Pedestrian Motion Classification for Autonomous Vehicles (6.867 Final Project)

Michael Everett & Björn Lütjens

***Abstract*— In this project, we applied techniques from the course to a dataset of pedestrian trajectories recorded on a golf cart around MIT's campus. Our objective was to classify whether a pedestrian's trajectory would enter a 4x10m rectangle in front of the vehicle, which could be used as an active safety feature on various types of vehicles that interact with pedestrians. We trained a SVM with Gaussian RBF Kernel and optimized hyperparameters to achieve 74% test accuracy. We then trained an RNN for the same task and achieved [M. E.] RNN accuracy? accuracy. Finally, we experimented with pedestrian motion prediction, to predict the future pedestrian trajectory.**

I. Introduction

Autonomous vehicles use onboard sensors to perceive their surroundings, and use the data to make decisions about where it is safe to drive. Today's research vehicles typically rely on Lidar and cameras as the main sensors, as these provide information about where/what obstacles are, and can help the vehicle maintain safe position in its lane and with respect to obstacles. In addition to static obstacles, vehicles interact with pedestrians in a variety of environments: people in crosswalks, jaywalkers, kids running across neighborhood streets. A campus shuttle could even drive on sidewalks among pedestrians, so the vehicle would be interacting with pedestrians almost all the time. These regular interactions mean the vehicle must be able to predict pedestrians' next moves in order to maintain safety.

Pedestrian motion prediction is difficult because people often behave very unexpectedly. Humans use many strategies to predict pedestrian intent while driving, such as body language, eye contact, and other non-verbal cues between human driver and human pedestrian, but autonomous vehicles can't easily communicate or read these signals. The information from sensors is typically fused: Lidar is used to measure the pedestrian position, and the camera is used to label the particular obstacle as a pedestrian. Therefore the only measurements collected are position over time (from which velocity can also be computed).

This project uses a dataset collected on MIT's campus over several months by three golf cart shuttles providing Mobility on Demand service to students, while simultaneously collecting pedestrian trajectory data to optimize vehicle routing strategy. The dataset contains about 65,000 pedestrian trajectories as well as the vehicles' trajectories, all in a global frame across the MIT campus.

[M. E.] **related works**

The objective of this project is to develop a classifier that can determine whether a person will step in front of a vehicle, based on a few seconds of their trajectory. We use a portion of our data set to train different classifiers, optimize hyperparameters with a separate portion, and evaluate performance with yet another portion. This classifier could be a useful component of an autonomous vehicle, or part of an active safety feature on a human-driven vehicle that could take over in case the driver does not see a pedestrian in time.

(a) Ground robot in Stata

(b) Golf carts (MIT MoD fleet)

Fig. 1: Many types of autonomous vehicles operate among pedestrians and must account for pedestrian motion in the vehicle motion planning algorithms.

The main contributions of this work are (i) a pedestrian trajectory dataset with 65,000 trajectories over three months, (ii) an SVM classifier for predicting when pedestrians will step in front of a vehicle, (iii) a classifier using Deep RNNs for that same objective, and (iv) a pedestrian motion predictor using Deep RNNs.

II. Dataset

An important realization we made during this project is the challenge of working with a real dataset. Understanding the raw data became a significant portion of the project, so this chapter provides a thorough explanation of our findings and methods to process the data.

A. Raw Data

Our data comes from golf carts equipped with Lidar and cameras [1], [2]. Fortunately, it is relatively straightforward to visualize trajectory data, as opposed to some high-dimensional datasets that exist for other applications.

The raw dataset's fields are shown in Table I, where (easting,northing) are the latitude/longitude global coordinates, and (x,y) are the coordinates in our global campus map. Veh id indicates which of the three vehicles corresponds to that data point, or in the pedestrian case, which vehicle sensed that pedestrian. Ped id is a unique id given to each pedestrian seen.

