Taxicab Trajectory Classification using Meta Learning

Ashley M. Schuliger
Department of Computer Science, Worcester Polytechnic Institute, amschuliger@wpi.edu

Additional Keywords and Phrases: Taxi trajectories, Classification, Meta learning, Recurrent neural networks

1. INTRODUCTION

Taxis have been a major source of public transportations in large cities, such as New York City, for decades. However, the recent rise of rideshare companies, such as Uber and Lyft, has steadily decreased the popularity of taxis. In 2014, taxicabs serviced approximately 170,900,000 passengers with Uber only servicing 4,500,000 passengers. The scene changed drastically over three years as the approximate number of passengers served by Uber climbed to 159,900,000 and the number of passengers served by taxi-cabs declined to 125,500,000 [1]. This decrease in taxi usage is best explained by the wait time difference between each service. In a study by Forbes, passengers stated that they chose a ride-sharing service instead of a taxi most commonly due to the short wait times and travel times of the ride-share [2]. Due to this, it is imperative that the taxi industry find ways to decrease their overall wait time and travel time in order to regain the confidence of their customers. One way they could improve these statistics is by using their final destination to make decisions regarding their subsequent passengers. As a result of recent technology innovations, taxi cabs are equipped with GPS technology that allows the taxi dispatchers to better coordinate the passengers that taxis would serve by knowing the location of each taxi. However, GPS technology has no way of determining the final destination of a taxi driver, rendering the current GPS coordinates of a taxicab useless. Ride-share services, such as Uber and Lyft, have been able to clear this hurdle by allowing drivers to choose who their next passenger will be. This allows drivers to use their own final destination to determine which passenger they can service with the shortest wait time. While this framework makes it possible for Uber and Lyft to service passengers quickly, this framework is not possible for the taxi industry. While taxi drivers are employees of a company, Uber and Lyft drivers are independent contractors, which allows them to build their own schedule and choose their own passengers. Taxi drivers are given passengers from a taxi dispatcher and thus, cannot use their final destination to choose their next passenger. However, past trajectories of these taxi cab drivers contain an overwhelming amount of information regarding their driving behaviors, including their speed along a route, their most frequently visited regions, and the average distance that they drive to serve a passenger. Due to this, it may be possible to use these driver behaviors to determine whether or not a trajectory should be assigned to a specific driver or not. Recent related works have proposed the use of a deep neural network in order to classify a trajectory as being from a specified driver. Classification applications such as these are easy to implement if a large training set is available that consists of several trajectories each for a small set of drivers. However, this method is unreasonable for this application due to the complexities of real-world conditions. In order to accurately use this model, a large number of trajectories would need to be collected from each individual taxi driver. Since the taxicab industry has a constant flow of taxi drivers leaving and becoming employed each day, collection of these trajectories would require an amount of time and resources that do not justify the profits gained from an overall decreased wait time for passengers. Thus, an analytics method that requires less time and resources to build an accurate model should be considered. Since the overall goal is to assign a trajectory to a driver based on their driving behaviors, it may be possible to build a model that can measure the similarity between a given trajectory and a trajectory from a given driver in the taxi system. In the subsequent sections, we propose the use of a deep neural network for meta learning in order to learn that model that can predict whether two trajectories are from the same taxi driver or not. Our neural network structure consists of two recurrent neural networks (RNN) that process the seeking and serving sub-trajectories of both input trajectories, a similarity measure that compares the RNN outputs and five general extracted features of both trajectories to each other, and a linear activation function. We provide the methodology used for building this neural network as well as the training and validation process that we used in order to obtain our model. We then train our model using cross validation to measure the accuracy. We also experiment with adjusting our hyperparameters in order to increase our accuracy

further. Once trained, we test our complete neural network on a held out dataset and achieve an accuracy of 81.33%.

2. METHODOLOGY

In this section, we provide a methodology by which to classify and accurately predict whether two taxi driver trajectories are from the same driver. We also describe the data processing and feature generation necessary for building a robust meta-learning neural network for prediction.

