International Journal of
Geo-Information

Article

Automatic Discovery of Railway Train Driving Modes Using Unsupervised Deep Learning

Han Zheng, Zanyang Cui and Xingchen Zhang *

School of Traffic and Transportation, Beijing Jiaotong University, No.3 Shang Yuan Cun, Hai Dian District, Beijing 100044, China
* Correspondence: xczhang@bjtu.edu.cn; Tel.: +86-137-0131-6758

Received: 16 May 2019; Accepted: 24 June 2019; Published: 27 June 2019

Abstract: Driving modes play vital roles in understanding the stochastic nature of a railway system and can support studies of automatic driving and capacity utilization optimization. Integrated trajectory data containing information such as GPS trajectories and gear changes can be good proxies in the study of driving modes. However, in the absence of labeled data, discovering driving modes is challenging. In this paper, instead of classical models (railway-specified feature extraction and classical clustering), we used five deep unsupervised learning models to overcome this difficulty. In these models, adversarial autoencoders and stacked autoencoders are used as feature extractors, along with generative adversarial network-based and Kullback–Leibler (KL) divergence-based networks as clustering models. An experiment based on real and artificial datasets showed the following: (i) The proposed deep learning models outperform the classical models by 27.64% on average. (ii) Integrated trajectory data can improve the accuracy of unsupervised learning by approximately 13.78%. (iii) The different performance rankings of models based on indices with labeled data and indices without labeled data demonstrate the insufficiency of people's understanding of the existing modes. This study also analyzes the relationship between the discovered modes and railway carrying capacity.

Keywords: modes of driving railway trains; unsupervised learning; deep learning; generative adversarial networks; stacked autoencoders

1. Introduction

1.1. Background

Modes of driving railway trains (MDRTs) represent the potential patterns characterizing drivers' behaviors when driving railway trains. MDRTs are an important concept in the research field of transportation and play a vital role in the understanding of the stochastic nature of railway systems [1,2].

A railway system contains large-scale heterogeneous hardware as well as corresponding complex software systems and is operated by staff consisting of differently skilled individuals and individuals with varying educational backgrounds. Therefore, the system inevitably has complex inherent stochastic properties. A good understanding of these inherent stochastic properties could help improve operational performance, from the planning phase to the implementation phases. Without such an understanding, implementations of plans will be hindered even if the plans are carefully scheduled. Any small-scale disturbance that appears in the system will cause a wide range of impacts (e.g., a slight train running delay at a certain station will cause a wide range of delays) [3–7]. The inherent stochastic nature comes from several sources, which can be roughly categorized into two main types: (i) Objective randomness (e.g., train running time fluctuations caused by changes in track surface adhesion during rain) and (ii) subjective uncertainty (e.g., different driving strategies that drivers may take when facing similar scenarios; in the absence of appropriate methods, we cannot predict their behavior in advance). Unlike objective randomness (which is determined by the natural attributes of hardware), subjective uncertainty stems from the behaviors of staff (e.g., dispatchers and drivers), where train driving behaviors are the most direct influences [8]. Thus, discovering the hidden modes in driving behaviors is significant when addressing such subjective uncertainty.

Train driving is a dynamic process with many constraints but also with a certain degree of autonomy. It reflects a trade-off between efficiency and robustness [1,6,8,9]. To ensure the robustness of railway systems, the operators of railways give autonomy to drivers in driving processes (this phenomenon widely exists in railway freight transportation and even in China's highly information-influenced high-speed railway system). Under these circumstances, drivers are able to drive trains according to their experience and basic constraints. For example, MDRTs existing in sections (the connecting parts of adjacent stations) make the running time multivariate, even under the same running plan. If operators have minimal understanding of MDRTs and use poor models (e.g., constant values) to represent the distributions of parameters (e.g., running time and additional operating time), the resulting plans would be inaccurate and unenforceable in the future [2,10–13].

Train driving directly impacts the overall performance of the system [1,4,8]. However, due to the lack of suitable methods, the modes contained therein have been difficult to quantify. The main difficulties faced in discovering MDRTs are as follows.

Large number of modes. Compared to existing transportation modes, MDRTs are more inclined to describe differences and commence at the level of microcosmic driving operation [2,14]. Thus, the number of modes in MDRTs is much greater than that of existing mode detections. Figure 1 illustrates some of the modes corresponding to two running plans (i.e., stop at Station1-stop at Station2 and stop at Station1-pass Station2) as an example. Modes 0–2 stop at both Station1 and Station2. Modes 3–5 stop only at Station1 and pass Station2. These modes will be more complicated when the plans are expanded (e.g., pass Station1-stop at Station2 and pass Station1-pass Station2). Furthermore, the railway consists of a large number of stations and sections. MDRTs that exist in different sections are characterized by large differences. This situation further increases the need for unsupervised learning.

i Large number of modes. Compared to existing transportation modes, MDRTs are more inclined to describe differences and commence at the level of microcosmic driving operation [2,14]. Thus, the number of modes in MDRTs is much greater than that of existing mode detections. Figure 1 illustrates some of the modes corresponding to two running plans (i.e., stop at Station1-stop at Station2 and stop at Station1-pass Station2) as an example. Modes 0–2 stop at both Station1 and Station2. Modes 3–5 stop only at Station1 and pass Station2. These modes will be more complicated when the plans are expanded (e.g., pass Station1-stop at Station2 and pass Station1-pass Station2). Furthermore, the railway consists of a large number of stations and sections. MDRTs that exist in different sections are characterized by large differences. This situation further increases the need for unsupervised learning.

ii Complex structures in the modes. Operators have previously used the running time to distinguish MDRTs. This method is inefficient; many modes are too close in running time to be distinguished [2]. Therefore, a method that fully considers the internal structure of the driving processes is required. However, the inner structure of these modes is complex, especially when multiple phases exist. For example, both mode 3 and mode 4 in Figure 1 have two unfixed-location acceleration phases, which are difficult to classify manually. The difficulty of MDRT analyses caused by the multiple phases has also been mentioned in other studies [6,7,15]. There remains no effective way to automatically discover a large number of modes from this type of complex-structure data.

iii Multi-profile data. The integrated trajectory data are aggregated time-series data that contain multiple profiles obtained from built-in locomotive control and information systems. We call this integrated trajectory data because all these profiles rely on spatio-temporal GPS trajectories for storage.

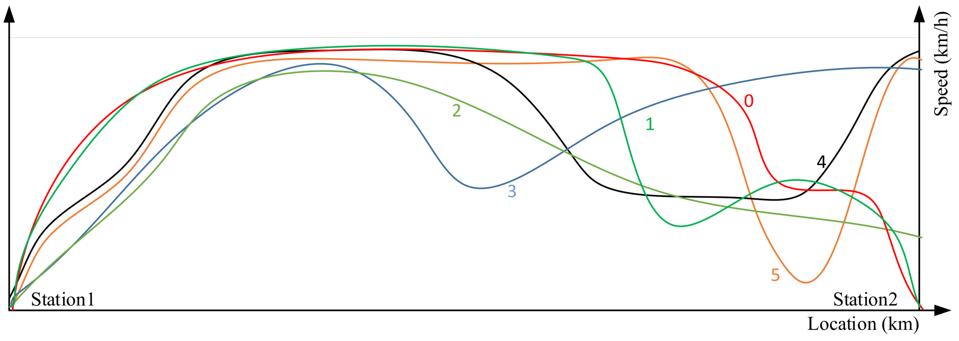


Figure 1. Examples of modes of driving railway trains (MDRTs) from the perspective of speed-location profile. Each line represents a mode in this figure. We use a method based on phase [6] to illustrate each example mode. Mode 0, the expected profile, has the phase structure A-Cr-B-Cr-B. Mode 1 has the phase structure A-Cr-B-A-Cr-B. Mode 2 has the phase structure A-Cr-Co. Mode 3 has the phase structure A-B-A-Cr. Mode 4 has the phase structure A-B-Cr-A. Mode 5 has the phase structure A-Cr-B-A-B, where 'A' represents the acceleration phase, 'Cr' represents the cruising phase (maintaining constant speed with throttle manipulation), 'Co' represents the coasting phase (train operation while the throttle is idle before braking), and 'B' represents the braking phase (for a station stop, for speed restriction or for a restrictive signal).

In modern freight railway systems, locomotives have equipped built-in control and information systems. Benefiting from these systems, large-scale integrated trajectory data can be obtained for the study of latent modes underlying railway driver behaviors. Taking a built-in locomotive control and information system applied in China as an example, Figure 2 illustrates the basic structure of the system, as well as the resulting integrated trajectory data.

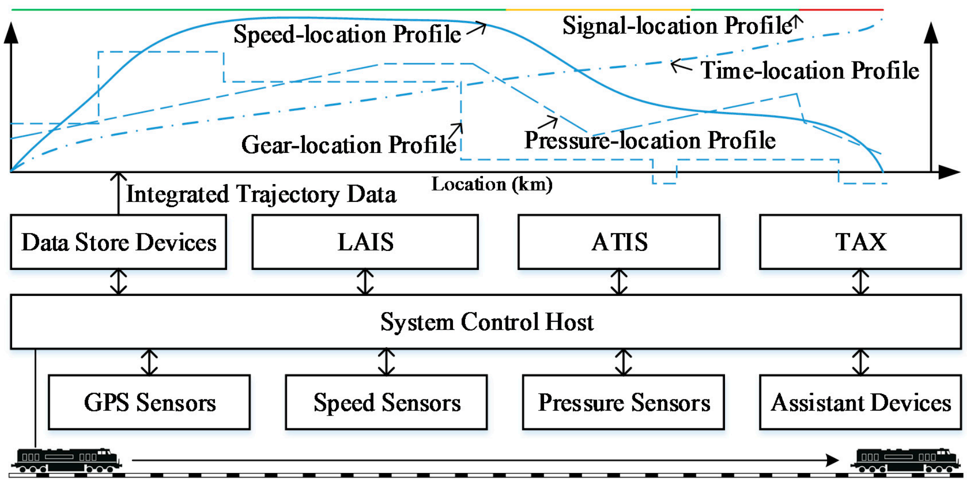


Figure 2. Built-in locomotive control and information systems and integrated trajectory data. LAIS refers to the train operation status information system, ATIS refers to the automatic railway car number identification system, and TAX refers to a comprehensive locomotive safety information monitoring device. All devices and sensors are arranged around the control host, which plays a role as a control center. The solid line represents a speed-location profile example. The dotted line represents a time-location profile example. The dashed line represents a gear-location profile example. The colored line represents a signal-location profile example. The line-dashed line represents a pressure-location profile example.

The integrated trajectory data contain several profiles obtained from sub-systems (e.g., signal and gear profiles) and sensors (e.g., speed, location, and pressure profiles) in the system. The sensors involved in this study are GPS sensors, speed sensors, and pressure sensors. The GPS sensor provides the system with the time and space information of the train by analyzing the relationships between the train and the satellite system. There are two types of speed sensors: The locomotive shaft end photoelectric speed sensor and the Hall magnetoelectric speed sensor. Two sensors are mounted on the axle end of the locomotive to detect the axle speed and convert it into an electrical pulse signal proportional to the wheel speed. The railway locomotive has an independent speed sensor that can be mutually corrected with GPS data to overcome the problem of unstable signal acquisition caused by complex terrain and weather. The pressure sensor provides the system with pressure signals for the train pips, brake cylinders, and balance air cylinders (the pressure states of these devices directly reflect the train's braking condition). The basic idea is to convert the measured gas (liquid) physical pressure into a standard electrical signal through a resistive pressure sensitive element. Through the signal adjustment circuit, the sensor outputs a voltage signal that is linearly proportional to the measured pressure.

The recording mode of the built-in control and information system is based on events (i.e., it records attributes with respect to changes in speed, pressure, signals, acceleration, etc.), and the relations among these profiles are complex. Thus, over a long period of time, the data are stored but not utilized for analyzing modes; there is still no specific and efficient method for finding modes from the multi-profile data.

iv. Lack of labeled data. The information system used by railways was originally designed as a safety-guard system that can assist drivers or dispatchers in avoiding over-speed or violation of signals. Neither the interface nor the built-in function modules are designed for automated analysis. Therefore, there are no ready-made automated analysis results. In addition, as sequence data with complex structures as well as multiple profiles, integrated railway trajectory data are difficult to label manually. On the one hand, only a small number of personnel with certain experience can recognize the modes existing in the data. On the other hand, multi-profile integrated trajectory data can hardly be analyzed directly by humans, which makes manual labeling more difficult. Given such limited labeled data, supervised models ultimately suffer from overfitting when adapted to very flexible deep neural networks.

Motivated by this observation, an unsupervised approach that can automatically discover modes is desired. Such an approach needs to learn important features from integrated train driving trajectory data and then discover the existing modes within an acceptable amount of time. Therefore, this paper attempts to use the unsupervised deep learning method to satisfy the actual requirements of automatically discovering train driving modes. To achieve this goal, we need to overcome two main difficulties:

i Lack of benchmark. Research on MDRTs is still in its infancy; thus, there is no well-accepted benchmark data, which will hinder the evaluation of the unsupervised learning models.

ii Prior knowledge is not accurate. Even experienced employees on site may have an incorrect or inadequate understanding of the mode distribution.

The unsupervised approach should contain unsupervised feature learning as well as unsupervised mode discovery. Once the train has completed a portion of the work, its trajectory data will be immediately processed and stored. After a certain period of accumulation, the operator can analyze the MDRTs in terms of proportion and internal characteristics. The information obtained can be used for multivariate running time calculations (for planning) or driver driving strategy extraction (for new driver training). These utilizations of MDRT can be found in the following fields: Driver behavior habits [6,7,15,16], auto-driving system design [6], and capacity utilization optimization [17,18].

The contributions of this paper are four-fold:

- We propose five unsupervised deep learning models and prove that they achieve better performance (by 27.64% on average) than classical unsupervised learning models on real and artificial datasets with different scales. Additionally, we prove that the adversarial autoencoder clustering model obtains the best performance.

- We prove that integrated trajectory data can improve the accuracy of unsupervised learning compared to pure GPS trajectory data (by approximately 13.78%).
- We discover the mode distribution in real dataset and analyze their characteristics based on phases.
- We measure the difference between model-predicted distributions of data and the labeled distributions by operators, namely, the gap between the unsupervised learning outcomes and the subjective recognition results, based on indices with ground-truth labels.

The remainder of this paper is organized as follows. The relevant literature is reviewed in the following section (Section 1.2) before a detailed introduction of the methodologies used in this study (Section 2). Related experiments from the study and discussion (Section 3) are then presented before our conclusions (Section 4) on the study.

1.2. Related Works

The analysis and application of concepts similar to MDRTs have been hot spots in studies on railway data mining [6,7,15–18]. A multi-phase analysis framework [6] was built to support many complex railway studies such as large-scale micro-simulations [15] and timetable optimizations [7]. In this framework, a driving process is separated by motion phases: Acceleration, cruising, coasting, and braking. These phases are analyzed from the perspectives of speed and distance to obtain various characteristics. Multi-phase phenomena have been discovered, especially in the braking process. However, these studies are based on limited-scale datasets and are focused on the applications of MDRTs (e.g., to develop a stochastic blocking time model for capacity utilization [6,7]). Before these studies, there was a lack of an approach to automatically and effectively discovering MDRTs in a big data context. This has also led to the fact that in the current study, there is no benchmark for everyone to recognize.

Studying modes requires proxies that can provide sufficient information. In existing study researches, both GPS trajectory data and mixed trajectory data obtained from other sensors are used as proxies to study modes. Pure GPS trajectory data have been used as conventional proxies to identify modes of transportation [19]. At the same time, the use of comprehensive data has also received attention [20]. Trajectory data acquisition via smartphones and multi-sensor systems has proven to be feasible in real-world applications. These two kinds data have their own pros and cons. Pure GPS trajectory data have many advantages, such as being low cost [21], having few spatiotemporal limitations [22], providing sufficient information [19,21,22], and relatively maturing techniques [23–26]. However, this single source trajectory data is less stable in complex environments. There is a lot of noise in the GPS signal in a railway environment such as caves. In contrast, the integrated data are more robust, and can cope with complex acquisition environments, and the information contained is more abundant. However, the data collection cost is relatively higher [20,27]. In the context of this article, the used data can be obtained from the existing locomotive sensor system. Therefore, using integrated data will result in better accuracy without increasing costs. The integrated trajectory data used in this paper are a type of mixed GPS data; all profiles are attached to the GPS trajectory data.

