Question 2

1 Team

Team Members	*Shuyang Ye*	*Ning Shen*
Student ID	*96481163*	*70533633*
CS ID	*l8n8s*	*i0c1p*

Kaggle Team Name	*123321*

2 Solution Summary

We adopted the k-nearest neighbors (KNN) algorithm for prediction of a target catr's future position. A preliminary criterion of similarity between two snapshots of intersection over past the one second was determined as follows:

$$s = (1 - w) * s_1 + w * s_2 \tag{1}$$

$$s_2 = \|\vec{x}_{\text{ego1}} - \vec{x}_{\text{ego2}}\|^2 + \|\vec{y}_{\text{ego1}} - \vec{y}_{\text{ego2}}\|^2 \tag{2}$$

where the weight w is a hyperparameter to be determined, s_1 and s_2 are the similarities from "others" and "agent" accordingly. The subscripts ego1 and ego2 denote to the "agent" from intersection 1 and 2 correspondingly. We will explain how to calculate s_1 in section 3.

Hyperparameters included the number of nearest neighbors k and weight w for balancing the influence of the ego car and the others. We used the validation set to tune k from 2 to 14 and w in $\{0.1, 0.2, ..., 0.9, 1.0\}$. It turned out that $k = 6$ and $w = 1.0$ yielded the lowest validation error. Although such an result seemed surprising at first glance, it was reasonable to only focus on the trajectory of the target car, considering the massive variability introduced by other cars and the fact the ego car's trajectory was affected by other cars already. For details of our experiment please refer to setion 3. In a nutshell, a distance between snapshots of intersection defined with only the trace of the ego car was adequate for predicting the future track.

3 Experiments

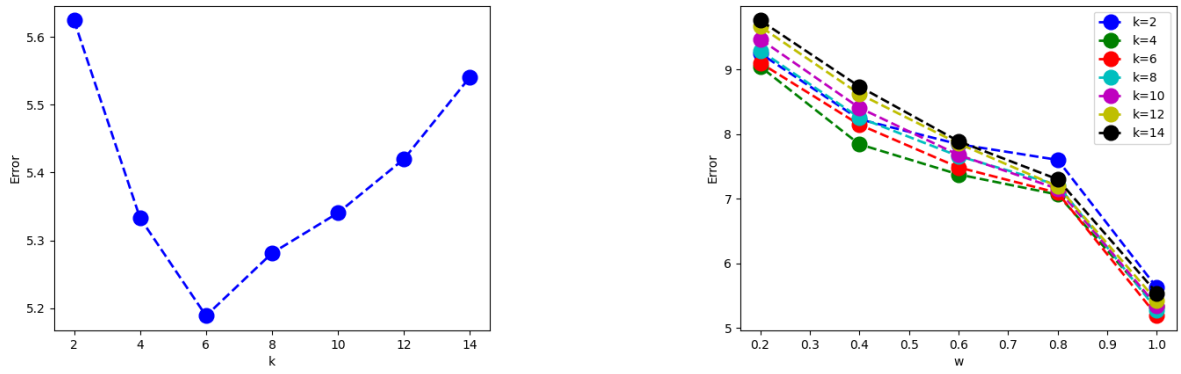
To begin with, we decided on which machine learning method to be applied. An autoregressive (AR) model was the first thing came to our mind naturally. However, considering that the training set only covered the track of the last one second while the task was to predict three seconds forward, an AR model seemed not appropriate for the potential bias from predicting a distant future with little exsting data. On the other hand, KNN as an non-parametric model possessed a great advantage given the multitude of the training samples and the limited number of test examples. The major difficulty of the application of KNN was to find a proper mathematical difinition of "similarity". In the context of this project, "similar" meant that the trajectories of ego cars over the last one second are close enough.

We conducted several experiments to find the proper hyperparameters. As explained in section 2, the similarity between two intersections consisted of two components: the ego car and others. During the comparison, we realized that an ego car must pair with an ego car in two snapshots, others must pairs with others as well. For other cars, we established the distance matrix of size 9×9. Each element d_{ij} is the distance between the track of car_i from intersection 1 and of car_j from intersection 2. For the intersection with less than 9 other cars, the d_{ij}'s that don't correspond with any car were filled with `np.inf`. Once the distance matrix was constructed, we applied a greedy search algorithm to find the similarity. We first sought for the smallest element in the matrix $d_{i_0 j_0}$, meaning that car_0 and car_{j_0} were paired. Then we deleted row i_0 and column j_0 and searched for the smallest $d_{i_1 j_1}$ from the modified matrix again. Repeating above steps

until there is no available d_{ij}. Averaging all the d_{ij}'s found yielded the similarity (s_1) between the other cars from two snapshots. The workflow is showed as below.

- step 1: Calculate the distance matrix $D = [d_{ij}]$ between two snapshots where d_{ij} is the Euclidean distance of two other cars.
- step 2: Find the minimum value of D, $d_{i_{min},j_{min}}$. (By searching for the minimum value of D)
- step 3: Delete the corresponding row i and column j from D.
- step 4: Repeat step2 and step3 x times, $x = \min\{\#\text{car1}, \#\text{car2}\}$. $\#\text{car}i$ represents the number of cars in i^{th} intersection.
- step 5: The average of values found in step2 yields s_1.

For the part of ego car, we directly computed Euclidean distance between two ego cars (s_2). The total similarity was weighted by w: $s = (1-w)s_1 + ws_2$. For each example from validation set, we sorted the total similarity s to find k nearest neighbors, in other word, k most similar intersections from training set. The averaged y from these intersections was the predicted track of the target car, y_{pred}. The validation error was evaluated by the different between y_{pred} and y_{val}. Figure 1 shows how the hyperparameters impacted the validation error. As we could see from Figure 1b, the experiment result revealed the fact that there was no need to include other cars into the similarity metric. Therefore, we have done the feature selection: only the data of ego car were utilized.



(a) (b)

Figure 1: Results of the hyperparameter scan: (a) validation error vs. k when $w = 1$; (b) validation error vs. w with respect to different k's.

4 Result

Team name	Score
123321	0.54513

5 Conclusion

During the exploration of this project, we strongly felt the power of Occam's razor. At the very beginning, we tried to include the impact of the other cars. The rationale behind was straightforward: any vehicle should be affected by the surrounding vehicles. However, with limited information provided, introducing

trajectories of other cars brought in too much noise. The autonomous driving is indeed a complicated task in reality. To solve this problem, we might need a more sophiscated model as well as access to much more information such as the destination, the traffic light, etc. To improve the performence of the prediction in the future, we might dig out more useful information from the original data source, considering not only the coordinates but also the speed of the cars.