A. Data Processing

Our dataset consists of a set of 5 trajectories each for 500 drivers. These trajectories are represented as a series of records where each record represents the GPS location of a taxi driver at the specific time in their trajectory. Each of these points is labeled with a 0 or 1 depending on whether the taxi cab is seeking or serving a passenger respectively. As such, this dataset contains long sequences of data points with the same serving or seeking label, indicating a subsequence of GPS coordinates along a seeking or serving sub-trajectory for the taxi driver. These sub-trajectories are important as they provide insight as to how a taxi driver behaves when seeking versus serving. Due to this, we chose to process the data into two separate lists: seeking and serving sub-trajectories. Each element in these lists contains a list of GPS coordinates that represent a route that a taxi driver took to serve or seek out a passenger. This processing method allows us to make more sense of our data and take a closer look at the behavior of taxi drivers on route. Another important processing technique that was used included shifting our GPS coordinates into a grid layout. Longitude and latitude coordinates are precise to the degree by which no two data records are likely to share the same coordinates. Due to this, it was important to simplify our trajectory space to a 500 by 500 grid to reduce the precision of this data and provide a more general location for each data point. The grid size was set to 500 as this size generalizes the location of a point without losing a large degree of precision from the GPS coordinates. In order to bound this grid, we determined the maximum and minimum latitudes and longitudes for the entire dataset. Each data point was then assigned a cell with an x and y coordinate based on their latitude and longitude, where $0 \leq x \leq 500$ and $0 \leq y \leq 500$. These preprocessing techniques provided us with an insightful representation of our data. However, data cleaning was needed before preprocessing due to outliers in our GPS coordinates as well as abnormal lengths of subsequences found in our dataset. When calculating the bounds of our grid space, abnormal longitude and latitude coordinates skewed the size of our grid. Due to this, our grid space consisted of many sparse spaces and a small number of extremely dense spaces. Individual cells in dense regions provided useless data as many points resulted in the exact same cell space, indicating that a taxi driver was not moving. Due to this, we chose to remove outliers in the data and limit the grid space to only the top 95% of GPS locations. This provided us with a grid space that better separated the majority of our data into cells that provided useful data. Along with this, our data contained short sequences of seeking and serving sub-trajectories where the length was less than 2. This represents a driver that only seeked another passenger for a few seconds, which does not logically make sense. We chose to label these sequences as outliers and removed them from our dataset. Due to this, we concatenated the seeking or serving sub-trajectories on either side of this outlier to continue the sequence smoothly. Once we had pre-processed and cleaned our data, we chose to mold our data into the shape and size that we needed for our neural network layers. In a subsequent section, we discuss our use of an RNN on our sub-trajectory data. In order to properly use such a neural network, our data needs to contain lists of consistent length. In our case, we needed to input a list of sub-trajectories where each trajectory has the same length and each driver has the same number of sub-trajectories. We reconciled this by setting a constant for the number of sub-trajectories needed for each driver and another constant for the number of points needed in each sub-trajectory. We chose our constants based on their average values across all of the trajectory data we obtained. In terms of preprocessing, we deleted extra data points from the sub-trajectories and deleted subjectories from a driver if the length was over the set limit. If the length of a sub-trajectory was below the constant, we filled the remaining length by duplicating the last data point in the original sub-trajectory. If the number of sub-trajectories for a driver was below the limit, we artificially added sub-trajectories of the limit length where each data point was located at [-1, -1] on the grid. From here, we were able to use our dataset and trajectories to effectively extract important features and build an accurate model.

B. Feature Generation

For feature generation, we focused on extracting features in two different ways: from the individual points and over the general trajectory. We chose to

extract features for the individual points along the trajectory in order to gain a better understanding of how different taxi drivers behave at each time-step along a sub-trajectory. Specifically, we calculated the speed of the driver at each point as well as the hour within which the point was documented. Both of these features are important as they provide some insight as to how a driver moves during a sub-trajectory as different parts of the day. This information also allows us to take into account the type of traffic that a taxi driver may experience at certain times in a day, which can help us to better compare taxi drivers based on what type of route they take (i.e. one with more traffic, one with less traffic but a longer distance). These features are stored in the lists of seeking and serving sub-trajectories along with the cell coordinates of the data point. As for the scope of the entire trajectory, we wanted to extract features that described the general behavior of a taxi driver throughout the day. These features include the coordinates of the most visited cell, the average distance of a seeking sub-trajectory (before pre-processing), the average distance of a serving sub-trajectory (before pre-processing), and the number of passengers served in a trajectory. The most frequently visited cell provides us insight as to where the taxi driver is most likely to drive through or most likely to stay between passengers. The average distances show us the behavior of a driver in terms of whether they favor driving a shorter distance through traffic or avoiding traffic by driving the extra miles and how far they are willing to drive to seek out a passenger. The number of passengers that a taxi driver serves in a trajectory is also important in that it shows us how efficient a taxi driver is compared to another. We use these extracted features in subsequent sections to build and train our deep neural network.

