Real-time Crash Prediction

Chris Lucas
Stanford University
cflucas@stanford.edu

Sam Premutico
Stanford University
samprem@stanford.edu

Neel Ramachandran
Stanford University
neelr@stanford.edu

1 Introduction

Our project, Real-time Crash Prediction, aims to detect risky driving behavior by predicting whether a driver will crash within a certain time window from the current time step. We aim to answer questions such as, Given a driving sequence of the last 5 seconds, can we predict whether a driver will crash 5 to 10 seconds from now?

We believe this is an interesting yet challenging task with a variety of applications. These applications include aiding insurance companies when they are deciding whether or not to insure someone based on the likelihood of crashing or improving real-time safety software found in cars today that alert drivers if they are driving dangerously.

In our report, we first define the task concretely, and then describe the dataset and important preprocessing steps. We then describe our models, experiments, and results. Finally, we outline future work and discuss the significance of our findings in the broader context of driver-behaviour analysis.

1.1 Problem Definition

Here, we describe our task precisely, and present our hypotheses about our eventual results. Our goal in this project is to build a model that can make continuous, real time collision predictions based on sequences of vehicle sensor data. We can define the task as follows. First, we assume access to d sensor readings from a vehicle, measured at 30 Hertz (ie, 30 times per second): As such, at any present timestep t_0, we receive a vector $x_{t_0} \in \mathbb{R}^d$ with the sensor readings from that timestep. Then, the input to our model at time t takes as input a sequence of sensor data of length 30ℓ, (where ℓ is masured in seconds), from $x_{t_0-30\ell}$ to x_{t_0}. We then seek to predict $y \in \{0, 1\}$, the occurrence of a collision, within the window $(t_0 + t_1, t_0 + t_2)$ (measured in seconds).

As such, we can answer questions such as the one posed in the previous section: Given a driving sequence of the last 5 (ℓ) seconds from present moment, can we predict whether a driver will crash 5 (t_0) to 10 (t_1) seconds from now? Formulating the task in this way allows us to eventually perform real-time crash prediction; since a vehicle is recording streaming sensor data, we can run inference with our model at repeated intervals during the course of a drive, in order to make continuous predictions and detect risky states.

Note that under this formulation, there exist three natural parameters: 1) the length of the sequence ℓ, the start of the collision window t_1, and the end of the collision window t_2. Choices for these parameters form different "settings" under which we can train our model and have interesting real-world implications: the choice of l determines how much streaming sensor data we must keep in memory at any given timestep, while the choice of t_1 and t_2 determine how far into the future we are trying to make a prediction. In this work, we consider all combinations of $\in \{0.5, 1, 3, 5, 10\}$ seconds and $(t_1, t_2) \in \{(1, 6), (5, 10), (15, 20)\}$ seconds, for a total of 15 settings. We note that in this project we keep the window size $(t_2 - t_1)$ constant at 5 seconds in order to compare results between settings, but the work is easily extendable to shorter or longer relevant window sizes.

Given this problem formulation, we make two hypotheses about our model's eventual results under different settings. First, we hypothesize that results should improve as we increase the input sequence ℓ, as the model can access a longer history of the state of the driver and vehicle. Second, we hypothesize that results should decrease as we predict further into the future (by increasing (t_1, t_2)), as the current driving state is less indicative of the eventual state we are predicting.

2 Dataset

The dataset is generated from a high-end driving simulator provided by driving simulation company Nervtech. It consists of fifteen distinct users, who each complete a $<$1 hour course with a variety of difficult circumstances and environments. The data is bucketed into groups of users who experienced at least one accident in the simulation and users who did not. There are 9 users who crashed at least once and 6 users who did not crash at all. For the majority of this work, we consider only the data from the 9 users who experienced a collision.

