College of Mechanical Engineering

Report of course design

Course Name Intelligent vehicle and autonomous driving

Design Topic Research on Driving Style Representation

Major class Artificial intelligence

Student number 58121124

Name Boyan Zhang

Instructor Keke Geng

score

December 30, 2023

Contents

1 Introduction

1.1 Background

The driving style recognition of intelligent vehicles refers to the technology that uses sensors and algorithms to extract features from driving data to classify and identify drivers' driving habits and behaviors. The purpose of driving style recognition is to ensure vehicle safety, improve vehicle fuel economy, and improve driver comfort and satisfaction. Driving style recognition can also provide a reference for the planning and decision-making of smart cars, enabling smart cars to adapt to different driving environments and scenarios, as well as effectively interact with other vehicles and human drivers.

The main challenges of driving style recognition are the diversity and complexity of driving data, as well as the multi-dimensional and dynamic nature of driving style. Driving data includes the motion state of the vehicle, such as speed, acceleration, steering Angle, etc., as well as the external environment of the vehicle, such as road conditions, traffic signals, surrounding vehicles and pedestrians. Driving data is influenced by the driver's personal characteristics, such as gender, age, driving age, psychological state, etc., as well as driving tasks, such as destination, route, time, etc. Driving style is a comprehensive reflection of the driver's driving habits and behaviors, with multiple dimensions, such as safety, efficiency, comfort, courtesy, confidence and so on. Driving style also changes with different times and situations during the driving process.

In order to solve the problem of driving style recognition, researchers have proposed different methods, which can be mainly divided into two categories: rule-based method and data-based method. The rule-based approach is to define some driving style indicators and thresholds based on expert knowledge or experience, and then judge the category of driving style according to the matching degree of driving data with these indicators and thresholds. The advantage of the rule-based approach is that it is simple and intuitive, easy to understand and implement, but the disadvantage is that it is difficult to cover all driving situations, and it is sensitive to the choice of parameters, which requires a lot of debugging

and verification. The data-based approach uses data mining technologies such as machine learning or deep learning to automatically learn the characteristics and patterns of driving styles from driving data, and then classifies and recognizes driving styles according to these characteristics and patterns. The advantage of data-based approach is that it can adapt to different driving data and scenarios, and has high accuracy and generalization ability, but the disadvantage is that a large amount of labeled data is required, and it is difficult to explain and verify the internal logic of driving style.

1.2 Course design purpose

Base on the situation, I try to find some paper on driving style identification. Miyajima et al. employed car following patterns and pedal operations (i.e., gas and break) as two important sources of data for the driver identification task. In terms of car following patterns, the paper suggests to use the distance to the following car and velocity, map them to a 2D space, and then use a Gaussian Mixture Model(GMM) to obtain an optimal velocity model specific to each driver. Fung et al. proposed to use acceleration and deceleration events extracted from naturalistic driving behavior for driver identification. Extracted events from driving data were characterized using five groups of attributes that are duration, speed, acceleration, jerk, and curvature (13 attributes in total). The authors used a dataset of 230K hours of driving data collected from 14 elderly drivers, which comprised 17K acceleration and 17K deceleration events. An accuracy of 50% was achieved to classify test trajectories taken from 14 drivers by employing a linear discriminative analysis (LDA) model. Fugiglando et al. introduced driving DNA, where it covers four aspects of driving, namely cautious driving (measured by braking data), attentive driving (measured by turning data), safe driving (measured by speeding data), and fuel efficiency (measured by RPM data). These aspects were measured per driver per trip, and then average of scores per multiple trips represented a driver's score. The idea was tested on a collection of 2, 000 trips collected from 53 drivers based on a wide scenario of road types and open traffic conditions. Their analysis showed the uniqueness of behavior of different drivers when represented by the four aspects. However, no quantitative analysis nor results were presented in the paper to show how accurately one can predict a driver's identity from her driving data. Dong et al. presented a deep-learning framework to learn driving style, which used GPS data to encode

