event. Data was fabricated to resemble location signals coming from mobile devices, providing an impression of what it is like to work with real data complicated by inaccurate and noisy values.

We implemented neural networks (NN) and K-nearest neighbors (KNN) algorithms to predict the place-id .

3 Data

Kaggle and Facebook fabricated an urban area with check-ins and places over a period of time (0-1,000,700). To create the training and test sets, they split the data by time. All data with time $< 786{,}200$ were in the training data set and all data with time $\geq 786{,}200$ were in the test set. The training dataset consisted of 29,118,021 observations and 108,390 unique places. The test dataset had 8,607,230 observations of unknown place-id. Each observation was considered a "check-in" event. Check-in events do not represent individual or groups of people. For the training set, each "check-in" consisted of xy-coordinates, signal accuracy, time, and a place-id.

To understand the data better and discover possible relationships between check-ins and features, we performed exploratory data analysis on the training set.

3.1 Place ID

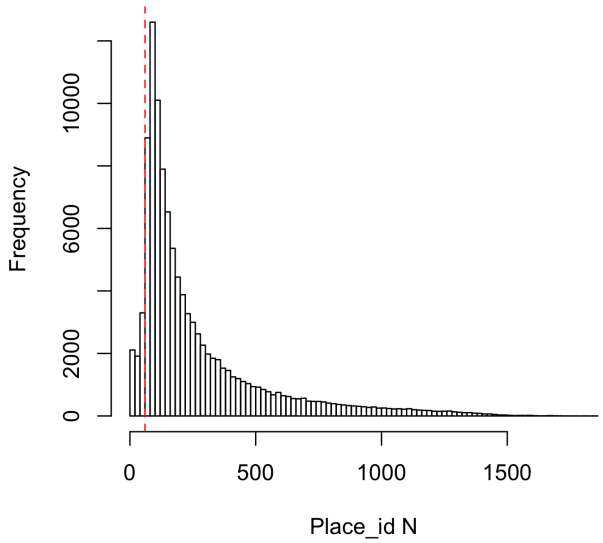


Figure 1: Place-id N frequency.

The number of check-ins per place-id range from 1 to 1,849 with the mean of 268.64. There appears to be a steep drop-off in place-id frequency at N < 60 (Figure 1). The right-skew of the distribution has shifted mean place-id N frequency away from the mode (N $= 100$). Most place-id have between 100-200 check-ins (Figure 1).

3.2 X,Y-Coordinates

Table 1: Place-id mean range and variance of X and Y coordinates.

Mean Range of X	Mean Range of Y	Mean Variance of X	Mean Variance of Y
5.9617	0.2124	0.7303	0.0012

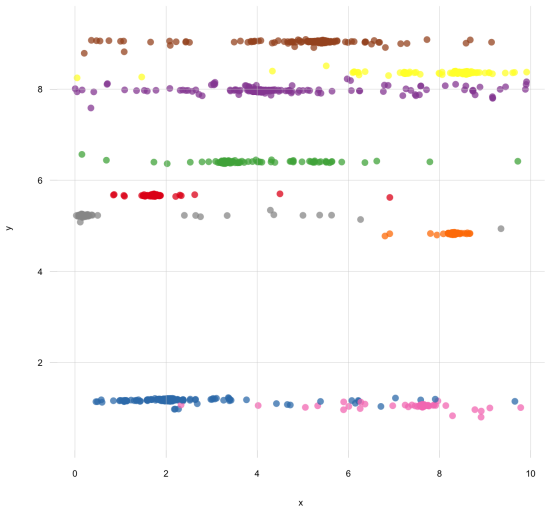
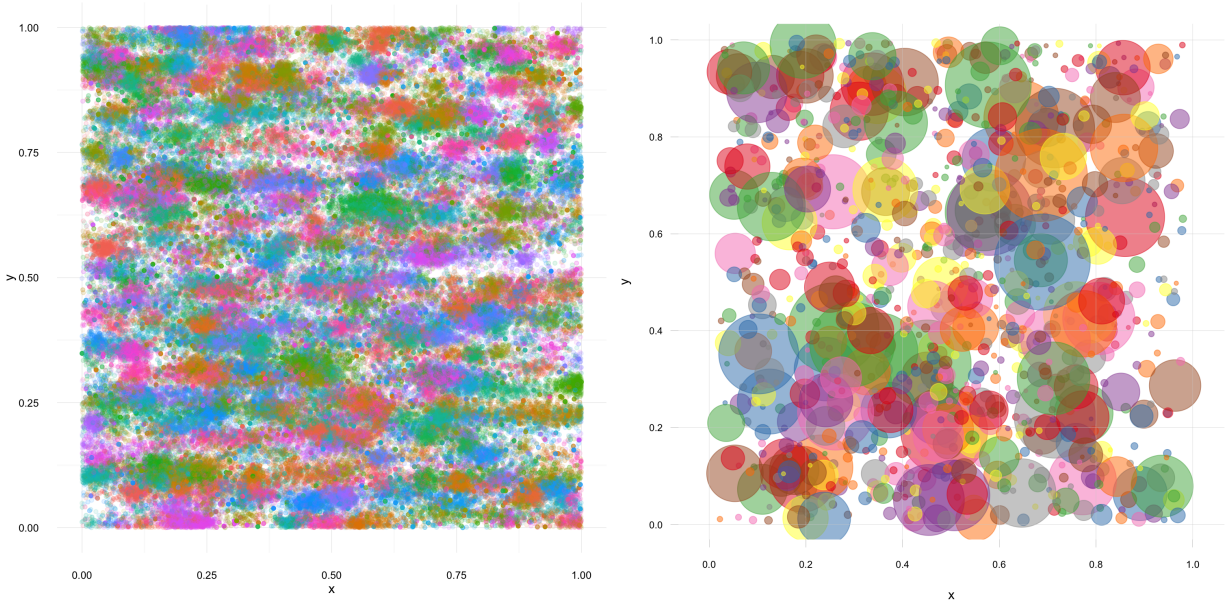


