Modeling Users' Powertrain Preferences

by

Jongu Shin

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July 2010
[September 2010]

Author .....
.....
Department of
Electrical Engineering and Computer Science
July 30, 2010

Certified by .....
.....
Leslie Pack Kaelbling
Professor
Thesis Supervisor

Accepted by .....
.....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Modeling Users' Powertrain Preferences

by

Jongu Shin

Submitted to the Department of
Electrical Engineering and Computer Science
on July 30, 2010, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Our goal is to construct a system that can determine a drivers preferences and goals and perform appropriate actions to aid the driver achieving his goals and improve the quality of his road behavior. Because the recommendation problem could be achieved effectively once we know the driver's intention, in this thesis, we are going to solve the problem to determine the driver's preferences.

A supervised learning approach has already been applied to this problem. However, because the approach locally classify a small interval at a time and is memoryless, the supervised learning does not perform well on our goal. Instead, we need to introduce new approach which has following characteristics. First, it should consider the entire stream of measurements. Second, it should be tolerant to the environment. Third, it should be able to distinguish various intentions.

In this thesis, two different approaches, Bayesian hypothesis testing and inverse reinforcement learning, will be used to classify and estimate the user's preferences.

Bayesian hypothesis testing classifies the driver as one of several driving types. Assuming that the probability distributions of the features (i.e. average, standard deviation) for a short period of measurement are different among the driving types, Bayesian hypothesis testing classifies the driver as one of driving types by maintaining a belief distribution for each driving type and updating it online as more measurements are available.

On the other hand, inverse reinforcement learning estimates the users' preferences as a linear combination of driving types. The inverse reinforcement learning approach assumes that the driver maximizes a reward function while driving, and his reward function is a linear combination of raw / expert features. Based on the observed trajectories of representative drivers, apprenticeship learning first calculates the reward function of each driving type with raw features, and these reward functions serve as expert features. After, with observed trajectories of a new driver, the same algorithm calculates the reward function of him, not with raw features, but with expert features, and estimates the preferences of any driver in a space of driving types.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Acknowledgments

I owe my deepest gratitude to Professor Leslie Pack Kaelbling, my academic, UROP, and thesis advisor, for her unending support, guidance, and inspiration, which have enabled me to develop a deep understanding in the field of computer science and machine learning.

I would also like to thank Professor Tomas Lozano-Perez, for his guidance and insightful comments that enabled me to grow intellectually and finish the thesis.

Finally, this thesis would not have been possible without the endorsement of Ford Motor Company and the help of Dr. Dimitar P. Filev and Dr. Kwaku O. Prakah-Asante. I offer my regards to all of those who supported me in any respect during the completion of this project.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

1 Introduction **13**

1.1 Motivation and Objective 13

1.2 Approaches 14

1.3 Dataset 14

2 Supervised Classification of Single Intervals **17**

2.1 Features 17

2.2 Classifiers 18

2.3 Performance 19

2.3.1 Single-User Case 19

2.3.2 Two-User Case 21

2.4 Discussion 22

3 Driver Type Classification **25**

3.1 Preliminaries 25

3.1.1 Bayesian hypothesis testing 25

3.1.2 Iterative Bayesian hypothesis testing 26

3.2 Pilot Experiment 26

3.3 Driver Type Classification 27

3.3.1 Data models 27

3.3.2 Transition model 29

3.3.3 Filtering 31

3.3.4 Results 31

3.4 Classification with Online Update of Models 37

3.4.1 Learning Parameters Slowly 38

3.4.2 Online Clustering 38

3.4.3 Expectation Maximization 39

3.4.4 Simplified Baum Welch 40

3.4.5 Results . . . 41
3.5 Discussion . . . 46

4 Utility Function Estimation **47**
4.1 Preliminaries . . . 48
4.1.1 Markov decision process . . . 48
4.1.2 Linear Approximation of Reward function . . . 48
4.1.3 Inverse Reinforcement Learning and Apprenticeship Learning . 49
4.2 Simulation and Dynamics . . . 50
4.2.1 Car Model : State and Action . . . 50
4.2.2 Simulation of Driving Environment . . . 52
4.2.3 Dynamics : Simulation of Nearby Environment . . . 52
4.3 Features . . . 57
4.3.1 Raw Features / basis functions . . . 57
4.3.2 Expert Features / composite functions . . . 58
4.4 Policies . . . 58
4.4.1 One-Step-Look-Ahead Policy . . . 59
4.4.2 Least-Squares Policy Iteration . . . 59
4.5 Learning Algorithm . . . 60
4.5.1 Apprenticeship Learning via Inverse Reinforcement Learning . 60
4.5.2 Q learning . . . 61
4.5.3 LSTDQ and LSPI . . . 61
4.5.4 Estimation with Expert Features . . . 62
4.6 Simulation Experiment . . . 63
4.6.1 Setup . . . 63
4.6.2 Results of Simple Policy . . . 64
4.6.3 Results of Least-Squares Policy Iteration . . . 66
4.7 Realistic Experiment . . . 68
4.7.1 Setup . . . 68
4.7.2 Results of Simple Policy . . . 70
4.7.3 Results of Least-Squares Policy Iteration . . . 72
4.8 Discussion . . . 73

5 Conclusion and Future Work **75**
5.1 Conclusion . . . 75
5.2 Future Work . . . 76

List of Figures

1-1 Example of Type-2 Dataset 15

2-1 Pattern map of gas pedal rate and gas pedal position 17
2-2 Pattern map of windows 18

3-1 Pilot Experiment of Bayesian hypothesis testing 27
3-2 Probability distributions of three measurements for different driving types 28
3-3 Result of Bayesian hypothesis testing 33
3-4 Result of Bayesian hypothesis testing 35
3-5 Result of Bayesian hypothesis testing when swapped 36
3-6 Beliefs on newly observed dataset 37
3-7 Change of probability distributions's parameters during EM iterations 43
3-8 Log-likelihoods and error rates over iterations 45

4-1 The interaction between an agent and its environment 48
4-2 State 51
4-3 Simulator 53
4-4 Factored Dynamics Model 55
4-5 Statistics of the second lane transition model 56
4-6 Flow of algorithm to estimate a driver's preference with expert features 63
4-7 Distance to feature expectation as a function iteration 64
4-8 Result of Simple Policy with Raw Features 65
4-9 Result of Simple Policy with Expert Features 66
4-10 Result of LSPI with Raw Features 67
4-11 Result of LSPI with Expert Features 67
4-12 The measured states (green lines) and simulated states (blue lines) . . 69
4-13 Distance to feature expectation over iteration 70
4-14 Result of Simple Policy with Raw Features 71

4-15 Result of LSPI with Raw Features 72

List of Tables

1.1 Available measurements for each type of datasets 16

2.1 Result of SVM of Driver Type 19
2.2 Result of Decision Tree of Driver Type 20
2.3 Result of SVM for Power Mode 20
2.4 Result of Decision Tree for Power Mode 20
2.5 Result of Decision Tree of Two-User Case 21

3.1 Possible Models for Measurements 29
3.2 Models and Calculated Parameters of Measurements 30
3.3 Result of Bayesian hypothesis testing 32

4.1 The coefficients of the dynamics 69
4.2 Result of Simple Policy with Expert Features 71
4.3 Result of LSPI with Expert Features 73

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Ultimately, we want to build a system that acts as a "helper" agent to a driver. Such a helper must solve two problems:

- Determine what the driver's preferences and goals are; and
- Take actions to help the user achieve his or her goals.

Although this problem can be treated monolithically, it may be very computationally expensive to do so. Thus, we would like to begin by addressing the first part of the problem in this thesis, as the second part of the problem can only be addressed once the first part has a good solution.

1.1 Motivation and Objective

Our goal is to construct a system that can determine a driver's preferences and goals and perform appropriate actions to aid the driver achieving his goals and improve the quality of his road behavior. To determine a user's intention, we need to build an inference engine that can take a stream of measurements from the environment and a user's actions as input, as well as models for a driver and environment, and return the classification of the user's preference.

One interesting approach has already been applied to this problem. A supervised learning approach takes a small interval (5, 10, or 30 seconds) from a stream of measurements and learns to classify each interval according to the type of driver [7]. However, as this method only focuses on a small portion of the data each time and is thus memoryless, it cannot take any longer term behavior of a driver into account.

Therefore, our objective is to introduce a new approach which has the following characteristics. First, it should consider the entire stream of measurements. While

looking back to the past input sequence using hidden states or other types of memory, the algorithm should focus on the driver's recent activities. Second, it should be sensitive to the environment. Because the criterion depends on the environment, such as traffic and road type, the algorithm should be adaptive to distinguish between different intentions appropriately. Third, it should be able to distinguish various intentions. Other than binary classification, the algorithm should classify multiple intentions or estimate in a space of driving types.

1.2 Approaches

In this thesis, two different approaches, Bayesian hypothesis testing and inverse reinforcement learning, will be used to classify and estimate the user's preferences.

Bayesian hypothesis testing classifies the driver as one of several driving types. The algorithm can work as binary classification between aggressive and cautious , or multi class classification among speedy , economical and safe . Assuming that the probability distributions of the features (i.e. average, standard deviation) for a short period of measurement are different among the driving types, and the measurements for each interval are conditionally independently identically distributed of the type, the algorithm maintains a belief distribution for each driving type and updates it online as more measurements are available.

On the other hand, inverse reinforcement learning estimates the users' preferences as a linear combination of driving types. Unlike the Bayesian approach, in which the algorithm maintains the belief states regarding the driver's type, the inverse reinforcement learning approach assumes that the driver maximizes a reward function while driving, and his reward function is a linear combination of driving types' reward functions. To determine the three different driving types, apprenticeship learning uses sample trajectories obtained from the representative driver for each driving type. Likewise, apprenticeship learning is also used to calculate the reward function of any one driver based on that driver's measured trajectory. By comparing that driver's reward function to the reward functions of the three driving types, our system can estimate the preferences of any driver in a space of driving types.

1.3 Dataset

To experiment with our new approaches, we begin our work with synthesized datasets to confirm that those approaches are valid. Since the simulated datasets vary depend-

ing on the method, they will be explained in relevant chapters. Once the approach proves its usefulness, we experiment with realistic datasets acquired from Ford: Type-1 has seven different measurements, whereas Type-2 has five measurements. **Table 1.1** explains available measurements for each type of dataset, and **Figure 1-1** shows one trajectory of a Type-2 dataset as an example.

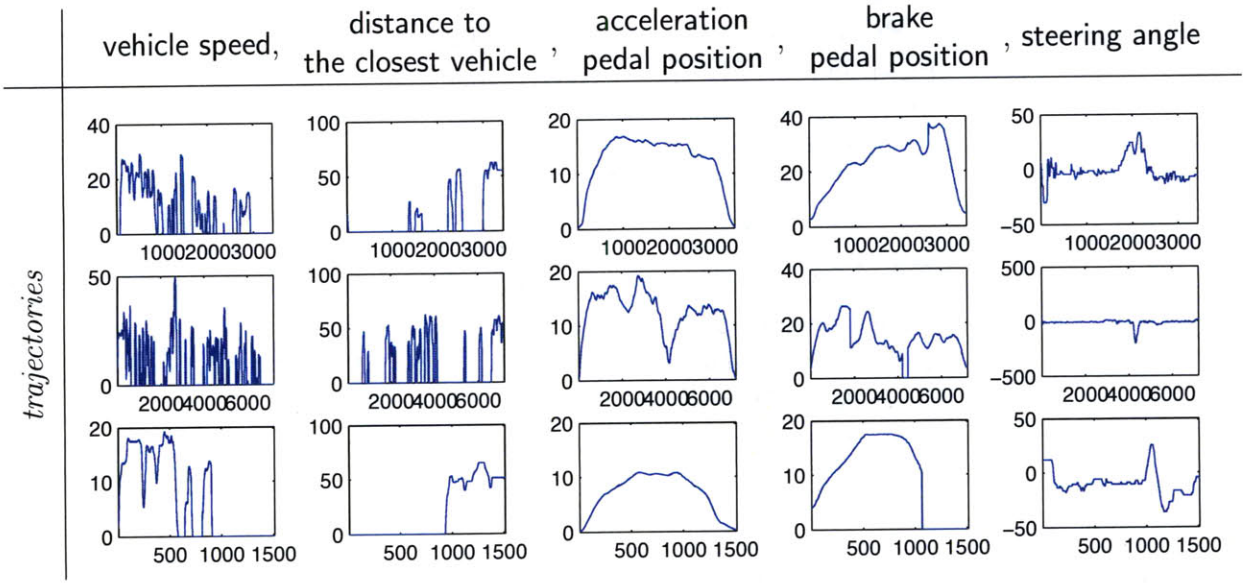


Figure 1-1: Three trajectories from one of the Type-2 datasets. The x axis is time, (50=1sec); the units for the y axis differ for each measurement.

The Type-1 datasets have various measurements, especially regarding the current status of the car and driving options chosen by a driver. The Type-2 datasets are more interesting, because they include distance to the vehicle in front, which we conjecture is particularly useful in diagnosing a driver's aggressiveness.

Each dataset consists of a series of datapoints, and the sampling rates are different for two datasets. Type-1 has 1000Hz sampling rate, whereas Type-2 has 50Hz sampling rate. Since the data are sampled at very high frequency, single data points are highly correlated with one another, and may be noisy. To get a dataset of more manageable size, with meaningful samples, we often aggregate datapoints in a small interval, called a window, with length of 5, 10, and 30 seconds, and calculate features of the interval, such as the sample mean and variance.

Collecting Datasets

In addition, several different schemes for collecting dataset were chosen to suit the needs to train, test, or update algorithms.

Table 1.1: Available measurements for each type of datasets

(a) Type-1

Measurement	*Value*	*Unit*
Accelerator Pedal Position	real 0-100	%
EDAS Performance Mode	1/2/3	boolean
Engine Speed	real 0-5000	rpm
PCM Brake On	Switch On/Off	boolean
PCM SpdCon EDAS	Active/Inactive	boolean
Pedal Rel RPS Ena	Rqst	boolean
Vehicle Speed	real 0-100	kph

(b) Type-2

Measurement	*Value*	*Unit*
vehicle speed	m/s	real 0 - 50
distance to the closest vehicle	m	real 0 - 100
acceleration pedal position	%	real 0 - 100
brake pedal position	%	real 0- 100
steering angle	deg	real

- Type-1 A : Each dataset was collected from one person who was driving with one type. Each dataset was labeled with the type and the driver. It was mainly to train the parameters. 6 datasets were collected.
- Type-1 B : Each dataset was collected from one person who changed his behavior in the middle. Each dataset was labeled as the two driving types with the time when the behavior changed and the driver. It was used to test the parameters. 6 datasets were collected.
- Type-2 A : Similar scheme to Type-1 A , but with Type-2 measurements. 8 datasets were collected.
- Type-2 B : Similar scheme to Type-1 B , but with Type-2 measurements. 6 datasets were collected.
- Type-2 C : Each dataset was collected from one person who changed his driving type several times. It was labeled with the degree of aggressiveness over time and the driver. It was used to test the EM algorithm, which updates parameters without knowing labels, and to verify updated parameters. 10 datasets were collected.

When we handle the real dataset, we choose the the dataset type and the scheme to best suit the purpose of methods, and they will be specified throughout the thesis.

Chapter 2

Supervised Classification of Single Intervals

A previous attempt to classify drivers' performance modes used a supervised learning approach [7].

2.1 Features

As the probabilistic pattern map of the gas pedal rate and gas pedal position are quite different between aggressive and cautious drivers (**Figure 2-1**), one would expect that the measurements can be successfully classified. However, because these values tend to be sampled at a high rate (1000Hz for the dataset 1 and 50Hz for the dataset 2), and because each datapoint is not independently identically distributed, it is meaningless to classify each data point independently.

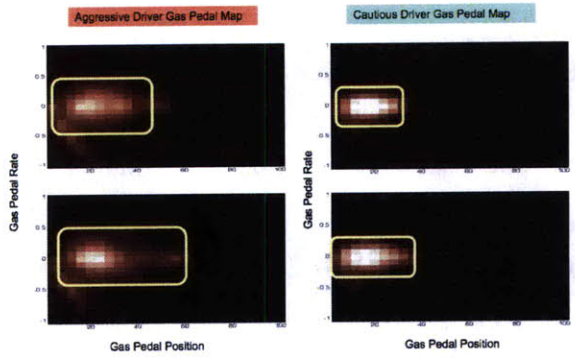
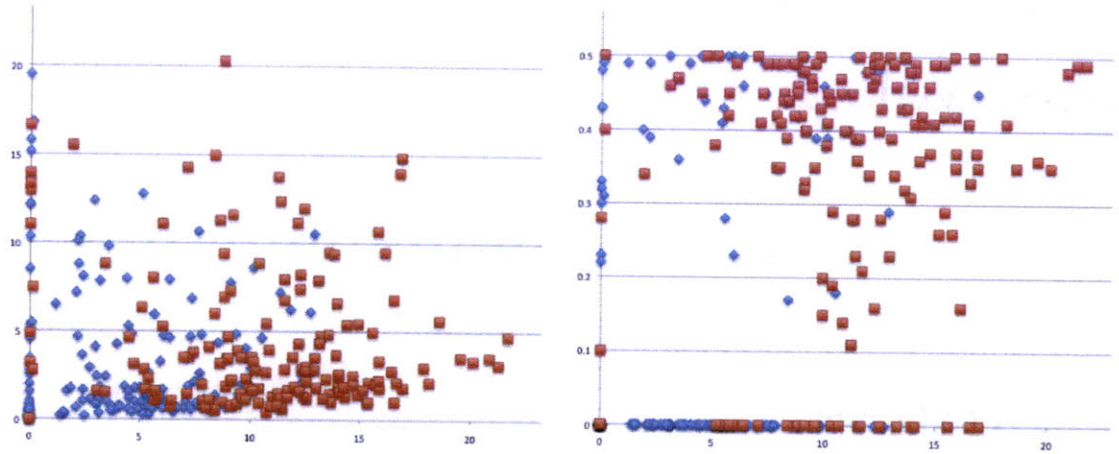


Figure 2-1: Pattern map of gas pedal rate and gas pedal position
Source: [12]

Instead, datapoints within a small interval, called a window, with length 5, 10,

and 30 seconds from a series of raw measurements, were aggregated and described by features (average and standard deviation) of each measurement, such as accelerator pedal position and vehicle speed. After filtering out meaningless windows that have a speed near zero, they were labeled as belonging to the cautious or aggressive driving types, and as belonging to a specific performance mode. **Figure 2-2** shows the relationship among calculated features; each dot represents one window, and its color represents the label. You may notice that the dots are reasonable well separated using those features.



(a) Accelerator standard deviation vs. speed standard deviation

(b) Accelerator standard deviation vs. brake standard deviation

Figure 2-2: Pattern map of windows. Each dot represents one window, and its color represents the label, where red: aggressive and blue: cautious .
Source: [7]

2.2 Classifiers

Two types of classifiers were employed in this experiment:

- SVM with linear kernel: the SVM^{light} implementation
  (available from http://svmlight.joachims.org/).
  More complex kernels did not provide improved performance.

- C4.5 (Release 8) Decision Tree builder
  (available from http://www.rulequest.com/Personal/).

2.3 Performance

This approach was tested on the Type-1 datasets [1], using two approaches. In the single-user case, training and testing data were selected from the same driver. In the two-user case, data from different drivers was mixed to generate the training and testing sets.

2.3.1 Single-User Case

The results for classifying between aggressive and cautious driving type for the single-user (koppa) case are shown in **Table 2.1**. The linear SVM and the decision tree algorithm had comparable performance. Note that the 5 and 10 sec windows perform much better than the 30 sec window.