A set of discriminative features plays a vital role in finding modes hidden in transportation trajectory data. In existing studies, supervised features (i.e., artificially designed features) such as global statistical features [14,20,27–32], local features [19,33], time-domain features [20], frequency-domain features [20,27], and specific features [28,30,34–37] have been extracted. Zheng et al. [2] proposed a type of feature that can reveal the differences between railway driving behaviors, therein being designed for a railway mode identification context. These features are designed by hand and are quick and easy to calculate. However, these features do not fully accommodate the demands of discovering MDRTs; these features have poor ability to characterize microscopic differences and are either inapplicable for addressing integrated multi-profile trajectory data directly. In addition they are insufficiently integrated with clustering methods, which means the joint training cannot be achieved.

The development of unsupervised learning models and algorithms has given us more choices in design methods in terms of both feature extraction and mode discovery. Classical unsupervised feature extraction methods, such as parameter t-SNE [38] and PCA [39], have been used in the field of unsupervised feature learning. Recently, deep learning methods have been developing rapidly; stacked autoencoders (SAE) [40,41], generative adversarial networks (GAN) [42], etc. have become important feature learning methods. Both GAN and SAE aim to match the real data distribution and simultaneously provide a mapping from a latent space to the input space. However, the mechanisms behind these two methods are not the same: SAE using an explicit approximation of maximum likelihood and GAN through implicit sampling [43]. Which method is more appropriate for the integrated trajectory data is a question we need to explore.

Clustering is the most effective method for discovering modes in feature spaces without sufficient labeled data. Deep learning also plays an important role in clustering [40]. Adversarial autoencoders [42], Kullback–Leibler (KL)-divergence-based networks [41], and categorical generative adversarial networks [44] have applications in the field of clustering. The joint training of the feature extractors as well as cluster models has been proven to be better than independent training [41]. Therefore, the performances of the above clustering methods combined with different feature extraction methods have become the focus of our attention.

In this paper, we propose an approach using deep unsupervised learning models to discover hybrid transportation modes from integrated trajectory data, which does not require prior labeled data. Five models, being combinations of different feature extractors and clustering models, are proposed. Deep learning models, such as adversarial autoencoders and stacked autoencoders, are designed as feature extractors, while generative adversarial network-based and KL divergence-based networks are used as clustering models. The reason for choosing these five models is that we want to explore the performance of two different types of deep learning methods (i.e., SAE, and GAN) in the face of integrated trajectory data. We also hoped to find a better combination of feature extraction and clustering method in joint training. In addition, integrated multi-profile data are used to improve learning performance.

2. Methodologies

2.1. Methodology Set Up

In this paper, we proposed an unsupervised deep learning approach to discover MDRTs from integrated trajectory data. The main modules of our discovery methodologies are shown in Figure 3, which includes three main parts: I. Preprocessing, II. The unsupervised deep learning models, and III. The evaluation metric.

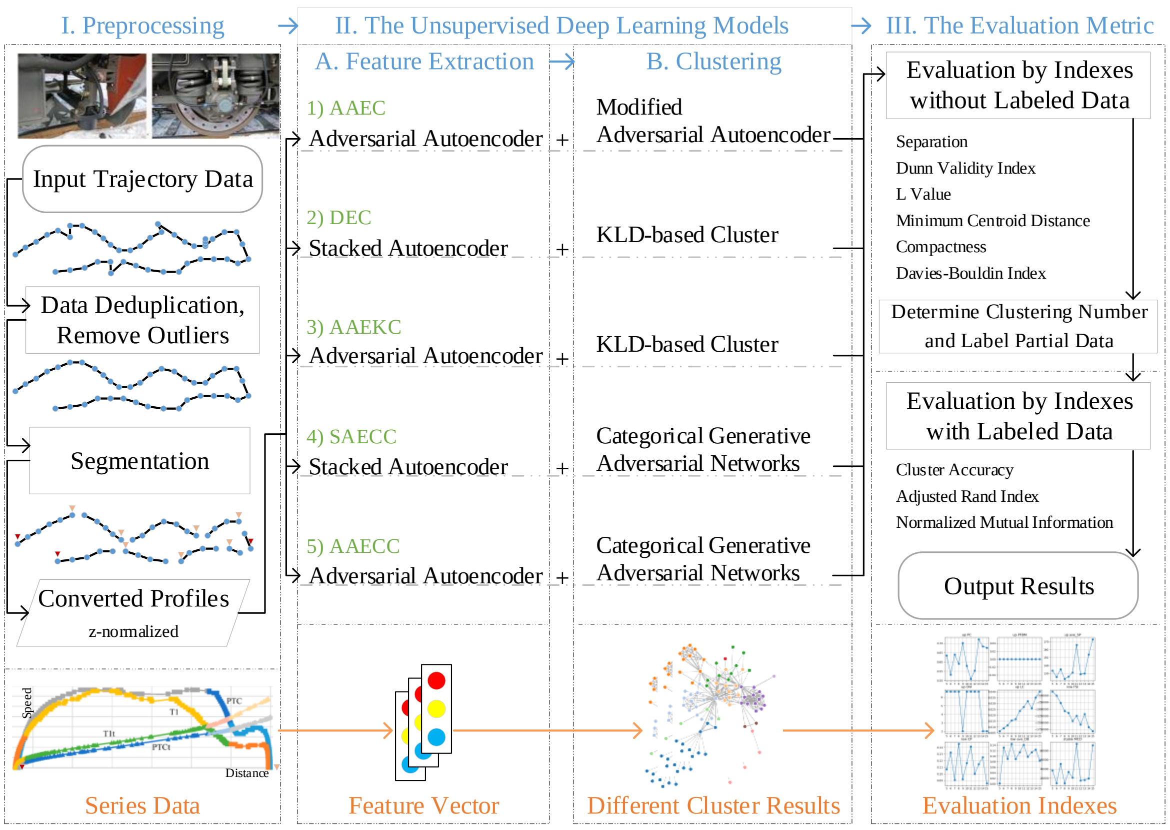


Figure 3. Framework of the proposed approach. The figure is roughly divided into three parts from top to bottom. The first part (blue) represents the basic parts of this methodology, including preprocessing, the unsupervised deep learning models, and the evaluation metric; the second part (black) is the specific details of each methodology part, where the flow is indicated by the arrows; and the third part (orange) illustrates the outputs of each part.

During preprocessing four data preprocessing techniques were employed. First, we removed duplicate data in the trajectories, as some data points were recorded more than once due to the designed redundancy mechanism [2]. Second, based on experience, we removed outlying trajectories that were deemed abnormal. For instance, if the average speed of a trajectory exceeded the designed speed, we identified it as an outlying trajectory and removed it from the dataset. In addition, we detected and removed outlying data points from the raw trajectories. Figure 4 illustrates the existence of outlying data points in trajectories. The method proposed by [45] was used to detect outlying data points. Third, we divided the full recorded trajectories (which recorded all information without interruption) into segments. The criterion of the divisions depended on the demands of the mode analyses. In this study, we focused on analyzing modes in sections other than at stations; therefore, we used the train stop criteria proposed in [2] to obtain segments from the trajectories. Fourth, we used z-normalization [46] to eliminate the corresponding distortions before mode discovery. Using this method, we can achieve scaling and translation invariances. The formula for z-normalization is illustrated in Equation (1).

$$x_{normalization} = (x - \mu)/\sigma \tag{1}$$

where μ denotes the mean value of the sample data and μ denotes the standard deviation of the sample data.

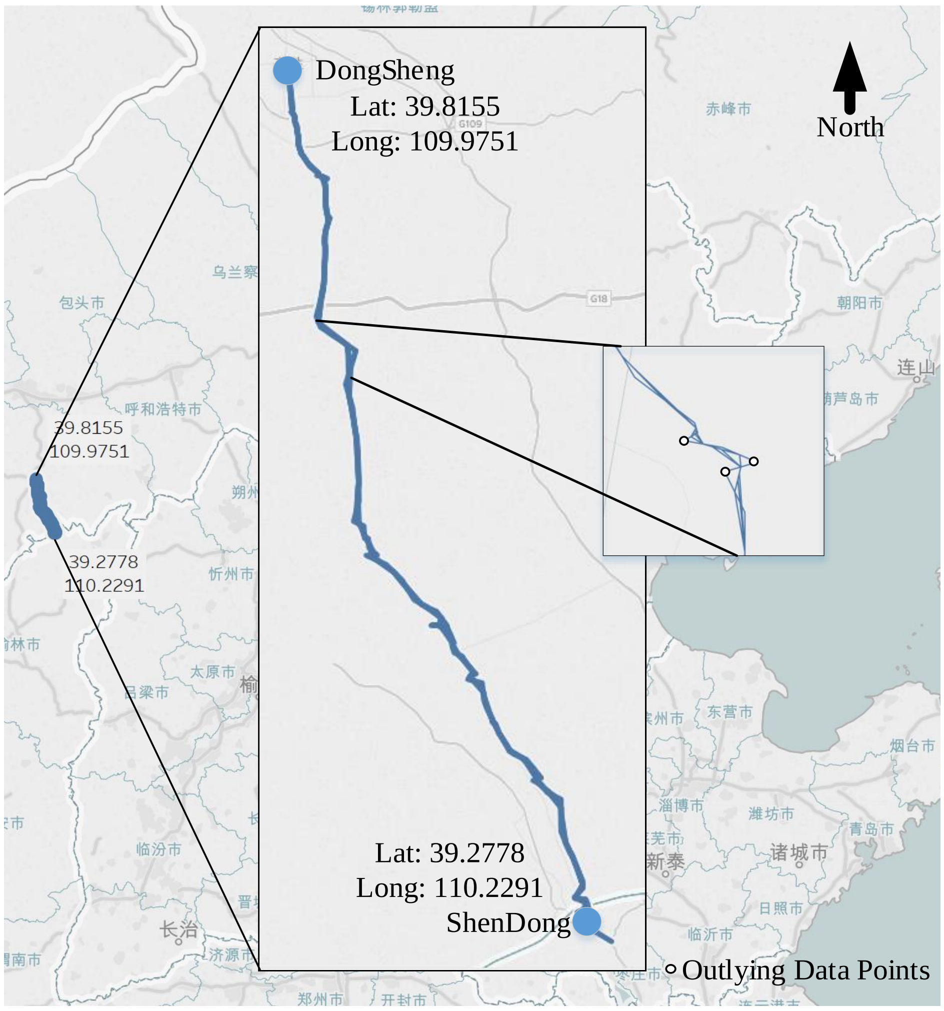


Figure 4. Outlying points in GPS trajectories (of integrated trajectory data). The latitudes and longitudes in the figure represent the spatial extent of the example trajectory data.

For the unsupervised deep learning models, five deep learning models were proposed. These five models were combinations of different deep learning-based feature extractors and clustering models, including adversarial autoencoders and stacked autoencoders as feature extractors as well as generative adversarial network-based and KL divergence-based networks as clustering models. We detailed each of the proposed unsupervised deep learning models in Section 2.2. Although we analyzed the proposed models from two perspectives, feature extractors and clustering models, they were jointly trained; the joint training of the feature extractors as well as cluster models has been proven to be better than independent training [41].

As to the evaluation metric, we used two systems of indices to evaluate the performances of the different methods: indices without ground-truth labels (discussed in Section 2.3.1) and indices with ground-truth labels (discussed in Section 2.3.2). The former can reveal the performance of each method from the aspect of cluster distributions. The latter can reveal similarities between clustering results and real demands. The detected modes need to be interpretable in the context of railway production. The stronger the similarities are, the closer the clustering result is to people's understandings. Ground-truth labels are obtained from experts who are veterans in driving or controlling railway trains.

2.2. Unsupervised Deep Learning Models

In this section, we introduced five models of unsupervised deep learning, along with corresponding structures, parameters, and learning methods, to address the mode discovery problem.

2.2.1. Parameters Tuning

Tuning the optimal hyperparameters of the deep learning model is not an easy task [41]. In practice, Bayesian optimization has been shown to obtain better results in fewer evaluations compared to grid search and random search due to the ability to reason with respect to the quality of experiments before they are run [47–50]. Therefore, we chose Bayesian optimization as the parameter tuning method. The details of Bayesian optimization can be found in references [47–50]. We selected the number of neurons in each layer of the deep learning network as hyperparameters and DVI as the evaluation index. The detailed formula of DVI is shown in Section 2.3.1. The results of the parameter tuning are presented in the Appendix A, from Tables A1–A5. According to the results, the deep learning method was proved to be not very sensitive to the parameters; the index did not change significantly. The selected hyperparameters are shown in Sections 2.2.2–2.2.6.

2.2.2. Adversarial Autoencoder Clustering Model (AAEC)

The adversarial autoencoder [42], which is a derivative of the generative adversarial network [51], was first proposed as a tool for extracting features. To perform clustering, Alireza et al. [42] modified the adversarial autoencoder such that the continuous latent feature variables are replaced by discrete latent class variables and such that the adversarial part imposes a categorical distribution on the cluster representation instead of a Gaussian distribution. Using the adversarial autoencoder as a feature extractor and the modified adversarial autoencoder as a cluster method, we obtained the adversarial autoencoder clustering model (AAEC), which can disentangle discrete class variables from the continuous latent feature variables in a purely unsupervised manner [42].

In this model, the data are generated by a latent class variable y that comes from a categorical distribution, as well as a continuous latent variable z that comes from a Gaussian distribution:

$$\begin{aligned} p(y) &= Cat(y), \\ p(z) &= N(z|0, I). \end{aligned} \tag{2}$$

Figure 5 illustrates the basic framework of AAEC. The inference network of the AAEC predicts both the discrete class variable y and the continuous latent variable z. The decoder then utilizes both the class label as a one-hot vector and the feature to reconstruct the profile. There are two separate adversarial networks that regularize the hidden representation of the autoencoder. The top adversarial network imposes a categorical distribution on the label representation. This adversarial network ensures that the latent class variable y does not carry any style information and that the aggregated posterior distribution of y matches the categorical distribution. The bottom adversarial network imposes a Gaussian distribution on the style representation, which ensures that the latent feature z is a continuous Gaussian variable. The details of each sub-network are shown in Table 1.

Table 1. Structure of the Network.

Index	Sub-Networks	Structure [1]
1	Encoder (Generator)	(3000, Leaky ReLU) − (3000, Leaky ReLU)− −(F [2] + C [4], Linear + Softmax) [5]
2	Decoder	(3000, Leaky ReLU) − (3000, Leaky ReLU) − (D [3], Linear)
3	Discriminator 1	(3000, Leaky ReLU) − (3000, Leaky ReLU) − (1, Sigmoid)
4	Discriminator 2	(3000, Leaky ReLU) − (3000, Leaky ReLU) − (1, Sigmoid)

[1] We used (Output, Activation function) as the format to illustrate a layer of the sub-networks. [2] F. F is the dimension of the feature, which is set as $100 \times n$ in this study, where n is the number of considered profiles. [3] D. D is the dimension of the input data. [4] C. C is the number of clusters. [5] The Encoder has two outputs referring to features and clusters, as shown in Figure 5.