C. Data Generation for Meta Learning

Before introducing the network structure that we used to carry out this meta learning task, it is important to note the additional data generation and manipulation that was needed. In normal classification applications, a dataset with a large number of records for each class is passed into the network structure with a set of labels that provide a classification k for each record, such that $0 \leq k < n$ where n is the number of classes. In meta learning, the goal is to provide a prediction for an unseen task based on a set of training tasks. The datasets used in this case contain a large amount of data overall but not within a specific “class”. Our dataset contains many drivers with only a few trajectories each, which is ideal for meta learning. In our case, we would like to compare two driver trajectories and predict whether or not they are the same driver. As a result, our training will look different than the training of a normal classification application. While normal classification processes each record individually to predict its class, our network must process two records or trajectories simultaneously in order to predict whether they are from the same driver or not. Thus, we created a training set in which each record contains two trajectories from two different or the same driver. As a result of this structure, the label for each record is either a 1 or a 0 if the trajectories are from the same driver or two different drivers respectively. Since our dataset was not initially structured in this way for meta learning, we restructured and relabeled our data after the preprocessing step. In order to maintain a balanced distribution of classes, we started by randomly pairing two trajectories within the same driver to guarantee the presence of this class. These records were labeled with 1. The remaining records were randomly shuffled and paired. These records were labeled with either a 0 or 1 depending on which driver they belonged to. We repeated this entire process again with the original dataset in order to obtain a larger training set. This generated an entirely new set of data since each trajectory was randomly paired. After this data generation step, our dataset was prepared for training with our meta learning neural network structure.

D. Network Structure

As mentioned in subsequent sections, each preprocessed record consists of two trajectories, each consisting of two lists of subjectories (both seeking and serving) with various features included as well as a separate list of features for the general trajectory. Due to the nature of GPS coordinates, both lists of sub-trajectories are set-up in such a way that information can be extracted based on the sequence of points. Thus, we chose to use two recurrent neural networks to obtain and process more information from both sub-trajectories within each passed in trajectory. From here, we concatenated the RNN outputs to each other and the general trajectory feature set for each trajectory. These concatenated outputs served as the embeddings that we used to measure the similarity between the two trajectories. We executed a simple absolute valued difference between the two embeddings, and we fed that difference into a simple linear activation function in order to reduce the dimension of the output to 1. We then fed this output into a sigmoid activation function in order to assign the record a label between 0 and 1. Since the absolute difference taken earlier resulted in an output

closer to 0 for similar trajectories and 1 for differing trajectories, we subtracted the resulting output by 1 in order to produce the accurate label. We chose this structure so that we would be able to extract as much information from the sequences of the trajectories as possible while also using the general features to classify these drivers. We also built a simple neural network that measures the similarity of the drivers based on the difference between their general trajectory features and then feeds the output through one linear activation function to produce a label for each record, but as we will see in subsequent sections, our deep neural network far outperformed this simple neural network.

E. Training and Validation Process

When training our neural network, we used a combination of loss measurements and gradient descent techniques to obtain the optimal model. We use Binary Cross Entropy Loss to determine the performance of the given model due to the binary nature of our labels, and then we perform backwards propagation to update our weights accordingly. From here, we step forward once again using our gradient descent mechanism with the given learning rate. For gradient descent, we chose to use Pytorch's Adam gradient descent function. This version of gradient descent obtains an optimal model by performing the normal RMSProp gradient descent with momentum taken into account. This form of gradient descent performs well when there are multiple local minimum losses, as it allows for more exploration to find the global minimum. We regulate the number of times that gradient descent is performed and the weights of the neural network are updated using a for loop with a number of epochs. In subsequent sections, we discuss how we chose the number of epochs and the learning rate for our gradient descent function. In regards to validation, we performed 5-fold cross validation in order to analyze a wider range of models. We then performed validation on a hold our dataset in order to obtain a final testing accuracy for our chosen model.