Figure 2: Random place-id plotted in 10-km x 10-km grid.

There was much more variance in place-id check-ins in the x coordinates than in the y coordinates (Table 1). Figure 2 shows check-ins for nine randomly selected place-ids in the training set. Many of the place-ids have check-ins that span nearly the entire 10-km x-plane. The lack of variance in the check-in y coordinates suggests that y coordinates are more informative with respect to place-id and therefore are better predictors than the x coordinates.



(a) Scatterplot of check-ins in 1-km x 1-km grid colored by place-id.

(b) Scatterplot of place-id mean(xy), size $\sim$ number of check-ins.

Figure 3: Scatterplot of check-ins

Figures 3a and 3b are scatterplots of check-ins in a 1 km^2 square. They show that check-ins for different place-ids overlap, bleeding together and leaving no obvious geographic boundaries between place-id's. This indicates that correctly predicting place-id check-ins by x,y coordinates alone would be difficult.

3.3 Time

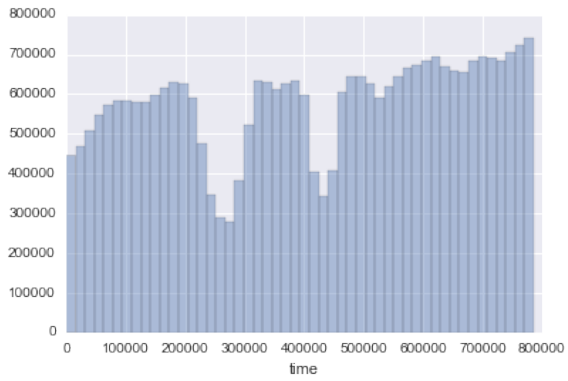


Figure 4: Histogram of Training data time feature.

The training data time feature ranged from 1 to 786,239 seconds. Figure 4 is a plot of the training data check-ins over time. There was a decrease in check-ins between time= 200,000-300,000 seconds and time=400,000-475,000. We hypothesized that a circular time feature would capture cyclic behavioral patterns. Most human activities occur at regular time periods, such as getting coffee in the morning, shopping on weekends, and going to school during the day. To investigate check-in behavior during such regular time intervals, we created new time features: hours (0-23 hours/day) and minutes (0-1439 minutes/day). These features did capture cyclic check-in behavior and check-ins did generally occur during specific times of the day (Figure 5a). Hours and minutes both captured the same general pattern in daily check-in frequency, with minutes obviously providing more granular information about daily check-ins than hours (Figures 5b and 5c).

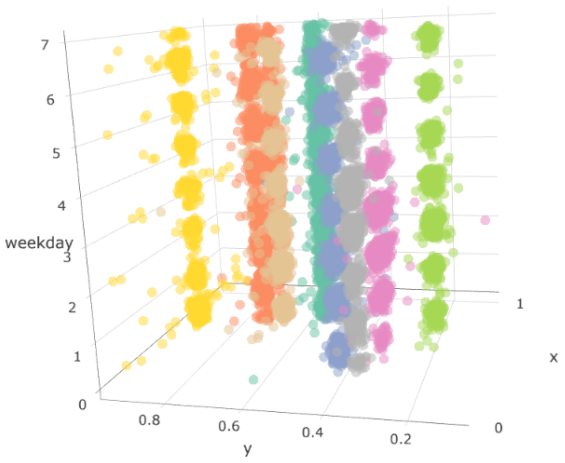


Figure 5: 3D-Scatterplot: place-ids, 1 km x 1 km x Day.