Method	*Accuracy*		
	5	10	30 sec window
SVM Linear Kernel (single train/test split)	80%	84%	70%
C4.5 (10-way cross-validation)	83%	82%	77%

Table 2.1: Each number represents one window.
Source: [7]

Support vector machines performs moderately well. Most of time, the algorithm achieves approximately 20 to 25% of error rate in binary classification between aggressive and cautious driving type, and between power mode 3 and power modes 1 or 2.

On the same data, the decision tree algorithm achieves a slightly lower error rate of 15%. The resulting decision rule is shown in **Table ??**. However, the decision tree heavily on the standard deviation of acceleration pedal position; all windows whose the standard deviation of acceleration pedal position below 7.79 are classified to a cautious driving type. This criterion classifies 181 windows correctly, but fails at 38 windows, which exceeds the average error rate. From this, we conclude that acceleration pedal position alone does not have enough information to classify all windows correctly.

[1] When Lozano-Perez experimented, the Type-2 datasets were not available.

```
Acc_Sd <= 7.79 : -1 (181.0/38.3)
Acc_Sd > 7.79 :
| Brk_Sd > 0.18 : 1 (117.0/12.8)
| Brk_Sd <= 0.18 :
| | Rpm_Av <= 1633.74 : -1 (12.0/5.7)
| | Rpm_Av > 1633.74 : 1 (23.0/6.0)
```

Table 2.2: Each number represents one window. +1: aggressive , -1: cautious .
Source: [7]

Classifiers were also constructed to distinguish between window during which power mode 3 is in use, on the one hand, and those where either power modes 1 or 2 is in use, on the other. The performance results are similar.

Method	*Accuracy*		
	5	10	30 sec window
SVM Linear Kernel (single train/test split)	79%	81%	71%
C4.5 (10-way cross-validation)	85%	82%	84%

Table 2.3: Each number represents one window.
Source: [7]

The decision tree for this problem (**Table ??**), shows a similar pattern as well, note that the top-level branch is nearly the same as in the previous tree.

```
Acc_Sd <= 7.9 : -1 (185.0/17.0)
Acc_Sd > 7.9 :
|   Acc_Sd > 14.4 : 1 (31.0/3.0)
|   Acc_Sd <= 14.4 :
|   |   Acc_Av <= 7.48 : 1 (25.0/3.0)
|   |   Acc_Av > 7.48 :
|   |   |   Rpm_Sd > 454.57 : -1 (18.0/1.0)
|   |   |   Rpm_Sd <= 454.57 :
|   |   |   |   Brk_Av <= 0.14 : -1 (43.0/13.0)
|   |   |   |   Brk_Av > 0.14 : 1 (31.0/9.0)
```

Table 2.4: Each number represents one window. +1: Power Mode 3, -1: Power Mode 1 or 2.
Source: [7]

2.3.2 Two-User Case

In the two-user case, where the data from two users (kopa and finn) are merged, the C4.5 classifier performs comparably (e.g. 81% for the 10 sec. windows in caution/aggressive classification). The linear SVM does substantially worse (e.g. 75% for the 10 sec. windows in caution/aggressive classification), however. This suggests that the data from these drivers is substantially different. The decision-tree classifier can readily deal with data that has multiple clusters (e.g. **Figure 2.4**); the linear SVM is less able to deal with that. A more powerful SVM kernel, such as radial basis functions, should be able to deal with the more complex data, but that was not attempted in this preliminary study.

```
Acc_Sd <= 7.79 :
|   Rpm_Av <= 1571.21 : -1 (189.0/44.5)
|   Rpm_Av > 1571.21 :
|   |   Acc_Sd <= 3.63 : -1 (23.0/2.5)
|   |   Acc_Sd > 3.63 :
|   |   |   Acc_Av <= 17.63 : 1 (5.0/1.2)
|   |   |   Acc_Av > 17.63 :
|   |   |   |   Acc_Sd <= 4.72 : 1 (4.0/2.2)
|   |   |   |   Acc_Sd > 4.72 : -1 (17.0/3.7)
Acc_Sd > 7.79 :
|   Brk_Sd > 0.18 : 1 (150.0/21.5)
|   Brk_Sd <= 0.18 :
|   |   Rpm_Av > 1637.89 : 1 (44.0/11.5)
|   |   Rpm_Av <= 1637.89 :
|   |   |   Spd_Av <= 71.27 : -1 (9.0/2.4)
|   |   |   Spd_Av > 71.27 :
|   |   |   |   Spd_Sd <= 2.34 : 1 (5.0/1.2)
|   |   |   |   Spd_Sd > 2.34 : -1 (3.0/2.1)
```

Table 2.5: Data from two users.
+1: aggressive , -1: cautious .
Source: [7]

Because different users have different driving patterns, the aggregated pattern map is naturally computed by calculating the average of various driving pattern maps. Such approach will certainly achieve the best performance on entire data. On the other hands, for an individual whose pattern map is different from the average, the algorithm fails to classify as much as he is deviated from the average. However, it is very likely that driving patterns vary between drivers. Therefore, the two-user case performs worse than the single-user case. Especially, the decision tree algorithm

fails significantly, because it is more susceptible to the inconsistency between training and test data. Moreover, it is not reasonable to have same criterion for two different drivers.

2.4 Discussion

Most importantly, the supervised learning approach has a limitation regardless of the error rate.

First, previous work is limited to binary classification. However, the users' intentions are often significantly diverse, and so binary classification is not effective. One may broaden the work by multi-class classification through binary classification as these methods are not limited to binary classification. However, such a dataset, in which classes are not entirely separable, often performs worse in the multi-class classification through binary classification. For example, assume that there are three driving types: aggressive , neutral , and cautious . Even though the criterion between aggressive and cautious is firm, the criteria between aggressive and neutral , and between neutral and cautious are subtle to determine and thus unstable. Therefore, binary classification for neutral often fails, thus failing the entire multi-class classification.

Second, the algorithms classify per window based on the immediate window, only focusing on a small portion of data each time. However, focusing on one window is not the optimal way to make a decision for time series measurements, such as driving. To resolve this issue, one way to aggregate all information from the beginning of the driving episode until the time of an inference is to calculate the ratio of each driving type so far and choose the higher one. For example, assume that out of 200 windows, 150 windows are classified cautious and 50 windows are aggressive . Then, the ratio for each driving type is 75% and 25%, respectively, and assuming the process does not change, the maximum likelihood estimation for the driving type is cautious . However, the assumption that the process does not change is very strong, and such an approach often fails to catch a sudden change in driving behavior. For example, if the user has driven aggressively for 10 minutes and changes his intention, the machine might take another 10 minutes to detect the change in driving behavior.

Third, the decision rule is susceptible to training data. In some scenarios, two driving types are naturally hard to distinguish, thus corresponding intervals should be marked as indistinguishable . A good algorithm should ignore those intervals and should not be affected by them. However, because the labels are marked per dataset

basis, these intervals are actually marked either aggressive or cautious . In this case, because supervised learning does binary classification between two labels, and because supervised learning tries to minimize the empirical risk, the algorithm often over-fits the training data. Due to this, a subtle variation of the training data may result in the significant change of the decision rule. One naive approach is to remove indistinguishable windows manually before running the algorithm; however, it is infeasible to check all windows by hand because of the size of datasets.

Therefore, we need to suggest an improved algorithm that solves the previously discussed limitations on supervised learning. Fortunately, researchers have solved similar problems in various techniques.

The first problem can be solved by introducing multiple classes, or a space of types. Instead of multi-class class through binary classification, direct multi-class classification may solve the issue of failing binary classification due to one indistinguishable class. Or, estimating the type in a space of types solves the issue and improves the performance even further. The second problem suggests that the improved algorithm must consider not only the immediate measurements, but also the historical measurements and future predictions. For example, an iterative algorithm with memory or discounted rewards enables such approach. The third problem also hints that the models or classification criterion should be trained so that the algorithm can generally perform well, but should be adjusted depending on situation.

In the following chapters, improved algorithms which use parts of these solutions will be discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Driver Type Classification

In this chapter, we discuss an algorithm that uses Bayesian hypothesis testing to diagnose the type of a driver based on a temporal sequence of the perceptual inputs (velocity and distance to car in front) and control actions (pedal positions).

Unlike the supervised learning approach which classifies the type of a driver per observation, our new approach maintains priors of hypotheses and updates as a new observation comes. By remembering the belief state and updating it online, the approach can perform much better than the supervised learning approach in our case.

3.1 Preliminaries

3.1.1 Bayesian hypothesis testing

Bayesian hypothesis testing gives the posteriors of hypotheses from an observation and the priors of hypotheses, following the exposition in the 6.437 class note [3].

Denote a set of hypotheses as $\mathcal{H} = \{H_1, ..., H_i, ..., H_m\}$ and its size as $|\mathcal{H}| = m$. Then, the prior of each hypothesis H_i is denoted as $p_{\mathrm{H}}(H_i)$. Under each hypothesis H_i, the likelihood of an observation y is denoted as $p_{\mathrm{y}|\mathrm{H}}(y|H_i)$ and often given beforehand. Finally, the posterior of an hypothesis H_i under an observation y is (by Bayes' theorem):

$$p_{\mathrm{H}|\mathrm{y}}(H_i|y) = \frac{p_{\mathrm{y}|\mathrm{H}}(y|H_i)p_{\mathrm{H}}(H_i)}{\sum_{j=1}^{m} p_{\mathrm{y}|\mathrm{H}}(y|H_j)p_{\mathrm{H}}(H_j)} \tag{3.1}$$

Given an observation, the maximum a posteriori hypothesis is the one that has

the highest posterior.

3.1.2 Iterative Bayesian hypothesis testing

Denote the prior of each hypothesis H_i at time t as $p_{\mathrm{H,t}}(H_i, t)$. Similarly, denote the posterior of an hypothesis H_i under an observation y_t at time t as $p_{\mathrm{H,t|y}}(H_i, t|y_t)$. For a time series of observations, because the belief propagates over time, the posterior at time t becomes a prior at time $t+1$. Thus, the following relationship holds: $p_{\mathrm{H,t}}(H_i, t+1) = p_{\mathrm{H,t|y}}(H_i, t|y_t)$. Using this relationship, we can iteratively estimate the posteriors of hypotheses give a series of observations.

Similarly, given a series of observations, the maximum a posteriori hypothesis is the one that has the highest posterior at the end.

3.2 Pilot Experiment

We assumed the driver belonged to one of three basic types:

1. speedy (trying to move as fast as possible)
2. economical (trying to save fuel)
3. safe (trying to maintain safe distance in front)

We then hand-coded potential policies (mappings from perceptual inputs to distributions over control actions) for each of these driver types.

Next, we built a system to do online diagnosis of the current driver's "type". It does belief-state estimation, maintaining a current belief distribution (assignment of probabilities to each of the driver types) and updates it online based on each new action of the driver. This process is quite efficient. The estimate can be displayed to the driver as a 'gauge' that shows, implicitly, what their preferences seem to be, based on their actions. If a driver who would like to save fuel sees that he is not being characterized as behaving in a fuel-saving mode, he can try to change his behavior as a result (and in future systems, we would have the computer agent make suggestions to the driver about how to do that).

We tested this model by using the different policies to generate artificial data, and making sure that we could recover the type of the particular policy that was used to generated data. An example trace is shown in **Figure 3-1**.

A crucial feature of this approach, in contrast to the supervised-learning method that we applied initially, is that it is online and can deal with dynamically changing

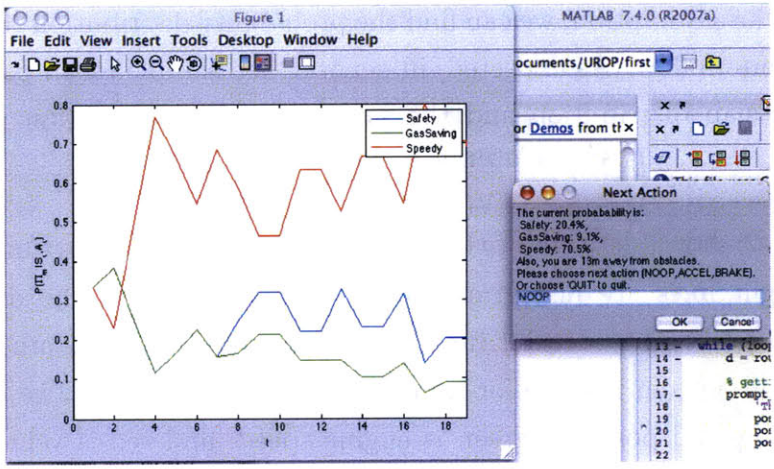


Figure 3-1: Belief in different driving types over time, based on ideal policies and simulated data.

behavior of the driver (who might switch modes on different days or different parts of a trip).

3.3 Driver Type Classification

Our next step was to build a similar dynamic filter based on real driver data. In the previous work, we assumed we knew policies for the different driving types in advance. We can think of a policy as a conditional distribution $\Pr(a|s,\tau)$ that specifies a distribution over actions given the current state, for each driver type, τ.

Now, we need to first 'train' our models based on some of the given data, and then use the rest of the data to 'test' (to determine whether we can diagnose what type of driver is currently driving). To simplify the problem, we trained joint distributional models, rather than conditionals, meaning that we estimated $\Pr(a,s|\tau)$. Finding appropriate probabilistic models for the real data was surprisingly difficult.

3.3.1 Data models

As it is meaningless to classify each data point independently, like supervised learning methods, Bayesian hypothesis testing also aggregates information of datapoints in a small interval into a window by calculating features, such as average and standard deviation, and assumes that each window is independent given the state.

We start by assuming that the distributions for each attribute are independent. This assumption is clearly not true, but it greatly simplifies the modeling problem, and we can re-address it later if our models are not sufficiently discriminative.

Based on the assumption, we can find the probability distribution that the features of each attribute follow. As expected, in general, the sample mean parameter values follow a Gaussian distribution and the sample standard deviation values follow a Gamma distribution: $\begin{pmatrix} s^2 \sim Gamma(\alpha, \beta) \\ m \sim N(\mu, \sigma^2) \end{pmatrix}$

Figure 3-2 shows data and fitted parameters for several distributions. In each figure, the light lines are histograms of the actual data and the dotted lines are the estimated probability density functions. The top two graphs are for the sample means, and the bottom two are for sample standard deviations. The blue curves are for accelerator pedal position; green is engine speed; and red is vehicle speed. The graphs on the left are for an aggressive driver and the graphs on the right are for a cautious driver.

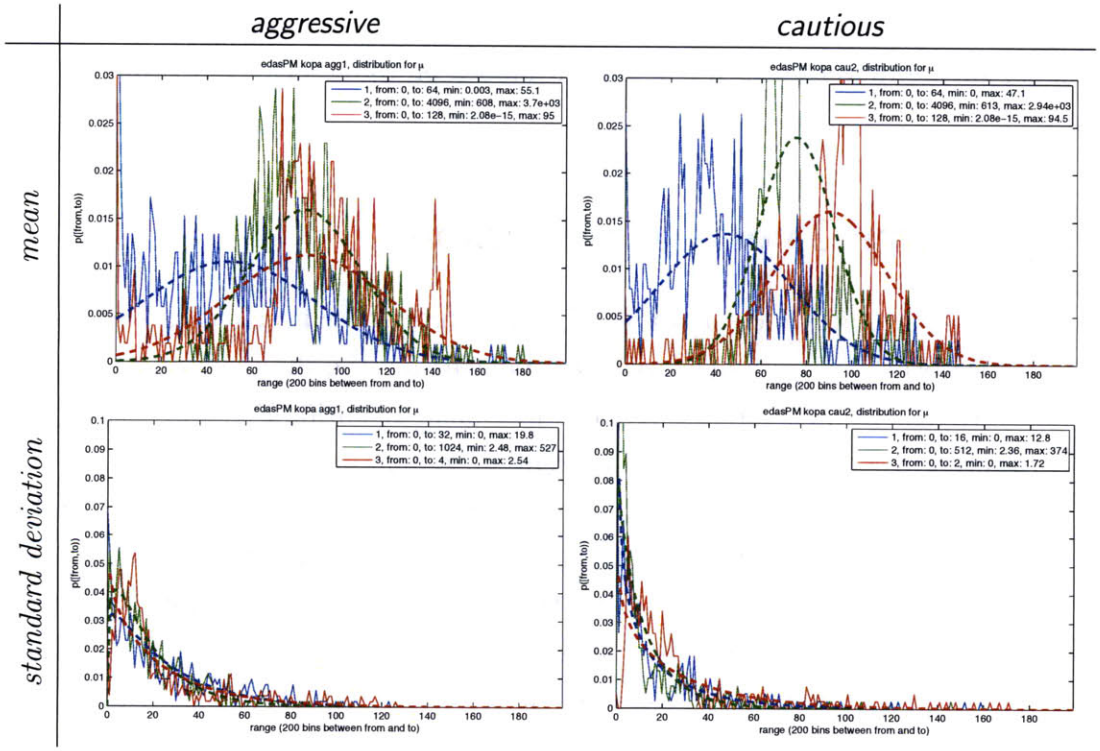


Figure 3-2: Probability distributions of three measurements for different driving types. Clearly those distributions differ between driving types. The left graphs show distributions for aggressive driving type, whereas the right graphs show for cautious . The colored lines indicate measured histograms, while the dashed thick line indicate empirical distributions. The fitted distribution in general mimics the measured histogram. See detailed description in text.

Many measurements cannot be effectively modeled with a single Gaussian or Gamma distribution. For example, velocity is 0 a significant proportion of the time, and so is the variance of the brake pedal. We model these situations with *mixture*

distributions, which give a value, such as zero, a fixed proportion p of the time, and otherwise yield a value drawn from a Gaussian distribution. We ultimately use these families of distributions for modeling the various properties: **Table 3.1**.

Table 3.1: Possible Models for Measurements

Name	*Model*
Normal	$\left(\begin{array}{rcl} m & \sim & N(\mu, \sigma^2) \end{array} \right)$
DeltaNormal	$\left(\begin{array}{rcl} m & \sim & p_m\delta(m) + (1-p_m)(1-\delta(m))N(\mu, \sigma^2) \end{array} \right)$
NormalGamma	$\left(\begin{array}{rcl} m & \sim & N(\mu, \sigma^2) \\ s^2 & \sim & Gamma(\alpha, \beta) \end{array} \right)$
DeltaNormalGamma	$\left(\begin{array}{rcl} m & \sim & p_m\delta(m) + (1-p_m)(1-\delta(m))N(\mu, \sigma^2) \\ s^2 & \sim & Gamma(\alpha, \beta) \end{array} \right)$
DeltaNormalDeltaGamma	$\left(\begin{array}{rcl} m & \sim & p_m\delta(m) + (1-p_m)(1-\delta(m))N(\mu, \sigma^2) \\ s^2 & \sim & p_s\delta(s^2) + (1-p_s)(1-\delta(s^2))Gamma(\alpha, \beta) \end{array} \right)$

As shown in **Figure 3-2**, different driving types have distinct distributions and therefore must be parameterized by dissimilar parameters. We have found appropriate models for measurements of each dataset type, and empirically calculated the parameters for the model for aggressive and cautious driving types (**Table 3.2**).

We assume that we have training data classified according to the type of the driver, and use maximum-likelihood estimation procedures to compute the parameters for each of these distributions from training data, yielding distributions of the form $\Pr(a, s|\tau)$ for different driving types τ.