3. EVALUATION & RESULTS

This section will cover the performance of our above mentioned neural network structure for meta learning. We will first discuss the training and validation results using 5-fold cross validation of the original data set. Next, we evaluate the performance of our model against a baseline simple neural network. We also provide an explanation for how we chose the correct parameters for our resulting neural network.

A. Training & Validation Results

We start by providing the results of training our model in order to justify our chosen network structure. Our final model was trained using two RNNs with 4 input features and 2 outputs, an absolute difference similarity measurement, and a simple linear activation function with the RNN outputs concatenated to 5 general features prior to the similarity measure. The output was fed into a sigmoid activation function and subtracted from 1 in order to obtain the final results. In regards to training, we trained our model with 7 epochs and a learning rate of 0.01. Initially, we split our data into a training set and a testing set, but we wanted to try out different folds of our dataset. Thus, we performed 5-fold cross validation to provide us with a wider range of models to choose from. Figure 1 below shows the resulting training and testing accuracies after performing 5-fold cross validation.

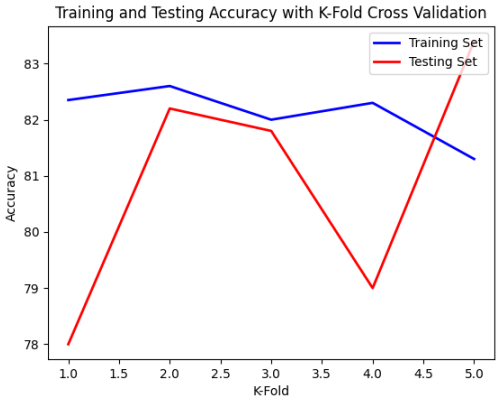

Figure 1: Training and testing accuracies obtained through training our deep neural network

As shown in the figure above, our highest test accuracy was achieved by our fifth validation fold with a testing accuracy of 83.4%. The low training accuracy at this fold, however, suggests that the testing accuracy achieved was by chance. As a result, we are more inclined to choose the model at the second validation fold since the model performs well with a training accuracy of 82.6% and a testing accuracy of 82.2%. These results show that this model provides a strong prediction for the similarity of taxi drivers based on their trajectories. In order to further validate our model and obtain a final testing accuracy, we evaluated the model with a held-out validation set. Validation with this dataset achieved a testing accuracy of 81.33%, which further implies a strong performance of our model.

B. Performance Comparison to Baseline

In order to provide further proof that our model is performing well, we compared our final results to the results produced by a baseline model. In our case, our baseline model is a simple neural network with only the five general features and none of the specific sequence data and features taken into account. As such, our simple neural network measures the absolute difference between the 5 general trajectory features of the two trajectories and then feeds them into a simple linear layer. The outputs are then fed into a sigmoid activation function and subtracted from 1 for the final prediction. We performed 5-fold cross-validation on this baseline model to obtain prediction results similar to those obtained in the previous section. Figure 2 below shows the results from using this baseline model.

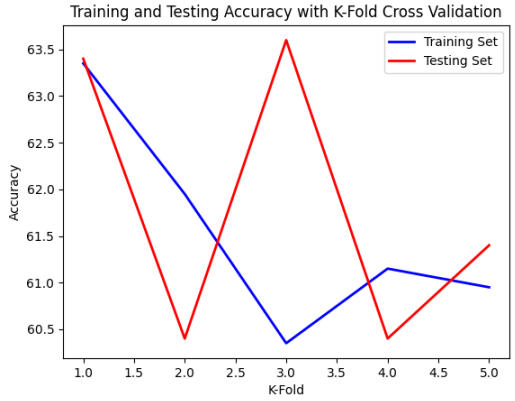

Figure 2: Training and testing accuracies for the simple neural network without the use of RNNs

Based upon the figure above, our baseline model barely performs better than a random guess, as the highest testing accuracy obtained with this model is 63.6%. The poor performance of this neural network provides justification for a deep neural network that takes into account information about the sequences of GPS coordinates along the driver's route. Thus, our model provides a more accurate prediction for the driver based on a trajectory than a simple neural network.