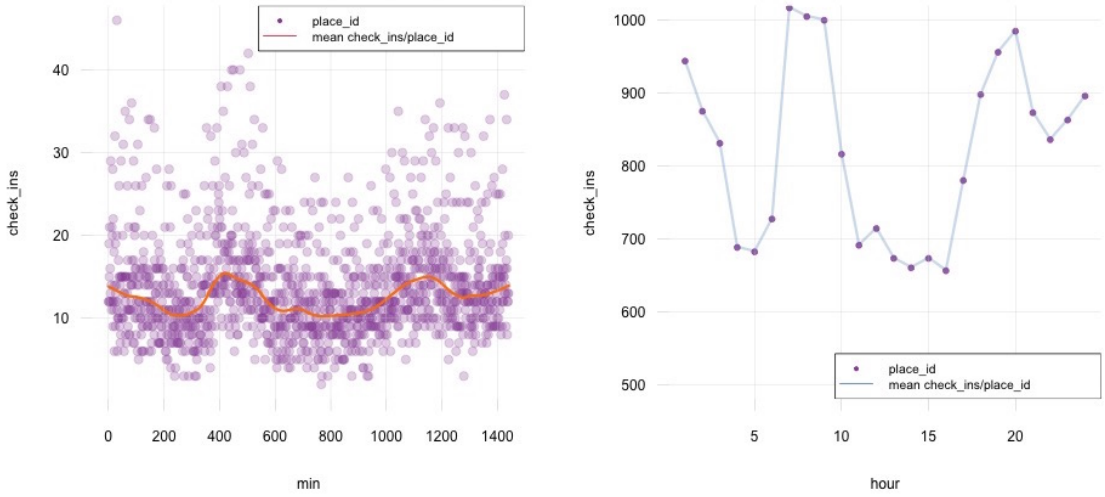


Figure 6: Time and Hours Plots

The 3-D scatterplot in Figure 5a shows clusters of check-ins at particular times of weekdays interspersed with periods of low frequency of check-ins. This pattern is consistent across all seven weekdays and emulates the typical work and sleep pattern of normal daily life and indicates that hours and minutes were effective features for capturing cyclic check-in behavior.

Time data, such as hours and minutes, are circular - times such as 23:59 and 00:01 are adjacent. Ideally, we would have transformed hours and minutes onto circular coordinates (e.g., radians or degrees). However, we were not able to incorporate circular transformation of hours and minute into our predictions before this report was submitted.

3.4 Accuracy

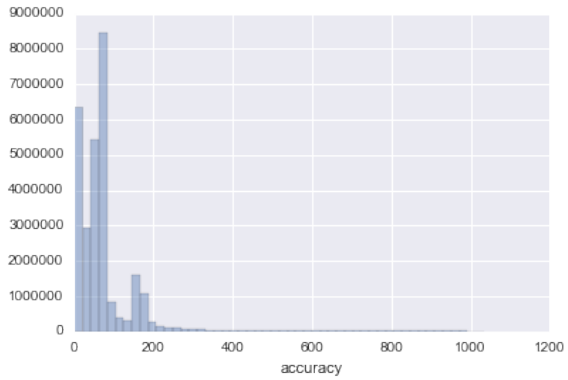


Figure 7: Histogram of Accuracy Feature.

Signal accuracy ranges from 0 to 1,033 and has a mean of 85.7. The vast majority of signal accuracy values are less than 200 and most range from 0 to 100.

We hypothesized that signal accuracy was a measure of xy-coordinate accuracy. This is analogous to GPS, where signal accuracy affects the precision of latitude and longitudinal coordinates. To determine if signal accuracy and check-in coordinates were correlated we calculated the Manhattan distance between the check-in xy-coordinates and the mean x and mean y coordinates of the corresponding place-id. Then, we examined if this distance was correlated with signal accuracy. The correlation between the Manhattan distance and signal accuracy was 0.024. We concluded that the Manhattan distance from mean center was not correlated with signal accuracy.

Next, we calculated the mean of the Manhattan distances between one check-in and all other check-ins for each place-id and calculated correlation between place-id check-ins mean Manhattan and signal accuracy. They were also not correlated (0.019). Figure 7 is a density plot of these mean Manhattan distances. The typical distance between check-ins within the same place-id was less than 0.15 km, but it ranges from 0 to 10 km. This indicates that most place-id check-ins are clustered relatively closely together.

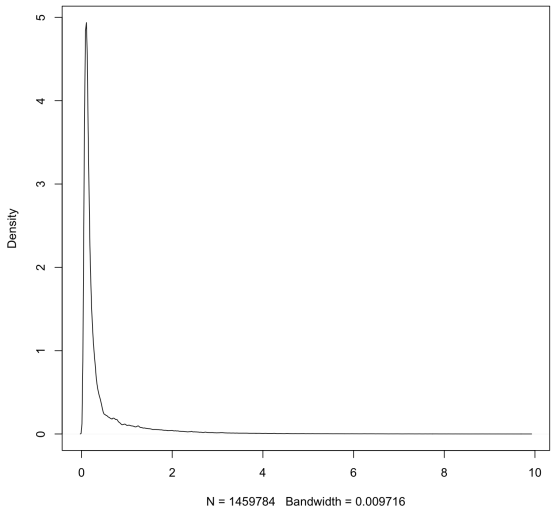


Figure 8: Density of mean Manhattan distance between place-id check-ins.