3.3.2 Transition model

We will assume that drivers can possibly change their type over time. Our initial belief, before seeing any data, is uniform over the driver types. If we have n driver types, then $\Pr(T_0 = \tau_i) = 1/n$, initially. The random variable T_0 represent the driver's type at time 0. Still, if data is available, this initial distribution can be estimated empirically as well.

The transition model specifies how the driver's type is likely to change over time. It is specified with a matrix of transition probabilities, indexed by i and j, that specify

$$\Pr(T_{t+1} = \tau_j | T_t = \tau_i) \ .$$

Table 3.2: Models and Calculated Parameters of Measurements. As common knowledge suggests, the sample mean of Accelerator Pedal Position, Engine Speed, and Vehicle Speed of Type-1 are larger for aggressive type than cautious type, while cautious uses more brake than aggressive type.
Type-2 shows similar trends on measurements. Notably, cautious type maintains almost twice as much distance to the closest vehicle as does aggressive type.

(a) Type-1

Measurement *Mode*	*Parameter* aggressive	cautious
Accelerator Pedal Position DeltaNormalGamma	$p_m = 0.12$ $\mu = 15.6, \sigma = 11.5$ $\alpha = 3.8, \beta = 2.6$	$p_m = 0.28$ $\mu = 14.3, \sigma = 9.1$ $\alpha = 2.0, \beta = 2.5$
Engine Speed NormalGamma	$\mu = 1601, \sigma = 503$ $\alpha = 1.5, \beta = 155.3$	$\mu = 1476, \sigma = 403$ $\alpha = 0.70, \beta = 164.6$
PCM Brake On DeltaNormal	$p_m = 0.32$ $\mu = 0.46, \sigma = 0.29$	$p_m = 0.73$ $\mu = 0.68, \sigma = 0.36$
Vehicle Speed NormalGamma	$\mu = 60.2, \sigma = 26.0$ $\alpha = 0.8, \beta = 2.5$	$\mu = 58.73, \sigma = 26.2$ $\alpha = 1.5, \beta = 1.8$

(b) Type-2

Measurement *Mode*	*Parameter* aggressive	cautious
vehicle speed NormalGamma	$\mu = 15.9, \sigma = 7.7$ $\alpha = 1.3, \beta = 0.70$	$\mu = 14.4, \sigma = 6.6$ $\alpha = 1.1, \beta = 0.58$
distance to the closest vehicle DeltaNormalGamma	$p_m = 0.17$ $\mu = 20.7, \sigma = 17.6$ $\alpha = 0.50, \beta = 17.2$	$p_m = 0.23$ $\mu = 34.6, \sigma = 23.0$ $\alpha = 0.55, \beta = 22.2$
acceleration pedal position DeltaNormalDeltaGamma	$p_m = 0.15$ $\mu = 13.5, \sigma = 10.2$ $\alpha = 2.6, \beta = 3.0$	$p_m = 0.16$ $\mu = 10.8, \sigma = 6.7$ $\alpha = 1.9, \beta = 2.0$
brake pedal position DeltaNormal	$p_m = 0.49$ $\mu = 26.3, \sigma = 19.6$	$p_m = 0.68$ $\mu = 30.4, \sigma = 21.4$
steering angle DeltaNormalGamma	$\delta = 0.01$ $\mu = -4.6, \sigma = 29.7$ $\alpha = 0.47, \beta = 18.2$	$\delta = 0.02$ $\mu = -3.4, \sigma = 30.3$ $\alpha = 0.42, \beta = 16.2$

In our example, we will have two τs: *aggressive* and *cautious* , and set the probability of staying in the same type as 0.9, and of changing types as 0.1.

3.3.3 Filtering

Now, we can do filtering, in which we are given a stream of data from a new driver, and have to output, on each step, the current probability distribution over this driver's type, conditioned on the historical data.

On each step, we compute a new distribution on driver type, $\Pr(T_{t+1})$, based on the most recent observed data and on the old driver type distribution $\Pr(T_t)$.

Given a window and prior probability, we can calculate the posterior probability using Bayes rule, where a is aggressive driver type, c is cautious driver type, and d is the observed data:

$$\begin{pmatrix} \Pr(T_{t+1}=a|d) \\ \Pr(T_{t+1}=c|d) \end{pmatrix} = \frac{1}{z} \underbrace{\begin{pmatrix} \Pr(T_{t+1}=a|T_t=a) & \Pr(T_{t+1}=a|T_t=c) \\ \Pr(T_{t+1}=c|T_t=a) & \Pr(T_{t+1}=c|T_t=c) \end{pmatrix}}_{\text{transition matrix}} \begin{pmatrix} p(d|T=a)\Pr(T_t=a) \\ p(d|T=c)\Pr(T_t=c) \end{pmatrix}$$

where z is a normalizing constant.

3.3.4 Results

For each driving type, we divide our datasets into training and testing datasets. The training dataset is used for parameter estimation, and the testing dataset is used for filtering.

Figure 3-3(a) shows the results of filtering on each of six separate Type-1 data sets. **Figure 3-3(b)** shows basically the same type of results for the Type-2 data (which has fewer features, but includes distance to the vehicle in front). The model for aggressive driving was trained on the training half of each of the aggressive data sets, and the model for cautious driving was trained on the training half of each of the cautious data sets.

The top row of each graph shows filtering results for aggressive drivers and the bottom row for cautious drivers. The blue curve is the probability that the driver is aggressive , conditioned on the testing data seen so far, $\Pr(T_t = a)$, and the green curve is the probabilty that the driver is cautious , conditioned on the testing data. We can see that in five of these cases, the filtering algorithm quickly converges to the appropriate driver type. In one case, with Finn driving aggressively, it is confused with cautious for part of the time.

We then calculate the error rate of our new approach by choosing the most likely driving type and comparing it with real labels, as shown in **Table 3.3**. In general, Bayesian hypothesis testing performs very well on both Type-1 and Type-2 datasets, achieving roughly 4% of error rates. Comparing to supervised learning whose error rates are 15% to 20%, the algorithm has one fourth of error rates.

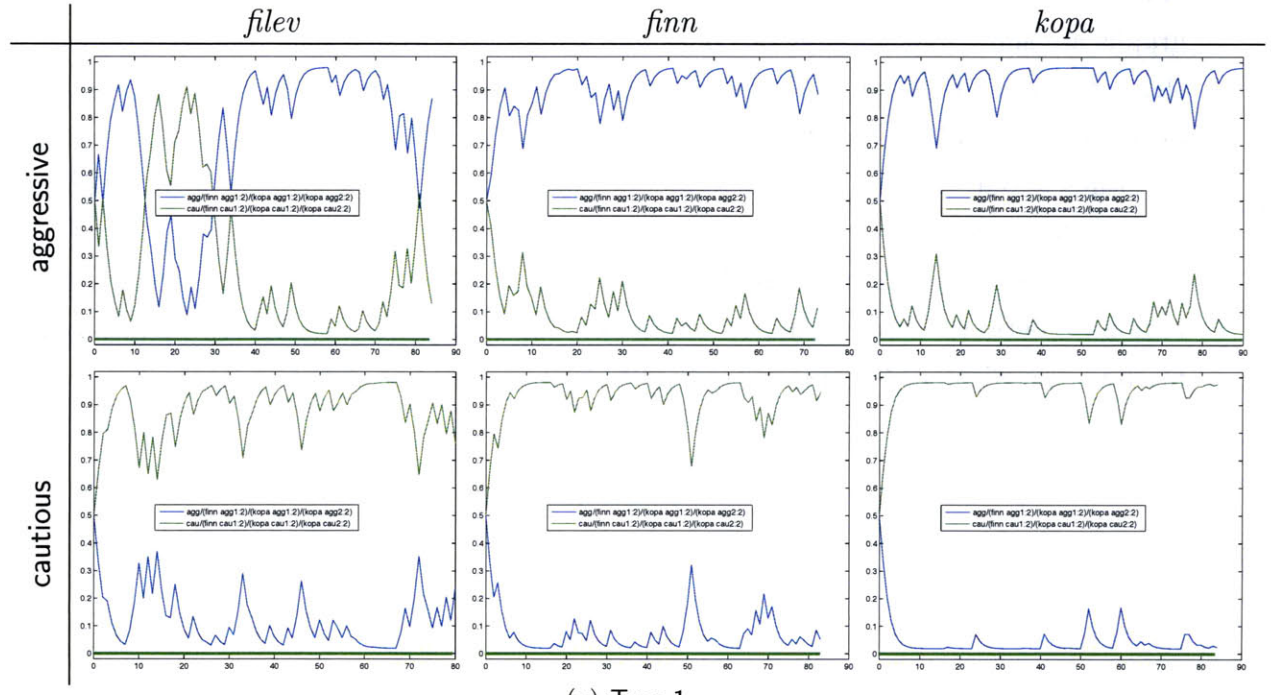
Table 3.3: Result of Bayesian hypothesis testing. On average, the algorithm achieved roughly 4% error rates on both types of datasets, which are significantly better than supervised learning algorithm's error rates of 20% to 25%. **Figure 3-3** shows the detailed belief changes of each dataset.

(a) Type-1

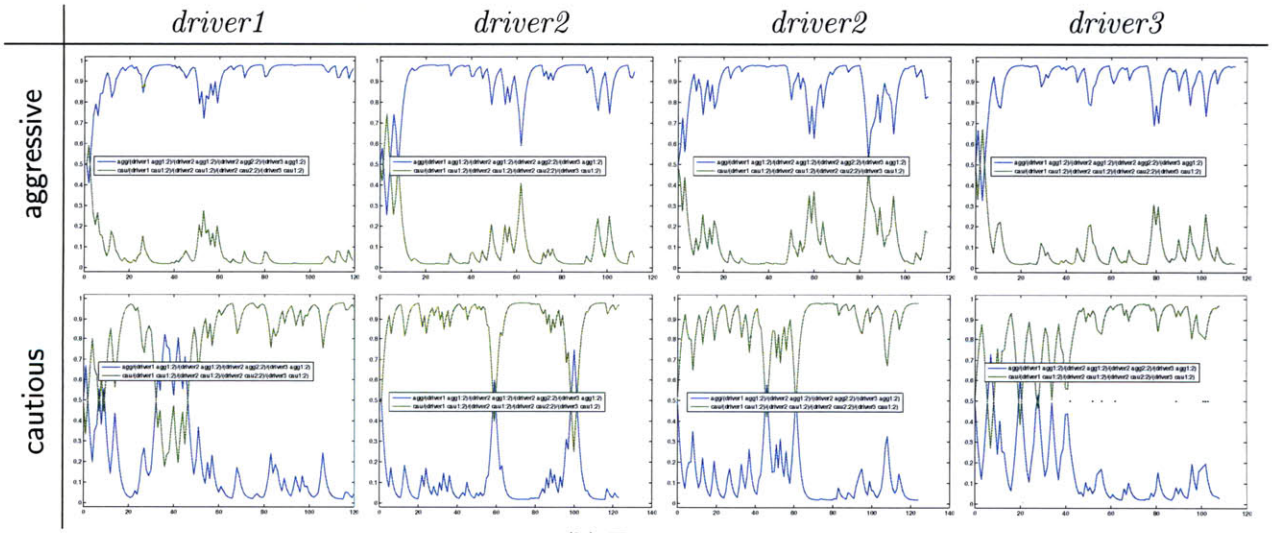
Driver	*Label*	*Error Rate*
filev	aggressive	0.2143
filev	cautious	0
finn	aggressive	0
finn	cautious	0
kopa	aggressive	0
kopa	cautious	0
average		0.0357

(b) Type-2

Driver	*Label*	*Error Rate*
driver1	aggressive	0.0167
driver1	cautious	0.0331
driver2	aggressive	0.0442
driver2	cautious	0.0484
driver2	aggressive	0
driver2	cautious	0.0400
driver3	aggressive	0.0172
driver3	cautious	0.1193
average		0.0399



(a) Type-1

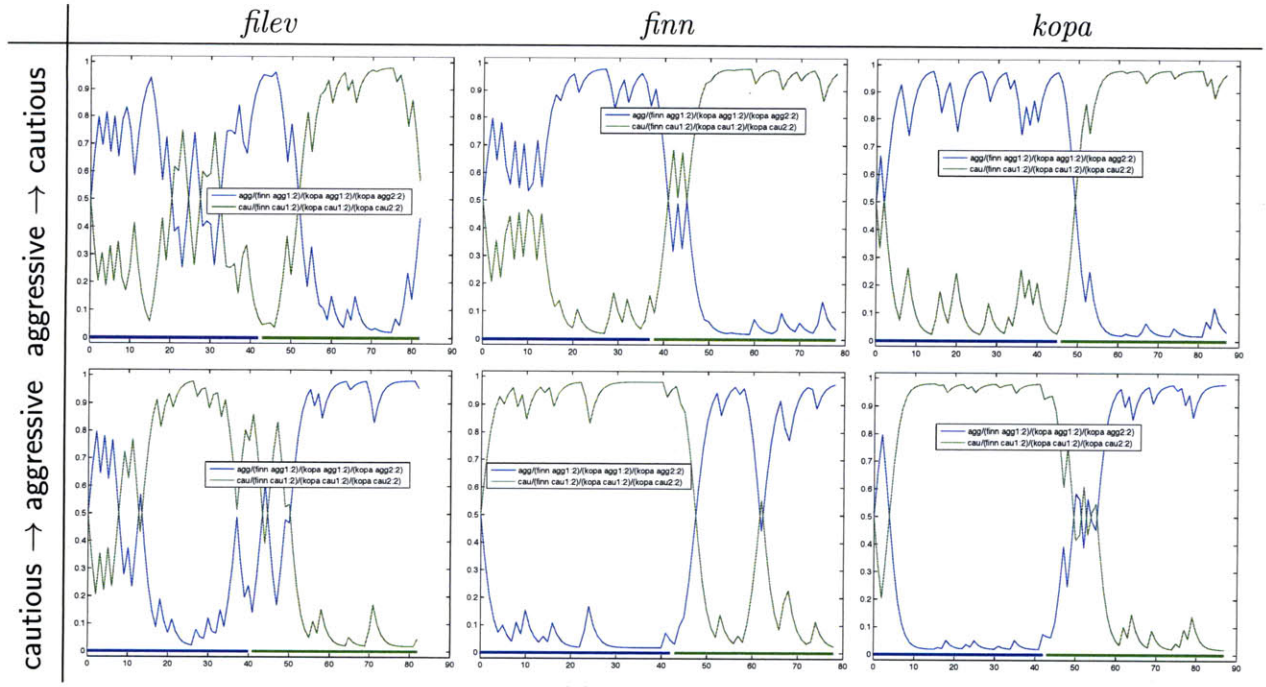


(b) Type-2

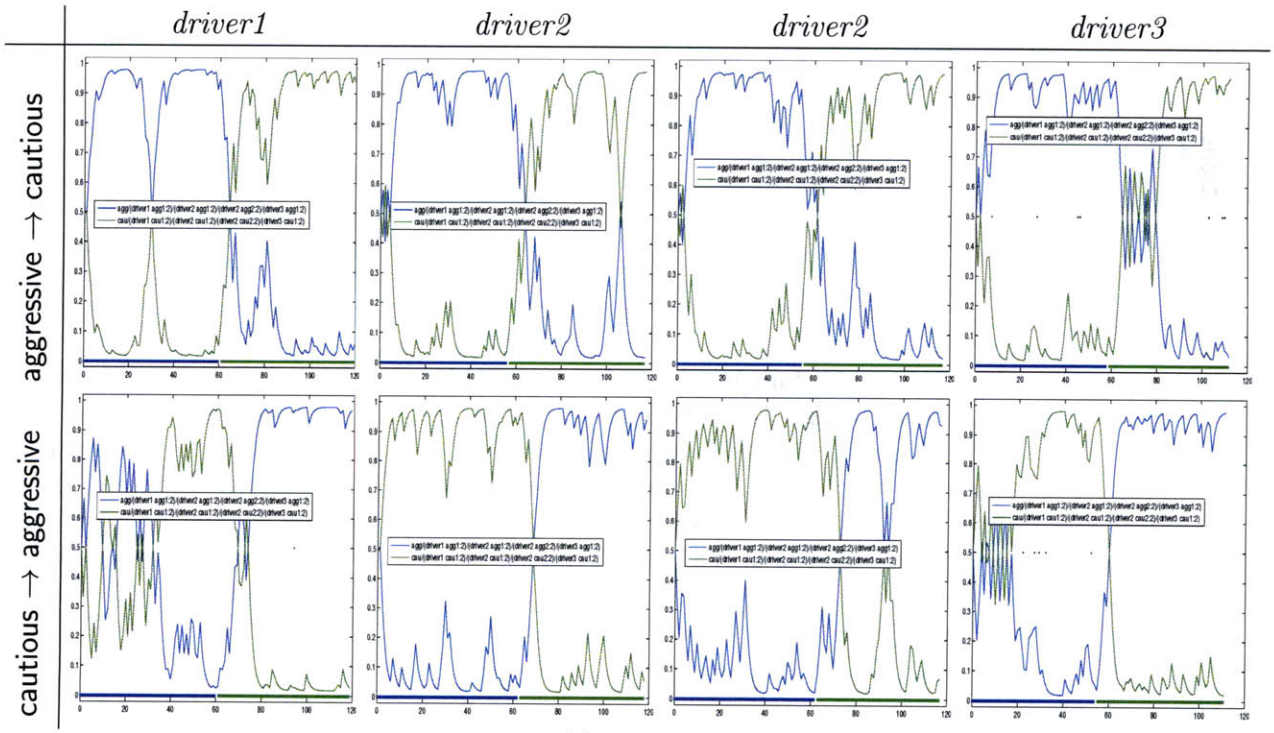
Figure 3-3: The training set for each driving type is generated by collecting all drivers' demonstrations. Then, the algorithm is tested on each dataset. Each subfigure corresponds to a dataset. The blue line indicates the posterior of aggressive driving type, whereas the green lines indicates that of cautious driving type.

In addition, to test whether the driving-type diagnosis could really track changing type, we performed a set of experiments in which we switched the data stream from one type to the other during the filtering process. **Figure 3-4** shows the results. The graphs are similar to the previous ones, but the change from a blue to a green line at the bottom of the graph indicates the driving type has changed. Due to the strong belief of one mode before the transition, the filter doesn't instantaneously change hypotheses, but after seeing a small amount of data of the new type, it fairly reliably changes hypotheses.

We tried an additional experiment in the Type-2 data, to see if we could discriminate between different individual drivers (as opposed to driving types) with our method. As **Figure 3-5** shows, there is no stable posterior distribution.



(a) Type-1



(b) Type-2

Figure 3-4: The training set for each driving type is generated by collecting all drivers' demonstrations. Then, the algorithm is tested on a swapped dataset whose driver changes his behavior in the middle of driving. Each subfigure corresponds to a dataset. The blue line indicates the posterior of aggressive driving type, whereas the green lines indicates that of cautious driving type. The sold blue and green lines at the bottom of each plot indicate the actual driving type.