trajectories in terms of statistical feature matrices. Then, using two deep-neural-network models, one CNN and one RNN, driver identity is predicted for a given trajectory. Chen et al. proposed an auto-encoder-based deep neural network model for driver identification. They use a negatively-constrained auto-encoder neural network model is used to find the best window size when scanning through the telemetry data and another deep negatively-constrained auto-encoder neural network is employed to extract latent features from telemetry data. To perform classification, some off-the-shelf classifiers were used that are Softmax, Random Tree, and Random Forest. Ezzini et al utilized a set of tree-based models for driver prediction by using equipped vehicles with a variety of sensors and cameras installed inside and outside the vehicle, as well as information coming from the CAN-bus.

Finally, I was interested in the paper called “Driving Style Representation in Convolutional Recurrent Neural Network Model of Driver Identification”. Therefore, I will reproduce the code and analyze the experimental results according to this paper.

2 Course design content

2.1 Theoretical analysis

First, the data used in this research is a large, private dataset provided by an insurance company. Data was collected in Columbus Ohio, from August 2017 to February 2018, using designated devices connected to the OBD-II port of vehicles which decode CAN-bus data. Additionally, each device encapsulates several sensors, including GPS, accelerometer, and magnetometer. The data collection rate is one record per second (i.e., 1Hz). In total, the dataset includes about 4, 500 drivers and 836K trajectories.

Table2.1.1 1: Details on trajectory data

Drivers	Trajectories	Total Travel Time	Total Travel Distance	P50 of Duration	P50 of Distance
4,476	835,995	221,895 hours	11,174,400 km	11 min	6 km

Secondly, the paper used several data sampling strategies to limit the spatial similarity between trajectories and obtain a diverse set of trajectories for each driver. Also, by studying the importance of different attributes to explore driving style, it showed that the greater

discriminative power of a selected set of attributes coming from CAN-bus in comparison to the GPS and other sensory data. Thirdly, to perform driver identification via encoding driving style, the paper presented a deep-neural-network architecture, named D-CRNN, which combines several components, including convolutional neural network (CNN), recurrent neural network (RNN), and fully connected (FC) component. Using the CNN component, the model can derive semantic information about driving patterns (a turn, a braking event, etc.). Using the RNN component of the architecture, the model can capture temporal dependencies between different driving patterns to encode behavioral information. Finally, the FC uses the encoded behavioral information to build a model that properly predicts the driver's identity when given a trajectory as input.

2.2 Design scheme

First, I will process the original data set of the paper and convert the trajectory into feature coding; Next, I will build the corresponding gradient enhanced decision tree according to the features; Then, according to the paper, the loss function is written and the model is trained by backpropagation. Finally, the trained model is tested on the test set to obtain the performance of the model.

2.3 Experimental process

First, I pre-process the raw data set. The data used in this paper is a large, private dataset provided by an insurance company. The dataset includes 4,500 drivers and 836K trajectories. According to the paper, to ensure the quality of this dataset, I remove the first and the last two minutes of each trajectory and then set every trajectory's time between 10 and 30 minutes. Last, I remove those trajectories with any missing attribute to preserve consistency in data.

Secondly, given a trajectory $T = \langle p\_1, p\_2, ..., p\_n \rangle$, I first partition that to smaller sub-trajectories (or segments) by a window of size L, with a shift of $L/2$. The overlap between neighboring segments helps to prevent information loss during the partitioning. I choose L=256. After partitioning T to a set of segments $S\_T = \{s\_1, s\_2, ..., s\_k\}$, where $k=2n/L-1$, the next step is to generate a feature map (matrix) for each segment $s \in S\_T$ which encodes several attributes for each data point. Potentially, I use the following attributes to describe

each data point: (1) Speed, (2) Acceleration, (3) GPS_Speed, (4) GPS_Acceleration, (5) Angular_Speed, (6) RPM, (7) Head, (8) AclX, (9) AclY, and(10) AclZ. These are the basic features. Then, I transform a basic feature map to an aggregate feature map which encodes more high-level driving data instead of point-wise, low-level data, and also deals with outliers.