There are some noise-related issues with the raw data, as it was collected on a research vehicle under development. One issue is that the vehicle's (x,y) position sometimes jumps, because the vehicle's localization system does not use

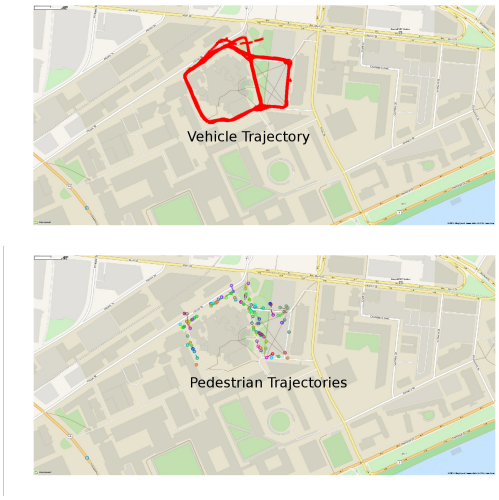


Fig. 2: The raw trajectories from one day of data collection visualized on a campus map. All trajectories are recorded in the global map coordinate frame.

Type	Fields						
Vechicle	time	easting	northing	x	y	veh id	
Pedestrian	time	easting	northing	x	y	veh id	ped id

TABLE I: Raw data fields

GPS and is imperfect. Other minor issues include pedestrian trajectories that incorrectly split/merge or are too short to be useful for this project.

In addition to addressing noise, we also pre-process the data by converting pedestrian trajectories into the vehicle's local frame. Our objective is to classify whether pedestrians cross in front of the vehicle, so the pedestrian trajectories must be converted into the vehicle's local frame. That is, our dataset is initially unlabeled, and we must generate the ground truth label that we wish to learn to classify later.

It might be possible to somehow feed both the vehicle trajectory and pedestrian trajectory into a classifier, and have it learn the transformation, but it seemed much more straightforward for us to rotate the data ahead of time. Using global coordinates could potentially allow the classifier to learn a global understanding of the map (e.g. where sidewalks/intersections are), but we chose to focus on a more structured problem for this project. Since global coordinates should have an impact on pedestrian motion, we could in the future try a hybrid approach where we feed local coordinates along with some context features (e.g. distance to curb, traffic light state) as in [3].

B. Global-to-Local Transformation

The global-to-local transformation relies on knowledge of vehicle orientation (heading angle) and smooth vehicle trajectories, neither of which we have by default. Line 1 describes the procedure for filtering, transforming, and labeling the raw dataset. M. E. **explain fig 3** M. E. **explain how to do global-to-local transform**

Algorithm 1: Algorithm for extracting local trajectories

Input: V_g, P_g: global vehicle, pedestrain trajectories (Table I)

Output: P_l: pedestrian trajectories in local vehicle frame

1: **for** each vehicle **do**

2: $I_{posjump} = \{i \in [1, len(V_g) - 1] \mid euclid\_dist(V_g(i) - V_g(i+1)) > 1.0\}$

3: $I_{timejump} = \{i \in [1, len(V_g) - 1] \mid time(V_g(i) - V_g(i+1)) > 0.5\}$

4: J = $I_{posjump} \cup I_{timejump}$

5: $T_{valid} = \{[J(i), J(i+1)] \mid time(J(i+1) - J(i)) > 5.0\}$

M. E. **finish**

6: **return** pedestrian trajectories

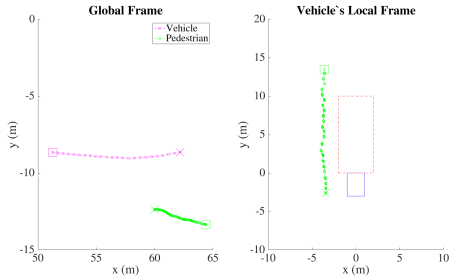


Fig. 3: On the left, the vehicle trajectory (magenta) and pedestrian trajectory (green) in the global frame. Both start from the square and move to the X. On the right is the local vehicle frame: the vehicle is the blue rectangle, and the red dashed rectangle is the "cross" zone. The green pedestrian trajectory doesn't cross into the red rectangle, so this trajectory is given binary label 0.

C. Processed Dataset

The processed dataset is visualized in Fig. 4. The vehicle (yellow taxi) has a blue rectangle representing the "cross" zone (similar to Fig. 3). Green trajectories are local pedestrian trajectories that do not cross into the blue zone, and red trajectories are ones that do enter the blue zone.