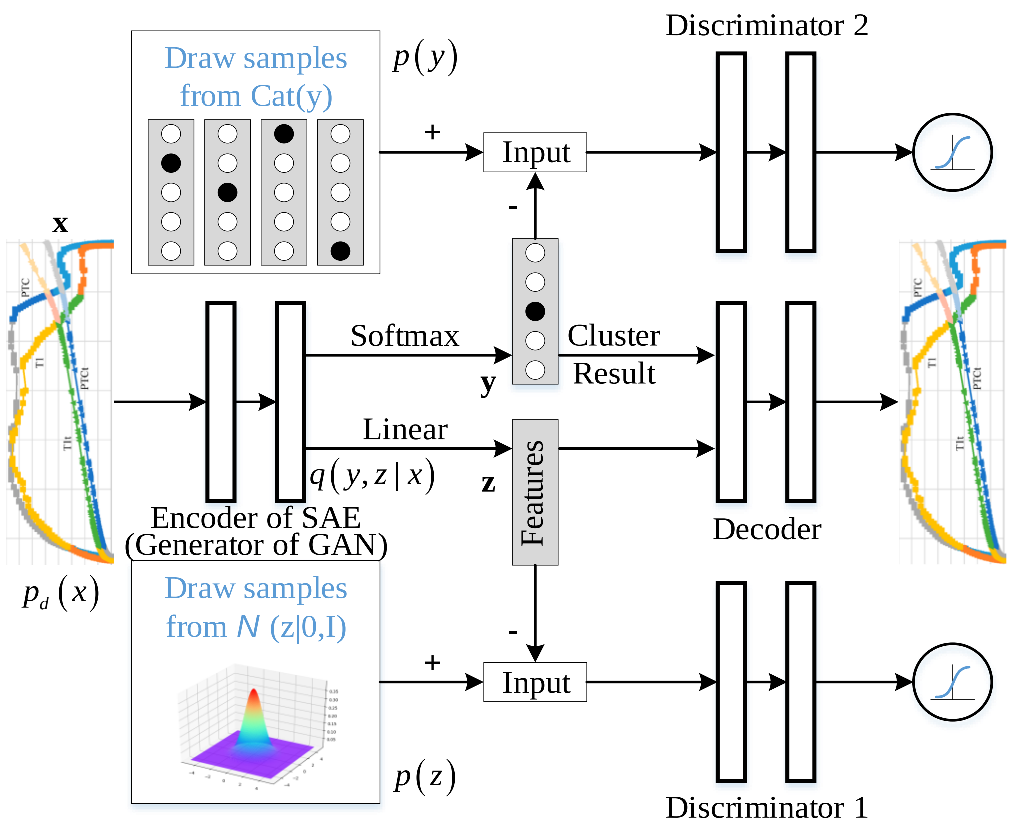


Figure 5. Network structure of the adversarial autoencoder clustering model (AAEC): The top adversarial network imposes a categorical distribution on the cluster representation, and the bottom adversarial network imposes a Gaussian distribution on the feature representation.

Both the adversarial networks and the autoencoder are trained jointly with stochastic gradient descent (SGD) in two phases: Phase I and Phase II. In Phase I, the autoencoder updates the encoder and decoder to minimize the reconstruction error of the inputs on an unlabeled mini-batch. In Phase II, each of the adversarial networks first updates their discriminative network to discriminate the true samples (generated using the categorical and Gaussian priors) from the generated samples (the cluster and feature computed by the autoencoder). The adversarial networks then update their generator to confuse their discriminative networks. The learning flow is shown in Table 2, where the training steps are illustrated in order. The detailed parameters of each learning step are shown in Table 3.

Table 2. Learning Flow.

Index	Learning Type	Learning Sub-Networks	Iterations	Algorithm
I	Layer-wise	Encoder-Decoder	I: 2000	SGD [1]
II.1	Layer-wise	Discriminator 1		SGD [2]
II.2	Layer-wise	Discriminator 2	II: 2000	SGD [2]
II.3	End-to-end	Generator		SGD [1]

[1] SGD. Stochastic gradient descent with a momentum value of 0.9. [2] SGD. Stochastic gradient descent with a momentum value of 0.1.

Table 3. Learning Parameter Settings.

Index	Dropout Rate	Mini-Batch Size	Learning Rate	Convergence Threshold	Loss Function
I	0.2 [5]	256	0.01 [1]	——	MSE [3]
II.1	0	256	0.1 [2]	——	BCE [4]
II.2	0	256	0.1 [2]	——	BCE [4]
II.3	0.2[5]	256	0.1 [2]	——	MSE [3]

[1] We used the initial learning rate of 0.01 and, after 50 epochs, reduced it to 0.001. [2] We used the initial learning rate of 0.1 and, after 50 epochs, reduced it to 0.01. [3] MSE. Mean Square Error. [4] BCE. Binary Cross-entropy. [5] We used the dropout operation at the input layer with a dropout rate of 0.2.

2.2.3. Deep Embedded Clustering (DEC)

In the deep embedded clustering (DEC) model, we used a stacked autoencoder to extract features from profiles and used the KLD (Kullback–Leibler divergence)-based method to conduct clustering. Inspired by the t-SNE parameter [38], this model was first proposed by Xie et al. [41] and outperforms other methods such as LDGMI [52].

The DEC clusters data by simultaneously learning a set of k cluster centers $\left\{u_j \in Z\right\}_{j=1}^{k}$ in the feature space Z and the parameters θ of the DNN that maps data points into Z. The DEC has two phases: Phase I, i.e., feature learning with a stacked autoencoder (SAE) [40], and Phase II, i.e., clustering, where we iterate between computing an auxiliary target distribution and minimizing the Kullback–Leibler divergence to it (given an initial estimate of θ and $\left\{u_j\right\}_{j=1}^{k}$). Phase II is optimized by the stochastic gradient descent (SGD)-based joint algorithm [41]. The objective function of Phase II is shown in Equation (3):

$$\begin{gathered} L = KL(P\|Q) = \textstyle\sum_i \sum_j p_{ij} \log p_{ij}/q_{ij}, \\ q_{ij} = \left(1+ \| z_i - u_j \|^2 /\alpha\right)^{-\frac{\alpha+1}{2}} / \textstyle\sum_{j'}\left(1+ \| z_i - u_{j'} \|^2 /\alpha\right)^{-\frac{\alpha+1}{2}}, \\ p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_j}, \end{gathered} \tag{3}$$

where $z_i = f_\theta(x_i) \in Z$ corresponds to $x_i \in X$ after embedding, α are the degrees of freedom of the Student's t-distribution (we set $\alpha = 1$ in this study based on previous studies [38,41]), q_{ij} can be interpreted as the probability of assigning sample i to cluster j (i.e., a soft assignment), and p_{ij} is the auxiliary distribution proposed by Xie et al. [41].

Inspired by Maaten [38], the structure of the DEC model is constructed (shown in Figure 6), and the details of each sub-network are shown in Table 4. The learning flow is shown in Table 5, where the training steps are illustrated in order. The parameters of each learning step are detailed in Table 6. To initialize the centroids, we ran k-means with 20 restarts and selected the best solution.

Table 4. Structure of the network.

Index	Sub-Networks	Structure [1]
1	Encoder	(1000, ReLU) − (1000, ReLU) − (2000, ReLU) − (F [2], Linear)
2	Decoder	(2000, ReLU) − (1000, ReLU) − (1000, ReLU) − (D [3], Linear)
3	Cluster Layer	(C [4], KL Divergence)

[1] We used (Output, Activation function) as the format to illustrate a layer of a sub-network. [2] F. F is the dimension of the feature, which was set as $100 \times n$ in this study, where n is the number of considered profiles. [3] D. D is the dimension of the input data. [4] C. C is the number of clusters.

Table 5. Learning flow.

Index	Learning Type	Learning Sub-Networks	Iterations	Algorithm
I.1	Layer-wise	Encoder-Decoder	I: 2000	SGD
I.2	End-to-end	Encoder-Decoder		SGD
II	End-to-end	Encoder-Cluster Layer	II: 50,000	SGD [1]

[1] SGD. Stochastic Gradient Descent-based joint algorithm with a momentum value of 0.9.

Table 6. Learning parameter settings.

Index	Dropout Rate	Mini-Batch Size	Learning Rate	Convergence Threshold	Loss Function
I.1	0.2	256	0.1 [1]	——	MSE [2]
I.2	0	256	0.1 [1]	——	MSE
II	0	256	0.01	0.10%	KLD [3]

[1] The learning rate was divided by 10 every 1000 iterations. [2] MSE. Mean Square Error. [3] KLD. Kullback–Leibler Divergence.

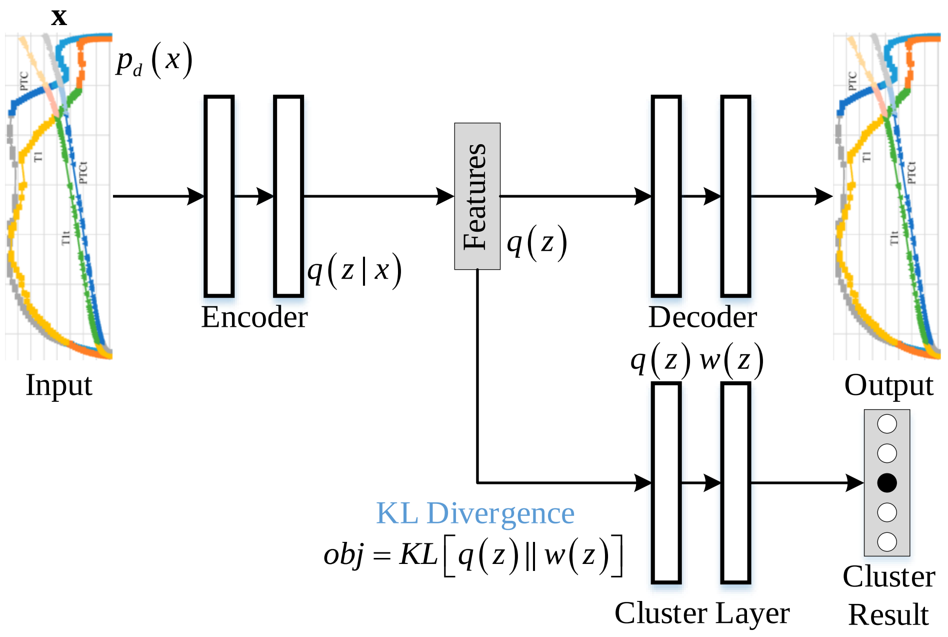


Figure 6. Network structure of deep embedded clustering (DEC). The first row is a stacked autoencoder that is initialized layer by layer, where each layer is a denoising autoencoder trained to reconstruct the previous layer's output after corruption [40]. The second row is the cluster layer, whose objective is minimizing the Kullback–Leibler (KL) divergence.

2.2.4. Model Consisting of AAE and KLD-based Cluster (AAEKC)

We used a model consisting of an AAE (as a feature extractor) and a KLD-based network (as a cluster), denoted as AAE and KLD-based cluster (AAEKC).

The structure of the AAEKC model is constructed (shown in Figure 7). There are two phases in the AAEKC training process: Phase I updates the autoencoder (Encoder + Decoder) to minimize the reconstruction error of the inputs. Phase II trains the discriminator and then the generator. Phase III performs clustering, where we iterate between computing an auxiliary target distribution and minimizing the Kullback–Leibler divergence to it, as discussed in Section 2.2.2. The details of each sub-network are shown in Table 7. The learning flow is shown in Table 8, where the training steps are illustrated in order. The detailed parameters of each learning step are shown in Table 9. To initialize the centroids, we ran k-means with 20 restarts and selected the best solution.

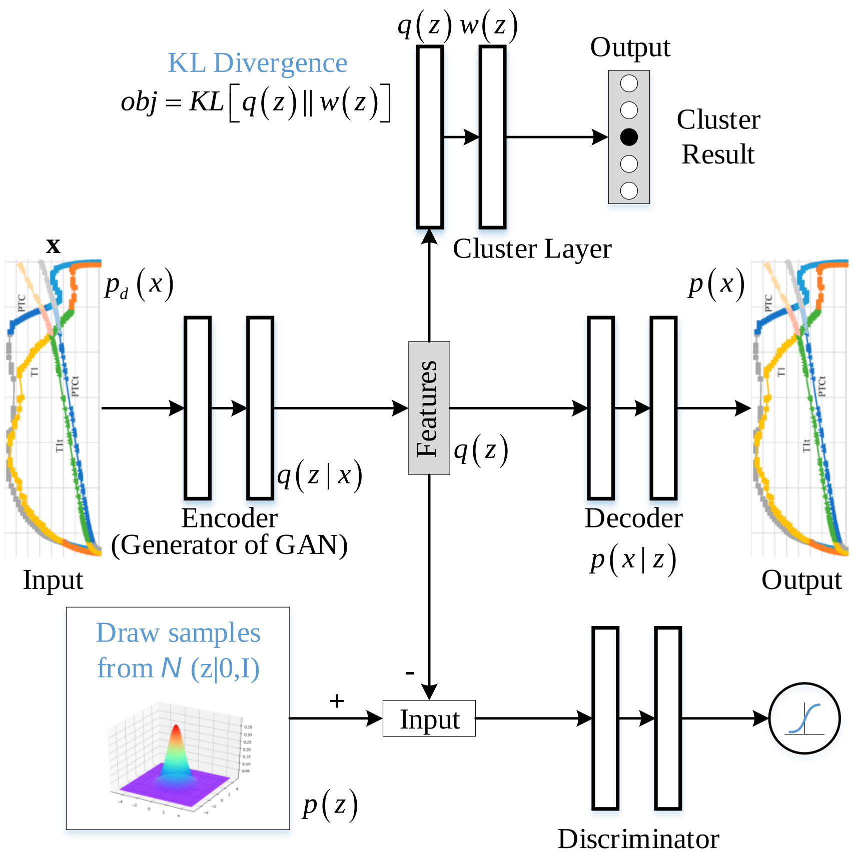


Figure 7. Network structure of AAE and KLD-based cluster (AAEKC). The first row is the cluster layer, whose objective is minimizing the Kullback–Leibler divergence. The remainder of this network is an adversarial autoencoder [42], where the second row is a standard autoencoder that reconstructs data x from a latent feature z. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the feature of the autoencoder or from a Gaussian distribution.

Table 7. Network structure.

Index	**Sub-Networks**	**Structure** [1]
1	Encoder (Generator)	(3000, Leaky ReLU) – (3000, Leaky ReLU) – (F [2], Sigmoid)
2	Decoder	(3000, Leaky ReLU) – (3000, Leaky ReLU) – (D [3], Linear)
3	Discriminator	(3000, Leaky ReLU) – (3000, Leaky ReLU) – (1, Sigmoid)
4	Cluster Layer	(C [4], KL Divergence)

[1] We used (Output, Activation function) as the format to illustrate a layer of the sub-network. [2] F. F is the dimension of the feature, which was set as $100 \times n$ in this study, where n is the number of considered profiles. [3] D. D is the dimension of the input data. [4] C. C is the number of clusters.

Table 8. Learning flow.

Index	**Learning Type**	**Learning Sub-Networks**	**Iterations**	**Algorithm**
I	Layer-wise	Encoder-Decoder	I: 2000	SGD [1]
II.1	Layer-wise	Discriminator [3]	II: 2000	SGD [2]
II.2	End-to-end	Generator [3]	III: 50,000	SGD [1]
III	End-to-end	Encoder-Cluster Layer		SGD [1]

[1] SGD. Stochastic gradient descent with a momentum value of 0.9. [2] SGD. Stochastic gradient descent with a momentum value of 0.1. [3] Samples from the dataset as well as from the Gaussian distribution were used to train the generator and then the discriminator.

Table 9. Learning parameter settings.

Index	Dropout Rate	Mini-Batch Size	Learning Rate	Convergence Threshold	Loss Function
I	0.2 [6]	256	0.01 [1]	——	MSE [3]
II.1	0	256	0.1 [2]	——	BCE [4]
II.2	0.2 [6]	256	0.1 [2]	——	MSE [3]
III	0	256	0.01	0.10%	KLD [5]

[1] The learning rate was divided by 10 every 1000 iterations. [2] We used the initial learning rate of 0.1 and, after 50 epochs, reduced it to 0.01. [3] MSE. Mean Square Error. [4] BCE. Binary Cross-entropy. [5] KLD. Kullback–Leibler Divergence. [6] We used the dropout operation at the input layer with a dropout rate of 0.2.