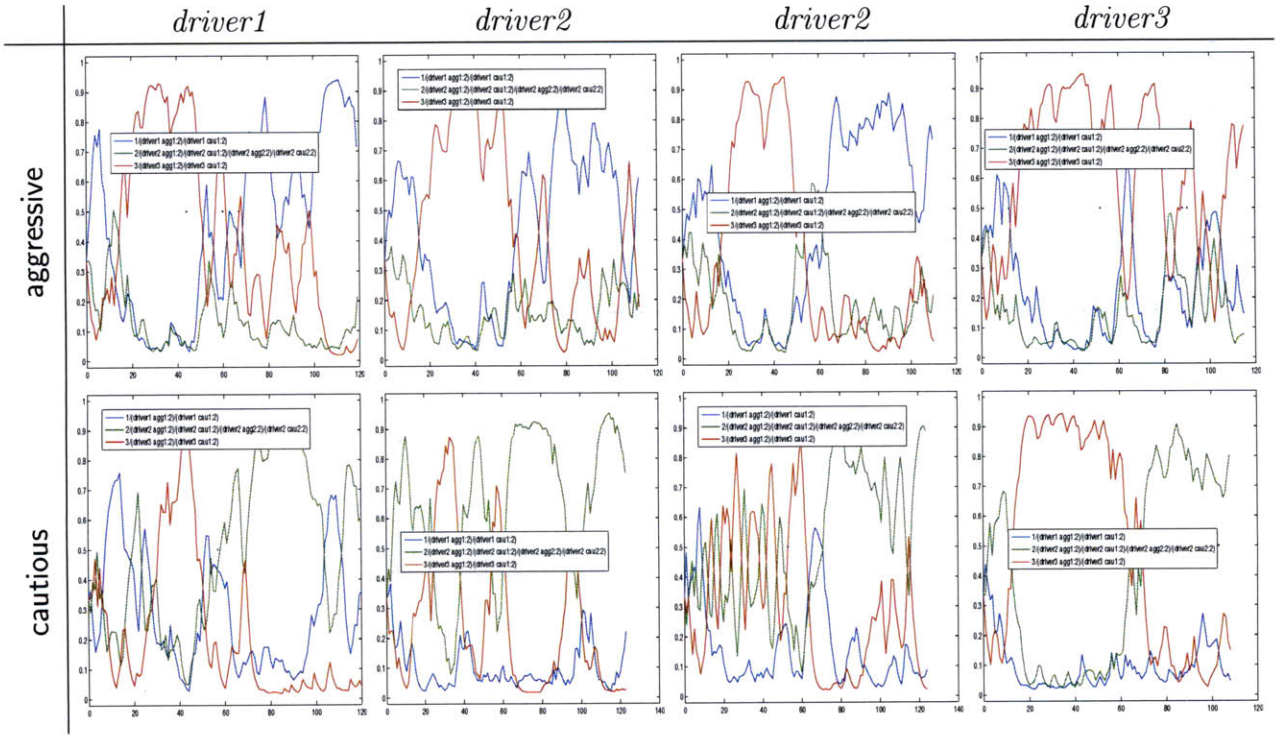


Figure 3-5: The training set for each driver is generated by collecting each driver's demonstrations. Then, the algorithm is tested on each dataset. Each subfigure corresponds to a dataset. The blue, green, and red lines indicate driver 1, driver 2, and driver 3, respectively. Unfortunately, the algorithm cannot determine the driver correctly. However, the result gives some insight about each driver's behavior. As the beliefs for driver 1 and 3 (blue and red) dominate at the aggressive driving type, whereas those for driver 2 and 3 (green and red) dominate at the cautious driving type, we can claim that the driver 1 is the most aggressive, and 2 is most cautious, and 3 is in between.

3.4 Classification with Online Update of Models

Using the model and parameters found from the previous chapter, we ran the algorithm with a newly collected data. Unlike the previous data which do not have transitions, the new data were collected to test whether

1. the algorithm can work for any new driver, and
2. the algorithm can detect a transition between driving types

Specifically, three drivers have driven in their own styles and then changed their driving styles in the middle of a dataset (Type-2 B). **Figure 3-6** shows the beliefs over time for each driver and each transition.

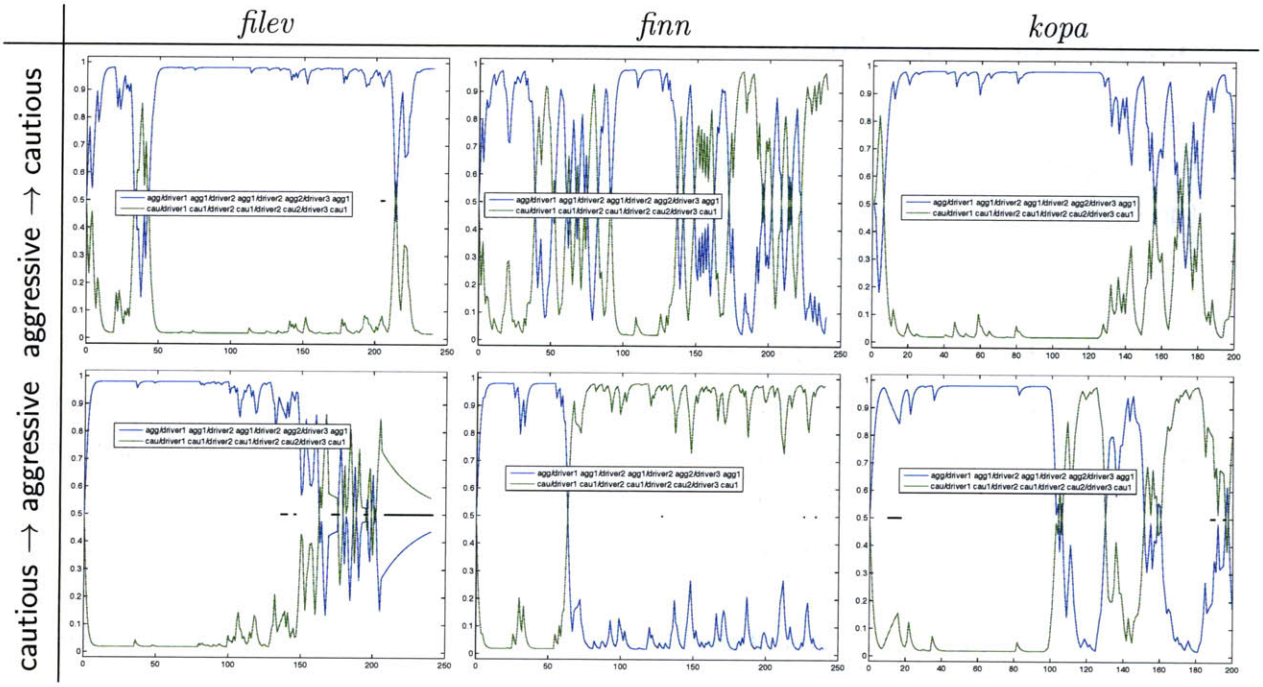


Figure 3-6: Beliefs on newly observed dataset

Even though there is noise, the driving style of Finn (the second column) is well distinguished. However, Filev's style is more aggressive and the algorithm classifies his entire driving as aggressive. Similarly, Kopa's cautious driving is not detected fully.

This problem comes from the fact that the criterion of aggressive versus cautious level is different across people. Even though the training datasets can give a criterion that works for most people, it might not work well for a specific individual. Specifically, when a new person drives a car, his driving style is not fully distinguishable using the model that was statically trained. That is, although the models derived

from training datasets can serve as a good initialization, we need to adaptively adjust the model online depending on a new person's style, as more data are available over time.

There are two important obstacles to such a scheme.

1. How to update the model and classify at the same time?
2. Without labels, how to update the parameters online?

Because we do not know the labels, regardless of the classification of each window, whether it is aggressive or cautious , we should rely on the unsupervised, or semi supervised methods. In this chapter, we propose three ways to resolve this issue.

3.4.1 Learning Parameters Slowly

One way is to learn the parameters slowly over time [12]. If the classification for one driver has remained in one class for a long time, we can update the parameters to fit the behavior of a specific driver better. For example, when one driver has driven in very high speed, even faster than the standard aggressive drivers, for a long time, we should raise the mean of the velocity.

Using this logic, we can adjust the parameters when the classification is saturated. Formally, when the posterior for one class exceeds a certain threshold, we can slowly update the parameters. The update of one parameter can be done simply with a learning rate γ by $a_{t+1} = a_t + (1 - \gamma)\Delta a_t$, where a_t is the parameter at t.

The mean of the normal distribution is one of the critical parameters that distinguishes various intentions and it can be updated this way. However, other parameters, such as the standard deviation of the normal distribution, and alpha and betas of the gamma distribution, are very subtle and hard to update incrementally.

3.4.2 Online Clustering

Another method is to perform unsupervised learning online. For every iteration, a clustering method, such as k-means clustering, is performed to divide datapoints into several classes. Based on the result from clustering, training is performed again to fit parameters for each class, and a new belief is calculated.

However, in this approach, the belief might be very unstable, because clustering differs significantly over time. Thus, we need to have a better approach, which instead of statically assigning each datapoint to one class, but makes a soft assignment to various classes in a way that maximizes the likelihood of the data.

3.4.3 Expectation Maximization

A more sophisticated way to update the parameters is the unsupervised method, expectation maximization, to update the model.

Unlike our previous method, in which each datapoint is labeled and the pdf for each class is estimated independently, the EM method maintains beliefs of each datapoint having been generated by each class: b_{ij} where i is the index for datapoint and j is the class number. Note that the sum of beliefs for all classes for one datapoint must be 1: $\sum_{j=1}^{m} b_{ij} = 1$.

However, often the new information from recent driving data is more important than the old information. Thus, we need to add another weight w_i to consider how important each datapoint is. Note that the sum of weights for all datapoints must be 1: $\sum_{i=1}^{n} w_i = 1$.

The target likelihood function that we want to maximize is:

$$L = \prod_i \prod_j f(x_i|\theta_j)^{(b_{ij}w_i)}$$

where f_j is the likelihood function of the class j.

As the name implies, the EM algorithm iteratively executes a E step and a M step, so that the algorithm can updates both the beliefs and the parameters at the same time.

EM step

In the E step, the beliefs of each datapoint are calculated. Using the current parameters θ_j^{iter}, we can calculate the likelihood of the datapoint being in each class by $f(x_i|\theta_j^{iter})^{(b_{ij}^{iter})}$ (note that w_i is dropped because it does not affect each datapoint).

$$b_{ij}^{iter+1} \propto f(x_i|\theta_j^{iter})^{(b_{ij}^{iter})}$$

In the M step, the parameters for the pdf are re-calculated based on the beliefs. That is,

$$\theta_j^{iter} = \underset{\theta_j}{\operatorname{argmax}} \prod_i f(x_i|\theta_j)^{(b_{ij}^{iter}w_i)}$$

Since we assumed that each datapoint is iid given the class and measurements are independent each other as well, we can estimate the parameters with maximum likelihood estimator with considering weights.

1. **Normal model:**

$$\begin{aligned}\mu &= E[x_i] = \frac{\sum x_i b_{ij}^{iter} w_i}{\sum b_{ij}^{iter} w_i} \\ \sigma &= E[x_i^2] - E[x_i]^2 = \frac{\sum x_i^2 b_{ij}^{iter} w_i}{\sum b_{ij}^{iter} w_i} - \mu^2\end{aligned}$$

2. **DeltaNormal model:**

$$\begin{aligned}p_m &= P[x_i = 0] = \frac{\sum_{x_i=0} b_{ij}^{iter} w_i}{\sum b_{ij}^{iter} w_i} \\ \mu &= \text{same above} \\ \sigma &= \text{same above}\end{aligned}$$

3. **Gamma model**: This model does not have a closed form. Thus, we need to find a numerical solution [8].

   The likelihood function is: $p(x, w|\alpha, \beta) = \prod_i \Gamma(x_i, w_i|\alpha, \beta) = \left\{\frac{x^{\alpha-1}}{\Gamma(\alpha)\beta^\alpha}\exp(-\frac{x_i}{\beta})\right\}^{w_i}$

   Taking a log-likelihood function, $\log p(x, w|\alpha, \beta) = (a-1)\sum_i w_i \log x_i - \sum_i w_i \log\Gamma(\alpha) - \sum_i w_i \alpha \log(\beta) - \beta^{-1}\sum_i w_i x_i$.

   By taking the derivative with respect to β and setting to 0, $0 = \sum_i w_i\alpha/\beta + \beta^{-2}\sum_i w_i x_i$. Thus, the maximum can be achieved when $\hat{\beta} = \frac{\sum w_i x_i}{\sum w_i}/\alpha$.

   Substituting $\hat{\beta}$ into the log-likelihood, we get:
   $\log p(x, w|\alpha, \hat{\beta}) = (a-1)\sum_i w_i \log x_i - \sum_i w_i \log\Gamma(\alpha) - \sum_i w_i \alpha \log(\frac{\sum w_i x_i}{\sum w_i}/\alpha) - \sum w_i\alpha$.

   This equation, unfortunately, cannot be solved analytically. Rather, using Newton's method, we can numerically find an α which maximizes the log-likelihood.

   Summarizing,

$$\begin{aligned}\alpha &= \underset{\alpha}{\text{argmax}}\left[(a-1)\sum_i w_i \log x_i - \sum_i w_i \log\Gamma(\alpha) - \sum_i w_i\alpha\log(\frac{\sum w_i x_i}{\sum w_i}/\alpha) - \sum w_i\alpha\right] \\ \beta &= \frac{\sum w_i x_i}{\sum w_i}/\alpha\end{aligned}$$

3.4.4 Simplified Baum Welch

The basic EM algorithm is a nice way to update the parameters online. However, it does not consider the transition probabilities nor the belief history. However,

the transition probabilities may differ from person to person, Thus, in addition to updating the parameters of the pdf, we also need to update the transition probabilities and and the belief history online.

The target likelihood function that we want to maximize is:

$$L = \prod_i \prod_j f(x_i|\theta_j)^{(b_{ij}w_i)} \prod_{t=1} w_t \left[\prod_{j1,j2} T_{j1,j2} b_{tj1} b_{t+1,j2} \right],$$

where T is the transition matrix. Note that in this approach, not all of the datapoints are being used to calculate the transition probability; specifically, only the datapoints that were observed online should be considered.

The solution of this problem is well known as the Baum-Welch algorithm [2]. Nevertheless, by assuming some properties of the transition matrix, such as symmetric, we can simplify the approach slightly.

3.4.5 Results

Setup

The algorithm initializes the the pdf of each class to be the distribution previously estimated on pooled data. In addition, as the algorithm requires some pool datapoints so that the distributions are not too saturated, the algorithm might need to wait until certain number of datapoints are collected.

As a new data is available (for each estimation), the algorithm adds a new datapoint to the datapoint pools, and re-weights the datapoints so that each of the new and old data is weighted appropriately. Then, by iteratively running EM until the saturation point, the algorithm recalculates the beliefs and adjusts pdfs for classes.

During the first run, thanks to the 'nice' initialization, the EM algorithms reaches the saturation point very quickly and does not modify pdfs much. However, as time passes and more new datapoints are collected, the parameters should be adaptively modified to reflect the true distribution of the data.

Expectation Maximization

For this experiment, we use newly collected datasets that were not used in the previous section. We have 10 Type-2 C datasets with different drivers, each of which has two behaviors.

Figure 3-7 shows the change of models' parameters during EM iteration. As the result of EM depends on how the parameters are initialized and how the inputs are weighted, to show the change more clearly, we gave each initial 'good' parameters from previous section for all datasets and equal weights on inputs. Note that the initial parameters serve as a ground for everyone, and the resulting parameters should be a specific adaptation for a given driver. Therefore, the shape of the graph should look like multiple lines, originating from the center, spreading outward.

The graphs of parameters of Normal / Gamma distribution of vehicle speed and Gamma distribution of steering angle show such a shape clearly. If the graph does not heave that way, we can conclude that the parameters were not initialized properly. For example, the spread of the parameters of Normal distribution of acceleration pedal position (the blue line spread upward) indicates that cautious type's initial standard deviation was set to be lower than all of the individual drivers for whom we have data.

In addition, the shape of the spread also gives an insight about measurements. For example, the spread of the parameters of Normal distribution of vehicle speed is larger vertically (standard deviation) than horizontally (mean). Also, there is a clear distinction between cautious and aggressive drivers based on mean. That is, there exists an unanimous among all people about the average speed, but the idea about the change of speed might differ. Specifically, datasets 3 and 4 show the tendency of very low change of speed.

Similarly, the shape of the spread of Gamma distribution of steering angle is larger horizontally (alpha) than vertically (beta), and mostly separable between two types. Thus, an unanimous also exists for the general shape of the standard deviation's pdf.

However, the graph of brake pedal position only shows blue/red lines that spread against each other. Note that EM only tries to increase the likelihood of the observed datapoints. Since the cautious type tends to use the brake more often than does the aggressive type, the EM algorithm chooses a low mean for aggressive , and a high mean for cautious as iterations progress. This behavior suggests that the EM algorithm may suffer serious overfitting problems for high iteration numbers, and that the models are not necessarily a very good fit for the distribution, which may contribute to making the algorithm go unstable. Thus, we exclude brake pedal position from updating parameters during EM, and achieve better performance.

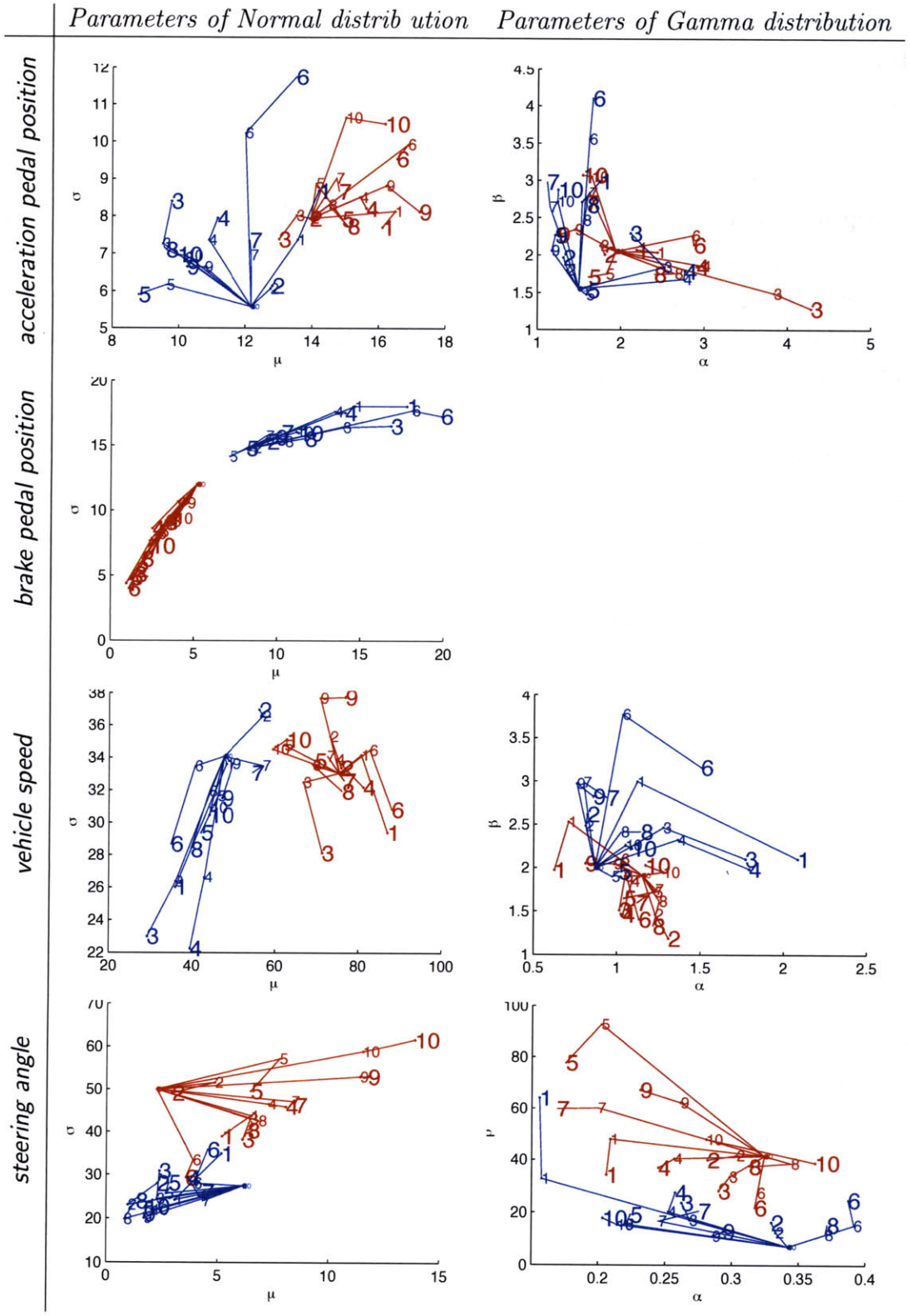
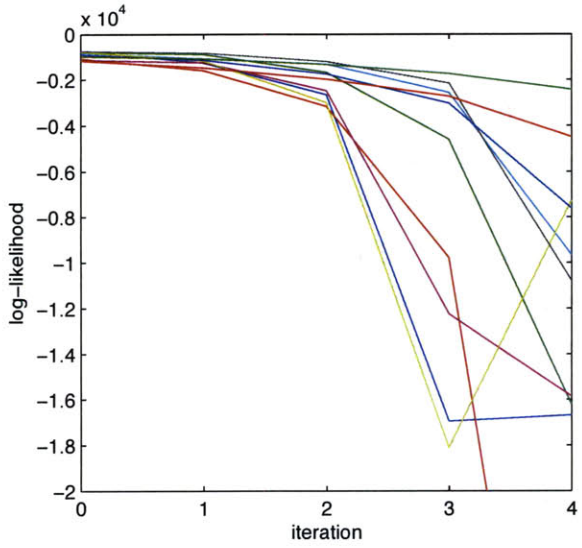


Figure 3-7: Change of probability distributions's parameters during EM iterations. Red lines indicate aggressive type's parameters, whereas blue lines indicate cautious type's parameters. Colored numbers denote the dataset number. Several lines of same color start with one origin (the original distribution was statistically initialized), and deviate to accommodate each driver's behavior.