A few important observations about the processed dataset are the jaggedness and locations of trajectories. The jaggedness is due to the transformation and sensor rates: the Lidar provides pedestrian trajectory at high rate (50 Hz), but the vehicle position is updated at a slower rate (10 Hz), so the pedestrian trajectory jumps a bit each time vehicle time step. Since the rotation is dependent on vehicle heading angle, and vehicle heading angle is computed from consecutive vehicle positions, this process is subject to some noise. It is also somewhat difficult to imagine what local trajectories *should* look like, because these trajectories are showing how the vehicle and pedestrian move relative to one another, and an observer doesn't know what the vehicle's velocity was for any particular trajectory. For example, a trajectory that arcs along the front of the vehicle could be the vehicle turning as a pedestrian walks straight, or a pedestrian walking in a curve while the vehicle is parked. All of this could have been

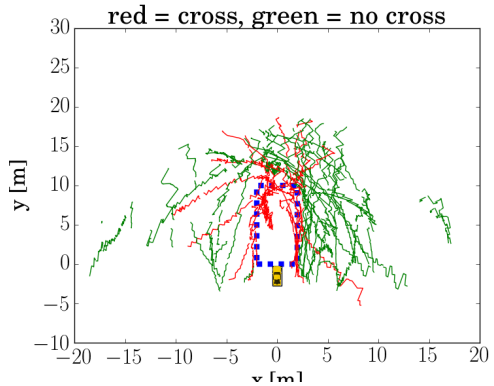


Fig. 4: A small portion of the training set for the binary classifier. The vehicle (yellow taxi) has a blue rectangle representing the "cross" zone. Green trajectories are local pedestrian trajectories that do not cross into the blue zone, and red ones do.

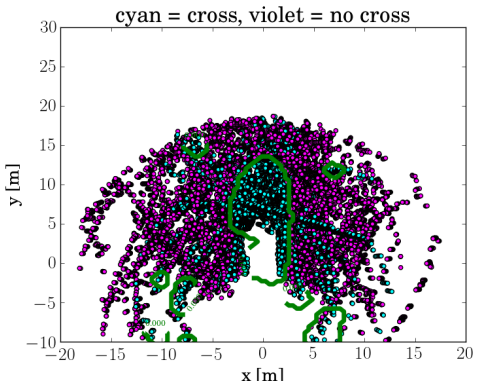


Fig. 5: SVM trained on snippets with $l = 1$ allow visualization of decision boundary. The important part of this plot is the green decision boundary in the center that resembles our arbitrary "cross zone". The SVM has roughly learned the location/shape of zone for simple input data. There is some overfitting evident by other green regions, but this is likely more of an artifact of $l = 1$ hyperparameter.

avoided if we had access to raw sensor measurements, as these provide local coordinates by default.

The second observation is that the trajectories are in front/along the sides of the vehicle, and have a limited range of about 20m. This makes sense given our sensor: it is mounted on the vehicle's front and sees a 270° cone. Its maximum range is more than 20m, but tracking pedestrians reliably is difficult beyond this distance.

III. SVM Binary Classifier

We trained a Support Vector Machine (SVM) to do binary classification on local pedestrian trajectories, to predict whether the trajectory will/will not enter a 4x10m rectangle in front of the vehicle. A portion of the pedestrian trajectory is fed into the classifier, and it outputs a binary label. This could be a useful safety feature on a car to identify which pedestrians might cross in front of the vehicle.

The trajectories are all of different length, so we split them up into equal "snippets" of length l. Each trajectory has a single 0/1 label, and this same label is applied to each snippet. This allows us to learn some notion of direction from (x,y) sequences, because a snippet that never crosses in front of the vehicle can still be labeled as such, if a later portion of its trajectory eventually does cross in front. These l-long lists of (x,y) points are fed into the classifier as the feature vector of length $2l$, as: $(x_i = [x_1, y_1, x_2, y_2, ..., x_l, y_l]; y_i = 0/1)$.

We used the SKLearn implementation of SVM [4] with Gaussian RBF Kernel.