2.2.5. Model Consisting of SAE and CatGAN (SAECC)

SAE and CatGAN (SAECC) is a model in which stacked autoencoders play a role in extracting features, and categorical generative adversarial networks (CatGANs) [44] play a role in clustering.

The discriminator in the standard GAN is designed to determine whether a given example x belongs to dataset X; this is a binary soft classification task. To address cluster issues, Springenberg [44] extended the standard GAN framework as CatGAN such that the discriminator can be used for clustering. Instead of learning a binary discriminative function, the CatGAN has a discriminator that can separate the data into C clusters by assigning a label y to each example x. Formally, the discriminator $D(x)$ is defined for this setting as a differentiable function predicting logits for C clusters: $D(x) \in \mathbb{N}^K$. The probability of example x belonging to one of the C mutually exclusive clusters is then given through a softmax assignment based on the discriminator output:

$$\mathrm{p}(y = k|x, D) = \frac{e^{D_k(x)}}{\sum_{k=1}^{K} e^{D_k(x)}} \tag{4}$$

However, the pure CatGAN underperforms the AAEC [42]. Thus, a stacked autoencoder is introduced to extract features for the CatGAN. We illustrate the network structure of SAECC in Figure 8, including the Generator, Discriminator, Encoder, and Decoder. The details of each sub-network are shown in Table 10. The training of this model has two phases. Phase I performs feature learning with the stacked autoencoder (Encoder + Decoder). Phase II performs clustering by the CatGAN, including two sub-phases: Training Generator and Discriminator iteratively. During the iterations of Phase II, samples from a dataset as well as from a Gaussian distribution are used to train the generator and discriminator, respectively. Due to the discussed change to the Discriminator, two types of entropies are introduced to calculate the losses of the trainings [44]. The learning flow is shown in Table 11, where the training steps are illustrated in order. The parameters of each learning step are detailed in Table 12.

Table 10. Network structure.

Index	Sub-Networks	Structure [1]
1	Encoder	(1000, ReLU) − (1000, ReLU) − (2000, ReLU) − (F [2], Sigmoid)
2	Decoder	(2000, ReLU) − (1000, ReLU) − (1000, ReLU) − (D [3], Linear)
3	Generator	(500, Leaky ReLU) − (1000, Leaky ReLU) − (F [2], Sigmoid)
4	Discriminator	(1000, Leaky ReLU) − (500, Leaky ReLU) − (250, Leaky ReLU) − (250, Leaky ReLU) − (250, Leaky ReLU) − (C [4], Softmax)

[1] We used (Output, Activation function) as the format to illustrate a layer of a sub-network. [2] F. F is the dimension of the feature, which was set as $100 \times n$ in this study, where n is the number of considered profiles. [3] D. D is the dimension of the input data. [4] C. C is the number of clusters.

Table 11. Learning flow.

Index	Learning Type	Learning Sub-Networks	Iterations	Algorithm
I	Layer-wise	Encoder-Decoder	I: 2000	SGD [1]
II.1	Layer-wise	Generator [2]	II: 5000	ADAM
II.2	Layer-wise	Discriminator [2]		ADAM

[1] SGD. Stochastic gradient descent with a momentum value of 0.9. [2] Samples from a dataset as well as from a Gaussian distribution were used to train the generator and discriminator, respectively.

Table 12. Learning parameter settings.

	Dropout Rate	Mini-Batch Size	Learning Rate	Convergence Threshold	Loss Function
I	0.2	256	0.01	——	MSE [1]
II.1	0	256	0.01	——	SE [2]
II.2	0	256	0.01	——	SE [2]

[1] MSE. Mean Square Error. [3] KLD. Kullback–Leibler Divergence. [2] SE. Specified entropies defined in study [44].

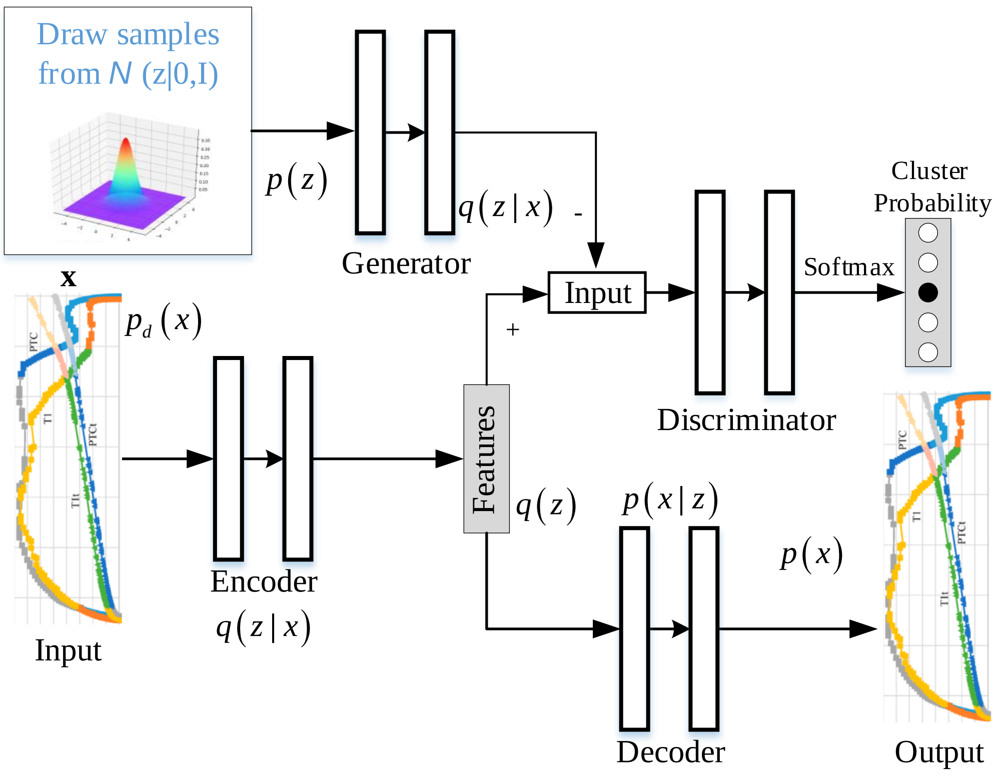


Figure 8. Network Structure of SAE and CatGAN (SAECC). The network consists of the Generator, Discriminator, Encoder, and Decoder. Among them, the Encoder and Decoder form the SAE, and the remainder constitutes the CatGAN.

2.2.6. Model Consisting of AAE and CatGAN (AAECC)

AAE and CatGAN (AAECC) is a model in which adversarial autoencoders play a role in extracting features, and the CatGAN plays a role in clustering. We illustrate the structure of AAECC in Figure 9. Each sub-network is detailed in Table 13.

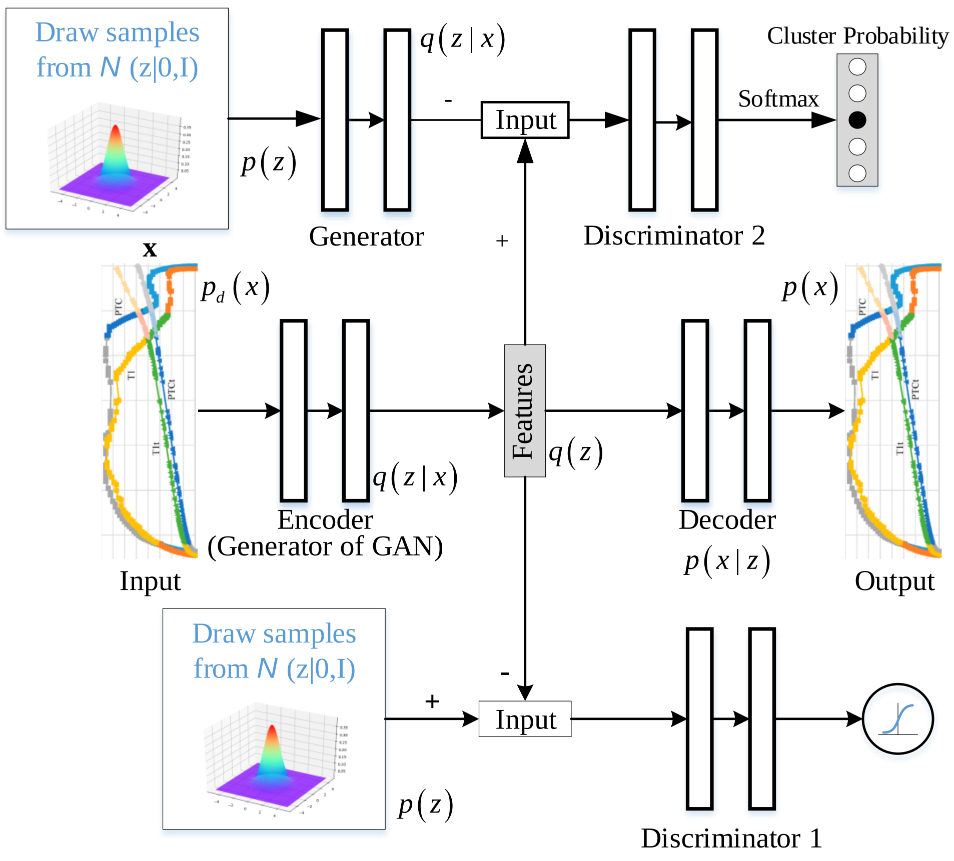


Figure 9. Network structure of AAE and CatGAN (AAECC).

Table 13. Network structure.

Index	Sub-Networks	Structure [1]
1	Encoder (Generator 1)	(1000, ReLU) – (1000, ReLU) – (2000, ReLU) – (F [2], Sigmoid)
2	Decoder	(2000, ReLU) – (1000, ReLU) – (1000, ReLU) – (D [3], Linear)
3	Generator 2	(500, Leaky ReLU) – (1000, Leaky ReLU) – (F [2], Sigmoid)
4	Discriminator 1	(3000, Leaky ReLU) – (3000, Leaky ReLU) – (1, Sigmoid)
5	Discriminator 2	(1000, Leaky ReLU) – (500, Leaky ReLU) – (250, Leaky ReLU) – (250, Leaky ReLU) – (250, Leaky ReLU) – (C [4], Softmax)

[1] We used (Output, Activation function) as the format to illustrate a layer of a sub-network. [2] F. F is the dimension of the feature, which was set as $100 \times n$ in this study, where n is the number of considered profiles. [3] D. D is the dimension of the input data. [4] C. C is the number of clusters.

The learning process of AAECC has three phases: Phase I updates the autoencoder (Encoder + Decoder) to minimize the reconstruction error of the inputs. Phase II trains Discriminator 1 and then Generator of the GAN. Phase III implements clustering by the CatGAN, consisting of two sub-phases: Training Generator and Discriminator 2 iteratively. During the iterations of Phase III, samples from the dataset and from a Gaussian distribution are used to train the generator and discriminator, respectively. The learning flow is shown in Table 14, where the training steps are illustrated in order. The parameters of each learning step are detailed in Table 15.

Table 14. Learning flow.

Index	Learning Type	Learning Sub-Networks	Iterations	Algorithm
I	Layer-wise	Encoder-Decoder	I: 2000 II: 5000	SGD [1]
II.1	Layer-wise	Discriminator 1 [3]		SGD [2]
II.2	End-to-end	Generator 1 [3]		SGD [1]
III.1	Layer-wise	Generator 2 [3]		ADAM
III.2	Layer-wise	Discriminator 2 [3]		ADAM

[1] SGD. Stochastic gradient descent with a momentum value of 0.9; [2] SGD. Stochastic gradient descent with momentum value of 0.1. [3] Samples from the dataset as well as from a Gaussian distribution are used to train the generator and discriminator, respectively.

Table 15. Learning parameter settings.

Index	Dropout Rate	Mini-Batch Size	Learning Rate	Convergence Threshold	Loss Function
I.1	0.2 [5]	256	0.01 [1]	——	MSE [3]
I.2	0	256	0.1 [2]	——	BCE [4]
I.3	0.2 [5]	256	0.1[2]	——	MSE [3]
II.1	0	256	0.01	——	SE [6]
II.2	0	256	0.01	——	SE [6]

[1] The learning rate was divided by 10 every 1000 iterations; [2] We used the initial learning rate of 0.1 and, after 50 epochs, reduced it to 0.01 [3] MSE. Mean Square Error. [4] BCE. Binary Cross-entropy. [5] We used the dropout operation at the input layer with a dropout rate of 0.2. [6] SE. Specified entropies defined in [44].

2.3. Evaluation Metric

In our study, we used two metric systems, with and without labeled data, as the tools to evaluate the performance of each method.

2.3.1. Evaluation Indices Without Ground-Truth Labels

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). Compactness and divergence are two essential aspects in evaluating the quality of the clustering results. However, these two aspects are not easily measured directly. We used an evaluation system that does not require ground-truth labeled data as a proxy of compactness and divergence to evaluate the performance of each method. This system contains six evaluation indices that do not require labeled data, as shown in Table 16.

Table 16. Evaluation indices without ground-truth labels.

ID	Index Name	Notation	D	C	EC	References
1	Separation	$SP(c) = \frac{1}{c^2-c}\sum_{i=0}^{c-1}\sum_{j=i+1}^{c-1} \parallel v_i - v_j \parallel$	1	0	↑	[53]
2	Dunn Validity Index	$DVI(c) = \frac{\min\limits_{0<m\neq n<c}\left\{\min\limits_{\substack{\forall x_i \in \Omega_m \\ \forall x_j \in \Omega_n}} \{\lVert x_i - x_j \rVert\}\right\}}{\max\limits_{0<m<c}\max\limits_{\forall x_i, x_j \in \Omega_m} \{\lVert x_i - x_j \rVert\}}$	1	1	↑	[53]
3	L Value	$L(c) = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{c-1} u_{ij}^m \lVert \overline{x} - v_j \rVert^2/(c-1)}{\sum_{i=0}^{n-1}\sum_{j=0}^{c-1} u_{ij}^m \lVert x_i - v_j \rVert^2/(n-c)}$	1	1	↑	[54]
4	Minimum Centroid Distance	$MCD(c) = min_{i\neq j} \parallel v_i - v_j \parallel^2$	1	0	—	[55]
5	Compactness	$CP(c) = \frac{1}{c}\sum_{j=0}^{c-1}\frac{1}{n}\sum_{i=0}^{n-1} \parallel x_i - v_j \parallel$	0	1	↓	[53]
6	Davies-Bouldin Index	$DBI(c) = \frac{1}{c}\sum_{i=0}^{c-1}\max\limits_{j\neq i}\left(\frac{\overline{SP}_i + \overline{SP}_j}{\lVert v_i - v_j \rVert}\right)$	1	1	↓	[53]

D: Divergence (Segregation); C: Compactness; EC: Evaluation Criterion; '↑' means the greater. the better; '—' means that we should choose the point at which the recession curve becomes stable; and '↓' means the smaller, the better.

2.3.2. Evaluation Indices with Ground-Truth Labels

In addition to the evaluation of the relationships among elements and clusters, we also wanted to observe the gap between the clustering results and the results of human recognition. Table 17 introduces indices with ground-truth labels, which can be used to measure the similarity between the true values and the predicted values of the unsupervised models. If the true values were labeled by experienced railway operators, we believed that the indices could accurately measure the gap. The indices are illustrated in Table 17.

Table 17. Evaluation indices with ground-truth labels.

ID	Index Name	Notation	References
1	Cluster Accuracy	$CA = \sum_{i=1}^{K} \frac{\max(C_i\|L_i)}{\|\Omega\|}$	[53]
2	Adjusted Rand Index	$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$ where RI $= \frac{a+b}{C_2^{n\ samples}}$	
3	Normalized Mutual Information	$NMI = U(X,Y) = 2R = 2\frac{I(X,Y)}{H(X)+H(Y)}$ where $I(X,Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)}$ $H(X) = \sum_{i=1}^{n} p(x_i) I(x_i) = \sum_{i=1}^{n} p(x_i) \log_b \frac{1}{p(x_i)} = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$	

where $C_2^{n\ samples}$ represents the number of pairs in the dataset. The range of *CA* is $[0,1]$, the range of *ARI* is $[-1,1]$, and the range of *NMI* is $[0,1]$. The evaluation criteria of these three indices are the greater, the better.