We could not find any patterns related to signal accuracy and xy-coordinates. Accuracy may not be linear and certain specific ranges of signal accuracy may correspond to stronger and weaker signal accuracy.

4 Method

Based on the exploratory data analysis, we selected the appropriate features to use for the predictions. We implemented KNN and NN algorithms to predict place-ids.

The large size of data (training =1.3 gb, test=0.4 gb) made running KNN and NN on the entire data in one iteration impossible on normal laptop computers. We decided that subsetting the data by xy-coordinate into grids and running prediction on each grid would be more computationally efficient. Each grid prediction could also be parallelized, reducing computation time further. To determine the optimal grid size, we investigated patterns in density (number of check-ins per place-id in a grid) for different grid sizes.

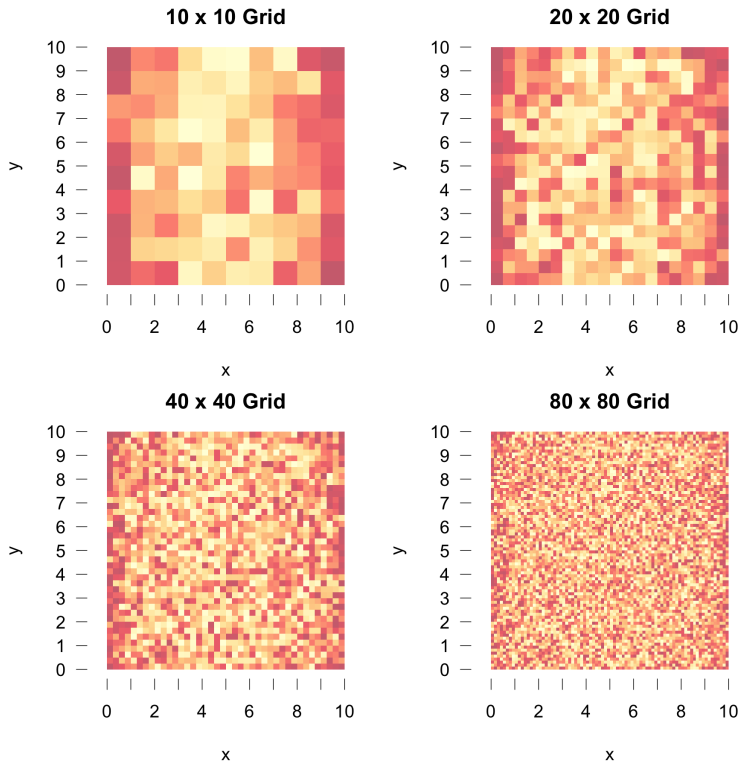


Figure 9: Grid Density Plots.

Figure 9 shows that there was higher density of check-ins in grids located near the x-boundary edges of the the 10km by 10km square. This pattern was consistent in larger (10x10) and smaller grid sizes (80x80). The distribution of mean Manhattan distance between check-ins within the same place-id (Figure 8) indicated that the 40x40 grid design would preserve place-id check-ins clusters. Additionally, we decided that grids that accounted for the large range in place-id x values (e.g, 1 km x 10 km) would not improve prediction. Check-ins that are far from the main cluster of check-ins would be difficult predict because they were far from their sister check-ins. Data in these larger grids of this size would also be quite large ($\sim$130 mb) and not significantly improve computation and processing time issues. We chose a 40 x 40 (0.25 km x 0.25 km) grid design because it was a good balance between cluster integrity and computational efficiency.

KNN and NN predictions were run on each grid and compiled into a single output.

4.1 Training

The Kaggle training and test data are only different by time. The test set was selected from the most recent check-ins. To mimic that in our KNN and NN training procedures, we split our training set into 90% training and 10% validation set, where the 10% validation set was composed of the 10% most recent check-ins.

Our task was to submit the top 3 likely check-in locations for each check-in in the test set. Given one true check-in location, the measure of performance used by Kaggle was mean average precision (MAP).

$$MAP@3 = \frac{1}{|U|}\sum_{u=1}^{|U|}\sum_{k=1}^{min(3,n)} P(k)$$

where $|U|$ is the number of check-in events, $P(k)$ is the precision at cutoff k, and n is the number of predicted businesses.

Let us first explain what average precision was in the context of our study. If the first guess is correct, we receive 1 point. If the second guess is correct, then we receive 1/2 points. If the third guess is current, then we receive 1/3 points. Otherwise we receive 0 points. MAP is simply the mean of average precisions for all test sample points.