The simulator reports the readings of 120 sensors collecting data at 30 Hz. The sensors output varied and granular information at each timestamp. Basic values such as coordinates, speed, and acceleration are recorded, as well as more complex features such as the car's distance to the next intersection, the torque of the steering wheel, or the current slope of the road. Beyond the sensors, there are 19 more features that record environmental variables. These include measurements relating to rain and fog, evidence of water on the road, or if the driver was wearing a seat-belt. The resulting dataset is comprised of 139 columns with values being recorded at the aforementioned high frequency over the course of the hour-long drive.

We experienced some issues in leveraging the full set of features present in the dataset due to a lack of official data documentation and difficulty communicating with Nervtech, the provider of the dataset.

However, there is some level of opacity regarding the column names and the data they hold. For example, categorical variables in the data such as *Area ID*, *Scenario ID*, and *Subgaze* contained integer values without documented interpretations. Although they potentially represent meaningful features for our predictions, we did not immediately have methods to disambiguate their significance. After working with out project mentor, we were able to discern the meaning of a some of these features, but decided to ignore others (such as *Subgaze*).

We anticipated that the mix of real-time sensor and environmental data would add useful texture when we are building models to solve the prediction question we outlined above. Thus, we ensured our models would leverage both kinds of readings and incorporate them accordingly.

We discuss the preprocessing of the dataset in section 3.1 and provide in-depth analysis of the data.

3 Experimental Setup and Methodology

3.1 Data Preparation

We dedicated the initial phase of our project to data preprocessing and exploration in order to render the data useful for our task and to gain possible insights into our task prior to building any models.

3.1.1 Cleaning $\&$ Normalization

First, we reduce the number of variables in the data (ie, the number of sensor recordings) from 196 to 76. We remove variables that 1) have low variance, 2) directly indicate an actual collision (such as the position and force of the collision), and 3) are specific to this particular simulation, and would prevent our model from generalizing well to other real-life or simulated driving data. A clear example of 3) is the x-y-z coordinate data marking the position of the vehicle. Removal of these variables is especially important here so that the model is not simply learning the coordinates of difficult or "risky" portions of the simulation, and so that it model can generalize well beyond this data. We do retain x-y-z coordinates in relation to the center of the road, which we calculate based on the given sensor data. We posit that this information is important to collision prediction, as it can

indicate drifting or a distracted driving state. After dropping these variables, we standardize each feature to have mean 0 and standard deviance 1.

3.1.2 Collision Segmentation

Next, we partition the data into collision and non-collision segments. Though the sequences input to our model are of length ℓ seconds, while partitioning the data we consider non-overlapping sequences of length $\ell + t_2$, as this represents the full length of time upon which the inference task is based (to answer the question of predicting a collision 5 to 10 seconds from present-time given the past 5 seconds requires considering a full sequence of 15 seconds). As such, the sequence lengths differ for each of the 15 settings mentioned in the previous section, and so we perform this segmentation step for each setting.

We note that it is a non-trivial task to extract distinct collisions from the original dataset. Every row of sensor readings in the raw data includes features related to the last user collision, including the coordinates and force of the collision. As such, we interpret a change in these values as marking the occurrence of a new collision; however, the values often change in several consecutive columns. We hypothesize that this occurs when a collision is not instantaneous but "dragged out" across time, such as two cars scraping against each other for a prolonged period. Thus, we use simple heuristics to distinguish distinct crashes from consecutive stages of the same crash. We consider distinct crashes as ones that 1) occur 30 seconds apart, and 2) occur 150 meters apart in Euclidean distance. We discuss the possible sensitivity of our model to the collision-segmentation heuristics in Section 4.

In addition, though the ground-truth time of the collision is known in our dataset, we aim to simulate the applicable task of predicting a collision within a window size (it is more reasonable to predict that a crash will occur 5 to 10 seconds from t_0, rather than exactly 5 seconds from t_0. As such, we choose collision segments such that the ground-truth collision is positioned at random in the (t_1, t_2) window.