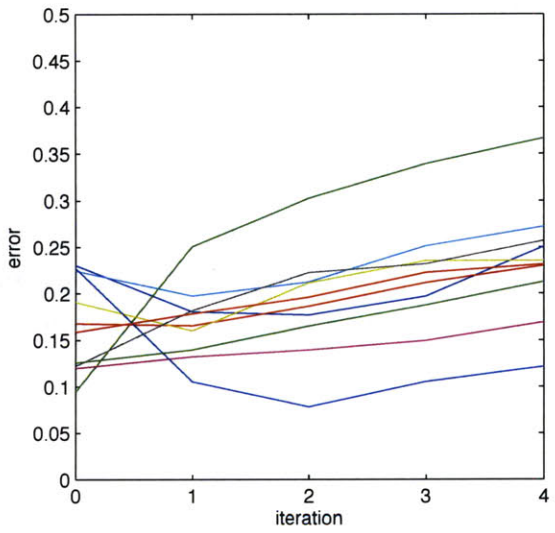
Figure 3-8(a) shows the change of likelihood rate during EM algorithm. The likelihood increases gradually until the second iteration, and suddenly jumps from third iteration. Also, the changes after the third iteration are not consistent among datasets. Therefore, we can conclude that only one or two EM iterations are valid, and to avoid overfitting, we should stop EM after 1 or 2 iterations.

Figure 3-8(b) plots the change of the error rate during EM algorithm. Thanks to the initialization, the algorithm achieves a good result since beginning, and the performances deteriorates because of overfitting after second iteration. However, only half of the datasets achieves the lower error rate in the first iteration, whereas second half unfortunately worsen the performance as a result of EM algorithm.

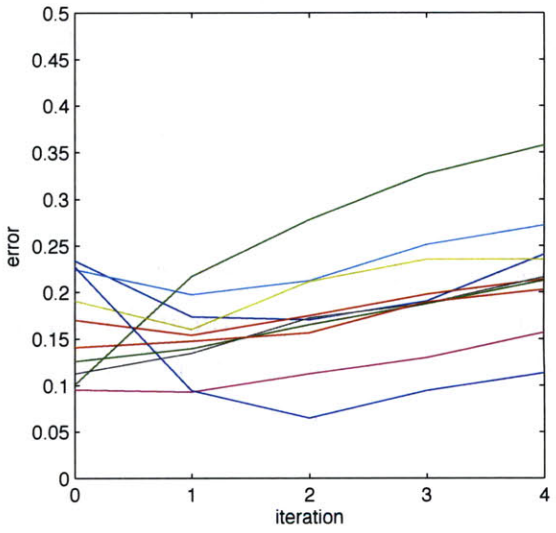
Nevertheless, note the purpose of the EM algorithm, in which the parameters are adjusted for each individual. Therefore, to evaluate the performance of the EM algorithm better, we should also adjust the criterion between aggressive and cautious for testing. Thus, **Figure 3-8(c)** shows the error rates after adjusting the criterion depending on the driver's behavior. Most of the datasets achieve better results after one or two iterations of the EM algorithm.



(a) log-likelihood



(b) uncompensated error



(c) compensated error

Figure 3-8: Log-likelihoods and error rates over iterations. Log-likelihoods decrease as iterations progress, but the significant drop beyond the third iteration is clearly a result of overfitting, and the error rates deteriorate as well. Note that Type-2 C datasets are labeled with the degree of aggressiveness over time, and the degree and whether it exceeds some threshold is used to verify the classification results of EM. The uncompensated error rates are computed with a fixed threshold, and the compensated error rates are with a variable threshold depending on the driver.

Note that one dataset (the light green line) decreases the likelihood after the third iteration, while the EM algorithm should monotonically increase the likelihood, due to the complicate model which consists of Normal, Gamma, and Delta functions.

3.5 Discussion

Bayesian hypothesis testing solves some limitations of a supervised learning approach. Instead of classifying per window based on the immediate window, the testing keeps the beliefs for each type and updates as new data are available. This method helps stabilize estimation against the sudden anomaly, but still enables the gradual change of driving types. In addition, the EM algorithm updates the model as well to adjust the criterion depending on the driver's behavior. Note, even though the testing is only done with aggressive and cautious types, the algorithm can be easily extended to the multi-class classification.

However, still a few limitations exist. First of all, the short interval length may harm the assumption that each window is drawn from an identical distribution. When the interval length is long, such as 30 seconds or more, this assumption is reasonable. However, due to the long interval length, the online algorithm only makes a lagged inference. By making this interval shorter, we can achieve a higher inference rate; however, too short an interval length might invalidate the assumption.

Second, the distributions are not exact. Because it is difficult to find a proper distribution of all measurements, we assume that each measurement and its features are independently distributed with respect to each other. With the assumption, we propose several distributions depending on the characteristics of each measurement. Then, we choose an appropriate distribution for each measurement, and its parameters are empirically estimated independently. This approximation significantly reduces the complexity of the problem without harming the result. However, this assumption that each measurement is independent is too strong.

Third, we failed to optimize the transition probabilities online. The simplified Baum-Welch algorithm saturates the transition probability as iterations progress, and achieves lower performance than the EM algorithm, in which the transition matrix is fixed. High number of iterations causes the likelihood of a datapoint given one driving type to be significantly larger or smaller than that given another driving type, due to overfitting. Then, the posteriors are saturated as either 1 or 0, regardless of priors and the transition matrix. That is, the classification solely depends on the immediate measurements, which is why the previous supervised learning algorithm did not work. In such situations, only slight changes in the measurements may cause the the transition between types and thus the learned transition probability becomes unstable for each iteration. Even the first few iterations before overfitting cause similar problems as well.

Chapter 4

Utility Function Estimation

In the previous system, we only considered two or three different driver types, and tried to categorize the driver as belonging to one of them. A richer model is to introduce a *space* of types, where, for example, we characterize each driver as trying to optimize a reward function of the form

$$R_{driver} = \alpha \cdot R_{\text{economical}} + \beta \cdot R_{\text{safe}} + \gamma \cdot R_{\text{speedy}} \text{ , where } ||\langle \alpha, \beta, \gamma \rangle|| \leq 1$$

More generally, each driver's reward function $R_{\text{driver}}(S, A)$ is a linear combination of features $\underline{\phi}(S, A)$, which the driver is willing to trade off, and therefore written as $R_{\text{driver}}(S, A) = \underline{w}_{\text{driver}}^T \cdot \underline{\phi}(S, A)$. Note that $\underline{w}$ is the character of a driver and describes his behavior.

Each feature is a function $\phi_i(S, A) : S \mapsto [0, 1]$ and measures a preference for a specific goal. For example, $\phi_{\text{distance}}(S, A)$ measures whether the distance to the closest obstacles consistently remains beyond five meters and so calculates the preference to avoid a collision. If the driver cares about safety, the coefficient will be relatively higher for the distance feature.

Based on this setup, the question will be, for this particular driver, what values of α, β, and γ or generally what $\underline{w}$, characterize his utility function. This problem can be solved by the inverse reinforcement learning [9].

4.1 Preliminaries

4.1.1 Markov decision process

The Markov decision process consists of a tuple (S, A, T, γ, D, R), cite?:

1. a set of states S;
2. a set of actions A;
3. a state transition function $T : S \times A \mapsto P_S(\cdot)$ which gives a state distribution when an agent takes an action a in a state s. Or, denote $T(s, a, s') = T(s, a)(s')$ for the probability of making a transition from a state s to a state s' by an action a;
4. an initial distribution D such that $s_0 \sim D$
5. a discount factor $\gamma \in [0, 1)$; and
6. a scalar reward function $R : S \mapsto [0, R_{max}]$.

At each time t, an agent observes its state $s_t \in S$ and the set of possible actions $A(s_t) \subset A$. It chooses an action $a \in A(s_t)$ and receives a new state $s_{t+1} \sim T(s_t, a_t)$ and a reward $r_{t+1} = R(s_{t+1})$. **Figure 4-1** describes how this interaction happens between an agent and its environment.

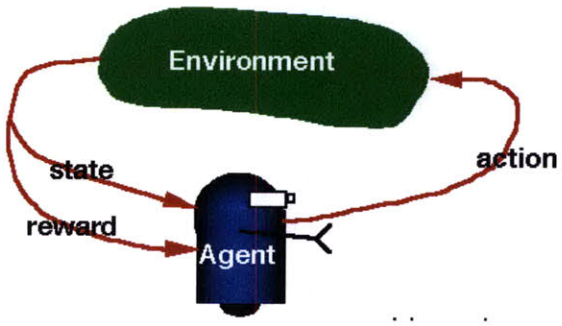


Figure 4-1: The interaction between an agent and its environment
Source: [4]

Note that the agent's preference can be fully specified by the reward function alone. The other information, MDP\R $= (S, A, T, \gamma, D)$, describes how the world, not the individual agent, behaves. The agent does not have control over the world; but, the information about the world is often directly measured or indirectly estimated from the result of the agent's actions.

4.1.2 Linear Approximation of Reward function

In the paper *Algorithms for Inverse Reinforcement Learning* [9], Ng states that even though infinite-state MDPs can be defined as being like finite-state MDPs, searching

the entire state space are often algorithmically infeasible. Rather, Ng approximates the reward as a linear function in large state spaces and solves Reinforcement Learning algorithms with liberalized reward. Specifically, let the true reward function be $R : \mathbb{R}^n \mapsto [0, 1]$, where $S = \mathbb{R}^n$, and assume that there are basic feature vector $\phi : S \mapsto [0, 1]^k$. Then, there is an approximated reward function $R^*(s) = w^* \cdot \phi(s)$, where $w^* \in \mathbb{R}^k$ and $||w^*|| \leq 1$ [1]. Typically, when the true reward function is unknown, the approximated reward function is also unknown.

Note that the linear approximation of state spaces only applies to a reward function and does not apply to a policy. The another important purpose of the linear approximation of state spaces is to emphasize that an agent trades off a reward function between various basic features. Instead, the state space of a policy is often discretized so that typical reinforcement algorithms can handle it. In our case, we also discretize the state space to find the policy; but we use basic features, such as Delta function, Gaussian function, to represent the components of a reward function.

4.1.3 Inverse Reinforcement Learning and Apprenticeship Learning

Unlike the reinforcement learning whose goal is to find a policy that maximizes the long-term reward with the (often full) knowledge of a reward function [4], inverse reinforcement learning is to determine the reward function given an MDP without a reward function, MDP\R, and an optimal policy π^* [10].

Apprenticeship learning is a subpart of inverse reinforcement learning. Specifically, we assume the ability to observe trajectories (state sequences) generated by an expert (in our case, the driver) starting from $s_0 \sim D$ and taking actions according to π_E. From these demonstrations, we calculate the approximated reward function $R_E = \underline{w}_E^T \underline{\phi}$ of the expert [1].

Most of the time, the coefficient $\underline{w}$ is an intermediate goal; it is used to derive an optimal policy that mimics the observed behavior of an expert. However, in our case, $\underline{w}$ serves a value to describe the preferences of a driver. In other words, as the coefficients are bounded $|\underline{w}| \leq 1$, the reward function naturally trades off linearly between basic features, thus indicating his preferences.

4.2 Simulation and Dynamics

In reinforcement learning, at each time or sampling, an agent interacts with the environment by perceiving states and performing actions. Thus, we need to define the realistic state and action space for simulation.

4.2.1 Car Model : State and Action

Broadly speaking, the state space consists of two parts; one part about the car itself, and the other part about the environment. The state about the car includes measurements such as vehicle speed and engine RPM. As the reinforcement learning focuses on the interaction with the environment, state about nearby environment includes much more:

1. current lane number
2. left/right lane changeable
3. distances to the front/rear vehicles in nearby lanes
4. speed of the front/rear vehicles in nearby lanes

As the driver does not need to care about all lanes, he should only focus on nearby (left/current/right) lanes. Thus, the state variables regarding the nearby lanes are limited to the three immediate lanes.

Unfortunately, not all information is suitable for simulation and easy to collect in a realistic fashion. Excluding some meaningless and hard-to-measure states, I have to use the following state variables: vehicle speed, left/right lane changeable, distances to the front/rear vehicles in nearby lanes, speed of the front/rear vehicles in nearby lanes.

The action space is much simpler, consisting of accelerator pedal position, brake pedal position, and lane change. Note that instead of steering angle, I have used lane change as an action. As it is very hard to realistically follow the procedure of changing lanes in simulation, I have assumed that the car can change lanes immediately if there is a room. Two pedal positions ranges from 0 to 100, whereas the lane change is either LEFT / STAY / RIGHT.

In the right side of **Figure 4-3**, an example of state and action and its graphics are shown.

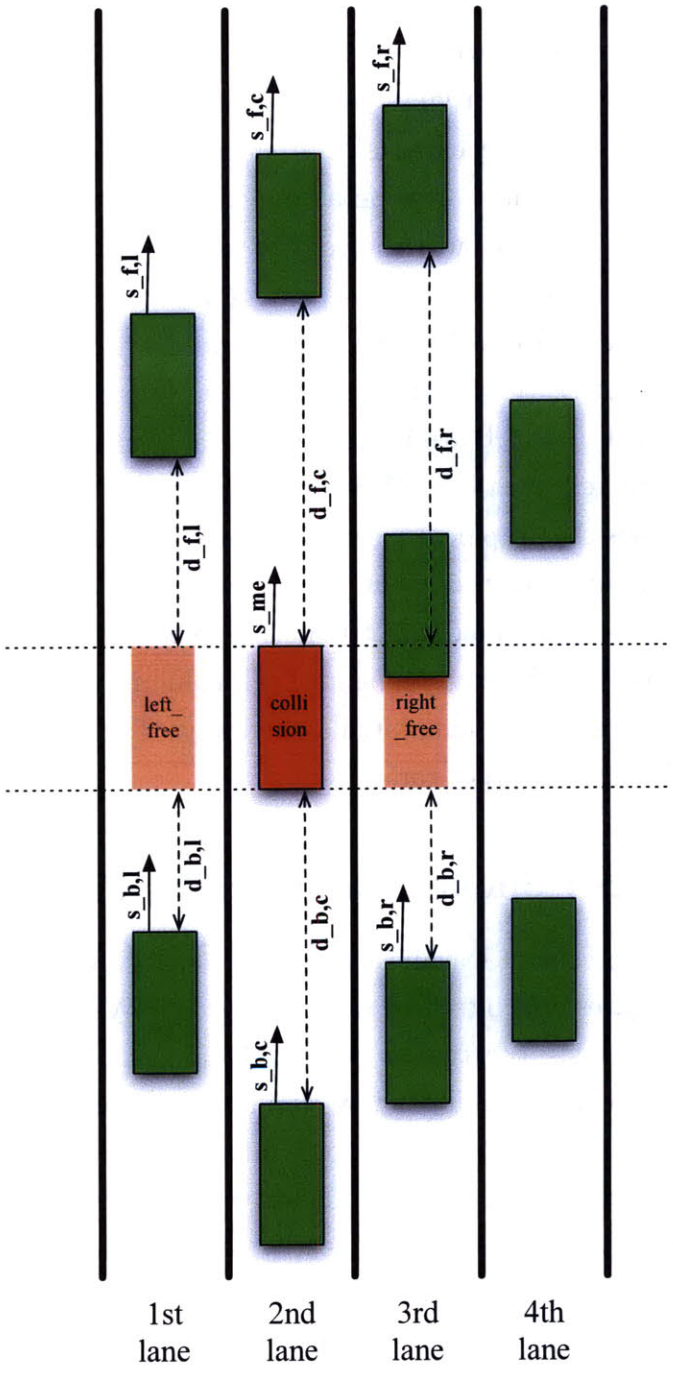


Figure 4-2: State

4.2.2 Simulation of Driving Environment

Using the state and action spaces defined above, we can simulate one person's driving. However, driving alone is meaningless in learning. Thus, we must first simulate the entire traffic environment, in which multiple cars are realistically and interactively driving. We chose a highway driving environment, so drivers can fully display their preferences without interfering externalities, such as traffic signals. For simplicity, cars do not enter nor exit; they drive on a long circular highway.

During simulation, cars follow a control law explained in *Congested traffic states in empirical observations and microscopic simulations* [11]. Each car has different parameters, including desired velocity $v0$, safe time headway T, Maximum acceleration a, Desired deceleration b, Acceleration exponent δ, Jam distance $s0$, Jam distance $s1$, and Vehicle length l, which are initialized at the startup. Typical values for the parameters are extracted from his JAVA implementation.

Figure 4-3 shows the graphical view of the simulator, in which roughly 100 cars are added in the highway environment. Four narrow columns in the left plot the part of the highway so that the entire traffic is viable, whereas middle column gives a close-up look of the nearby environment. Each column is connected other column and so all columns compose a long circular highway. The rightmost column shows the state and action of the target car.

4.2.3 Dynamics : Simulation of Nearby Environment

To solve inverse reinforcement learning problem, we need a world dynamics, in which an agent observes states and interacts through actions. We might use the simulator discussed in the previous subsection as a world dynamics. However, as the simulator models the entire driving environment by simulating all cars, each of which follows the designated control, it is not suitable for reinforcement learning.

First of all, our observation is local. The driver of a car cannot fully observe the entire situation; rather, he can only see nearby cars and decide action based on his limited observations. Thus, the dynamics should only involves nearby cars.

Second, most of the information available in the simulator does not reveal the preference of a specific driver. While simulating all cars in the environment is very costly, most of the information is useless for estimating a specific driver's behavior.