For our NN prediction, our strategy for dealing with the large number of classes we need to predict was to restrict classes to those with a minimum number t number of samples. This was done on a per-grid basis and a different threshold t was determined for each grid. Increasing the threshold t allowed a more accurate classifier to be trained but also increases the number of mistakes made on excluded classes. Our strategy for determining the optimal t was the same as that used to determine the best regularization parameter. An example plot of MAP with various values of t is provided for illustration:

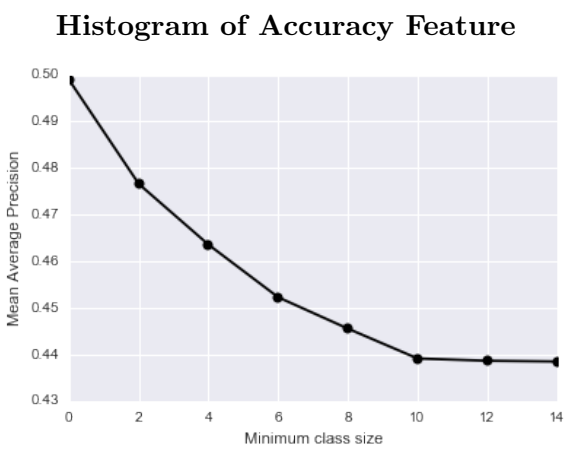


Figure 10: Hyperparameter tuning.

In this example, since $t = 0$ gives the highest MAP, $t = 0$ is chosen as the threshold. With all our methods described, a total of 1600 neural networks were trained for the 1600 grids we have.

4.2 Neural Networks

Previous contestants of the Facebook check-in challenge used relatively simple models such as K-nearest neighbors or naive Bayes. We wanted to see that given the current popularity of NN, whether the complicated neural network could potentially perform better.

Because our dataset only has 4 features, only one intermediate layer for our neural network was sufficient. In practice, the number of neurons in the intermediate layer is the average number of neurons between the input and output layers, and this is the rule we adhered to. From the histogram of the place-ids in Figure 1, we can see that most place-ids have more than 60 check-ins.

We considered only looking at place-ids with more than 60 check-ins to make our predictions.

With the architecture of the neural network determined, we only need to determine the hyper-parameters. Our hyperparameters are the learning rate for gradient descent and regularization parameter. Empirically, we found the learning rate of $\alpha = 0.1$ to work well with performance and time considerations. A regularization parameter is tuned for each model corresponding to each grid. The regularization parameter is chosen based on the model that has the highest MAP score over the regularization parameter space.

Neural network is a powerful supervised machine learning algorithm suited for both regression and classification problems. A unique feature about this algorithm is that it allows for the implicit learning of features without having the user explicitly design them. The neural network is constructed by several neurons with the following architecture:

Histogram of Accuracy Feature

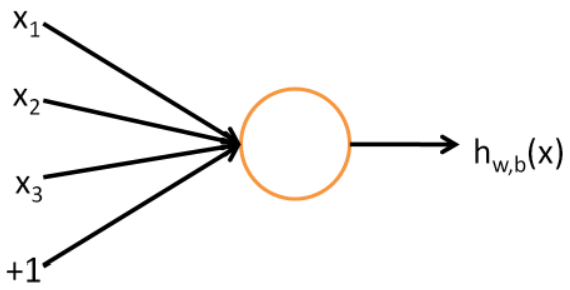


Figure 11: Neuron Structure.

In this representation, $x_1, ..., x_3$ are the input values for features 1 to 3, and $+1$ is the bias term to increase the model space. The neuron then accepts these input to output $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + W_4 b)$, where $W \in \mathbb{R}^{4x1}$ is a vector of weights for each of the input values and b represents the bias of $+1$. $f()$ is an activation function, typically chosen to be the sigmoid function or tanh function. The activation function serves the purpose that small changes in the weights and biases only results in small changes in the final output.

A neural network is built from these individual neurons. An example of a neural network with 3 layers looks like this:

Histogram of Accuracy Feature

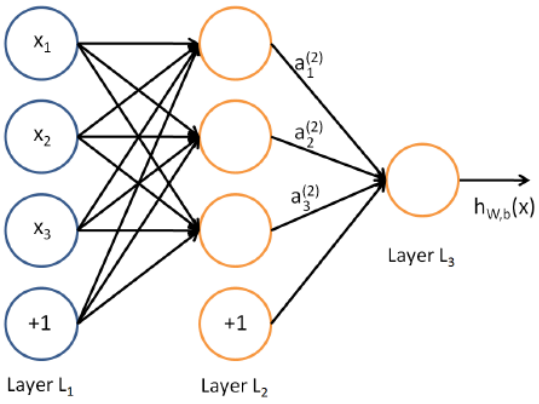


Figure 12: Example of neural network architecture.