In our initial experiments, we consider the $\ell = 5$ and $(t_1, t_2) = (5, 10)$ setting. Our resulting dataset comprises of 1489 non-overlapping sequences, with 1310 non-collision sequences and 179 collision sequences. We make two important observations regarding the size of the dataset. First, due to the small number of drives with collisions in the dataset, the size of the dataset is extremely small, and has a strong class imbalance. We therefore choose to use simple model architectures as to not overfit the training data; these architectures are further described in Section 3.2. Second, we note that there are 179 collisions in the dataset; these span across 9 drivers in simulations that last under an hour. This means that on average, each user crashes every three minutes, and approximately 20 total times. This finding is an important reminder that this dataset is generated from *simulated* rather than *real* drives, likely attributing to two important biases towards collisions in the data: 1) segments of the simulation are more challenging than typical real-life driving, and 2) drivers are less-risk averse in simulated driving, given there are no tangible repercussions to crashing. This prompts natural skepticism about the ability of this work to generalize to real-life driving data, which we return to in Section 4.

As mentioned above, we re-segment the data for each of the considered settings. We note that the number of collision sequences will remain constant since the ground-truth number of collisions is independent of our segmenting procedure, while the number of the non-collision sequences decreases as we widen the gap between the input sequence and the crash prediction window. As a result, the dataset size and class imbalance vary from setting to setting, and makes our results across varied settings difficult to compare. We fix this issue by adding supplementary data from the non-collision driving dataset to the necessary settings.

3.1.3 Data Exploration

After cleaning and segmenting our data, we perform exploration and visualization to get a sense of what the data looks like. We first visualize our driving sequences in two dimensions using t-Distributed Stochastic Neighbor Embedding (t-SNE) [2] to reduce the driving sequences to two dimensions and plotted the resulting datapoints, grouped by features of interest.

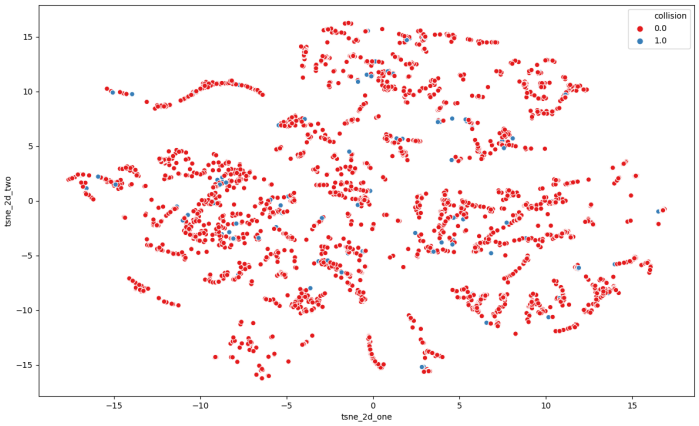

Figure 1: Sequences colored as either collision (blue) or non-collision (red). We see that the sequences fail to produce meaningful clusters with respect to collision and non-collision labels.

In 1, we group embedded collision sequences in blue, and non-collision sequences in red. We find no clear separation between the groups, indicating no obvious differences between the sequences for both labels.

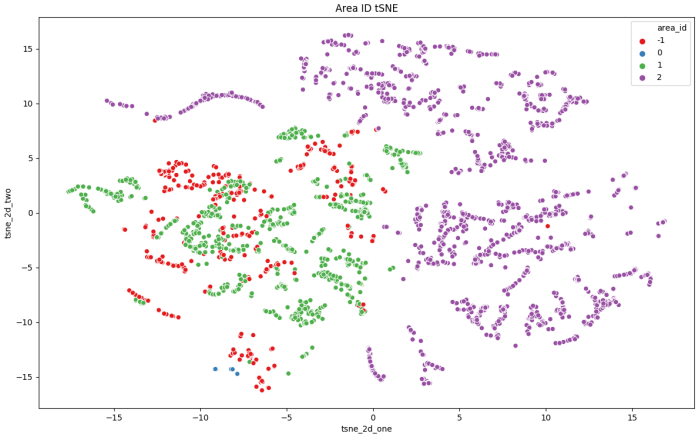

Figure 2: tSNE representation of driving sequences, colored by Area ID, showing clear separation in driving patterns by area.