C. Hyperparameters

Once we had the structure for our neural network, we worked towards increasing our accuracy by adjusting the hyperparameters used in our training process. The hyperparameters that we chose to tune include the number of output layers in our activation functions, the ordering of the similarity comparison and the linear layer, the learning rate for gradient descent, and the number of epochs used. For our RNN activation function, we were able to choose the number of outputs. Initially, we started with the same number of outputs as there were inputs. We then experimented by increasing and decreasing this value in order to analyze the effect it had on our accuracy. In doing this, we noticed that our optimal number of outputs was 2, which was lower than the number of inputs provided to the function. Along with this, we experimented with performing the similarity measurement on the concatenated RNN outputs and general feature set before and after being fed through the linear activation function. We found that the model performed better when the similarity measure was performed on each individual output and feature input rather than the reduced dimension space after the linear layer. For the learning rate, we started with a value of 0.05 and experimented with values above and below. As we increased this value, we noticed a sharp decrease in our accuracy, as our gradient descent was unable to train accurately with a larger learning rate. When we decreased our learning rate to 0.01, we found our optimal model. Thus, our final model uses a learning rate of 0.01 for gradient descent. For our epochs, we tested out different values of epochs to determine the optimal value for our model. In theory, as the number of epochs increases, the training error decreases; however, the testing error will decrease and then steadily increase. Due to this, it is important for us to test a wide range of epochs in order to find the optimal value where both our training and testing errors are low and our accuracies are high. In order to do this, we experimented with epoch values ranging from 1 to 20 and recorded the accuracies achieved with each one. Figure 3 below shows the resulting accuracies for each number of epochs.

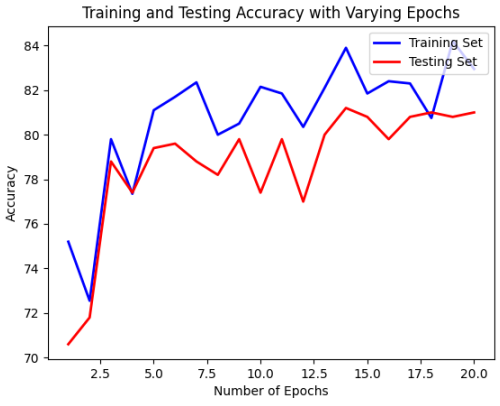

Figure 3: Training and testing accuracies for various epoch values using our deep neural network

The figure above appears to follow the theory mentioned above. The testing accuracy never consistently decreases, however. It appears to level out after 14 epochs, which potentially indicates that we reached our “optimal” epoch. Based on these results in the figure above, we are inclined to choose an epoch value of 14 since epoch values following it do not significantly improve the testing accuracy. As a result of these tests, we chose to adjust our model to perform the similarity comparison before the linear layer, and to have 2 outputs for our RNNs, a learning rate of 0.01 for our gradient descent, and 14 epochs for training.

4. CONCLUSION

In this paper, we proposed a deep neural network for meta learning as a way to predict whether two taxi trajectories were obtained from the same driver. Our structure consisted of two recurrent neural networks with the outputs of each trajectory compared to one another and then fed into one linear neural network and a sigmoid activation function. Training our model using Binary Cross Entropy loss and Adam gradient descent aided in finding the optimal weights for our given neural network. Implementation on our training and testing set also validated the accuracy of our resulting model. We also provide a comparison of our model to a baseline model in order to prove the strong performance of our model. The proposed neural network provides strong predictions for the similarity of two trajectories. In future work, we plan to explore different features that we can generate from our original dataset in order to improve the performance of our neural network.

REFERENCES

1. (n.d.). Uber Revenue and Usage Statistics - BuildFire. Retrieved March 18, 2021, from https://buildfire.com/uber-statistics/
2. (2014, September 8). Uber, Lyft Cars Arrive Much Faster Than Taxis, Study Says - Forbes. Retrieved March 18, 2021, from https://www.forbes.com/sites/ellenhuet/2014/09/08/uber-lyft-cars-arrive-faster-than-taxis/