With this architecture, our weight vector becomes a matrix instead. To go from layer L_1 to layer L_2, we would need a $W^{(1)} \in \mathbb{R}^{4x3}$ matrix, where 4 is the number of input values n_{in} and 3 is the number of output values n_{out}. The neurons in the second layer are denoted as $a_j^{(2)}$, where j denotes the j^{th} neuron. According to above diagram, $a_j^{(2)}$ is calculated as $f((\sum_{i=1}^{3} W_{ij}^{(1)} x_i) + W_{4j}^{(1)} b_j^{(1)})$. To go from layer 2 to layer 3, one would use a $W^{(2)} \in \mathbb{R}^{4x1}$ vector, and the value of $h_{W,b}(x)$ is computed as $f((\sum_{i=1}^{3} W_{i1}^{(2)} a_i^{(2)}) + W_{41}^{(2)} b_1^{(2)})$. The logic of these computational steps can be extrapolated to neural networks in general with a number of layers and a given number of neurons per layer. Note that a neural network with more than 3 layers is often called a deep neural network.

The number of features we have in our data matrix determines the number of neurons in the first layer, plus one extra neuron to represent bias. The number of output neurons in the last layer is determined by the number of classes to predict, with one neuron corresponding to each class.

Training a neural network involves iterating the values of the weight matrices via gradient descent to minimize error, which could be set as the mean square error or cross entropy error. Either way, the cost function is non-convex such that the result from training may represent a local solution instead of a global solution. One way to combat this is to randomly initialize the weight values prior to training.

To do gradient descent, we employed a cross entropy loss function. Cross entropy was used to quantify how similar two probability distributions Q and P are. Cross entropy is defined as:

$$R = \sum_i p_i log_2(\frac{1}{q_i}) = -\sum_i p_i log_2(q_i)$$

R is minimized when Q and P is the same. The reason why we chose cross entropy over mean squared error is because cross entropy increased the speed of training compared to gradient that is proportional to the difference between the predicted output and actual output.

To account for the large number of of prediction classes, we chose the softmax activation function to be the output activation function:

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Where k is the number of classes and i refers to class i and $z = Wx + b$. It can be easily verified that:

$$\sum_{i=1}^{k} a_i = 1$$

And that:

$$1 > a_i > 0$$

Which means a can be interpreted as a probability distribution of k distinct classes. In probabilistic terms, the joint distribution of the classes is a multinomial distribution with $n = 1$, since we are calculating the probability of a sample being one of k distinct classes.

For the rest of the activation functions, we chose the rectifier activation function, defined as:

$$f(x) = \max(0, x)$$

A neuron employing the rectifier is called the rectified linear unit (ReLU). The reason we are choosing the rectifier activation function is because it does not suffer from the vanishing gradient problem. In the vanishing gradient problem, if the output is close to 1 or 0, their derivatives approach 0. Another advantage of ReLu is inducing sparsity in the neural network. Sparsity means that a significant proportion of outputs from each neuron is 0.

4.3 K-Nearest Neighbor

To compare performance of our neural network prediction, we also implemented a KNN algorithm. KNN is a non-parametric supervised learning algorithm that uses the majority vote of an object's neighbors to predict classification (Ripley et al 1996). In the most simple case of k=1 neighbor, an unclassified object is given the classification of its nearest neighbor. An object's nearest neighbors are the closet k neighbors as determined by a distance function. As the density of points increases (i.e. cluster), it becomes more likely that the nearest neighbor(s) are good predictors of the object's true classification.

We used the FNN package in R to implement KNN (Alina et al 2013; R Core Team 2016). FNN's KNN algorithm uses Euclidean distance from test object to training objects to determine nearest neighbors. Euclidean distance is defined as:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{ip} - x_{jp})^2}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are vectors with n components.

As with the NN test run, the training dataset was sorted by time and divided into a new training set (90% of check-in) and test set (10% of check-ins). The features used for predictions were:

1. x, y coordinates

2. signal accuracy
3. time
4. hour (0 - 23) of check-in
5. min (0 - 1,439) of check-in

All feature data were normalized to between 0 and 1, and time features (time, hour, and minutes) were weighted by 1/3 to balance their combined importance with the other features in the prediction.

To determine an optimal number of nearest neighbors, k, 5% of the 1,600 grids were randomly sampled and place-ids were predicted using 3 to 49 neighbors. Figure 13 below shows the mean MAP k=3 to 49 neighbors. The highest average MAP(63.3) was achieved at k = 3. However, as the project required submission of three predictions per observation, we needed a large enough k that would return more than one predicted place-id. We selected k=21 neighbors because it produced a similar mean MAP as k=3 (62.1), and was large enough to return three place-id predictions.

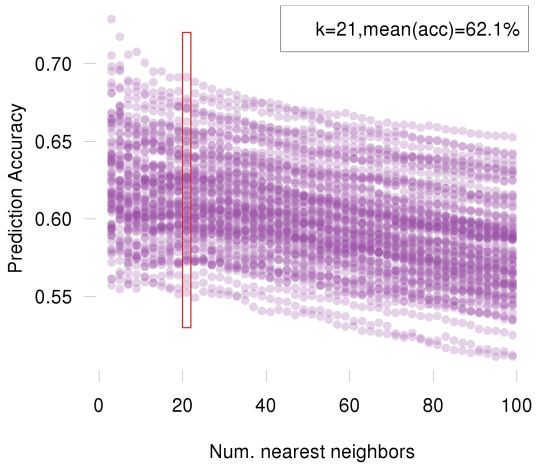


Figure 13: KNN k optimization.