We see in 2 that when we color driving sequences by the Area ID (denoting different driving environments such as loading zone, rural, urban, and highway) of the sequence, there is clear separation in the driving embeddings, confirming our intuition that driving style varies signifcantly from environment to environment. As sequences cluster meaningfully with respect to Area ID but not collision label, we run t-SNE on sequences from a single Area ID, and observe the collision versus non-collision groupings. However, we find no meaningful patterns in the visualizations, again indicating that in the reduced space, collision sequences are not distinguishable from non-collision sequences.

We also analyzed the individual normalized column values in the data to find features that were significantly different in collision and non-collision sequences.

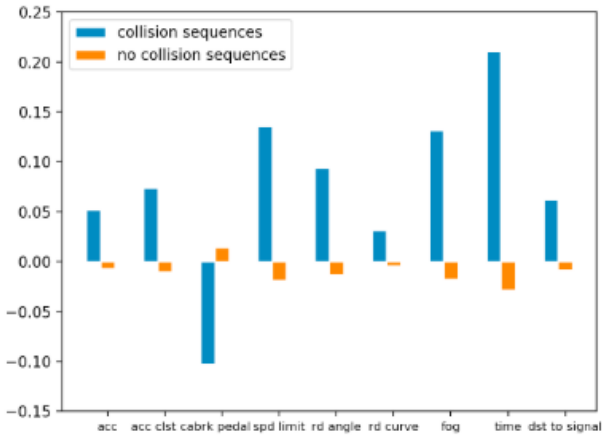

Figure 3: Mean feature values of normalized data, grouped by collision and non-collision sequences.

Interestingly, in 3 we find significant differences in both driver-related and environment-related variables. Collision sequences exhibit higher values for driver-related variables such as acceleration and acceleration of the closest vehicle, as well as environment-related variables such as speed limit, road angle and curve, fog, and distance to the next intersection. It is also interesting to note, that some of these features depend on the complexity of the sensor system and car software we have available. For example, precisely tracking the acceleration of the closest vehicle would likely require a connected car system, and determining the distance to the next intersection might require the availability of a navigation tool. In practice, not all current in-car systems have these features, but we seek to use as many relevant features as possible to make collision prediction more tractable.

3.2 Models

We implement two baseline models, and two deep learning architectures. For the latter, we seek to exploit the temporal nature of the data by implementing recurrent neural network (RNN) and convolutional neural network (CNN) architectures, and show that their improvement over the baseline models indicates successful leveraging of temporal information.

3.2.1 Baselines

For our baselines, we use a Logistic Regression binary classifier to evaluate our sucess with a simple linear model. We used L2-norm regularization and the max number of iterations for convergence was set to 200. We also train a Random Forest classifier with 10 decision trees.

As previously mentioned, we do not leverage the temporal nature of the data for these baseline models. A single input sequence generated from the segmentation procedure is a matrix of size (T, D), where T is given by 30ℓ and $D = 76$; here, we reshape the input sequence into a single T vector.

3.2.2 RNN

Our first deep learning architecture makes use of RNNs, a popular model choice for sequential or temporal data. An RNN processes input by ingesting data from each timestep sequentially, and maintaining a hidden state that it encodes information about the input over time. Here, we use we use a Long Short-Term Memory (LSTM) model [3], an RNN variant designed to better learn long-term temporal dependencies than the vanilla RNN implementation. Our initial LSTM architecture consisted of a 60-dimensional hidden vector h_t. We further experiment with the hidden size of the network, described below. We feed the final LSTM hidden state to a fully-connected linear layer that outputs scores for both classes.

3.2.3 CNN

We also implemented a 1D-CNN architecture. Though CNNs are historically more common in the two-dimensional input setting for tasks in computer vision, recent work [4] suggests that the one-dimensional variant is equally, if not more effective than RNN-based architectures in the temporal setting, due to faster paralleled learning, and increased gradient flow that allows the model to learn long-term dependencies more effectively than LSTMs.