Third, the simulator requires large number of iterations until the policy iteration algorithm converges. Since the simulator consists of the states of all cars, the result of each iteraion significantly depends on randomnes, and thus the policy iteration

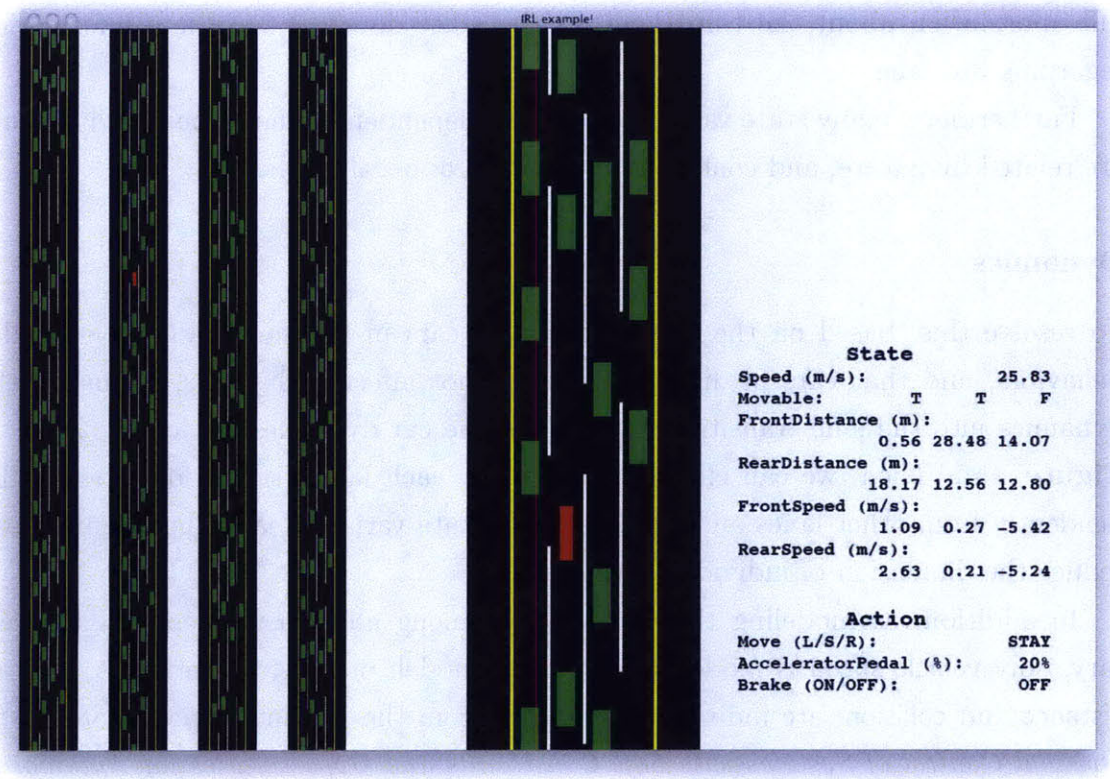


Figure 4-3: Simulator

algorithm requires excessive search of all cars until the algorithm finally converges to a stable result.

Rather, we need to make a dynamics model of one specific car, especially regarding the relationship between state and action. One straightforward way is to discretize the entire space and count the number of transitions. However, as the dimension of entire state space is nearly 10, learning an MDP naively was practically impossible. Also, there is an inevitable randomness, especially when the car changes its lane, as the previous state does not have any information about the newly observable lane. For example, when a car changes lane from 1st to 2nd, the dynamics cannot give full information about the third lane, as the state does not have any information regarding 3rd lane.

Furthermore, many state variables are not independent: vehicle speed and distance are related by nature, and controlled by accelerator pedal position.

Dynamics

To resolve this, based on the assumption that cars in the same lane have similar behaviors, and that cars in different lanes do not affect each other, we factor the dynamics into the lane transition models and the car dynamics model, as shown in **Figure 4-4**. Then, we can effectively learn the each lane's transition model independently from other lanes and from my car's state variables, and thus significantly reduce the dimension of individual dynamics.

In addition, as modeling the relationship among all measurements is unnecessary, only vehicle speed is modeled by Markov model; other state variables, such as distance and collision, are indirectly calculated from the predicted speed. *Stochastic Optimal Control of Systems with Soft Constraints and Opportunities for Automotive Applications* [5] explains the control based on Markov model.

Each lane's transition works as follows: the speed of the front car is modeled by a Markov Chain with 20 discrete levels, uniformly distributed between 15 and $35m/s$. The speed of the back car is modeled by another Markov Chain of two inputs: the speed of front car and the distance to the front vehicle with 20 and 10 discrete levels uniformly distributed 15 and $35m/s$ and $0m$ to $100m$. The transition probabilities are learned from multiple simulations, and shown in **Figure 4-5**.

Once the future speed of front and rear cars in each lane are chosen by Markov models, other state variables, such as distances and collision, are naturally calculated indirectly. Note that this approach cannot account for the case that a new car enters into the lane. Therefore, to simulate a new car entering, the dynamics also consider

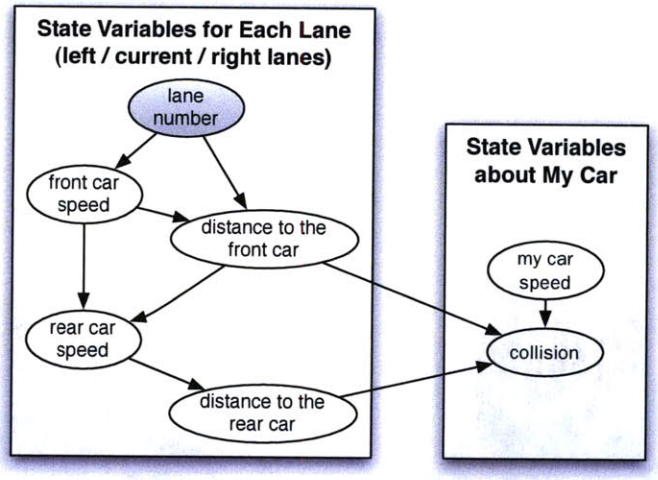
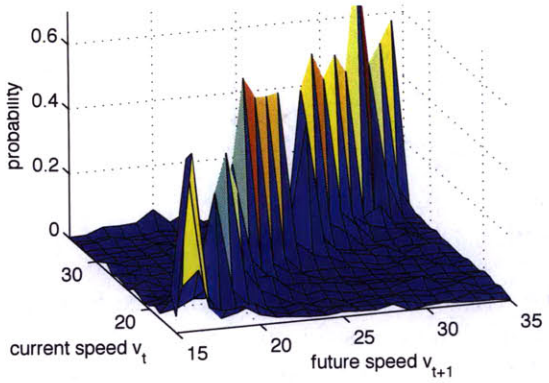


Figure 4-4: Factored Dynamics Model. The dynamics is factored into three lane transition models and my car's transition model. Each lane transition model includes Markov models for front / rear vehicle speed, and calculates front / rear distances and collision with the predicted future speeds.

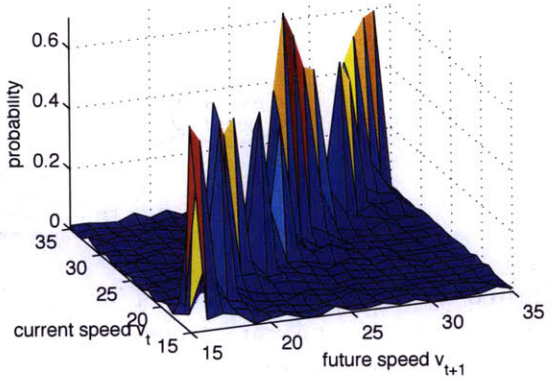
the probability of a new car entering given distance is required, and **Figure 4-5(d)** shows the simulated pdf of it.

Finally, the product of transition models of all lanes form the entire dynamics. Each transition model is used for predicting the next state of each lane, and the results are collected for the entire next state. Note that due to the independence assumption, a car's lane change might not be captured fully. For example, if a car disappears in one lane, it must move to another lane. However, due to the independence of lane models, this movement might not be captured.

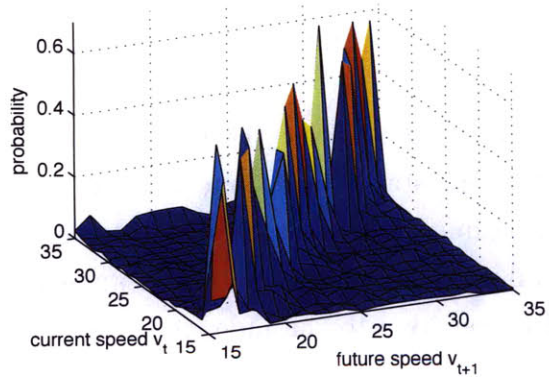
Note that this procedure does not consider the effect of an action. Rather, as the action only affect the state of the driver's car, its effect can be easily calculated independently from other cars. When there is a lane change, the lane's states are shifted in the opposite direction of lane change. As the dynamics does not have information about the newly observed lane, state variables for the new lane is generated randomly based on the statistics about the given lane. For example, suppose that a car was in the second lane and moved to the third lane. The state variables used to have data from the first to the third lane. However, after the lane change, the state variables now have data from the second to the fourth lane. Since dynamics did not have any data about the fourth lane before, it must be generated randomly.



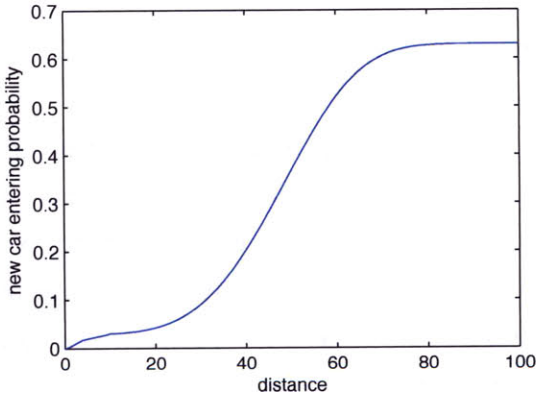
(a) Transition probabilities of the speed of the front car



(b) Transition probabilities of the speed of the rear car given the distance is $10 \sim 20m$



(c) Transition probabilities of the speed of the rear car given the distance is $70 \sim 80m$



(d) the simulated pdf of the entering probability given the distance

Figure 4-5: Statistics of the second lane transition model. Since the second lane is for high-speed driving, the front car accelerates until it reaches the average speed of the second lane, which is roughly $28m/s$, and slightly de-accelerates on the very high speed (a). Similar behaviors are observed for the rear car when the distance is far (c). However, when the distance is close, the rear car de-accelerates regardless of its current speed (b). In addition, when the distance is high, a new car is more likely to enter (d).

4.3 Features

Our goal is to find the reward function of one driver as a linear combination of other experts' reward functions. However, each expert's reward function is also characterized by a linear combination of basis functions. Both are essentially the same in the nature, and same algorithms can be applied to find them. To clarify, we call the basis functions as raw features, and each expert's reward function as an expert feature.

Raw features are used to capture the characteristics of the driver with respect to common measures, such as speed, whereas expert features are used to estimate the relative similarity of driver with respect to a given set of experts. Expert features are defined in terms of raw features.

4.3.1 Raw Features / basis functions

Among the variables from the state and action, four were selected to estimate the behavior of a driver: vehicle speed, accelerator pedal position, time to collision, and collision, where time to collisioin is calculated by distance to the front vehicle / vehicle speed. Other variables were omitted as they do not reveal driver's preferences.

Any functions of the selected variables can be basis functions. We use radial basis functions for vehicle speed, accelerator pedal posi tion, and time to collision and indicator function for collision. Several radial basis functions with different means, but with the same variance were used for each measurement. Such Gaussian membership function gives a smooth reward function and thus help to prevent sudden change of coefficients during policy update iterations and give a stable and better result. Also, as the reward function is not simply linear to the state variables, expert's preferences can be a non-linear function of the state variables.

$$\phi(s,a) = \begin{pmatrix} \exp(-\frac{||s-\mu_{s,1}||^2}{2\sigma_s^2}) \\ \exp(-\frac{||s-\mu_{s,2}||^2}{2\sigma_s^2}) \\ \cdot \\ \exp(-\frac{||s-\mu_{s,3}||^2}{2\sigma_s^2}) \\ - - - - - - - - \\ \exp(-\frac{||a-\mu_{a,1}||^2}{2\sigma_a^2}) \\ \exp(-\frac{||a-\mu_{a,2}||^2}{2\sigma_a^2}) \\ \cdot \\ \exp(-\frac{||a-\mu_{a,3}||^2}{2\sigma_a^2}) \\ - - - - - - - - \\ \cdots \\ collision \end{pmatrix}$$

4.3.2 Expert Features / composite functions

In addition to raw features, expert features are used to estimate how one driver's behavior is similar to the experts'. Unlike the basis functions, in which amy function can be used, expert features must be composites of basis functions; otherwise, we cannot use inverse reinforcement learning. The reward functions of different experts are used as expert features, and concatenated. Since the reward functions are linear combination of basis functions, we can easily calculate Φ by multiplying with concatenated weights of experts W.

$$\Phi(s,a) = \begin{pmatrix} R_{\text{speedy}}\ (s,a) \\ R_{\text{economical}}\ (s,a) \\ R_{\text{safe}}\ (s,a) \end{pmatrix} = \begin{pmatrix} w_{\text{speedy}}\ \cdot \phi(s,a) \\ w_{\text{economical}}\ \cdot \phi(s,a) \\ w_{\text{safe}}\ \cdot \phi(s,a) \end{pmatrix} = W^T \cdot \phi(s,a)$$

where

$$W = \begin{pmatrix} w_{\text{speedy}} & w_{\text{economical}} & w_{\text{safe}} \end{pmatrix}$$

4.4 Policies

Given a dynamics and a reward function, expressed as either raw or expert features, finding an optimal policy is also an essential part of reinforcement learning. Various algorithms have been proposed to solve this problem. Among them, we chose two

algorithms, one simple and one complicated, to find the optimal driving policy.

4.4.1 One-Step-Look-Ahead Policy

The simple one-step-look-ahead policy chooses the action that gives the most reward for the immediate future, and thus cannot take into account the delayed reward. Because it is easy to implement and test, the policy is useful for validating the algorithm. However, as it does not consider the future reward, it misses an important aspect of reinforcement learning.

4.4.2 Least-Squares Policy Iteration

Unlike the simple one-step-look-ahead policy, which only considers the immediate reward and so does not require value or policy iteration, more sophisticated algorithms often require explicit iteration over the entire state space. However, due to the high dimension of the driving state, such an approach is not feasible. Rather, a method similar to the one we used to simplify the reward functions can be adapted to find the policy as well. Least-squares policy iteration learns the state-action value function as a linear combination of basis functions and thus reduces the dimensionality of state spaces when searching for the optimal policy [6].

The basis functions for policy iteration ψ consists of various functions, such as radical, linear, and quadratic functions, to capture the complexity of a policy, whereas raw features mostly consist of Gaussian membership functions. Specifically, we concatenate raw features, linear action, and quadratic actions:

$$\psi(s,a) = \begin{pmatrix} \phi(s,a) \\ s \\ a \\ a^2 \end{pmatrix}$$

The detailed explanation for LSPI will be discussed on the next section.

4.5 Learning Algorithm

4.5.1 Apprenticeship Learning via Inverse Reinforcement Learning

The heart of apprenticeship learning is to find a policy, or a w, which satisfies the following equation through an iterative approach [1] :

$$\left|\mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)|\pi_E\right] - \mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)|\widetilde{\pi}\right]\right| = \left|w^T\mu_E - w^T\mu(\widetilde{\pi})\right| \quad (4.1)$$

$$\leq ||w||_2||\mu_E - \mu(\widetilde{\pi})||_2 \quad (4.2)$$

$$\leq 1 \cdot \epsilon = \epsilon. \quad (4.3)$$

μ_E is called the expert's feature expectation. Since we don't have direct access to the expert's policy, the feature expectation is often calculated empirically from the measured trajectories: $\hat{\mu}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{\infty}\gamma^t\phi(s_t^{(i)})$

$\mu(\pi)$ is called the policy's feature expectation and is calculated from trajectories simulated by Monte Carlo simulation.

Using the expert's feature expectation and the policy's feature expectation, we can find a policy that mimics the expert's observed behavior. At first, an arbitrary initialized policy is proposed, and the corresponding policy's feature expectation is calculated. Then, find w

The following algorithm 1 iteratively finds such a policy.

Algorithm 1 Apprenticeship Learning

1: initialize π^0 arbitrarily, compute $\mu^{(0)} = \mu^{(\pi^{(0)})}$, and set $i = 1$
2: **repeat**
3: $t^{(i)} = \max_{w:||w||_2\leq 1} \min_{j\in\{0,1,\ldots,n-1\}} w^T(\mu_E - \mu^{(j)})$, and
let $w^{(i)}$ be the value of w that attains this maximum
4: Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T\phi$
5: Compute (or estimate) $\mu^{(i)} = \mu^{(\pi^{(i)})}$.
6: **until** $t_i \leq \epsilon$

Source: *Apprenticeship Learning via Inverse Reinforcement Learning* [1]

4.5.2 Q learning

In practice, it is very rare that we have full knowledge of the model. In addition, as the simple case "exploitation vs. exploration" implies, we need to find an optimal policy by exploring the environment to obtain information and choosing actions [4].

The core of the algorithm is a simple value iteration update. It assumes the old value and makes a correction based on the new information. First, Q learning initializes the qualities as fixed values for all (s, a) pairs. Then, every time an agent performs an action and receives a reward with a new action, the algorithm updates the quality of the corresponding pair (s, a). The following equation explains how the new quality is recalculated :

$$\begin{aligned} Q(s_t, a_t) &\leftarrow \underbrace{Q(s_t, a_t)}_{\textit{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\textit{learning rate}} \times [\overbrace{\underbrace{R(s_{t+1})}_{\textit{reward}} + \underbrace{\gamma}_{\textit{discount factor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\textit{max future value}}}^{\textit{expected discounted reward}} - \overbrace{Q(s_t, a_t)}^{\textit{old value}}] \\ &\leftarrow Q(s_t, a_t)(1 - \alpha_t(s_t, a_t)) + \alpha_t(s_t, a_t)[R(s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)]. \end{aligned}$$

The learning rate determines to what extent the newly acquired information overrides the old information. A factor of 0 will make the agent learn nothing, while a factor of 1 would make the agent consider only the most recent information. The factor may be the same value for all pairs or may vary.

4.5.3 LSTDQ and LSPI

(this entire section)

Q learning gives a way to weigh delayed rewards while searching for an optimal policy. However, it does not generalize over large state and action spaces [4]. Michail G. Lagoudakis and Ronald Parr proposed an algorithm that learns the approximate state-action value function of a fixed policy (LSTDQ) using least-squares temporal-difference learning algorithm (LSTD) and iterates LSTDQ until an optimal policy is found (LSPI) [6].