To improve algorithm-processing efficiency, KNN prediction was parallelized to 20 processor cores. The top three place-id predictions correspond to the top three nearest neighbor votes. MAP of those predictions was assessed per grid and overall against the true place-ids. The test set predictions were evaluated by Kaggle.

5 Results

5.1 Validation Results

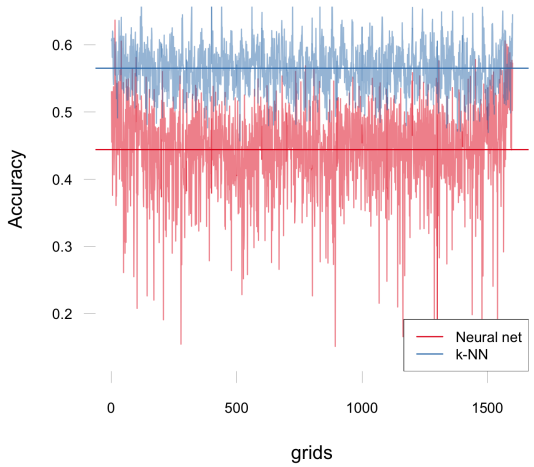
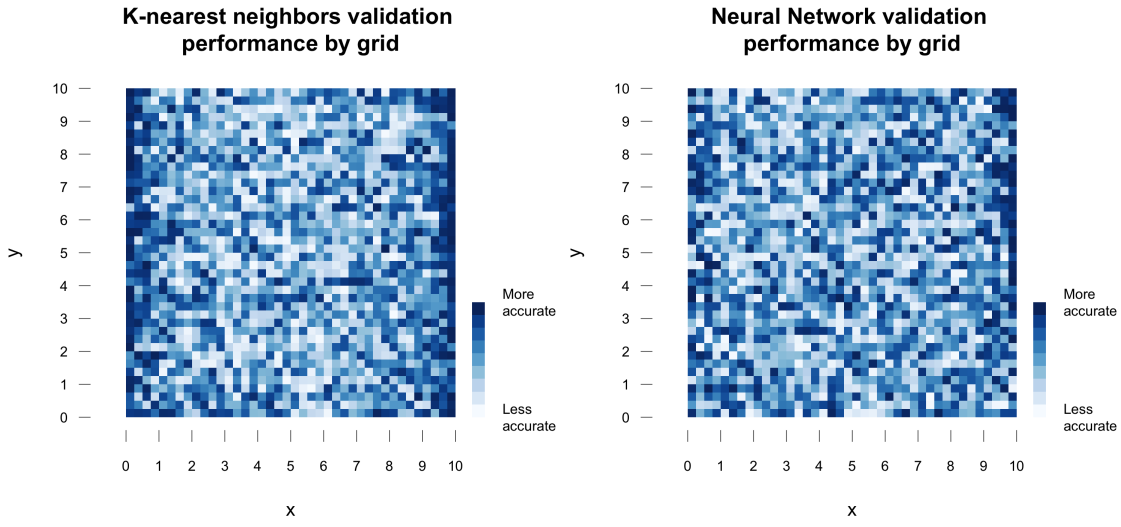


Figure 14: KNN vs NN validation performance.

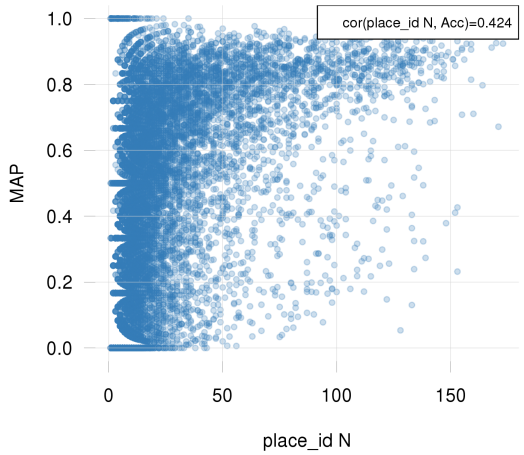
Prediction MAP for KNN was generally higher than the NN prediction MAP across the 1,600 grids. For NN, the validation test performed with MAP of 44.4. KNN had a MAP of 56.51 on the validation test. For both KNN and NN, prediction MAP was better on the grids near the x=0 and x=10 edges of the 10-km by 10-km square and worse in grids near the center of the feature space.