Thirdly, I build the model according to the DRCNN network structure proposed in the paper, which is composed of input layer, CNN layer, RNN layer and full connection layer, and the loss function uses cross-entropy function. The specific structure of DRCNN is as follows: for CNN layer, it includes two convolutional layers, each followed by a max-pooling to downsample. The activation function for both convolutional layers is ReLU and I use a dropout probability of 50% after each pooling layer; for RNN layer, it has two layers of GRU cells, each of layer has 100 GRU cells with a dropout probability of 50%;for full connection layer, it includes two layers. The first layer has 100 hidden neurons and uses sigmoid function as the activation function. The output of the first layer need batch normalization and dropout with 50% probability. The output of the second layer's is the probability values for different class labels and use softmax function to make the final prediction.

Fourthly, I use the processed dataset to train the model which I build. After training, I use the trained model to predict the result on the test set. Also, I used the model gradient Enhanced decision Tree (GBDT) which is the paper used for comparison, because GBDT is known to be effective for the task of driver identification. The model is built directly using the paper source code. Here the paper uses the set of hand-crafted features as input. This set includes 321 features. The paper also modified the original set of features by using the following basic features: Speed, Acceleration, Acceleration_Change, RPM, RPM_Change, and Angular_Speed.

Finally, I tested the two trained models separately on the test and got two results from the DRCNN and GBDT model. For the two results, I use the accuracy which is provided by the paper. The accuracy is as followed:

$$Accuracy = \frac{\sum_{i=1}^{n} 1_{(prediction(T_i)=d_i)}(T_i)}{n}$$

2.4 Results analysis

The training process is as follows:

Figure 2.4.1

The final DRCNN model's training result is as follows:

Figure 2.4.2

The result of DRCNN on the test set is as follows:

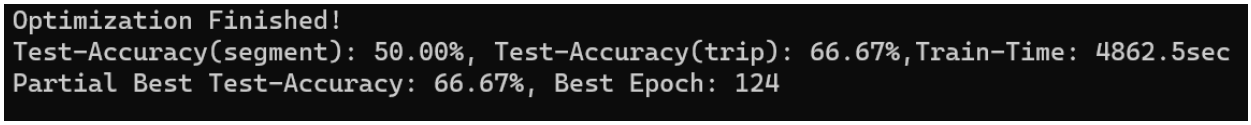

Figure 2.4.3

The result of GBDT on the test set is as follows:

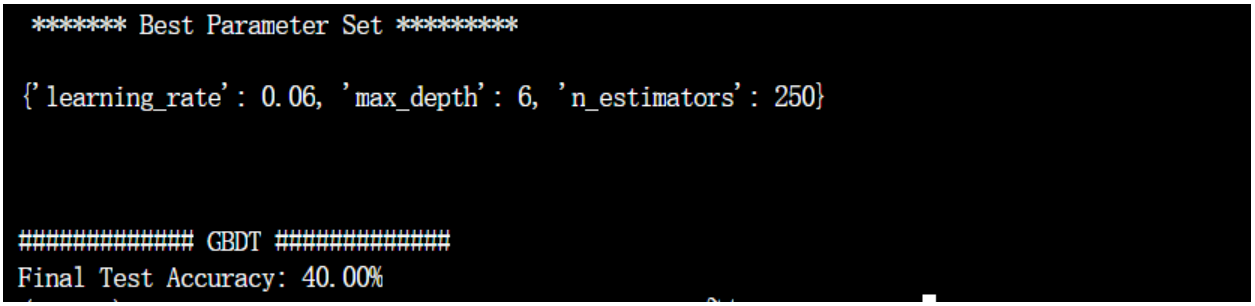

Figure 2.4.4

According to this result, it is obvious that DRCNN model has better performance in this task. Compared to the GBDT model, DRCNN improved the accuracy by nearly 27%. However, when I train the DRCNN and GBDT model, I find that DRCNN needs much more time than GBDT. Moreover, for the DRCNN model, before the 50th epoch, the accuracy is always below 5%. Therefore, the training method of DRCNN may need to be improved. In the process of experiment, especially in the data processing, I find that there are many noises in the actual data, and I need to spend a lot of time to preprocess the data. However, though I did the above, this model still falls short of the performance of the model constructed in the paper, which was able to achieve about 78% performance. Also, since my model reproduction differs from the original paper's model, the time of training the model is much more than the model in the paper.

3 Conclusion

In this research, I reproduces the experiment of “Driving Style Representation in Convolutional Recurrent Neural Network Model of Driver Identification”. The feasibility and effectiveness of the D-CRNN model proposed in this paper are verified, and a high-fidelity driving style representation is constructed from the original vehicle motion data, which delivers on the efficient driver recognition task. There is still a certain gap between the results of the reproduced experiment and those in the paper, and the relative error is about 10%. This paper also explores the influence of the different models on the experimental results, and finds that the D-CRNN model has good generalization ability for different data sets, but it is sensitive to different convolution kernel sizes and the number of hidden units in the Bi-LSTM layer, which needs to be adjusted according to specific data sets. At the same time, the adjustment of relevant parameters may be adjusted according to the actual adjustment.

The main gain of this paper is to deeply understand the principle and implementation of D-CRNN model, as well as the relevant knowledge and technology of driving style representation and driver recognition tasks. The main problems of the repeated experiments in this paper are the lack of more data sets and more evaluation indicators, as well as the lack of interpretability and visualization analysis of D-CRNN model. In this paper, the improvement direction of the repeated experiment is to try to use more data sets and more evaluation indicators to test the performance and generalization ability of D-CRNN model, and to use some interpretable and visual methods to analyze the internal mechanism and output results of D-CRNN model. At the same time, it is also necessary to continue to optimize the structure of the repeated D-RCNN model, so that the model can converge faster to save training time. The suggestion and prospect of future research in this paper is to explore more deep neural network architectures and methods to build a higher fidelity representation of driving style, better handle various forms of data sets obtained in real life, and apply driving style representation and driver recognition tasks to more scenarios and fields, such as driving safety, driving behavior analysis, etc.

Reference

[1] Sobhan Moosavi, Pravar D. Mahajan, Srinivasan Parthasarathy, Colleen Saunders-Chukwu, and Rajiv Ramnath; "Driving Style Representation in Convolutional Recurrent Neural Network Models of Driver Identification", 2020
[2] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, F. Itakura, Driver modeling based on driving behavior and its evaluation in driver identification, Proceedings of the IEEE 95 (2) (2007) 427–437.
[3] N. C. Fung, B. Wallace, A. D. Chan, R. Goubran, M. M. Porter, S. Marshall, F. Knoefel, Driver identification using vehicle acceleration and deceleration events from naturalistic driving of older drivers, in: 2017 IEEE International Symposium on Medical Measurements and Applications (MeMeA), IEEE, 2017, pp. 33–38.
[4] U. Fugiglando, P. Santi, S. Milardo, K. Abida, C. Ratti, Characterizing the" driver dna" through can bus data analysis, in: Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services, 2017, pp. 37–41
[5] W. Dong, J. Li, R. Yao, C. Li, T. Yuan, L. Wang, Characterizing driving styles with deep learning, arXiv preprint arXiv:1607.03611 (2016)
[6] J. Chen, Z. Wu, J. Zhang, Driver identification based on hidden feature extraction by using adaptive nonnegativity-constrained autoencoder, Applied Soft Computing 74 (2019) 1–9
[7] S. Ezzini, I. Berrada, M. Ghogho, Who is behind the wheel? driver identification and fingerprinting, Journal of Big Data 5 (1) (2018) 9
[8] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1234

Appendix

DRCCN.py:

```
from __future__ import print_function
from __future__ import division
import tensorflow as tf
from tensorflow.contrib import rnn
import numpy as np
import random
import _pickle as cPickle
import time
from sklearn.preprocessing import OneHotEncoder
import functools
import argparse
import os
import shutil


class DCRNN_MODEL:
    def __init__(self, data, target, dropout, num_layers, is_training, timesteps=128):
        self.data = data
        self.target = target
        self.dropout = dropout
        self.num_layers = num_layers
        self.is_training = is_training
        self._num_hidden = args.neurons
        self._timesteps = timesteps
        self.prediction
        self.error
        self.optimize
        self.accuracy
```

```
            self.predProbs

    @lazy_property
    def prediction(self):
            # CNN part
            input_layer = tf.reshape(self.data, [-1, self._timesteps, FEATURES, 1])
            conv1 = tf.layers.conv2d(inputs=input_layer, filters=16, kernel_size=[5,
FEATURES], strides=1, activation=tf.nn.relu, padding='SAME')
            pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[1, 8], strides=1)
            pool1 = tf.nn.dropout(pool1, (0.5 if self.is_training==True else 1.0))
            conv2 = tf.layers.conv2d(inputs=pool1, filters=16, kernel_size=[3, 3], strides=1,
activation=tf.nn.relu, padding='SAME')
            pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[1, 8], strides=1)
            last_pool_shape = pool2.get_shape().as_list()
            pool2_flat = tf.reshape(pool2, [-1, last_pool_shape[1],
last_pool_shape[2]*last_pool_shape[3]])
            pool2_flat = tf.nn.dropout(pool2_flat, (0.5 if self.is_training==True else 1.0))

            # concatenating CNN output and input
            pool2_flat_extended = tf.concat([pool2_flat, self.data], axis=2)

            # Recurrent network.
            stacked_rnn = []
            for i in range(self.num_layers):
                cell = tf.contrib.rnn.GRUCell(num_units=self._num_hidden)
                cell = tf.contrib.rnn.DropoutWrapper(cell, output_keep_prob=1.0-
self.dropout[i])
                stacked_rnn.append(cell)
            network = tf.contrib.rnn.MultiRNNCell(cells=stacked_rnn, state_is_tuple=True)
```

```
            x = tf.unstack(pool2_flat_extended, last_pool_shape[1], 1)
            output, _ = rnn.static_rnn(network, x, dtype=tf.float32)

            # Softmax layer parameters
            dense    = tf.layers.dense(inputs=output[-1], units=self._num_hidden,
activation=None)
            dense    = tf.contrib.layers.batch_norm(dense, center=True, scale=True,
is_training=self.is_training, activation_fn=tf.sigmoid)
            dense    = tf.nn.dropout(dense, (0.5 if self.is_training==True else 1.0))
            logits = tf.layers.dense(inputs=dense, units=int(self.target.get_shape()[1]),
activation=None)
            soft_reg = tf.nn.softmax(logits)

            return soft_reg, output[-1]

    @lazy_property
    def cost(self):
            soft_reg, _ = self.prediction
            cross_entropy = tf.reduce_mean(-tf.reduce_sum(self.target * tf.log(soft_reg),
reduction_indices=[1]))
            return cross_entropy

    @lazy_property
    def optimize(self):
            optimizer = tf.train.RMSPropOptimizer(learning_rate=0.00005, momentum=0.9,
epsilon=1e-6)
            return optimizer.minimize(self.cost)

    @lazy_property
    def error(self):
```

```
        soft_reg, _ = self.prediction
        mistakes = tf.not_equal(tf.argmax(self.target, 1), tf.argmax(soft_reg, 1))
        return tf.reduce_mean(tf.cast(mistakes, tf.float32))

@lazy_property
def accuracy(self):
        soft_reg, pool2_flat = self.prediction
        correct_pred = tf.equal(tf.argmax(self.target, 1), tf.argmax(soft_reg, 1))
        return tf.reduce_mean(tf.cast(correct_pred, tf.float32)), pool2_flat

@lazy_property
def predProbs(self):
        soft_reg, _ = self.prediction
        softmax_prob = soft_reg
        return softmax_prob
```