A. Simple Example

We first tried training with $l = 1$, so that we could easily visualize the decision boundary in Fig. 5. The data (magenta/cyan) aren't particularly meaningful, because it's hard to learn anything from a single (x,y) position of a pedestrian ($l = 1$). But, we can observe that the green decision boundary roughly resembles our arbitrarily-created rectangular "cross zone". This figure indicates the SVM is correctly learning that there is a region in space that causes trajectories to be labeled a certain way. After this quick sanity check, we can proceed to larger values of l, but lose the ability to easily visualize the results.

B. Hyperparameters

There are several hyperparameters that affect the behavior of the classifier. The main ones that we experimented with were l (snippet length) and C (SVM regularizer). A grid search for optimal parameter values is shown in Table II (optimal w.r.t. high validation accuracy).

As C increases, training accuracy increases for all scenarios, which is expected because this corresponds to more penalty on mis-classified points. We can tell when C is too high, because training accuracy is much higher than validation accuracy, a symptom of overfitting.

As l increases, more time steps of pedestrian positions are used in the feature vector, so there is more information on which to classify. Intuitively, large l would seem preferable than small, but if l is too large, the noise in the data may outweigh the value of the information. To be more concrete, it seems unlikely that one would need 50 points worth of trajectory data to do the binary classification (especially given the noise seen in Fig. 4).

This intuition aligns with our observed validation accuracies. The highest validation accuracy of 79.8% occurs with $l = 25$ and $C = 10$, so we choose those as the optimal hyperparameters.

The test accuracy with these values was 74.4%.

C. Results

With these optimized hyperparameters, we show the various combinations of true/predicted labels in Fig. 6.

M. E. **add plot with our trajectories labeled as 0/1**

D. Kernel Choice

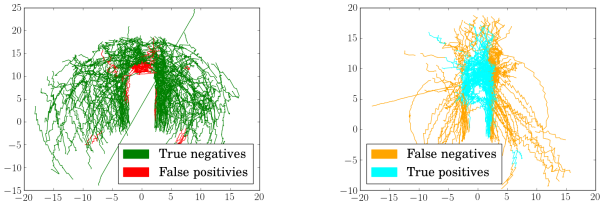
M. E. **discuss kernel options?**

Dataset	$l = 2$							$l = 10$						
	$C = 10^{-3}$	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
Train	70.8	83.2	84.8	85.9	87.1	89.1	91.9	63.8	76.1	84.6	86.8	92.6	97.2	99.6
Val	63.1	77.4	79.1	78.9	78.0	77.8	77.6	60.3	67.9	78.4	79.4	79.3	77.7	76.1
Dataset	$l = 25$							$l = 50$						
	$C = 10^{-3}$	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
Train	62.9	64.9	82.4	88.7	96.8	99.9	100	60.4	60.4	77.6	92.2	99.3	100	100
Val	59.2	59.3	75.0	79.7	79.8	78.8	78.8	52.3	52.3	64.8	75.0	74.8	74.5	74.5

TABLE II: Accuracy of SVM for various values of hyperparameters C, l.

Classification	Number	Percent
False Positives	136	5.0
False Negatives	554	20.4
True Positives	691	25.4
True Negatives	1339	49.2
Total	2720	100.0

TABLE III: Classification matrix



(a) Trajs. labeled "No Cross" (b) Trajs. labeled "Cross"

Fig. 6

Division of Labor & Link to Code

Michael mostly worked on the dataset processing and SVM optimization, and Björn did the RNN work and the rest of the SVM work. This project is not used for any other classes. Our code can be found at: https://github.com/mfe7/6.867.

References

[1] J. Miller, A. Hasfura, S.-Y. Liu, and J. P. How, "Dynamic arrival rate estimation for campus mobility on demand network graphs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[2] J. Miller and J. P. How, "Predictive positioning and quality of service ridesharing for campus mobility on demand systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[3] Anonymous, "CASNSC: A context-based approach for accurate pedestrian motion prediction at intersections," https://openreview.net/pdf?id=rJ26HSLRb, 2017, [Online; accessed 12-Dec-2017].

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.