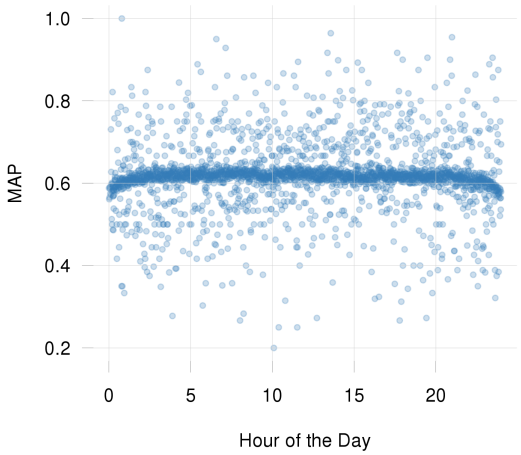
(a) KNN MAP.

(b) NN MAP.

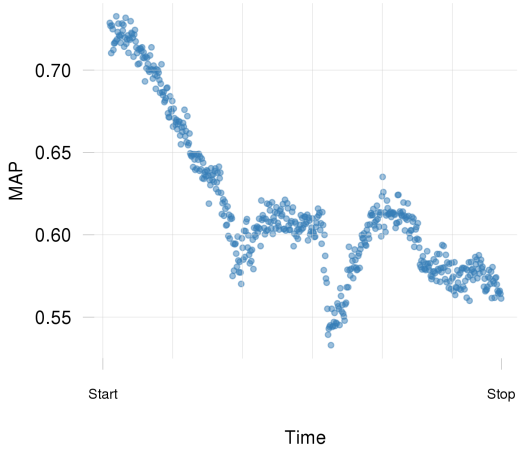
Figure 15: KNN and NN MAP by grids for validation test.



(a) Place-id N and MAP.



(b) Hours and MAP.



(c) Time and MAP.

Figure 16: KNN validation MAP results.

Prediction MAP appears to be correlated with frequency of place-id check-in. Place-ids with more check-ins generally had a higher MAP than less popular places. Prediction MAP decreases at the beginning and end points of the day cycle. Otherwise, MAP was constant across time. Best prediction MAP was at time=0 and generally decreased over time, with to brief peaks over the full time period.

5.2 Kaggle Results

NN and KNN prediction MAP on the test set was 4 and 47.6, respectively (Appendix).

6 Discussion

For the Kaggle Facebook check-in prediction competition, we implemented NN and KNN algorithms to predict where future Facebook check-ins would occur in a simulated urban area on a 10 km by 10 km square. We had better results on both the validation tests and Kaggle submissions with the KNN algorithm. One possible reason for this was because we filtered out place-ids with less than 60 check ins in our NN implementation and not KNN, making it impossible to for NN classify check-ins with these place-ids.

The higher prediction performance seen in grids along the boundaries of the x-coordinates indicate that MAP may be correlated with density. This indicates that a grid design that determines grid dimensions by density may improve performance.

The poor performance of NN in the Kaggle submission was likely due to a technical glitch in the export of the prediction to .csv format. Correction of the error was not completed before submission of this report.

Improvement of prediction performance could be achieved in a number of ways. As mentioned earlier, transformed cyclical time variables hours and minutes into a circular format would preserve the distances between adjacent time points. Additionally, determining optimal weights for features, such as up-weighting y coordinates, and down-weighting x coordinates might further improve performance.

Further analysis includes investigating relationship between signal accuracy and time. It is possible that specific ranges of signal accuracy correspond to strong and weak signal.

And lastly, we could also implement methods other than KNN and NN. Top performers on Kaggle used methods such as xgboost and random forest and achieved MAP > 60.

7 References

1. http://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf
2. http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
3. Alina Beygelzimer, Sham Kakadet, John Langford, Sunil Arya, David Mount and Shengqiao Li (2013). FNN: Fast Nearest Neighbor Search Algorithms and Applications. R package version 1.1. https://CRAN.R-project.org/package=FNN.
4. R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

8 Appendix

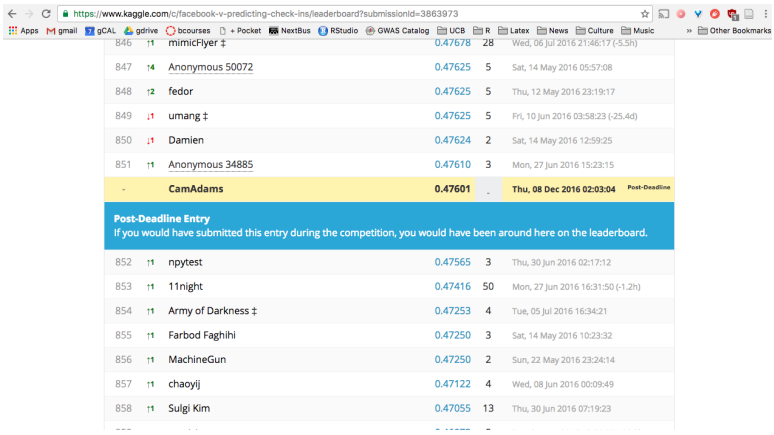


Figure 17: KNN Kaggle submission MAP.