2.3.3. Cluster Number Determination and Partial Data Labeling

Cluster number determination is the previous step of data labeling because of the lack of benchmark and prior knowledge. To find the optimal number of clusters, we trained the proposed deep learning models one by one under different cluster numbers and evaluated the results by the evaluation system discussed in Section 2.3.1.

Here, we used an example to elaborate the process of determining the number of clusters. To find the optimal number of clusters of a dataset named RD9 (its definition is given in Section 3), the five proposed deep learning models were trained with different cluster numbers n_c from a candidate set $\{5, 6, 7, \cdots, 13, 14, 15\}$. The settings of the models are shown in Table 18. Then, we obtained the indices illustrated in Figure 10.

Table 18. Example of settings for cluster number determination.

Index	Deep Models	Dataset [1]	Cluster Number
1.1	AAEC	RD9	$\{5, 6, 7, \cdots, 13, 14, 15\}$
1.2	DEC		
1.3	AAEKC		
1.4	SAECC		
1.5	AAECC		

[1] RD9. A dataset containing nine years (January 2009–January 2018) of integrated trajectories.

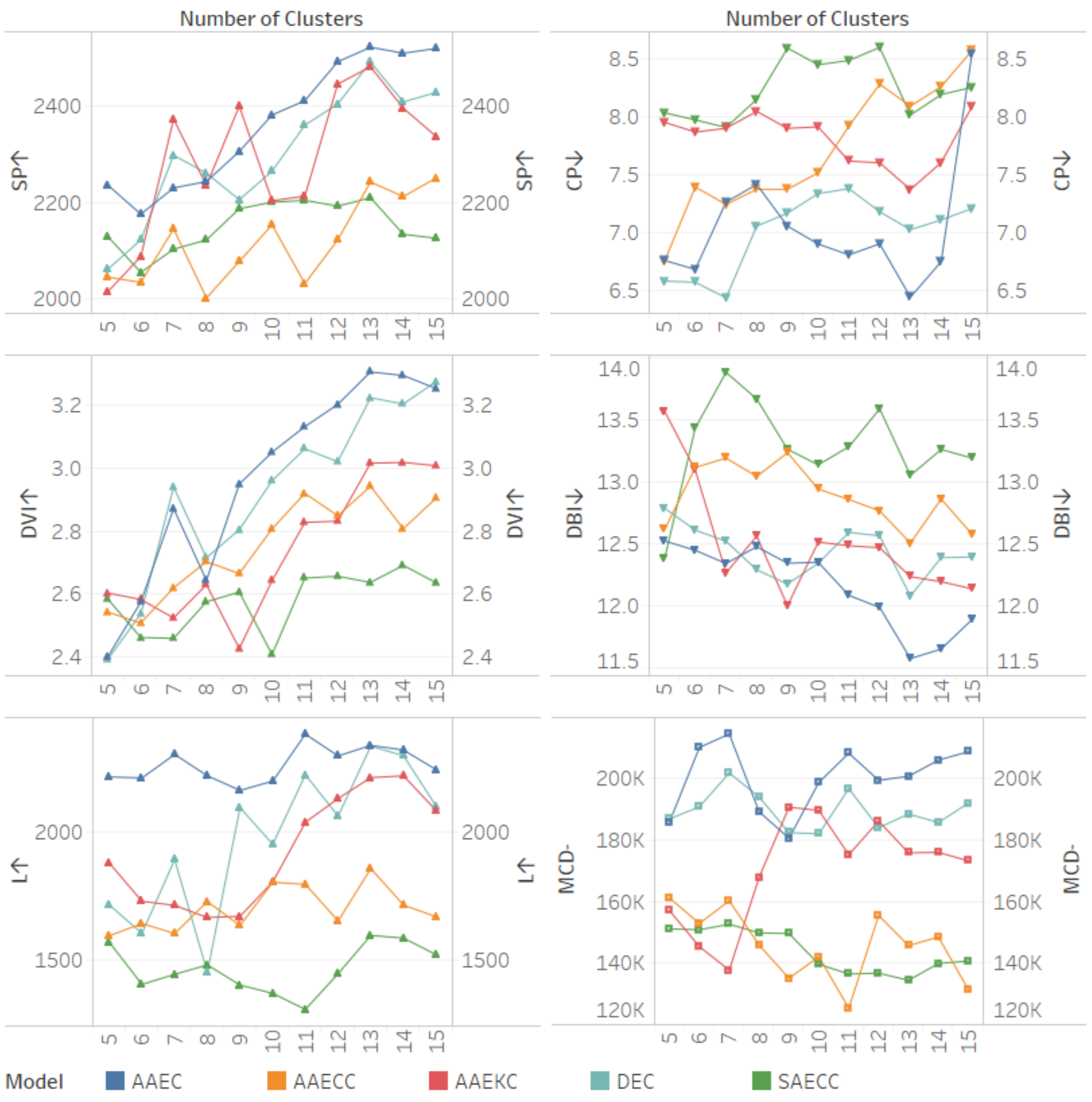


Figure 10. Cluster number determination by indices without ground-truth labels.

The optimal number of clusters that we were looking for is the number of clusters corresponding to the peaks of the *DVI*, *SP*, and *L* sequences and the valleys of the *CP* and performance difference (*DBI*) sequences. In addition, in the range around the optimal value, there should be a stable *MCD*. In Figure 10, when the cluster number was less than 13, the *DVI*, *SP*, and *L* sequences presented fluctuations in most of the five models. A peak was apparent at 13 clusters. After this point, these three indices began to decrease. On the other hand, the *CP* and *DBI* sequences had several valleys, for example, seven and 13 clusters for *CP* and nine and 13 clusters for *DBI*. Additionally, *MCD* began to stabilize after 12 clusters. Considering these indices together, we chose 13 clusters as the optimal cluster number.

After obtaining the optimal number of clusters, we invited experts from the railway system, including excellent dispatchers and train drivers, to manually label a portion of the data as the ground-truth labels.

3. Results and Discussion

A dataset of trajectories over a nine-year period (from January 2009 to January 2018) from a section (Dongsheng-Aobaogou) of the Baoshen Railway, Inner Mongolia, China was collected for numerical experiments. The data were recorded in an "event-by-event" manner, meaning that any change in state will add a new row of data. For the nine years of data, we obtained data for every six months. This dataset contained 44,282 trajectories (approximately 13.48 trajectories per day) produced by 113 locomotives, with four profile types (i.e., speed-location profile, gear location-location profile, signal-location profile, and time-profile profile).

Based on the actual data above, we defined three sub-datasets from the utilized dataset: (i) The RD2, a dataset containing two years (January 2009–January 2011) of integrated trajectories, (ii) the RD9, a dataset containing nine years (January 2009–January 2018) of integrated trajectories, (iii) the PRD9, a dataset containing nine years (January 2009–January 2018) of pure GPS trajectories, and (iv) the PRD2, a dataset containing two years (January 2009–January 2011) of pure GPS trajectories.

To further explore the performance characteristics of the proposed models, we wanted to introduce different types of datasets (including different data sizes and different mode distributions). However, due to the lack of a recognized benchmark, we artificially generated a series of datasets in which data were generated by simulated drivers (i.e., people simulating driving and producing output trajectories according to a series of instructions). Although these data were not as detailed as real data, they could be customized in size and type and could be used as important data for testing model performance. This study used artificial data accumulated over five years. Based on these artificial data, we defined the following datasets: (i) The AUBSD, artificial unbalanced small-scale dataset, contained artificially generated trajectories (small scale, i.e., 5000 trajectories), and the number of trajectories representing different modes varies significantly. (ii) The AUBLD, artificial unbalanced large-scale dataset, contained artificially generated trajectories (large scale, i.e., 40,000 trajectories), and the number of trajectories representing different modes varied significantly. (iii) The ABSD, artificial balanced small-scale dataset, contained artificially generated trajectories (small scale, i.e., 5000 trajectories), and the number of trajectories contained in different modes was approximately balanced. (iv) The ABLD, artificial balanced large-scale dataset, contained artificially generated trajectories (large scale, i.e., 40,000 trajectories), and the number of trajectories representing different modes was approximately balanced. After applying the preprocessing method discussed in Section 2.1, trajectories that were suitable for machine learning were obtained. Then, we conducted experiments as described in Section 3.1.

3.1. Experimental Settings

The numerical experiment consists of four steps. In step (i), we analyzed the mode distribution of a given dataset (RD9) according to the results of the cluster number determination (discussed in Section 2.3.3).

In step (ii), depending on the optimal number of clusters n_c, we analyzed the performances of the five proposed deep learning models and two comparison classical models on different datasets. These datasets were different in scale and mode distribution. The settings of this step are shown in Table 19, where 2.1–2.2 were the comparison models and 2.3–2.7 were the proposed models. The criterion for the comparison was the evaluation system discussed in Section 2.3.1. The results of step (ii) could indicate the performances of each model on different datasets.

Table 19. Settings of Step (ii).

Index	Feature Extractor	Cluster Model	Dataset [1]	Cluster Number
2.1	Railway-specified features [2]	Spectral Embedded Clustering [56]	AUBSD AUBLD ABSD ABLD RD9	Determined by the method in Section 2.3.3
2.2	Railway-specified features [2]	k-means [57]		
2.3	AAEC			
2.4	DEC			
2.5	AAEKC			
2.6	SAECC			
2.7	AAECC			

[1] The definition of each dataset can be found at the beginning of Section 3.

In step (iii), we tested the pure GPS trajectory data (from PRD9) and integrated trajectory data (from RD9). Based on the optimal number of clusters n_c and the performances of the five proposed models, we selected the top two models and trained them on different datasets; then we evaluated the corresponding results. The result of step (iii) could show the necessity of using integrated trajectory data. Additionally, by analyzing the results combined with those of previous steps, we could observe the detailed impact of integrated data on unsupervised learning. The settings of this step are shown in Table 20.

Table 20. Settings of Step (iii).

Index	Deep Models	Dataset [1]	Cluster Number
3.1	The top performing model	RD9	Determined by the method in Section 2.3.3
3.2	The second best performing model	PRD9	

[1] The definition of each dataset can be found at the beginning of Section 3.

In step (iv), we invited experts from the railway system, including excellent dispatchers and train drivers, to manually label the data in RD2 as the ground-truth labels. We trained the five proposed deep learning models on the integrated trajectory data (from RD2) as well as on pure GPS trajectory data (from PRD2); then, we evaluated the corresponding results by the index system discussed in Section 2.3.2. The results of step (iv) revealed (1) the gap between the unsupervised learning outcomes and the subjective recognition results, as well as (2) the differences in the training performances between pure GPS trajectory data and integrated trajectories. The settings of this step are shown in Table 21.

Table 21. Settings of Step (iv).

Index	Deep Models	Dataset [1]	Cluster Number
4.1	AAEC	RD2 PRD2	Determined by the method in Section 2.3.3
4.2	DEC		
4.3	AAEKC		
4.4	SAECC		
4.5	AAECC		

[1] The definition of each dataset can be found at the beginning of Section 3.

3.2. Results and Discussions

3.2.1. Result and Discussion of Step (i)

Figure 11 lists the found modes and the corresponding counts. From the 13 figures, we can see that the trajectories in one mode followed the same trend; this trend was also the embodiment of the driving phases. According to the definition of phases in [6], we could attempt to analyze the driving phases based on the mode. For example, in mode (a), the phase structure was A(42-44)-Cr(44-52)-B(52-53)-Cr(53-53.8)-B(53.8-54), and that of mode (b) was A(42,46)-Cr(46,49.5)-Co(49.5,52)-Cr(52,53.8)-B(53.8,54), where 'A' represents the acceleration phase, 'Cr' represents the cruising phase (maintaining speed with throttle manipulation), 'Co' represents the coasting phase (train operation while the throttle is idle before braking), and 'B' represents the braking phase (for a station stop, for speed restriction or for a restrictive signal). The numbers in brackets were approximate location ranges of the phases (km). From this application, the necessity of mode analysis could be seen. Additionally, we can see from (n) in Figure 11 that the distribution of modes in reality was not balanced; RD9 represents large-scale unbalanced data.

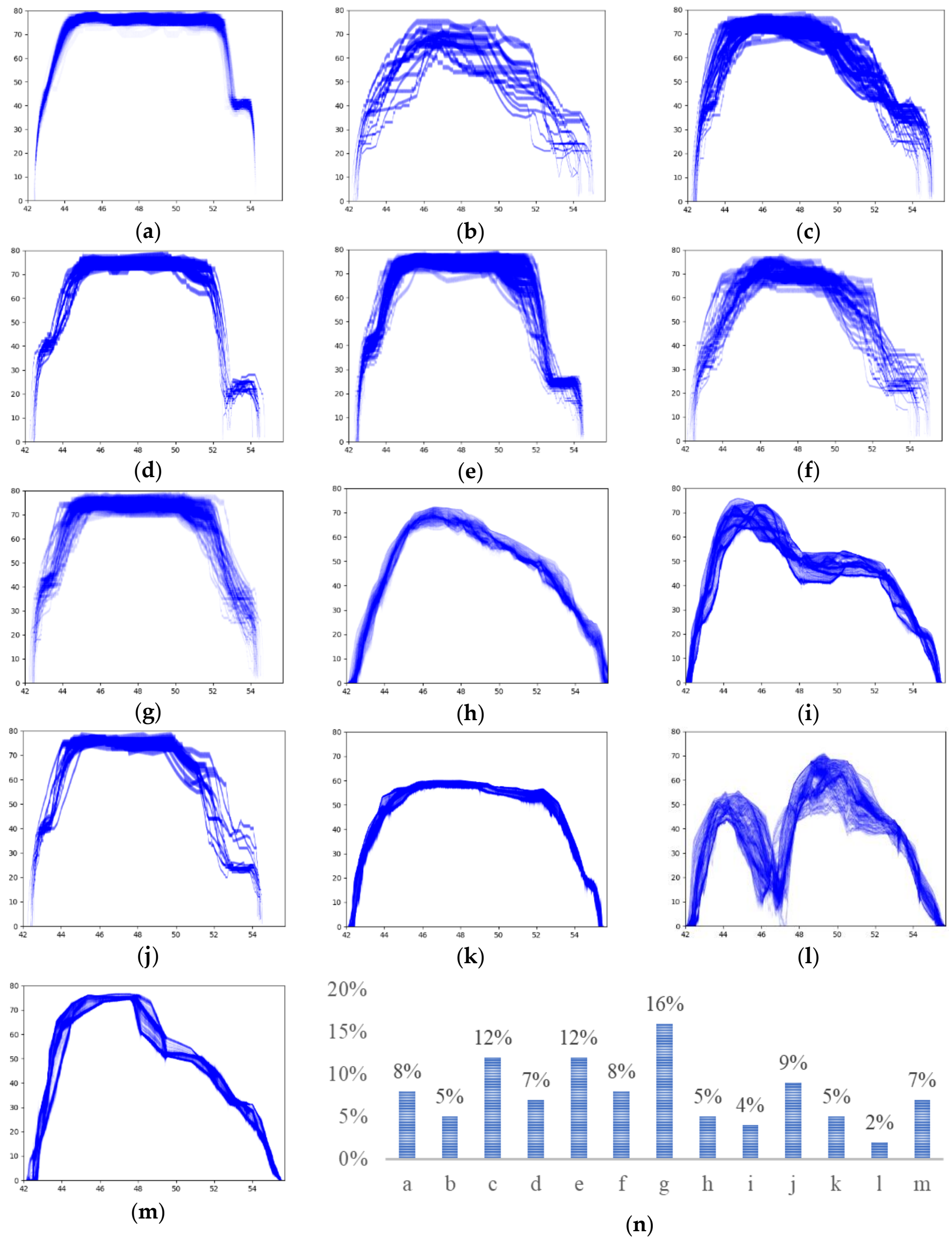


Figure 11. Illustration of modes in RD9. Figure (**a**–**m**) are images of the modes. Due to space limitations, we describe these modes in terms of speed-location profiles. Figure (**n**) is used to express the statistics of the data contained in each mode.