Given observed samples (s, a, r, s') and basis functions ψ, LSTDQ learns an approximated Q-function based on samples (instead of ϕ and w, ψ and v are used to distinguish them from features and coefficients of a reward function):

$$\hat{Q}(s, a; v) = \sum_{i=1}^{k} \psi_i(s, a) v_i = \psi(s, a)^T v$$

The following **Algorithm 2** learns an approximated Q-function:

Algorithm 2 LSTDQ $(D, \psi, \gamma, \pi, R)$

$B \leftarrow \frac{1}{\delta} I$
$b \leftarrow 0$
for $(s, a, s') \in D$ **do**
  $B \leftarrow B - \dfrac{B\psi(s,a)(\psi(s,a) - \gamma\psi(s', \pi(s')))^T B}{1 + (\psi(s,a) - \gamma\psi(s', \pi(s')))^T B\psi(s,a)}$
  $b \leftarrow b + \psi(s,a)R(s')$
end for
$v \leftarrow Bb$
return v

Source: *Least-squares policy iteration* [6]

Once the linearized Q-function is found, we can determine the optimal policy that maximizes the future rewards using:

$$\pi^{(t+1)}(s, v) = \underset{a}{\operatorname{argmax}}\, \hat{Q}(s, a; v) = \underset{a}{\operatorname{argmax}}\, \psi(s, a)^T v$$

Note that LSTDQ only provides a method to learn an approximated Q-function of a fixed policy. Therefore, Least-Squares Policy Iteration, LSPI, iteratively performs the LSTDQ algorithm until the policy converges: **Algorithm 3**.

Algorithm 3 LSPI $(D, \psi, \gamma, \pi, R)$

initialize π^0 arbitrarily
repeat
  $\pi^{t+1} \leftarrow LSTDQ(D, \psi, \gamma, \pi^t, R)$
until $\pi^t \approx \pi^{t+1}$
return π^{t+1}

Source: *Least-squares policy iteration* [6]

4.5.4 Estimation with Expert Features

Apprenticeship learning gives the coefficient $\underline{w}$ of the reward function, describing the observed behavior as a linear combination of given basic features, which can be gaussian, for example. Using the observed trajectories for each of the driving types, and the apprenticeship learning algorithm, we can calculate the reward function of the driving type as a linear combination of the basic features. For the observed trajectory of any driver, it is possible to calculate the reward function with basic features.

However, the apprenticeship algorithm is not limited to basic features. Rather, we can find a reward function as a linear combination of driving types' reward function, which is also a linear combination of basic features. Then, the weight determines the preferences for the each driving type. **Figure 4-6** shows the basic flow of the whole algorithm.

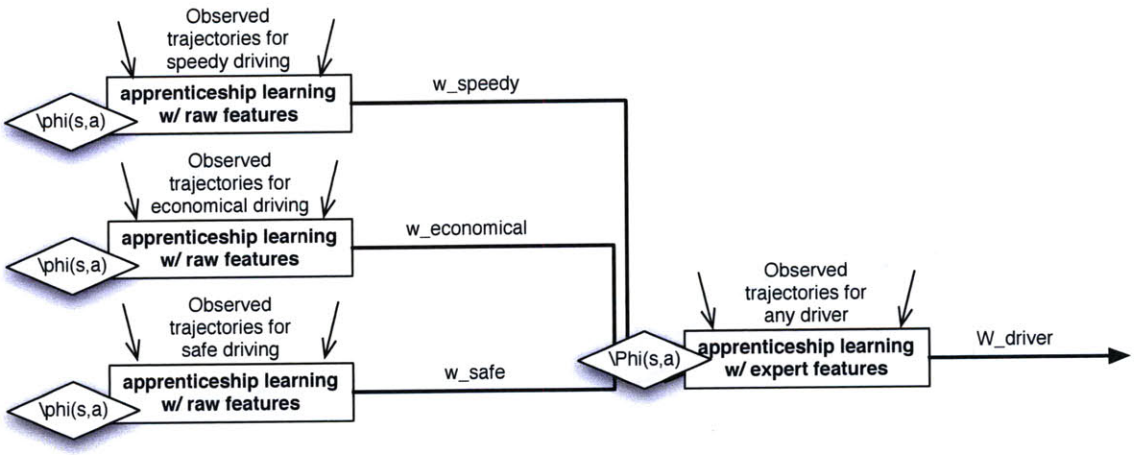


Figure 4-6: Flow of algorithm to estimate a driver's preference with expert features

Alternatively, we can find the reward function from basic features, and calculate the distance to each driving type.

4.6 Simulation Experiment

4.6.1 Setup

The entire algorithm was implemented in MATLAB, and tested with two policies.

We collected multiple trajectories of three different experts (speedy , economical , and safe) using the simulator explained in the previous section, and collected trajectories of a new driver whose preferences we want to estimate using the same simulator. Typically each trajectory consists of few hundred to one thousand samples, which is roughly one to a few minutes, as the sampling rate is $dt = 0.25$. For a stable result, we collected three trajectories for each expert. The dynamics was learned from the simulator as well. We generated trajectories of each car, moving autonomously, and used them to estimate the parameters of the dynamics.

4.6.2 Results of Simple Policy

Since the reward functions are in general very smooth, and steep delayed rewards do not exist in this domain, the simple policy gives a fairly good result. **Figure 4-7** shows the distance of current feature expectations to the expert's. After roughly 10 iterations, the inverse reinforcement learning algorithm finds the coefficient w_{type} of each expert. However, due to its simplicity, the one-step-look-ahead policy can only find a policy where $||\mu(\pi) - \mu_E|| \geq 10$.

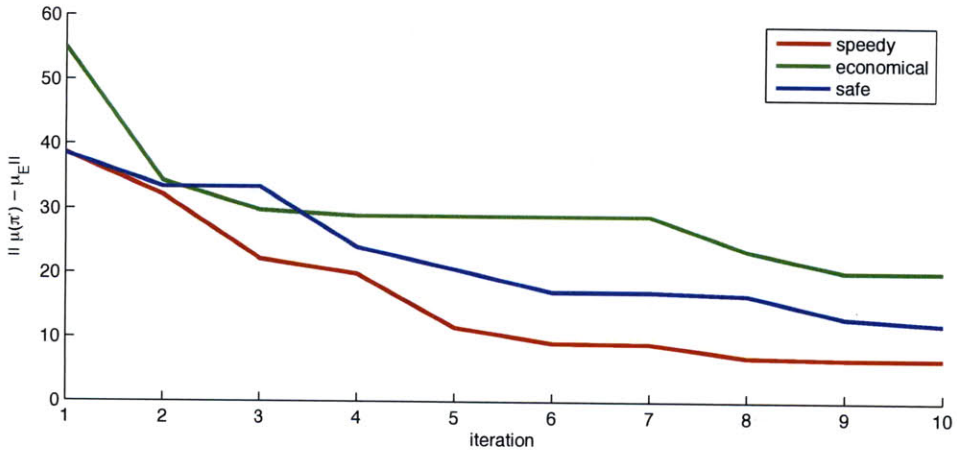
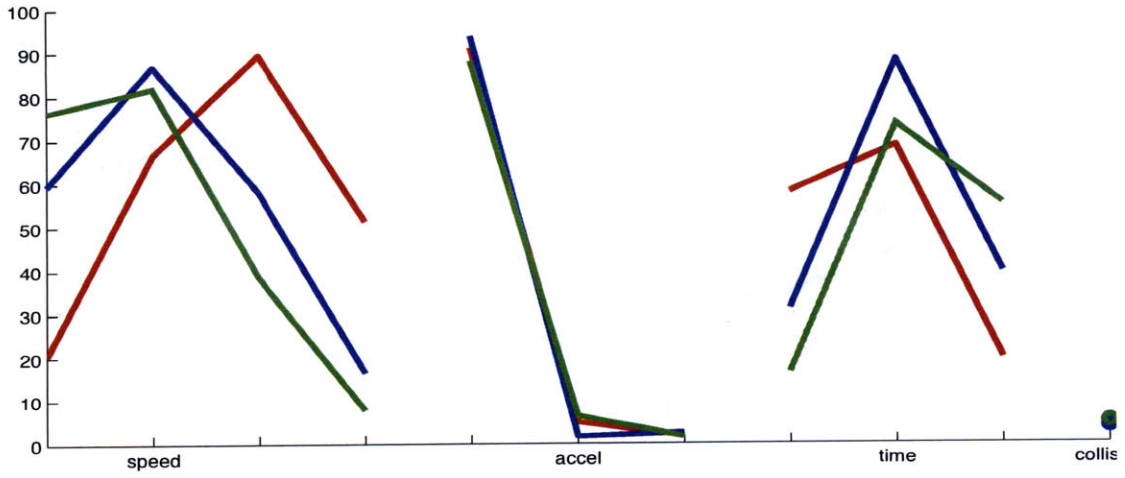


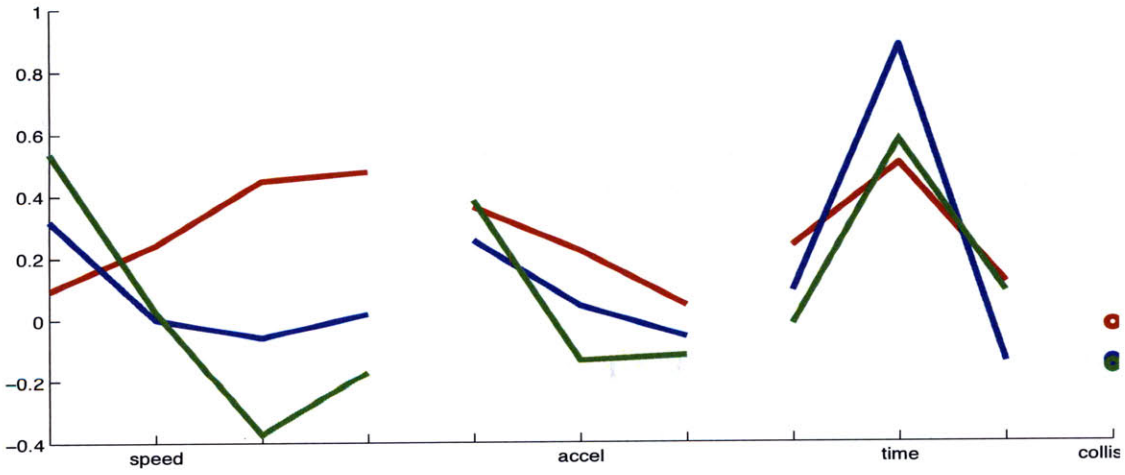
Figure 4-7: Distance to feature expectation as a function iteration. After roughly 5 to 10 iterations, the algorithm succesfully found the reward of each type, but the margin between $\mu(\pi)$ and μ_E remained large due to the simple policy searching algorithm.

Figure 4-8(a) gives the raw feature expectations of three experts. Different behaviors of experts were captured clearly. For example, a speedy driver remains in faster speed and low time-to-collision a lot, whereas a safe driver maintains high time-to-collision and low speed.

Figure 4-8(b) shows the calculated weights of each basis functions. As expected, speedy type prefers high speed, and tolerates low time-to-collision. Also, economical type shows notably low preferences in accelerating, and safe type prefers high time-to-collision.



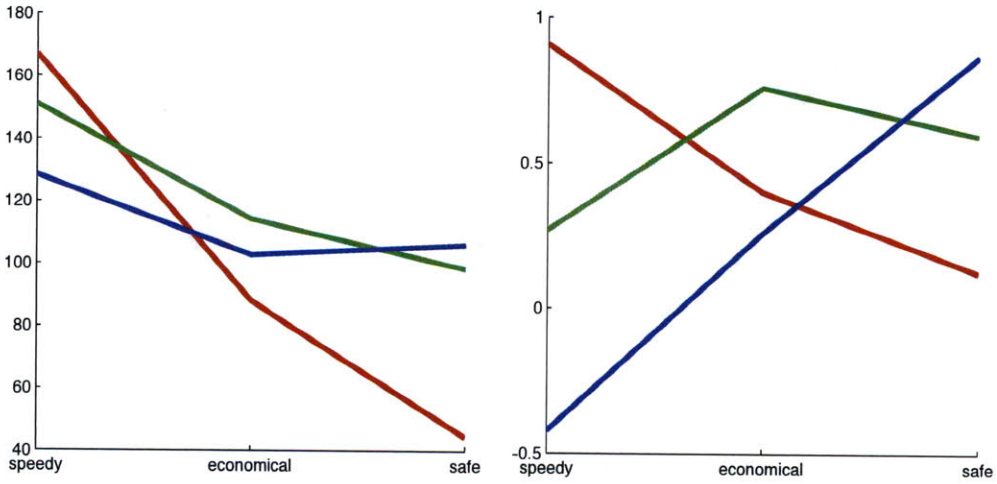
(a) feature expectations for three different experts



(b) features' weight for three different experts

Figure 4-8: Result of Simple Policy with Raw Features. Each expert's feature expection and coefficients of its reward function generally agrees with common sense.

Now, based on the weights of three experts, expert features can be calculated. and any driver's preferences can be estimated. Before moving forward, for a sanity check, we need to confirm that the calculated weights are valid. One fact is that each expert's trajectory must have strong preferences toward its own style. **Figure 4-9** verifies the claim. Each driving type achieves highest feature expectation in its own type, and is classified as its own type.



(a) feature expectations for three different experts

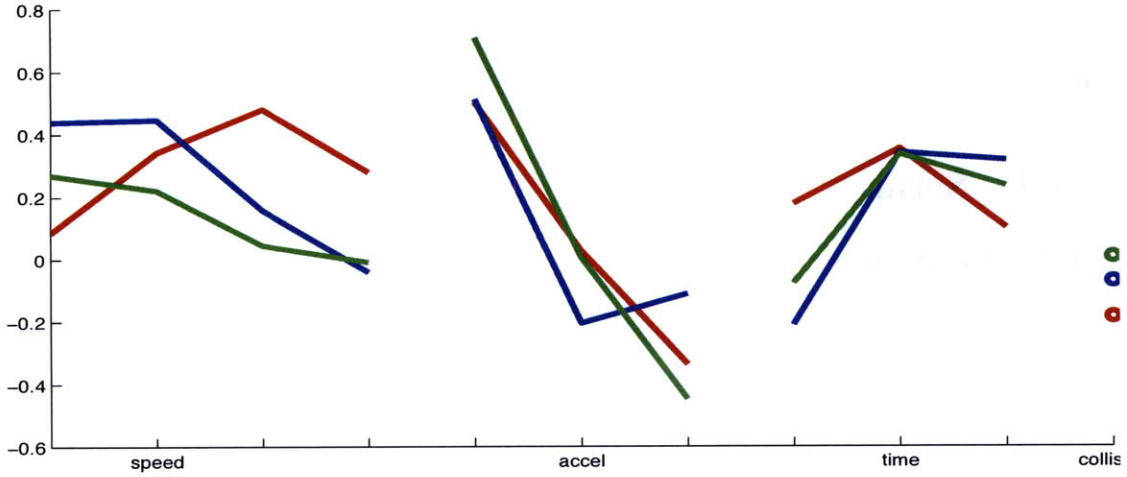
(b) features' weight for three different experts

Figure 4-9: Result of Simple Policy with Expert Features. Each expert's own trajectories must have high feature expectation on its own expert feature, and high coefficient for its own type, and all experts met this requirement.

4.6.3 Results of Least-Squares Policy Iteration

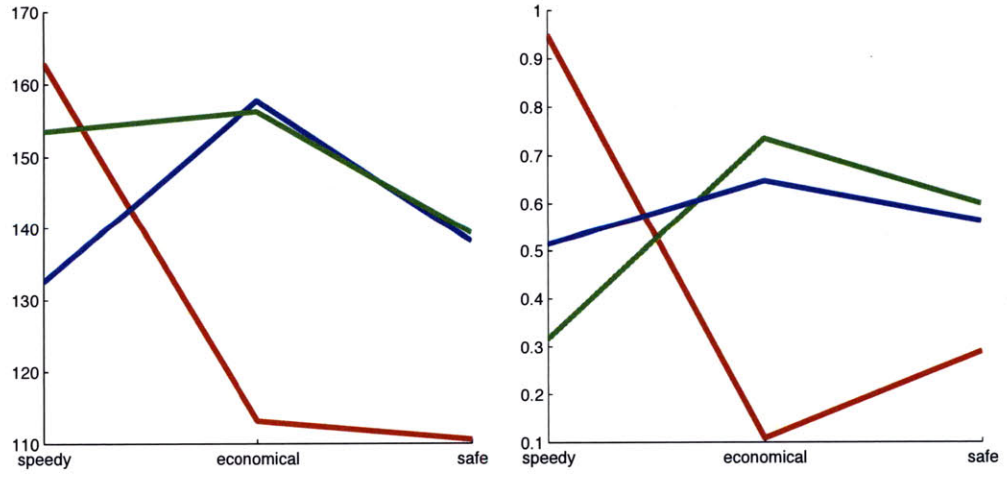
In general, LSPI gives a similar result to the simple policy, and **Figure 4-10(a)** shows the calculated weights of each basis functions. Major characteristics of each expert are well captured and coefficients are almost alike to the result of the simple policy, except the coefficients regarding collision are against the common sense: aggressive type least prefers collision.

However, LSPI could not distinguish economical and safe types in a sanity check, as shown in **Figure 4-11**. The feature expectations between two experts' trajectories are almost identical, and so the algorithm estimates safe type as economical .



(a) features' weight for three different drivers

Figure 4-10: Result of LSPI with Raw Features



(a) feature expectations for three different experts

(b) features' weight for three different experts

Figure 4-11: Result of LSPI with Expert Features

4.7 Realistic Experiment

For a more realistic experiment, we use Type-2 datasets. Because we assume driving decisions are made depending on the free space in front, Type-1 datasets cannot be used in this approach.

4.7.1 Setup

State and Action

Out of the available five measurements, vehicle speed and distance to the closest vehicle charaterize the state space, whereas acceleration pedal position, brake pedal position, and steering angle are actions.

Dyncamics

The data is collected by driving car in the real world. Thus, the dynamics should follow physical laws. However, it is difficult to precisely derive the dynamics, or the relationship, between states and actions, due to noise and measurement errors. Still, we can get a rough idea of how they should be related; for example, the distance should have a linear relation to the speed. Using this knowledge, we can formulate the dynamics by a simple steady state model:

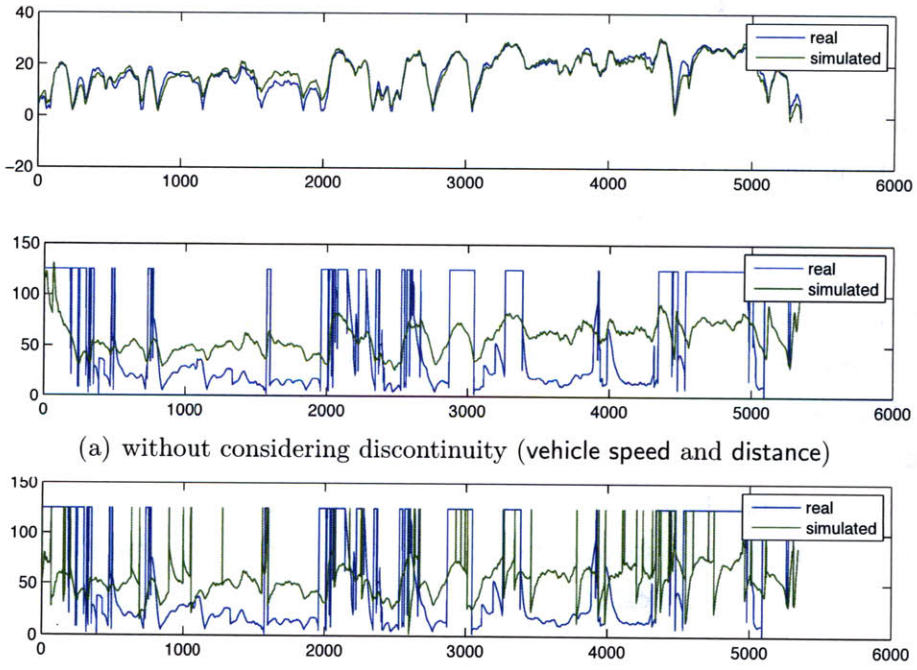
$$\begin{pmatrix} v_{t+1} \\ d_{t+1} \end{pmatrix} = \begin{pmatrix} \beta_v & \beta_d \end{pmatrix}^T \begin{pmatrix} v_t \\ d_t \\ a_t \\ b_t \\ s_t \\ a_t^2 \\ b_t^2 \\ s_t^2 \\ 1 \end{pmatrix} + \begin{pmatrix} \epsilon_v \\ \epsilon_d \end{pmatrix} \tag{4.4}$$

Based on the data, we can empirically calculate the coefficients of the dynamics by linear regression, as shown in **Table 4.1**. Once the coefficients are determined, we can simulate the world and **Figure 4-12(a)** show the measured states and simulated states. The simulation for vehicle speed performs well; the two lines have almost the same shape, with small errors. However, the simulation for distance does not reflect the reality accurately. In particular, the measured states often contain discontinuities

that the states fall into $d = 0$, which indicates that no vehicle is detected the radar sensor within the specification range. This discontinuity can be seen as a reasonable phenomenon, as the driver can change lanes, but cannot be fully simulated by the simple linear steady state model. Thus, the dynamics is modified to account for randomness that considers the discontinuity of distances, using a Markov model for the probability of a discontinuity given distance, as shown in **Figure 4-12(b)**.