We intentionally keep our neural models shallow to avoid overfitting due to the small size of our dataset. Our CNN model has two Convolutional $\rightarrow$ ReLU $\rightarrow$ MaxPool layers, follwed by a fully connected prediction layer. We used 3x3 kernels for the convolutional layers and 2x2 kernels for the max-pool layers. We initially use 80 and 40 filters for the first and second convolutional layers, respectively. We further experiment with hyper-parameter tuning across both models in the next section.

3.3 Experiments & Methodology

For both deep learning models, we use Binary Cross-Entropy loss, and an Adam optimizer. Due to the significant class-imbalance of our data, we weight the loss function to penalize missing collisions predictions more harshly than missing non-collisions predictions.

As previously mentioned, we generate datasets for each of the 15 settings we consider, ie each possible combination of $\ell = \{0.5, 1, 3, 5, 10\}$ and $(t_1, t_2) = \{(1, 6), (5, 10), (15, 20)\}$. For each setting, we train several models, performing random hyperparameter search across hidden state size (LSTM), number of filters (CNN), learning rate, and loss weights. We use an 80/10/10 training/validation/eval split across sequence datasets and models. To evaluate our models, we avoid using accuracy due to the severe class imbalance of our dataset (a naive prediction of non-collision for every sequence would result in an accuracy of nearly 90%). Instead, we focus on precision, recall, and F1-score metrics associated with the collision class. We consider recall and F1-score of particular importance here, because in this setting we can afford to be over-agressive with predicting collisions, so that 1) we have the best chance of detecting every collision and 2) even if a collision does not actually occur, we may be accurately predicting some notion of a risky driving state.

3.4 Results

Our results are summarized in the tables below. We note that we only run our baselines models on the $\ell = 5$, $(t_1, t_2) = (5, 10)$ setting. We can see that the LSTM and CNN models both outperform the baseline models by over 5%, suggesting that leveraging the temporal nature of the data plays an important role in making good predictions.

From our baseline models, we extracted the importance of each of the features in making crash predictions, shown in in 4. We find distance from the center of the road, vehicle roll, acceleration, and distance to the next intersection among the most important features. Matching our intuition, these results reaffirmed the idea that crash prediction is predicated on sensor data from both the driver's behavior as well as other environmental variables.

Table 1: Preliminary Results

Model	Precision	Recall	F1-Score
Logistic Regression	0.15	0.13	0.14
Random Forest	0.25	0.05	0.09

The observed results align with our initial hypotheses regarding the effects of changing the setting by tweaking the ℓ and (t_1, t_2) parameters. We find that as we increase the sequence length (moving down the columns in the table), F1-scores improve. This makes sense given that we are essentially using more data to inform our predictions about the future. Meanwhile, as we push the five-second crash prediction window further into the future, we see a decrease in performance. This matched our initial intuition that it is more difficult to make predictions about the distant future than it is to

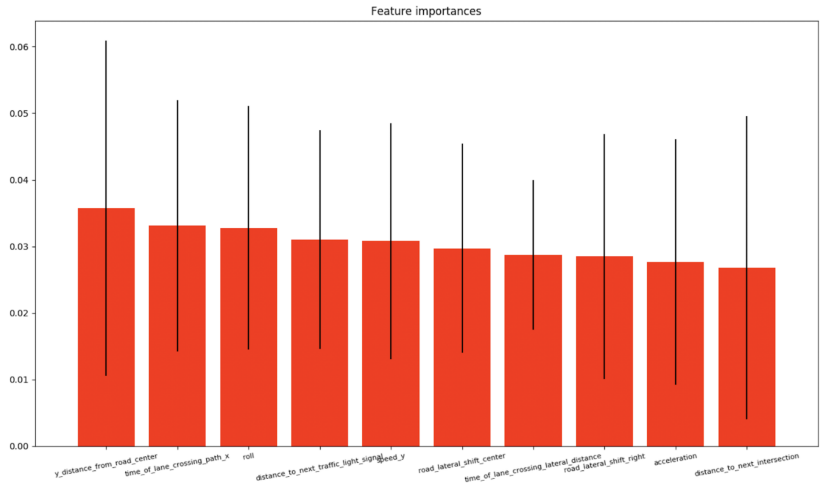

Figure 4: Feature importances in the baseline model.