3.2.2. Result and Discussion of Step (ii)

We tested the two comparison models and five proposed models on the AUBSD, AUBLD, ABSD, ABLD, and RD9 datasets, where the results are shown in Figure 12. Focusing on differences in model performances on the datasets, we observed the following phenomena:

(1) The performances of almost all models were relatively high on the balanced datasets (ABSD and ABLD) and relatively low on the unbalanced datasets (AUBSD, AUBLD, and RD9). For example, *DVI*, *SP*, and *L* on the balanced datasets were larger than on the unbalanced datasets. The imbalance of data had a greater impact on classical models than the proposed deep learning models. This was expressed in the corresponding reduction in the indices: For example,

the *SP* of k-means was 2331.6 on the ABSD and 2025.0 on the AUBSD, a reduction of 13.1%. Correspondingly, the *SP* of AAEC was 2399.9 on the ABSD and 2253.1 on the AUBSD, decreasing by 6.1%. A similar situation existed for the other indices.

(2) By analyzing the performance of each model individually, we found that when the dataset was small, there was only a small difference between the performances of the models. As the scale increased, the performances of the models began to decrease. For example, the CatGAN-based models performed well on small datasets but less so on the large datasets. Classical models poorly handled the large datasets; their performances decreased significantly on the AUBLD and ABLD datasets. For example, on the AUBSD dataset, the *DBI* of SEC was 7.2. However, on the AUBLD, the *DBI* of SEC increased by 56.9% (11.3). In contrast, the deep learning methods, especially DEC and AAEC, still achieved relatively high performances on the large datasets, although they did show a decline.

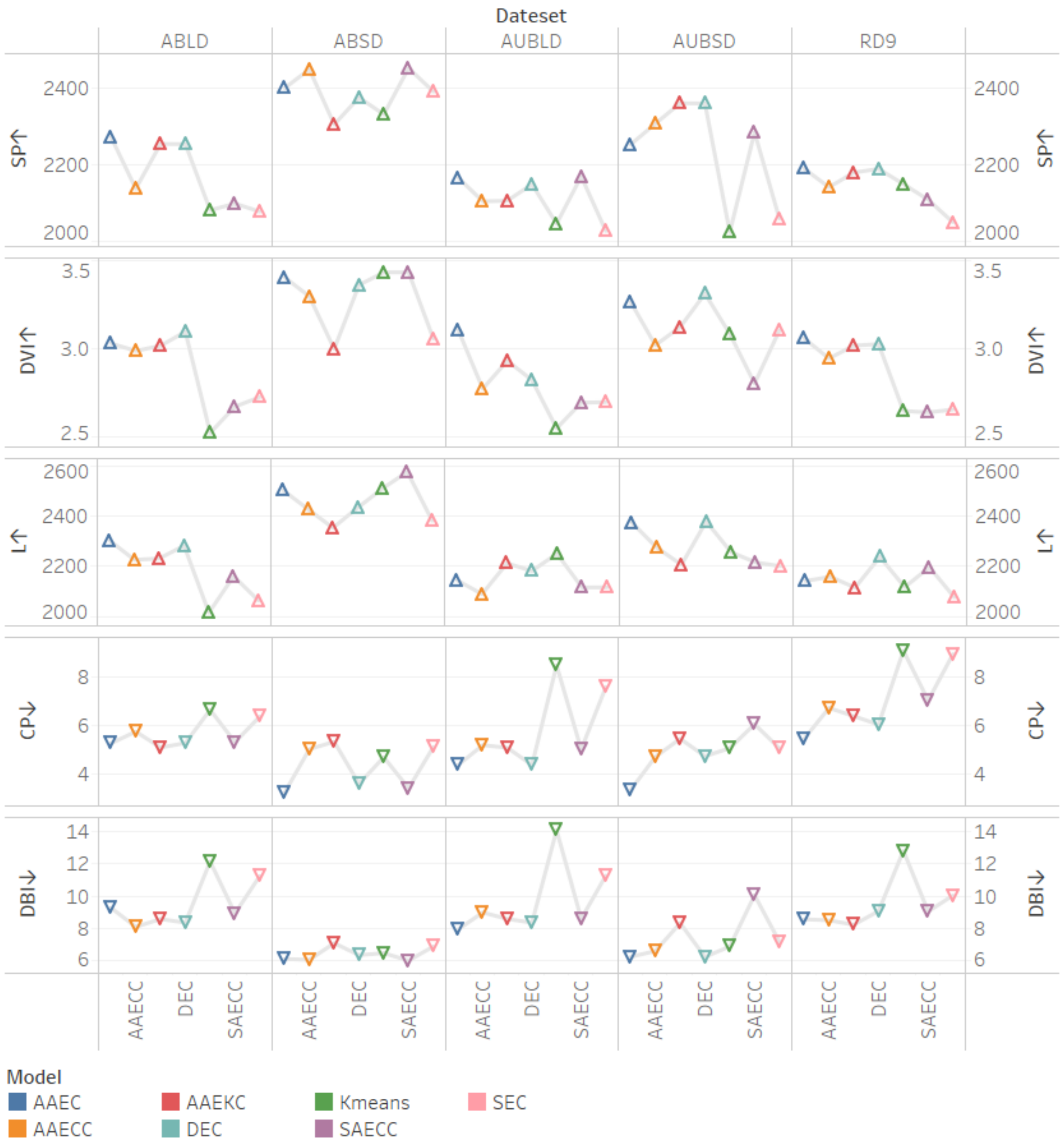


Figure 12. Illustration of evaluation indices of Step (ii).

In order to quantitatively compare the performance differences between the proposed models and the classical models. We used large-scale datasets as the analysis object and selected *DBI* as the index for analysis. From Table 22 we can see that the proposed models had a significant performance advantage over the classical models. Namely, indices of the proposed methods were better (lower) 32.94% (AUBLD), 26.01% (ABLD), and 23.95% (RD9) than the indices of the classical methods. In other word, the proposed deep learning models outperformed the classical models by 27.64% on average.

Table 22. Performance difference (DBI) between proposed models and classical models.

DBI	AAEC	DEC	AAEKC	SAECC	AAECC	k-Means	SEC
AUBLD	7.95	8.36	8.60	8.61	8.99	14.08	11.27
ABLD	9.28	8.36	8.59	8.94	8.11	12.15	11.25
RD9	8.58	9.08	8.24	9.06	8.51	12.81	10.05
			Average value				
AUBLD			8.50			12.68	
ABLD			8.66			11.70	
RD9			8.69			11.43	
			Performance improvement				
AUBLD				32.94%			
ABLD				26.01%			
RD9				23.95%			

In conjunction with Figure 10, we analyzed the performance differences between the proposed models. By analyzing the distribution of these indices shown in Figure 10, we could compare the performance of each method. Before 11 clusters, in almost all the indices, the performances of each model were close and low, and the rankings of each model fluctuated. After 11 clusters, the index sequences began to show more obvious stratification. Thus, we chose the indices after 11 clusters, especially the indices with 13 clusters (the optimal number of clusters), to analyze the performance rankings. From the perspectives of *DVI*, *SP*, and *L*, the top three models were the same: AAEC, DEC, and AAEKC. This ranking was also found for *CP* and *DBI*. In a comprehensive analysis of each index, the models' rankings in this dataset were AAEC, DEC, AAEKC, SAECC, and AAECC.

This table contains three rows. The first row is the *DBI* values of each model in different datasets, the second one is the average *DBI* value of the proposed models and the classic models, and the third one is the performance improvement of the proposed models in different datasets.

3.2.3. Result and Discussion of Step (iii)

The difference in the training performance between the pure GPS trajectory data and the integrated trajectory data could be seen by analyzing Figure 13, which illustrates the results of Step (iii). From the values of *DVI*, *SP*, *L*, *CP*, and *DBI*, we found that the two models on the RD9 dataset outperformed those on the PRD9 dataset. For example, by calculating the differences in *DBI*, we found that performances of models on RD9 were 13.78% better than on PRD9 on average. Combining the results of Step (i) (shown in Figure 11), we deduced how the integrated data affect unsupervised learning.

The integrated data enhanced the performance of unsupervised learning by providing more details about the driving operations. For example, the driving belonging to modes (d) and (e) in Figure 11 presents different enter-station behaviors: Mode (d) shows an obvious re-acceleration process (phase), rather than the cruising process (phase) in mode (e). However, these two modes cannot be clearly identified using pure GPS trajectory data, which were misidentified as belonging to the same cluster. Although pure GPS trajectory data could be used for unsupervised mode learning, their accuracy was far lower than integrated trajectory data.

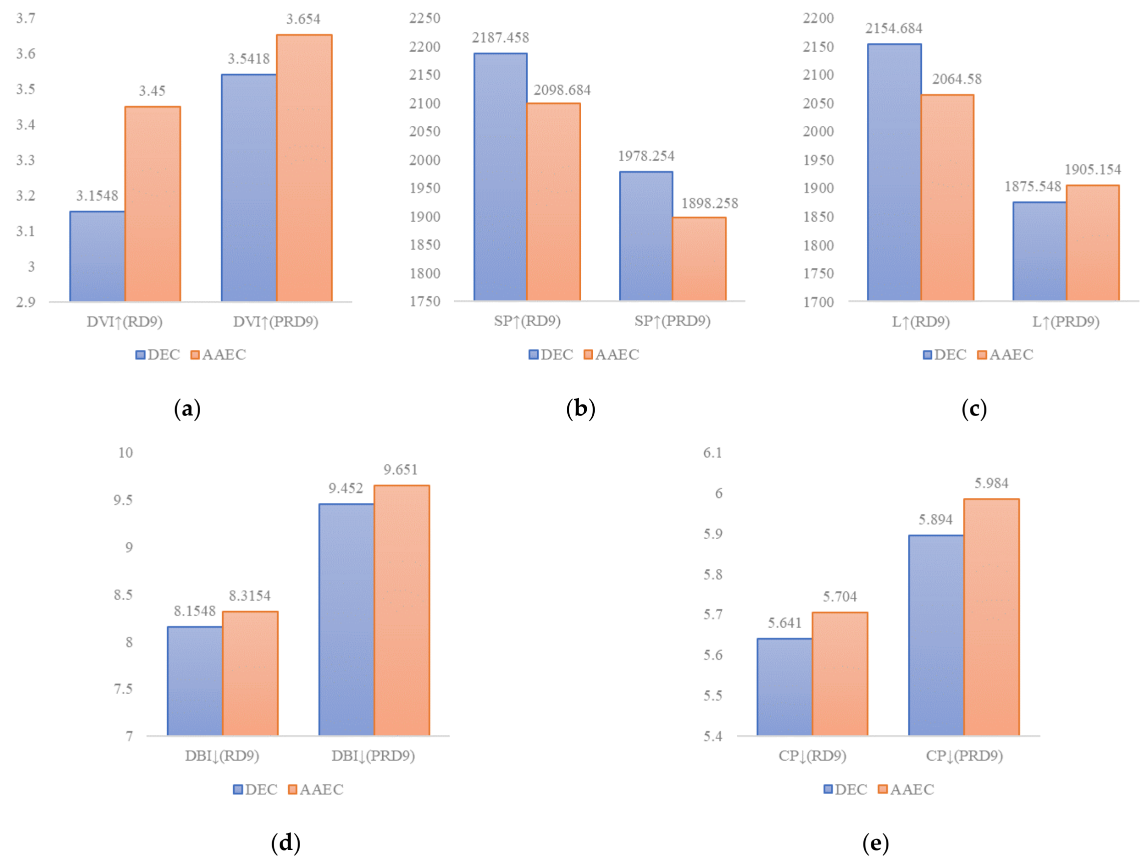
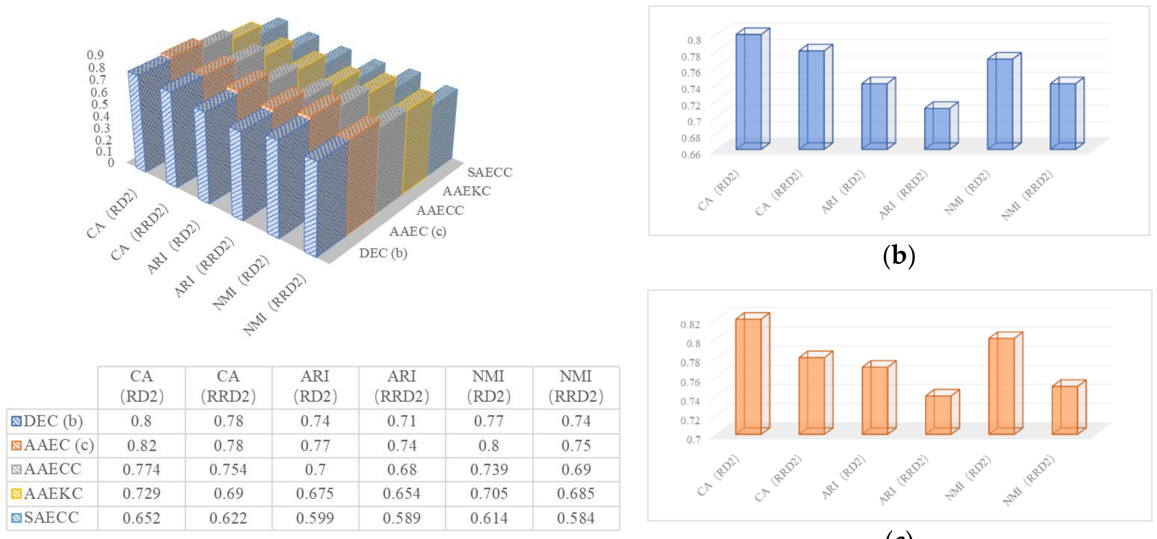


Figure 13. Illustration of evaluation indices of Step (iii). The first row (from (**a**–**c**)) illustrates distributions of *DVI*, *SP*, and *L*, and the second row (from (**d**–**e**)) illustrates those of *DBI* and *CP*.

3.2.4. Result and Discussion of Step (iv)

In Step (iv), we used the evaluation indices with ground-truth labels discussed in Section 2.3.2. Although the absolute value of these indices cannot be used to evaluate the performances of the models, the relative values of these indices can be used to measure the gap between the unsupervised learning outcomes and the subjective recognition results. Specifically, the higher (tending to 100%) the three indices are, the closer the unsupervised learning outcomes to the subjective recognition results.

Panel (a) of Figure 14 is the distributions of the indices (CA, ARI, and NMI) on the RD2 and PRD2 datasets. The performances of the models are presented in the same order, regardless of the index or dataset. The ranking of the models was DEC-AAEC-AAEKC-SAECC-AAECC, where DEC had a clear advantage over AAEC and AAEKC. This ranking differed from Step (i) and Step (ii) in the ranking of DEC and AAEC. In addition, even the model that performed best (DEC, with $CA = 0.82$, $ARI = 0.77$, and $NMI = 0.80$) still showed a large gap when compared to subjective recognition. From these two results, it could be seen that people's understanding of the driving mode was certainly different from the results obtained by machine learning, and the cause of this phenomenon might be the insufficiency of people's understanding of the MDRTs (because DEC, which had worse indices than AAEC in Step (i) and Step (ii), outperformed AAEC in this step) or failures existing in the labeling processes.



	CA (RD2)	CA (RRD2)	ARI (RD2)	ARI (RRD2)	NMI (RD2)	NMI (RRD2)
▨DEC (b)	0.8	0.78	0.74	0.71	0.77	0.74
▨AAEC (c)	0.82	0.78	0.77	0.74	0.8	0.75
▨AAECC	0.774	0.754	0.7	0.68	0.739	0.69
▨AAEKC	0.729	0.69	0.675	0.654	0.705	0.685
▨SAECC	0.652	0.622	0.599	0.589	0.614	0.584

(**a**) (**b**) (**c**)

Figure 14. Evaluation indices with ground-truth labels: (**a**) The indices of the five proposed models on the RD2 and PRD2 datasets. (**b**) The indices of the DEC model on the RD2 and PRD2 datasets. (**c**) The indices of the AAEC model on the RD2 and PRD2 datasets.

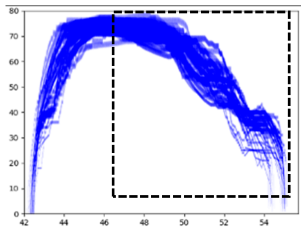
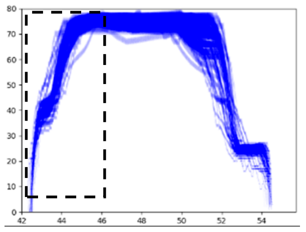
The result of Step (iv) also indicated the necessity of using integrated trajectory data. Picking the two top-performing models (i.e., DEC and AAEC), we present slices of the indices in (b) and (c) of Figure 14. We find that the indices of RD2 were better than those of PRD2. In the process of labeling data, we did not require that experts limit the type of information to be referenced; experts may only label the modes based on experience or based on some technical indicators. From the results shown in Figure 14, the performances of the models using integrated trajectory data were better than those using pure GPS data. This indicates that in processing during daily operation, experienced drivers would decide their own driving modes based on comprehensive information, not simply on time and space information.

3.3. The Relationship between Discovered MDRT and Railway Carrying Capacity

After discussing the performances of the five proposed models in conducting unsupervised MDRT learning tasks, we provided further discussions about the obtained MDRTs themselves. From the 13 obtained modes, we picked two modes (i.e., mode (c) and mode (e)), illustrated in Table 22. Using the analysis method from [6], we analyzed the impacts of these two modes on railway carrying capacity.

Mode (c) adopts the strategy of first accelerating and then decelerating: Rapid acceleration after exiting the station, reaching the speed limit of 44 km; decelerating in advance, with smooth operation upon entering the station; and finally no re-acceleration. Mode (e) accelerates slowly when exiting, reaches the speed limit of 45 km, cruises longer, and brakes later. The braking force is large, and there is re-acceleration when entering the station. The main phases of the operation of the two modes are shown in Table 23. The two modes had different impacts on the carrying capacities of the two stations and the section. The first mode produced a smaller occupancy of the station departure capacity but achieved a greater occupancy of the section carrying capacity and the station arrival capacity. The second mode, on the other hand, achieved a larger occupancy for the station departure capacity but a smaller occupancy for the section carrying capacity. In daily operation, the station-section-station carrying capacity adjustment was realized through these types of flexible behavior combinations. This is also the hidden nature that had been noticed previously; however, it lacked quantitative analysis.

Table 23. Capacity utilization analysis samples of MDRTs.

Index	Mode	Phases (with Location Range km-km) [1]	Speed-Location Profiles [2]
1	mode (c)	A(42,43)-Cr(43,43.5)-A(43.5,45)-Cr(45,49.8)-Co(49.8,52)-Cr(52,53.8)-B(53.8,54)	
2	mode (e)	A(42,45)-Cr(45,51)-B(51,52)-Cr(52,53.8)-B(53.8,54)	

[1] We referred to the phase system in [6], where 'A' represents acceleration, 'Cr' represents cruising (maintaining speed with throttle manipulation), 'Co' represents coasting (train operation while the throttle is idle before braking), and 'B' represents braking (for a station stop, for speed restriction or for a restrictive signal). [2] The boxes in the images are the main areas of capacity utilization.

4. Conclusions

In this paper, we proposed an unsupervised deep learning approach consisting of five models to learn MDRTs from integrated trajectories of in-operation railway trains. From the results of the four steps of the designed experiment, we drew the following conclusions: (i) The proposed deep learning models outperformed the classical models by 27.64% on average. AAEC achieved the best performance on balanced and unbalanced datasets according to the results of Steps (ii)–(iii). This may be related to the design idea of implicitly expressing the sample distribution. (ii) Integrated trajectory data could improve the accuracy of unsupervised learning by approximately 13.78%. (iii) Under the index system with labeled data and the index system without labeled data, the performance rankings of the proposed models were different, demonstrating the insufficiency of people's understanding of existing modes. In addition to the learning performance analysis, we also conducted a simple analysis of the relationship between modes and railway carrying capacity.

The discovery of driving modes will help us to understand the structure of the subjective uncertainty facing railway systems and thus guide capacity utilization optimization and automatic train driving algorithm design. The proposed approach was applied to the capacity utilization optimization and new driver training of the Baoshen railway. The approach enabled the construction of a MDRTs collection whereby a multivariate parameter set of running times was obtained so that operators could make distinct plans for different situations. New drivers are also now able to learn different modes and enhance their ability to address different situations. In the course of this research, we found that the model did not work well on unbalanced datasets. This is also one of the problems that we will overcome in the future.

Author Contributions: Conceptualization, Han Zheng; Data curation, Han Zheng; Resources, Han Zheng; Software, Zanyang Cui; Supervision, Xingchen Zhang; Visualization, Han Zheng and Zanyang Cui; Writing—original draft, Han Zheng; Writing—review & editing, Han Zheng.

Funding: The research was supported by the National Natural Science Foundation of China (Grant No. U1734204).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. Parameter Tuning Process of AAEC.

(i)

AAEC	0 [1]	0	0	0	0	1	2
Encoder &decoder (Layer1&2)	1500.0 [2] (1500)	1000.6 (1000)	2001.8 (2001)	1243.9 (1243)	2901.8 (2901)	1743.0 (1743)	2981.0 (2981)
	3101.6 (3101)	3890.0 (3890)	3120.7 (3120)	3000.0 (3000)	3081.0 (3081)	3500.0 (3500)	3181.0 (3181)
D1 (Layer1&2)	2001.0 (2001)	2301.0 (2301)	3001.0 (3001)	2201.0 (2201)	1001.0 (1001)	2391.0 (2391)	1031.0 (1031)
	3210.0 (3210)	3110.7 (3110)	2210.0 (2210)	3210.0 (3210)	2210.0 (2210)	3210.0 (3210)	2290.2 (2290)
D2 (Layer1&2)	2108.0 (2108)	2138.8 (2138)	3108.0 (3108)	2148.7 (2148)	1108.0 (1108)	2248.7 (2248)	1208.0 (1208)
	3980.8 (3980)	3080.0 (3080)	3080.8 (3080)	3280.8 (3280)	1980.8 (1980)	3180.8 (3180)	1080.8 (1080)
DVI	0.98	1.21	1.82	1.01	1.76	1.21	1.65

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(ii)

AAEC	3 [1]	4	5	6	7	8	9
Encoder &decoder (Layer1&2)	1700.0 [2] (1700)	1921.6 (1921)	2101.0 (2101)	2890.0 (2890)	3000.8 (3000)	3102.0 (3102)	2781.0 (2781)
	3201.6 (3201)	3490.0 (3490)	3120.7 (3120)	3020.0 (3020)	3000.3 (3000)	3500.0 (3500)	2881.0 (2881)
D1 (Layer1&2)	2101.0 (2101)	2801.0 (2801)	3013.0 (3013)	3091.2 (3091)	3000.9 (3000)	2980.0 (2980)	3131.0 (3131)
	3310.0 (3310)	3210.7 (3210)	3810.0 (3810)	3010.0 (3010)	3000.1 (3000)	3010.0 (3010)	3290.2 (3290)
D2 (Layer1&2)	2908.2 (2908)	2038.8 (2038)	2308.8 (2308)	2848.7 (2848)	3000.8 (3000)	3048.7 (3048)	3208.6 (3208)
	3780.8 (3780)	3280.0 (3280)	2080.8 (2080)	3180.8 (3180)	3000.5 (3000)	3280.8 (3280)	2080.8 (2080)
DVI	1.90	2.39	2.01	2.89	3.01	2.90	2.65

[1] Iteration; [2] Number of neurons (the value converted to type "int").

Table A2. Parameter Tuning Process of DEC.

(i)

DEC	0 [1]	0	0	0	0	1	2
Encoder &decoder (Layer1&2&3)	1000.0 [2] (1000)	1010.6 (1010)	2001.8 (2001)	2243.9 (2243)	1901.8 (1901)	1843.0 (1843)	2081.0 (2081)
	2101.6 (2101)	1890.0 (1890)	2120.7 (2120)	1090.0 (1090)	3821.0 (3821)	3570.0 (3570)	3081.0 (3081)
	1920.9 (1920)	2018.0 (2018)	2098.7 (2098)	1237.9 (1237)	3089.3 (3089)	2002.2 (2002)	1986.6 (1986)
DVI	1.32	1.01	0.82	1.10	1.26	1.31	0.98

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(ii)

DEC	3 [1]	4	5	6	7	8	9
Encoder &decoder (Layer1&2&3)	1790.0 [2] (1790)	2021.6 (2021)	2101.0 (2101)	1890.0 (1890)	1301.8 (1301)	1000.7 (1000)	1200.0 (1200)
	2201.6 (2201)	2490.0 (2490)	1820.7 (1820)	1320.0 (1320)	1031.0 (1031)	1000.4 (1000)	1012.3 (1012)
	2087.2 (2087)	1301.7 (1301)	2910 (2910)	2012.3 (2012)	1532.0 (1532)	2000.8 (2000)	1021.4 (1021)
DVI	1.58	1.86	2.30	2.70	2.96	3.13	2.90

[1] Iteration; [2] Number of neurons (the value converted to type "int").

Table A3. Parameter Tuning Process of AAEKC.

(i)

AAEKC	0 [1]	0	0	0	0	1	2
Encoder &decoder (Layer1&2)	1200.0 [2] (1200)	1900.6 (1900)	3401.8 (3401)	1243.9 (1243)	2201.8 (2201)	2043.0 (2043)	2481.0 (2481)
	3001.6 (3001)	3090.0 (3090)	3920.7 (3920)	2000.0 (2000)	3381.0 (3381)	3300.0 (3300)	3187.0 (3187)
D (Layer1&2)	2001.0 (2001)	3201.0 (3201)	3601.0 (3601)	2101.0 (2101)	3201.0 (3201)	3391.0 (3391)	3131.0 (3131)
	2210.0 (2210)	3010.7 (3010)	2010.0 (2010)	3010.0 (3010)	1210.0 (1210)	3190.0 (3190)	3290.2 (3290)
DVI	1.21	1.00	1.93	1.32	1.52	2.31	2.87

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(ii)

AAEKC	3 [1]	4	5	6	7	8	9
Encoder &decoder (Layer1&2)	2900.0 [2] (2900)	3021.6 (3021)	3000.0 (3000)	3190.0 (3190)	3030.8 (3030)	3182.0 (3182)	3381.0 (3381)
	3001.6 (3001)	3090.0 (3090)	3000.3 (3000)	3002.0 (3002)	3231.0 (3231)	3100.0 (3100)	2981.0 (2981)
D (Layer1&2)	2808.8 (2808)	3138.8 (3138)	3000.7 (3000)	3108.3 (3108)	3312.8 (3312)	3001.7 (3001)	3020.8 (3020)
	3380.8 (3380)	3080.0 (3080)	3000.8 (3000)	3100.2 (3100)	3187.8 (3187)	3180.8 (3180)	3000.8 (3000)
DVI	2.89	2.91	3.00	2.98	2.81	2.90	2.85

[1] Iteration; [2] Number of neurons (the value converted to type "int").

Table A4. Parameter Tuning Process of SAECC.

SAECC	**0** [1]	**0**	**0**	**0**	**0**	**1**	**2**
Encoder &decoder (Layer1&2&3)	1100.0 [2] (1100)	1000.6 (1000)	2901.8 (2901)	1043.9 (1043)	2201.8 (2201)	1503.0 (1503)	1481.0 (1481)
	2000.6 (2000)	1090.0 (1090)	2720.7 (2720)	1700.0 (1700)	1081.0 (1081)	2300.0 (2300)	2187.0 (2187)
	2890.7 (2890)	2970.7 (2970)	2086.0 (2086)	2098.6 (2098)	2365.8 (2365)	2543.0 (2543)	2290.0 (2290)
G (Layer1&2)	501.7 (501)	590.6 (590)	890.6 (890)	1000.6 (1000)	753.8 (753)	600.6 (600)	800.8 (800)
	689.6 (689)	708.7 (708)	1087.8 (1087)	1342.6 (1342)	1006.7 (1006)	1109.0 (1109)	1090.0 (1090)
D (Layer1& 2&3&4&5)	1201.0 (1201)	2901.0 (29001)	2601.0 (2601)	2101.0 (2101)	2801.0 (2801)	2901.0 (2901)	2131.0 (2131)
	2010.0 (2010)	2990.7 (2990)	2010.0 (2010)	2010.0 (2010)	1210.0 (1210)	2990.0 (2990)	2590.2 (2590)
	1021.9 (1021)	1000.0 (1000)	2012.7 (2012)	892.0 (892)	2000.8 (2000)	1002.7 (1002)	842.0 (842)
	2001.9 (2001)	501.8 (501)	3512.3 (3512)	722.0 (722)	3021.0 (3021)	1730.9 (1730)	1687.0 (1687)
	3019.0 (3019)	801.9 (801)	1092.9 (1092)	523.9 (523)	2712.0 (2712)	1420.8 (1420)	1021.9 (1021)
DVI	0.65	0.89	0.93	0.82	0.70	1.01	0.98

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(i)

SAECC	**3** [1]	**4**	**5**	**6**	**7**	**8**	**9**
Encoder &decoder (Layer1&2&3)	1000.0 [2] (1000)	1900.6 (1900)	1501.8 (1501)	1243.9 (1243)	1201.8 (1201)	1000.0 (1000)	1001.0 (1001)
	2001.6 (2001)	2890.0 (2890)	2020.7 (2020)	2000.0 (2000)	1381.0 (1381)	1000.8 (1000)	1187.0 (1187)
	2210.7 (2210)	2098.0 (2098)	2346.0 (2346)	2008.9 (2008)	1908.8 (1908)	2000.1 (2000)	2406.0 (2406)
G (Layer1&2)	700.6 (700)	600.0 (600)	654.8 (654)	598.0 (598)	500.6 (500)	500.8 (500)	510.7 (510)
	976.6 (976)	987.0 (987)	1000.0 (1000)	998.7 (998)	1020.9 (1020)	1000.0 (1000)	1050.4 (1050)
D (Layer1& 2&3&4&5)	1001.0 (1001)	1201.0 (1201)	1080.0 (1080)	1101.0 (1101)	1301.0 (1301)	1000.7 (1000)	1031.0 (1031)
	810.0 (810)	510.7 (510)	608.0 (608)	810.0 (810)	600.0 (600)	500.6 (500)	690.2 (690)
	1001.9 (1001)	800.0 (800)	912.7 (912)	702.0 (702)	500.8 (500)	250.7 (250)	842.0 (842)
	1701.9 (1701)	401.8 (401)	512.3 (512)	682.0 (682)	521.0 (521)	250.9 (250)	1687.0 (1687)
	2019.0 (2019)	701.9 (701)	902.9 (902)	623.9 (623)	512.0 (512)	250.8 (250)	1021.9 (1021)
DVI	1.21	1.30	1.88	2.09	2.32	2.43	2.27

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(ii)

Table A5. Parameter Tuning Process of AAECC.

AAECC	0 [1]	0	0	0	0	1	2
Encoder &decoder (Layer1&2&3)	1000.0 [2] (1000)	1300.6 (1300)	1901.8 (1901)	1243.9 (1243)	2201.8 (2201)	1203.0 (1203)	1281.0 (1281)
	1000.6 (1000)	1190.0 (1190)	2020.7 (2020)	1000.0 (1000)	2081.0 (2081)	1300.0 (1300)	1807.0 (1807)
	2090.7 (2090)	2070.7 (2070)	2986.0 (2986)	1098.6 (1098)	2065.8 (2065)	1543.0 (1543)	2090.0 (2090)
G2 (Layer1&2)	701.7 (701)	790.6 (790)	1090.6 (1090)	900.6 (900)	1093.8 (1093)	680.6 (680)	700.8 (700)
	889.6 (889)	1008.7 (1008)	887.8 (887)	1042.6 (1042)	806.7 (806)	1309.0 (1309)	1190.0 (1190)
D1 (Layer1&2)	3201.0 (3201)	2901.0 (2901)	1601.0 (1601)	2101.0 (2101)	2801.0 (2801)	1901.0 (1901)	2931.0 (2931)
	3010.0 (3010)	1990.7 (1990)	3010.0 (3010)	2010.0 (2010)	2210.0 (2210)	1990.0 (1990)	2990.2 (2990)
D2 (Layer1& 2&3&4&5)	2001.8 (2001)	3018.2 (3018)	2123.0 (2123)	2001.8 (2001)	1023.6 (1023)	1203.0 (1203)	2201.0 (2201)
	1082.8 (1082)	672.9 (672)	3238.2 (3238)	1023.8 (1023)	789.8 (789)	568.0 (568)	1023.9 (1023)
	1203.8 (1203)	2312.0 (2312)	2012.7 (2012)	788.9 (788)	1004.9 (1004)	1003.0 (1003)	989.0 (989)
	2031.8 (2031)	1003.8 (1003)	3129.0 (3129)	1002.4 (1002)	765.3 (765)	891.0 (891)	765.0 (765)
	3321.8 (3321)	1024.8 (1024)	1292.8 (1292)	560.7 (560)	3231.0 (3231)	2001.9 (2001)	1652.0 (1652)
DVI	0.75	0.79	0.90	0.96	0.80	0.91	1.02

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(i)

AAECC	3 [1]	4	5	6	7	8	9
Encoder &decoder (Layer1&2&3)	1100.0 [2] (1100)	1300.6 (1300)	1201.8 (1201)	1000.9 (1000)	1001.8 (1001)	1030.5 (1030)	1481.0 (1481)
	1000.6 (1000)	1190.0 (1190)	1200.7 (1200)	1000.0 (1000)	1001.0 (1001)	1000.1 (1000)	2287.0 (2287)
	2490.7 (2490)	2070.7 (2070)	2186.0 (2186)	2000.6 (2000)	2015.8 (2015)	2000.5 (2000)	2090.0 (2090)
G2 (Layer1&2)	901.7 (901)	980.6 (980)	890.6 (890)	500.6 (500)	553.8 (553)	500.6 (500)	600.8 (600)
	1089.6 (1089)	1008.7 (1008)	1000.8 (1000)	1000.6 (1000)	1106.7 (1106)	1000.2 (1000)	1290.0 (1290)
D1 (Layer1&2)	3201.0 (3201)	2901.0 (2901)	2801.0 (2801)	3000.0 (3000)	3201.0 (3201)	3000.8 (3000)	2631.0 (2631)
	3010.0 (3010)	2890.7 (2890)	3010.0 (3010)	3000.9 (3000)	3010.0 (3010)	3000.7 (3000)	2590.2 (2590)
D2 (Layer1& 2&3&4&5)	1201.8 (1201)	1018.2 (1018)	1023.0 (1023)	1000.8 (1000)	1023.6 (1023)	1000.1 (1000)	1201.0 (1201)
	820.8 (820)	972.9 (972)	838.2 (838)	500.8 (500)	689.8 (689)	520.8 (520)	623.9 (623)
	880.0 (880)	1327.0 (1327)	708.0 (708)	250.9 (250)	230.8 (230)	200.8 (200)	432.9 (432)
	700.8 (700)	800.3 (800)	500.9 (500)	250.8 (250)	300.9 (300)	280.9 (280)	320.9 (320)
	1023.9 (1023)	680.0 (680)	300.8 (300)	250.0 (250)	276.8 (276)	230.4 (230)	302.8 (302)
DVI	1.60	1.78	2.07	2.35	2.32	2.30	2.01

[1] Iteration; [2] Number of neurons (the value converted to type "int").

(ii)

References

1. Bešinović, N. Integrated Capacity Assessment and Timetabling Models for Dense Railway Networks. Ph.D. Dissertation, Delft University of Technology, Delft, The Netherlands, 2017.
2. Zheng, H.; Cui, Z.; Zhang, X. Identifying modes of driving railway trains from gps trajectory data: An ensemble classifier-based approach. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 308. [CrossRef]
3. Bešinović, N.; Goverde, R.M.P. *Capacity Assessment in Railway Networks*; Springer: Berlin, Germany, 2018.
4. Besinovic, N.; Roberti, R.; Quaglietta, E.; Cacchiani, V.; Toth, P.; Goverde, R.M.P. Micro-macro approach to robust timetabling. In Proceedings of the International Seminar on Railway Operations Modelling and Analysis, Tokyo, Japan, 23–26 March 2015.
5. Goverde, R.M.P. A delay propagation algorithm for large-scale railway traffic networks. *Transp. Res. Part C Emerg. Technol.* **2010**, *18*, 269–28711. [CrossRef]
6. Longo, G.; Medeossi, G.; Nash, A. Estimating train motion using detailed sensor data. In Proceedings of the Transportation Research Board 91st Annual Meeting, Washington, DC, USA, 22–26 January 2012; pp. 1–6.
7. Medeossi, G.; Longo, G.; Fabris, S.D. A method for using stochastic blocking times to improve timetable planning. *J. Rail Transp. Plan. Manag.* **2011**, *1*, 1–13. [CrossRef]
8. Bešinović, N.; Goverde, R.M.P.; Quaglietta, E.; Roberti, R. An integrated micro–macro approach to robust railway timetabling. *Transp. Res. Part B Methodol.* **2016**, *87*, 14–32. [CrossRef]
9. Zhou, L.; Tong, L.; Chen, J.; Tang, J.; Zhou, X. Joint optimization of high-speed train timetables and speed profiles: A unified modeling approach using space-time-speed grid networks. *Transp. Res. Part B Methodol.* **2017**, *97*, 157–181. [CrossRef]
10. Cacchiani, V.; Toth, P. Nominal and robust train timetabling problems. *Eur. J. Oper. Res.* **2012**, *219*, 727–737. [CrossRef]
11. Goerigk, M.; Schöbel, A. Recovery-to-optimality: A new two-stage approach to robustness with an application to aperiodic timetabling. *Comput. Oper. Res.* **2014**, *52*, 1–15. [CrossRef]
12. Chen, J.; Zhang, X.; Cai, H.; Zheng, Y. A monitoring data mining based approach to measuring and correcting timetable parameters. *Procedia-Soc. Behav. Sci.* **2012**, *43*, 644–652. [CrossRef]
13. Wang, J.; Zhang, X.; Yi, Z.; Chen, J. Method for the measurement and correction of train diagram parameters based on monitoring data mining. *China Railw. Sci.* **2011**, *32*, 117–121.
14. Xiao, Z.; Wang, Y.; Fu, K.; Wu, F. Identifying different transportation modes from trajectory data using tree-based ensemble classifiers. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 57. [CrossRef]
15. Fabris, S.D.; Longo, G.; Medeossi, G. Automated Analysis of Train Event Recorder Data to Improve Micro-Simulation Models. In Proceedings of the COMPRAIL 2010 Conference, Beijing, China, 31 August–2 September 2010; pp. 575–583.
16. Powell, J.P.; Palacín, R. Driving style for ertms level 2 and conventional lineside signalling: An exploratory study. *Ing. Ferrov.* **2016**, *71*, 927–942.
17. Goverde, R.M.P.; Daamen, W.; Hansen, I.A. Automatic identification of route conflict occurrences and their consequences. *Comput. Railw. XI* **2008**, *103*, 473–482.
18. Albrecht, T.; Goverde, R.M.P.; Weeda, V.A.; Luipen, J.V. Reconstruction of Train Trajectories from Track Occupation Data to Determine the Effects of a Driver Information System. In Proceedings of the COMPRAIL 2006 Conference, Prague, Czech Republic, 31 August–2 September 2006; pp. 207–216.
19. Dodge, S.; Weibel, R.; Forootan, E. Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Comput. Environ. Urban Syst.* **2009**, *33*, 419–434. [CrossRef]
20. Elhoushi, M.; Georgy, J.; Noureldin, A.; Korenberg, M. Online motion mode recognition for portable navigation using low-cost sensors. *Navigation* **2016**, *62*, 273–290. [CrossRef]
21. Schuessler, N.; Axhausen, K.W. Processing gps raw data without additional information. *Transp. Res. Rec.* **2009**, *2105*, 28–36. [CrossRef]
22. Zheng, Y.; Liu, L.; Wang, L.; Xie, X. Learning transportation mode from raw gps data for geographic applications on the web. In Proceedings of the International Conference on World Wide Web, WWW 2008, Beijing, China, 21–25 April 2008; pp. 247–256.
23. Wagner, D.P. *Lexington Area Travel Data Collection Test: Gps for Personal Travel Surveys*; Battelle Transporation Division: Columbus, OH, USA, 1997; pp. 1–92.

24. Yalamanchili, L.; Pendyala, R.; Prabaharan, N.; Chakravarthy, P. Analysis of global positioning system-based data collection methods for capturing multistop trip-chaining behavior. *Transp. Res. Rec. J. Transp. Res. Board* **1999**, *1660*, 58–65. [CrossRef]
25. Draijer, G.; Kalfs, N.; Perdok, J. Global positioning system as data collection method for travel research. *Opt. Express* **2000**, *1719*, 147–153. [CrossRef]
26. Wolf, J.L. Using Gps Data Loggers to Replace Travel Diaries in the Collection of Travel Data. Ph.D. Dissertation, Georgia Institute of Technology, School of Civil and Environmental Engineering, Atlanta, GA, USA, 2000; pp. 58–65.
27. Reddy, S.; Min, M.; Burke, J.; Estrin, D.; Hansen, M.; Srivastava, M. Using mobile phones to determine transportation modes. *ACM Trans. Sens. Netw.* **2010**, *6*, 1–27. [CrossRef]
28. Stenneth, L.; Wolfson, O.; Yu, P.S.; Xu, B. Transportation mode detection using mobile phones and gis information. In Proceedings of the ACM Sigspatial International Symposium on Advances in Geographic Information Systems, Acm-Gis 2011, Chicago, IL, USA, 1–4 November 2011; pp. 54–63.
29. Widhalm, P.; Nitsche, P.; Brändle, N. Transport mode detection with realistic smartphone sensor data. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 11–15 November 2012; pp. 573–576.
30. Das, R.D.; Winter, S. Detecting urban transport modes using a hybrid knowledge driven framework from gps trajectory. *Int. J. Geo-Inf.* **2016**, *5*, 207. [CrossRef]
31. Mardia, K.V.; Jupp, P.E. *Directional Statistics*; Wiley: Hoboken, NJ, USA, 2000.
32. Zheng, Y.; Li, Q.; Chen, Y.; Xie, X.; Ma, W.Y. Understanding mobility based on gps data. In Proceedings of the International Conference on Ubiquitous Computing, Seoul, Korea, 21–24 September 2008; pp. 312–321.
33. Deng, H.; Runger, G.; Tuv, E.; Vladimir, M. A time series forest for classification and feature extraction. *Inf. Sci.* **2013**, *239*, 142–153. [CrossRef]
34. Zhang, J.; Wang, Y.; Zhao, W. An improved hybrid method for enhanced road feature selection in map generalization. *Int. J. Geo-Inf.* **2017**, *6*, 196. [CrossRef]
35. Qian, H.; Lu, Y. Simplifying gps trajectory data with enhanced spatial-temporal constraints. *Int. J. Geo-Inf.* **2017**, *6*, 329. [CrossRef]
36. Ma, C.; Zhang, Y.; Wang, A.; Wang, Y.; Chen, G. Traffic command gesture recognition for virtual urban scenes based on a spatiotemporal convolution neural network. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 37. [CrossRef]
37. Gonzalez, P.A.; Weinstein, J.S.; Barbeau, S.J.; Labrador, M.A.; Winters, P.L.; Georggi, N.L.; Perez, R. Automating mode detection using neural networks and assisted gps data collected using gps-enabled mobile phones. In Proceedings of the 15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting, New York, NY, USA, 16–20 November 2008.
38. Maaten, L. Learning a parametric embedding by preserving local structure. In *Artificial Intelligence and Statistics*; Springer: Berlin, Germany, 2009; pp. 384–391.
39. Dabiri, Z.; Lang, S. Comparison of independent component analysis, principal component analysis, and minimum noise fraction transformation for tree species classification using apex hyperspectral imagery. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 488. [CrossRef]
40. Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **2010**, *11*, 3371–3408.
41. Xie, J.; Girshick, R.; Farhadi, A. Unsupervised deep embedding for clustering analysis. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 478–487.
42. Makhzani, A.; Shlens, J.; Jaitly, N.; Goodfellow, I.J.C.S. Adversarial autoencoders. *arXiv* **2015**, arXiv:1511.05644v2.
43. Mukherjee, S.; Asnani, H.; Lin, E.; Kannan, S. Clustergan: Latent space clustering in generative adversarial networks. *arXiv*, 2018; arXiv:1809.03627v2.
44. Springenberg, J.T. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv*, 2015; arXiv:1511.06390v2.
45. Lee, J.G.; Han, J.; Li, X. Trajectory outlier detection: A partition-and-detect framework. In Proceedings of the IEEE International Conference on Data Engineering, Cancun, Mexico, 7–12 April 2008.
46. Goldin, D.Q.; Kanellakis, P.C. *On Similarity Queries for Time-Series Data: Constraint Specification and Implementation*; Springer: Berlin, Germany, 1995.

47. Bergstra, J.; Bengio, Y. Algorithms for hyper-parameter optimization. In Proceedings of the International Conference on Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; pp. 2546–2554.
48. Hutter, F.; Hoos, H.H.; Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In Proceedings of the Learning and Intelligent Optimization—International Conference, Lion 5, Rome, Italy, 17–21 January 2011; pp. 507–523.
49. Thornton, C.; Hutter, F.; Hoos, H.H.; Leytonbrown, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, Chicago, IL, USA, 11–14 August 2013; pp. 847–855.
50. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. *Adv. Neural Inf. Proc. Syst.* **2012**, *4*, 2951–2959.
51. Goodfellow, I.J. Nips 2016 tutorial: Generative adversarial networks. *arXiv*, 2016; arXiv:1701.00160v4.
52. Yi, Y.; Dong, X.; Feiping, N.; Shuicheng, Y.; Yueting, Z. Image clustering using local discriminant models and global integration. *IEEE Trans. Image Proc.* **2010**, *19*, 2761–2773.
53. Fahad, A.; Alshatri, N.; Tari, Z.; Alamri, A.; Khalil, I.; Zomaya, A.Y.; Foufou, S.; Bouras, A. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *Emerg. Top. Comput. IEEE Trans.* **2014**, *2*, 267–279. [CrossRef]
54. Li, Y.; Yu, F. A new validity function for fuzzy clustering. In Proceedings of the International Conference on Computational Intelligence and Natural Computing, Wuhan, China, 6–7 June 2009; pp. 462–465.
55. Zhang, K.; Sun, D.; Shen, S.; Zhu, Y. Analyzing spatiotemporal congestion pattern on urban roads based on taxi gps data. *J. Transp. Land Use* **2017**, *10*, 675–694. [CrossRef]
56. Nie, F.; Zeng, Z.; Tsang, I.W.; Xu, D.; Zhang, C. Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering. *IEEE Trans. Neural Netw.* **2011**, *22*, 1796–1808.
57. Hartigan, J.A.; Wong, M.A. *A K-Means Clustering Algorithm: Algorithm as 136*; Wiley: Hoboken, NJ, USA, 1979; Volume 28, pp. 100–108.