Table 4.1: The coefficients of the dynamics.

$$\begin{pmatrix} \beta_v & \beta_d \end{pmatrix} = \begin{pmatrix} 0.9930 & 0.0074 \\ 0.0000 & 0.9591 \\ 0.0074 & -0.0099 \\ 0.0047 & -0.0188 \\ 0.0000 & -0.0009 \\ -0.0000 & 0.0002 \\ -0.0002 & 0.0002 \\ -0.0000 & -0.0000 \\ 0.0807 & 0.7487 \end{pmatrix}$$



(a) without considering discontinuity (vehicle speed and distance)

(b) with considering discontinuity (distance)

Figure 4-12: The measured states (green lines) and simulated states (blue lines). The x axis is time, $(50 = 1sec)$, and the units for the y axis are m/s and m.

4.7.2 Results of Simple Policy

Unlike the simulation experiment in which the simple policy performs well, the real experiment cannot achieve a good result with the simple policy. **Figure 4-7** shows the distance of current feature expectations to the expert's. After roughly 6 iterations, the inverse reinforcement learning algorithm converges, but can only optimize a policy where $||\mu(\pi) - \mu_E|| \geq 100$.

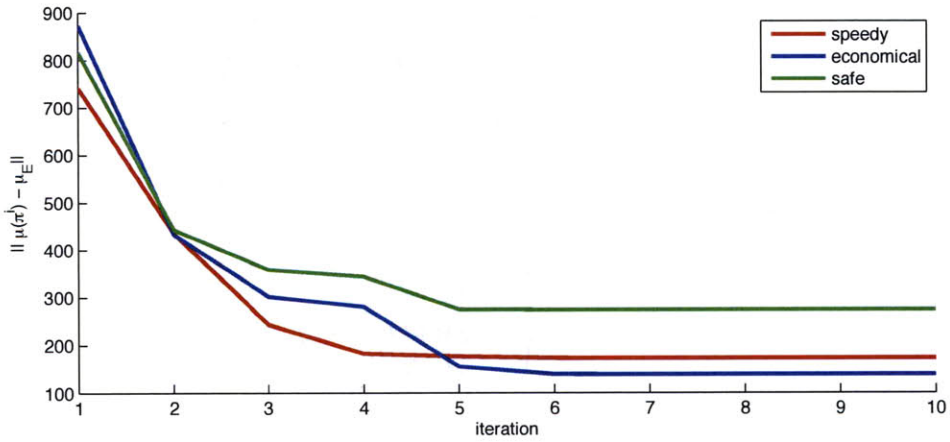
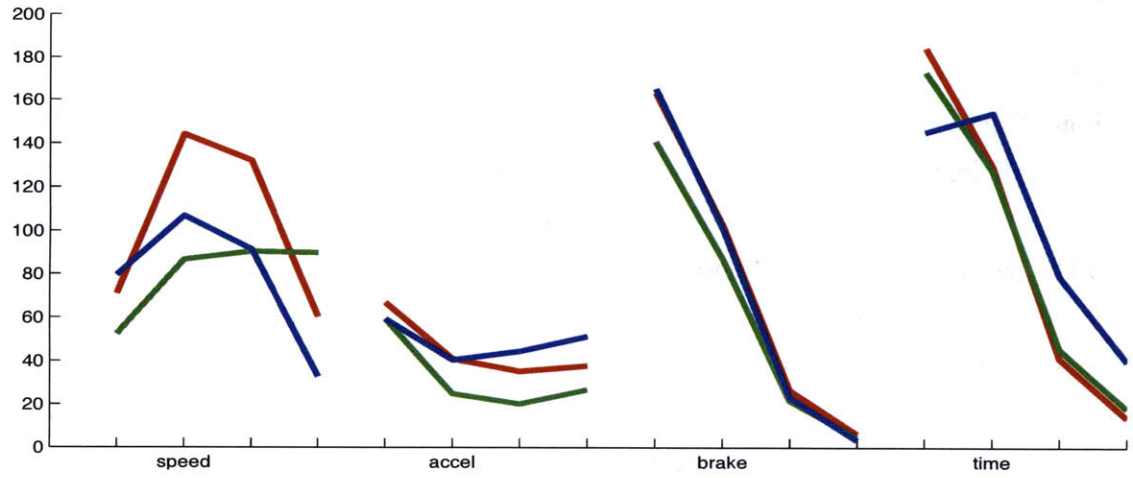


Figure 4-13: Distance to feature expectation over iteration

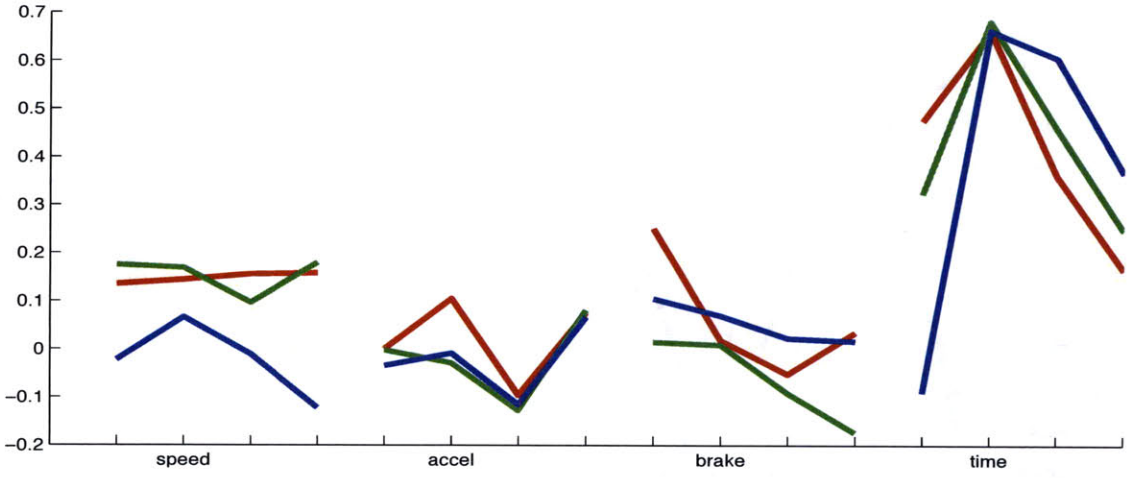
Figure 4-14(a) gives the raw feature expectations of three experts. The distinguishable behaviors of the experts are generally captured: speedy type prefers high speeds more than other types, economical type uses the accelerator the least, and safe type strongly prefers high time-to-collision. However, the difference in the realistic experiment is much more subtle than in the simulation experiment.

Figure 4-14(b) shows the calculated weight of each basis function. Notable facts are: speedy uses the accelerator and the brake often and tolerates low time-to-collision, economical type tends not to use the brake and the accelerator, and safe type strongly prefers low speed and high time-to-collision. In addition, the preferences regarding speed are similar between speedy and economical , and those regarding accelerators are similar between economical and safe . This tells us that an economical driver wants to drive in a similar speed to a speedy driver while not using the accelerator and the brake.

Table 4.2 shows the expert feature expectations and corresponding estimation result of Type-2 datasets, as a sanity check. Except for one dataset, all datasets are classified as their own types.



(a) feature expectations for three different drivers



(b) features' weight for three different drivers

Figure 4-14: Result of Simple Policy with Raw Features

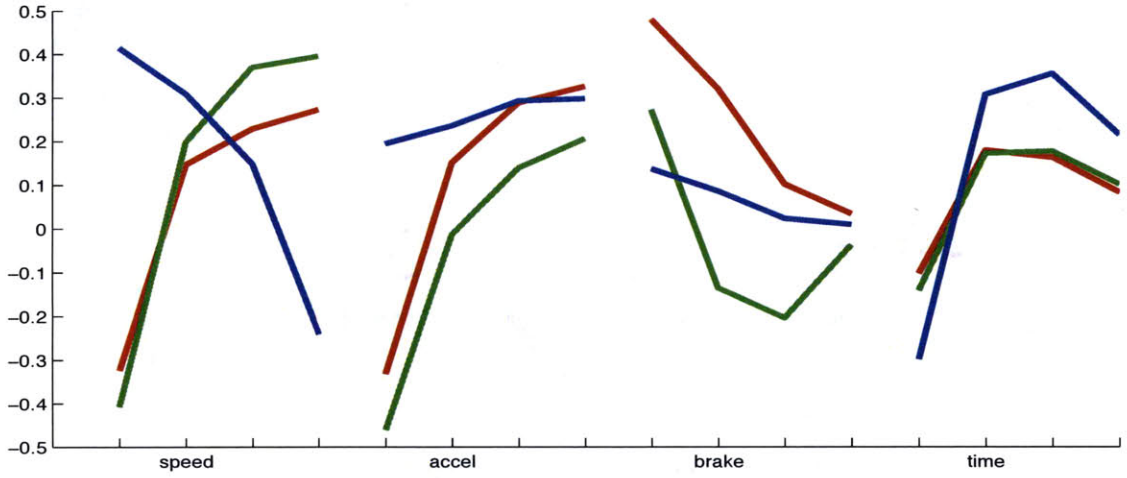
Table 4.2: Result of Simple Policy with Expert Features. Out of 8 datasets, 7 are successfully classified as its own type. speedy and economical types are often confused.

driver	*type*	*expert feature expectations*			*weights*			*correct*
		speedy	economical	safe	speedy	economical	safe	
driver1	economical	278	221	137	0.52	0.54	0.66	X
driver1	safe	322	268	187	0.53	0.56	0.64	O
driver2	speedy	286	218	109	0.69	0.58	0.42	O
driver2	speedy	289	224	125	0.63	0.57	0.52	O
driver2	speedy	313	239	123	0.70	0.58	0.41	O
driver2	safe	273	215	157	0.43	0.44	0.79	O
driver3	economical	266	209	91	0.66	0.68	0.31	O
driver3	safe	272	212	164	0.41	0.40	0.82	O
correct rate								7 / 8

4.7.3 Results of Least-Squares Policy Iteration

LSPI performs a decent job on estimating with raw features and **Figure 4-10(a)** shows the result. Notable properties of each expert are in general estimated; however, the coefficients are slightly different from those of the simple policy. Specifically, LSPI gives much smoother coefficients, while the simple policy gives a steep high coefficient to the second time feature and and irregular coefficients to the accelerator features.

As the simple policy does not learn the state-action value function, it often chooses extreme actions, for example, either no acceleration at all or full acceleration, to receive immediate rewards and thus have steep or irregular coefficients. However, thanks to the linearized Q-function, LSPI chooses appropriate actions for various states and achieve smother coefficients.



(a) features' weight for three different drivers

Figure 4-15: Result of LSPI with Raw Features

However, LSPI performs worse in a sanity check, as shown in **Table 4.3**. It fails to estimate 3 out of 8 datasets, especially between speedy and safe types. Also, it fails to estimate *driver3*'s behaviors at all. Note that both *driver3*'s trajectories are estimated as speedy . However, as LSPI requires intense learning and large number of datasets, if we add enough new datasets for safe type and *driver3*, the algorithm should be able to capture their behaviors eventually as well.

Table 4.3: Result of LSPI with Expert Features. Out of 8 datasets, 5 are successfully classified as their own types. The algorithm often fails to estimate between speedy and safe types, and *driver3* correctly.

driver	*type*	*expert feature expectations*			*weights*			*correct*
		speedy	economical	safe	speedy	economical	safe	
driver1	economical	177	94	110	0.56	0.82	-0.13	O
driver1	safe	198	101	219	0.57	0.44	0.69	O
driver2	speedy	169	76	150	0.70	0.71	0.09	O
driver2	speedy	165	69	157	0.53	0.43	0.73	X
driver2	speedy	203	108	168	0.68	0.60	0.43	O
driver2	safe	162	51	169	0.16	0.38	0.91	O
driver3	economical	128	57	103	0.74	0.54	0.40	X
driver3	safe	190	64	182	0.73	0.42	0.54	X
correct rate								5 / 8

4.8 Discussion

Inverse reinforcement learning solves some limitations of Bayesian hypothesis testing. Instead of classifying the driving behavior as one of few types, the algorithm can estimate the behaviors in a space of types, which is more informative and meaningful. Also, since we do not assume any specific form for the distribution nor the reward, the algorithm can fully utilize all the information possible from measurements and is not restricted by strong assumptions that we have to used in previous chapters.

However, still a few limitations exist.

First of all, the approach is limited in the situation in which dynamics is fully learnable. However, due to the hardness of predicting driving situation, and since the dynamic often changes for various reasons, such as traffic, road condition, road type, it is difficult to learn reasonable dynamics. We could make reasonable dynamics for simulations, with few assumptions, such as independent lanes, highway environment, but often they are not valid in real life.

Second, the algorithm is very expensive, thus hardly online. All algorithm were implemented in MATLAB and tested on a fast multicore machine. However, each run of the algorithm takes more than few minutes. Most of the running time is spent for updating policies and simulating the dynamics to find the policy that mimics the expert; however, such procedure is inevitable and so hard to run the algorithm in a computer with limited capacity.

Third, the sophisticated least-squares policy iteration algorithm only achieves similar performance to the simple one-step-look-ahead policy. While LSPI learns the

general state-action value function and thus theoretically able to weigh not only the immediate rewards but also delayed rewards, its performance is not far different from the simple policy, which only considers the immediate future. However, LSPI worsens the learning time significantly, comparing to the simple policy which does not requires any learning of a state-value function.

Note that as the raw features consist of Gaussian membership functions on various measurements, they are generally smooth and thus no steep delayed reward exists (except the collision), and so discounted future rewards are smooth as well. In such functions, weighing the future rewards becomes less important, or choosing extreme actions may serve as an alternative to consider the delayed rewards, thus making two policies perform similarly.

In addition, a real driver does not often think about states of several seconds later while driving. Rather, he constantly accelerates or de-accelerates until the car reaches his designated speed, or changes the lane whenever it is allowable. Note that when a driver intends to change lanes, he should speed up or down to a similar speed to a designated lane and then change lanes gradually. In this scenario, LSPI may help to capture this behavior. However, to make the simulation simpler and due to the limited measurements, changing lane is done immediately in the simulation and the lane change action is not possible in the realistic experiment; then, such a maneuver cannot be captured in our experiment. Thus, I believe that a real driver's policy is actually similar to the simple policy, not the complicated policy.

Apart from these limitations, the algorithm remains as the best option for modeling the preferences.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we investigated three different methods to model drivers' preferences: supervised learning, Bayesian hypothesis testing, and inverse reinforcement learning. While the approaches and performances vary, all method were successfully able to determine driver's preference.

Based on the features of a window, supervised learning can classify each window into one of two classes. However, since the proposed approaches were limited to the binary classification, since it only classifies on a per-window basis, the algorithm is unstable and cannot detect changes of the driving type. In addition, the decision rule was too sensitive to the training data, and thus cannot capture the general rule for all drivers.

Rather, Bayesian hypothesis testing maintains the beliefs for each type and update the beliefs using the new data. By considering both the likelihood of the new data and the priors, the algorithm classifies the driver's preference with not only the current behavior but also the history. The models are initialized to capture the general decision rule, and can be updated using a specific driver's data online. Thus, the algorithm is more stable and can capture the generic decision rules while possible to be adjusted to specific rules. However, the algorithm relies on the strong assumptions that sampling windows are independent given the type and probability distributions of the measurements are independent each other, which are hardly true.

Inverse reinforcement learning, on the other hand, finds the reward that most explains the driver's behavior through simulation. By finding the experts' rewards with raw features (basis functions) and the driver's reward with expert features (composite functions), the algorithm estimates the driver's preference in a space of driving

types, and gives most meaningful and stable estimation. In addition, as the algorithm performs explicit simulation and intense calculation, it requires understanding of the dynamics and expensive resources. However, it is difficult to fully simulate all possible driving situations, and expensive to run the algorithm online.

5.2 Future Work

While the work has been able to apply various machine learning techniques for modeling powertrain preferences, further studies can be taken to further improve them.

Further approach include to choose models with respect to the driving situation. Currently, both Bayesian hypothesis testing and inverse reinforcement learning share same model for all driving environments. However, the criterion should be different for different driving environments (such as highway, traffic, city) and adjust models accordingly. Note that the EM algorithm was used to adjust models with respect to the driver's intentions, not the environment. To resolve this, hierarchical models can be introduced to consider both driving environments and drivers intentions. The algorithm not only maintain the beliefs for driving types, but also the beliefs for environment, and thus performs better in various environments.

The work, in general, lacks some theoretical background. Rather than being based on rigorous theory, most of the ideas came from scrutinizing the dataset and from applying trial and error methods. For example, we did not justify the type of probability density function used in Bayesian hypothesis testing. Further, the features used in inverse reinforcement learning were not chosen for concrete reasons. Therefore, more clear reasoning of the work is required, and more thorough understanding of the driving behaviors might help to improve the algorithms.

In addition, the thesis only focuses on estimating the preferences. Once we can do basic estimation of the driver's preference, we will consider:

- Adding active diagnosis actions, like asking questions. We'd have to include a model of the cost (in annoyance and decreased safety) of asking questions, perhaps depending on an additional estimate of the driver's current cognitive load.

- If, through asking questions, we find that the driver's stated preferences are at variance with their apparent driving strategy, then the system should make suggestions about how to improve. This requires solving a decision problem to

decide, given the estimated utility function for the user and the current driving situation, what actions to recommend.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*, volume 69. ACM Press, 2004.

[2] Jeff Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, 1998.

[3] Polina Golland and Gregory W. Wornell. 6.437 inference and information. 6.437 class notes, 2009.

[4] LP Kaelbling, ML Littman, and AW Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4(237-285):102–138, 1996.

[5] Ilya V Kolmanovsky and Dimitre P Filev. Stochastic optimal control of systems with soft constraints and opportunities for automotive applications. *IEEE International Conference on Control Applications*, pages 1–6, May 2009.

[6] M Lagoudakis and R Parr. Least-squares policy iteration. *The Journal of Machine Learning ...*, Jan 2003.

[7] Tomas Lozano-Perez. Interim report to ford motor corporation. 2008.

[8] Thomas P. Minka. Estimating a gamma distribution, 2002.

[9] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

[10] Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 101–103. ACM Press, 1998.

[11] M Treiber, A Hennecke, and D Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, Jan 2000. selected control.

[12] Finn Tseng. Edas proactive performance mode advisory system. 2009.