Table 2: CNN F1 Scores for Sequence Length and Window Size Permutations

	(1,6)	**(5, 10)**	**(15, 20)**
0.5	0.37	0.05	0.04
1	0.13	0.09	0.05
3	0.17	0.14	0.09
5	0.24	0.19	0.11
10	0.40	0.33	0.20

Table 3: LSTM F1 Scores for Sequence Length and Window Size Permutations

	(1,6)	**(5, 10)**	**(15, 20)**
0.5	0.35	0.06	0.04
1	0.13	0.08	0.05
3	0.16	0.12	0.08
5	0.23	0.16	0.10
10	0.40	0.30	0.19

make predictions about the near future. One notable discrepancy in these results is the fact that both models perform relatively well in the $\ell = 0.5$, $(t_1, t_2) = (1, 6)$ setting; this is most likely due to the fact that the input sequence is short and very close to the relevant crash prediction window. As such, certain sensor values are likely already becoming extreme and indicative of a crash at this point, while this valuable information may be diluted in the settings with longer input sequences.

We also find comparable results between the CNN and LSTM models, though the CNN models perform slightly better (and took a fraction of the time to train), aligning with results of work done in [4]. We find that overall, the results achieved by the models are relatively poor in terms of absolute measure; we discuss further in the next section.

4 Future Work and Conclusion

In this paper, we train models to perform real-time crash prediction in order to answer questions like, Given a sequence of sensor readings from the past 5 seconds, can we predict whether a crash will occur 5 to 10 seconds from now? Answering this question has a number of important applications, ranging from general driving safety to car insurance decisions. We train two baseline models and two neural models, and find that the neural models, though simple, outperform the baseline models by leveraging the temporal nature of the data. We encounter two main limitations in this project:

first, the small nature of the dataset makes learning an effective, complex model difficult, and we are constrained to using simple models here as to prevent overfitting. Secondly, we find that these simulated drives are in fact irrepresentative of real-life driving, as the number of collisions each user makes is extremely high, likely due to the challenging nature of the driving simulation, and the decreased risk profile of the drivers. This makes our results difficult to generalize to the real-world driving data that would look quite different in terms of collision occurrences.

Ultimately, we find that while the results of our models satisfy our hypotheses, they perform somewhat poorly, as measured by F1-score. In this setting, however, it is important to consider the difficulty in predicting collisions. Collisions are rare (even with the increased ocurrences in this dataset) and often occur out of nowhere, and cannot be detected based on data from previous timesteps. Furthermore, our sensor data fails to encode lots of visual information that would be important to the collision prediction task. Thus, it is difficult to expect such a model to score highly on this task in terms of conventional metrics; false positive predictions may still be a valuable example of risky state detection, for example. Nevertheless, our work shows promising initial results for performing collision prediction.

One point of further exploration would be training our models perform in different kinds of environments. From the data, we know whether a driver is driving in rural or urban environments. It would be interesting to see how our models perform with each of the fifteen aforementioned settings with regards to window position and sequence length. For example, our intuition tells us that in fast-paced urban environments, we could make myopic predictions well. On the other hand, in more rural areas, we could potentially be better at predicting crashes further out in the future.

Finally, we would like to acknowledge and thank Vincent Duong from viaduct.ai for mentoring us over the course of the quarter.

References

[1] Hallac, David, et al.'Drive2Vec: Multiscale State-Space Embedding of Vehicular Sensor Data.

[2] Maaten, Laurens van der, and Geoffrey Hinton. 'Visualizing data using t-SNE.

[3] Hochreiter et. al. 'Long Short-Term Memory'

[4] Bai et. al. 'An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling