electronics

MDPI

Article

Business Process Outcome Prediction Based on Deep Latent Factor Model

Ke Lu [1], Xinjian Fang [2,*] and Xianwen Fang [1,3]

1 School of Mathematics and Big Data, Anhui University of Science and Technology, Huainan 232001, China; 2019100059@aust.edu.cn (K.L.); xwfang@aust.edu.cn (X.F.)
2 School of Surveying and Land Information Engineering, Anhui University of Science and Technology, Huainan 232001, China
3 Anhui Province Engineering Laboratory for Big Data Analysis and Early Warning Technology of Coal Mine Safety, Huainan 232001, China
* Correspondence: fangxinjian@aust.edu.cn

Abstract: Business process outcome prediction plays an essential role in business process monitoring. It continuously analyzes completed process events to predict the executing cases' outcome. Most of the current outcome prediction focuses only on the activity information in historical logs and less on the embedded and implicit knowledge that has not been explicitly represented. To address these issues, this paper proposes a Deep Latent Factor Model Predictor (DLFM Predictor) for uncovering the implicit factors affecting system operation and predicting the final results of continuous operation cases based on log behavior characteristics and resource information. First, the event logs are analyzed from the control flow and resource perspectives to construct composite data. Then, the stack autoencoder model is trained to extract the data's main feature components for improving the training data's reliability. Next, we capture the implicit factors at the control and data flow levels among events and construct a deep implicit factor model to optimize the parameter settings. After that, an expansive prefix sequence construction method is proposed to realize the outcome prediction of online event streams. Finally, the proposed algorithm is implemented based on the mainstream framework of neural networks and evaluated by real logs. The results show that the algorithm performs well under several evaluation metrics.

Keywords: Deep Latent Factor Model; business process outcome prediction; stacked autoencoder model; predictive process monitoring

check for updates

Citation: Lu, K.; Fang, X.; Fang, X. Business Process Outcome Prediction Based on Deep Latent Factor Model. *Electronics* **2022**, *11*, 1509. https://doi.org/10.3390/electronics11091509

Academic Editor: Antoni Morell

Received: 4 April 2022
Accepted: 3 May 2022
Published: 8 May 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Process mining is a discipline that analyzes historical event logs to discover process models, detect actual versus expected deviations, and thus optimize business process structures [1]. As the research interest in process mining continuously rises, the traditional a posteriori review can no longer meet the growing real-time demand. In this context, predictive business process monitoring has been proposed to provide operational support regarding detection, prediction, and recommendation [2].

The goal of predictive business process monitoring is to continuously predict the future state of a business system concerning current ongoing events and historical information [3], and an outcome-oriented predictive approach is one of the most intuitive tasks [4]. With the help of outcome-oriented predictive process monitoring techniques, managers can obtain relevant recommendations while the system is running. For example, whether the solution will be accepted, whether the customer is satisfied with the service quality, and which travel route will be chosen by the customer. Current approaches to outcome prediction mainly analyze the relationship between events in the process from the control-flow perspective [5,6]. Some studies have also incorporated attributes related to events to enrich the information of the researched object [4,7].

Realistically, the factors that affect the results are not limited to the explicit attributes embodied in the logs, such as users, execution time, and resources involved. Some latent factors that have not been explicitly quantified and analyzed can also have a non-negligible impact on the results, such as local policies and the mental state of the process executors. We consider the latent factor model to extract the implicit factors in the historical information to address this issue. The latent factor model is a collaborative filtering algorithm originating from recommender systems [8]. The latent factor model is widely used in product recommendation and movie rating prediction. It builds on the consensus that each user has their preferences. Each item also contains the preference information of all users. High user ratings for items indicate that the preference information contained in the item is exactly the information that the user prefers. This preference information cannot be identified obviously, and it is considered the factor that potentially influences the user's rating of the item, i.e., the latent factors. We introduce the latent factor model, which considers that the case occurs because it matches the supposed preference information of the event log and thus explores the prediction method under the action of the latent factor.

Inspired by the studies mentioned above, this paper proposes a Deep Latent Factor Model method Predictor (DLFM Predictor) to predict business process outcomes based on historical event behavior and data information. We realize this method in offline and online phases:

(1) Construct the deep latent factor model in the offline phase. First, analyze the processing activity and resource sequences from the control and data flow perspectives. Then they are mapped into a composite training data set, and the critical features of the training set are analyzed using a stacked autoencoder model. Finally, a deep latent factor model is constructed, updated, and optimized on the training set.

(2) Predict business process outcome in the online phase. This phase introduces a sliding window mechanism that caches the running online event streams as a prefix sequence (including the activity and resources of the event stream). We can then predict the process outcome using the model optimized in the offline phase.

The other sections of the article are structured as follows. We present some closely related research work in Section 2. Section 3 briefly introduces some of the notations, concepts, and theories involved. Section 4 breaks down each algorithm step in detail, followed by an evaluation of our algorithm using real logs in Section 5. Finally, Section 6 concludes this paper and presents plans for future work.

2. Related Works

The literature on process outcome prediction analyzes logs based on a known set of categorical labels to predict the outcome of a process by grouping real-time events into different subgroups. Maggi et al. constructed a decision tree based on historical execution information, reduced by user-defined linear time logic rules during prediction [9]. This method reduces the analysis space and predicts the case's outcome on time. Leoni et al. proposed a generic framework for classification/regression problems to predict process outcomes by considering multilevel features such as control, data, and organization [10]. Francescomarino et al. considers the data properties of event sequences and traces. It clusters each prefix of the historical traces in the offline part and then matches the real-time cases with the clusters to complete the prediction in a classification way [11]. The complexity of prefix computation is alleviated by limiting the prefix length to 21 and interval fetching. Leontjeva et al. proposed two complex coding sequences based on feature vectors by training classifiers to give two or more predictions for running cases [12].

With the development of deep learning, the prediction problem has received more attention. Many neural network-based methods have been introduced to the prediction process monitoring scenario. Metzger et al. extended the application of recurrent neural networks (RNN) by considering the sequential form of the process and incorporating recurrent and parallel structures for outcome prediction [5]. Pasquadibisceglie combined computer vision with process monitoring to predict outcomes [13]. The method represents the feature vector of the current trace in the form of an image, from which convolutional neural networks in computer vision can be introduced to obtain the prediction results in image processing.

Many studies manually captured the features of the process without differentiating the importance of the features. Wang et al. used the Att-Bi-LSTM algorithm, which incorporates a bidirectional long- and short-term memory network and an attention mechanism to capture representative features [4]. This method avoids manual screening of log features and can reduce subjective bias.

While the above methods consider outcome prediction as supervised learning, Folino et al. show a particular case [7]. The method proposes a semi-supervised approach that achieves outcome prediction on partially labeled datasets through a fine-tuning strategy that applies parameters obtained from training on unlabeled complete logs to the outcome prediction model.

The latent factor model is a model-based collaborative filtering algorithm in recommender systems [8], which has received much attention due to its high accuracy and scalability in implicit semantic analysis. Cheng extended the latent factor model to an aspect-aware latent factor model in predicting movie ratings [14]. They focus on the latent factors related to ratings and analyze the reviews related to ratings by aspect-aware topic models to address limitations such as cold starts and opacity. When constructing latent factor models on existing data, using stochastic gradient descent to optimize the model is a common means, and the selection of learning rate plays an essential role in this stage. Qingxian Wang et al. used the particle swarm optimization principle to automatically search for the learning rate, thus constructing a word use latent factor model to select the optimal learning rate [15].

In order to extract latent factors from high-dimensional sparsity with missing information, Luo et al. proposed a new intrinsic nonlinear factor model, inherently, an NLF model [16]. In comparison, Wu et al. added a data feature-aware module to the latent factor model for achieving high accuracy prediction under noisy data [17]. In recent years, the rapid development of deep learning has inspired scholars such as Mongia and Wu. The former proposed a deep latent factor model for the recommendation system [18]. Since the high computational complexity is due to the high dimensionality and sparsity of the user and item matrices, the latter constructed the deep latent factor model by connecting the latent factor models sequentially through nonlinear activation functions [19]. They replaced the neural network structure in the depth structure with the LFM, which allows multilayer latent factor models. These methods have high expressiveness and improve computational efficiency very well.

3. Preliminaries

In this section, we briefly introduce some domain knowledge and involved notations related to the study of this paper.

3.1. Event Log

Let A denote an active set, and a sequence consisting of elements $a_i, a \in A$ in the set is denoted as $\sigma =< a_1, a_2, \ldots, a_n >$, which length is n. If only some of the elements in σ are needed, we call it a subsequence and it is marked as $\sigma' \ =< a_i, \ldots, a_j >, i \le j \wedge i, j \in [0, |n|]$. In particular, when $i = j = 0$, the sequence $\sigma' \ =< a_i, \ldots, a_j >, i \le j \wedge i, j \in [0, |n|]$ is an empty set, denoted as ϵ. We use the symbol $\cdot$ to denote the concatenation operation. For example, if $\sigma_1 =< a, b >$and $\sigma_2 =< c, d >$, we can get the connection of σ_1 and σ_2 as $\sigma' = \sigma_1 \cdot \sigma_2 =< a, b, c, d >$, and the new sequence obtained is the original two sequence elements connected. The set formed by all these sequences is noted as A^*. The function $fr_\sigma(a_i)$ can express the frequency of each element in the sequence σ.

The prefix is used to extract the first activity of the sequence, formalized as follows.

Definition 1. *Prefixes [20]*

The prefix $f_{pre}^{(k)}(\sigma)$ of a sequence $\sigma =< a_1, a_2, \ldots, a_n >$ is a non-empty subsequence of length k, denoted as follows:

$$f_{pre}^{(k)}(\sigma) = \begin{cases} < a_1, \ldots, a_k >, & if\ 0 < k < |\sigma| \\ \sigma, & if\ k \geq |\sigma| \end{cases}$$

When an activity a is triggered in a system, it is usually accompanied by related information about resources and roles data. These are collectively called attributes and denoted by $attr$. We denote each activity trigger, and the accompanying attribute information by the symbol $e = (a, attr)$ called an event. Traces are denoted as sequences in which each activity trigger is recorded, i.e., $\sigma =< e_1, e_2, \ldots, e_n >$. Moreover, all the traces recorded during the execution of the system together constitute the event log L.

3.2. latent Factor Model

Latent factor models are often used to recommend favorite items for users. For example, when analyzing users' ratings of movies, the latent factor model assumes that explicit factors such as movie genre, actors, and directors, and some implicit features influence users' ratings.

We explain the latent factor model from a mathematical perspective. We take the analysis of movie rating data, a widely used scenario for recommendation systems, as an example. The user's rating of a movie is noted as $x_{i,j}$, and the latent factor model is analyzed by modeling an inner product of the user's latent factor u_i and the movie's latent factor of v_i, i.e., $x_{i,j} = u_i \cdot v_j, \forall i, j$ [18]. The latent factor model can be transformed into a matrix decomposition problem.

If all the data in the scoring matrix were known, the decomposition process would be fast and straightforward. However, the observed information is usually incomplete, making matrix decomposition less intuitive. The latent factor model can infer the missing ratings by minimizing the deviation of the inner product of the latent factors from the actual observed values and recommending the most likely favorite movies to the users.

This idea can also work in prediction scenarios, where the latent factor model is used to obtain the best recommendation about the label to obtain prediction results.

The idea based on the LFM model can be fitted with the product $\hat{M}_{CE}$ of the case matrix P_C and the event matrix Q_E [18]. To investigate the influence of potential factors on the occurrence of cases and behavior, we consider that there are K hidden factors influencing the cases and events, so the behavior matrix is further decomposed as follows.

$$\hat{M}_{CE} = P_C Q_E = \sum_{k=1}^{K} P_{C,k} Q_{k,E} \tag{1}$$

The goal of the training is to minimize the value of $\hat{M}_{CE}$ from the actual matrix M_{CE} value, where a loss function is used to quantify the deviation between the fitted and actual values.

Definition 2. *Loss function*

$$Cost = \sum_{(C,E)\in K} \left(M_{CE} - \hat{M}_{CE}\right)^2 = \sum_{(C,E)\in K} \left(M_{CE} - \sum_{k=1}^{K} P_{C,k} Q_{k,E}\right)^2 \tag{2}$$

In the actual training process, overfitting often occurs. A standard method is to add a regular term to the loss function, at which point the loss function is

$$Cost = \sum_{(C,E)\in K} \left(M_{CE} - \hat{M}_{CE}\right)^2 + \lambda\|P_C\|^2 + \lambda\|Q_E\|^2 = \sum_{(C,E)\in K} \left(M_{CE} - \sum_{k=1}^{K} P_{C,k} Q_{k,E}\right)^2 + \lambda\|P_C\|^2 + \lambda\|Q_E\|^2 \tag{3}$$

Therefore, the latent factor model's training objective is converted to minimize the loss function.

Definition 3. *Objective function*

$$\min : Cost = \min : \sum_{(C,E)\in K} \left(M_{CE} - \hat{M}_{CE}\right)^2 + \lambda\|P_C\|^2 + \lambda\|Q_E\|^2 \tag{4}$$

To optimize the objective function, we use gradient descent to find the fastest optimization direction by taking partial derivatives of the objective function, which gives us

$$\frac{\partial \operatorname{Cos} t}{\partial P_{CE}} = -2\left(M_{CE} - \sum_{k=1}^{K} P_{C,k}Q_{k,E}\right)Q_{KE} + 2\lambda P_{CK} \tag{5}$$

$$\frac{\partial \operatorname{Cos} t}{\partial Q_{KE}} = -2\left(M_{CE} - \sum_{k=1}^{K} P_{C,k}Q_{k,E}\right)P_{CK} + 2\lambda Q_{KE} \tag{6}$$

4. Deep Latent Factor Model for Predicting the Outcome

The prediction problem can often be viewed as a classification problem [21]. Our goal is to infer which outcome the process ultimately boils down to based on the available historical information. There are many realistic scenarios with various explicit or implicit factors that affect the system. Explicit factors include information recorded in the logs, such as activities, resources, and runtime. Implicit factors are not recorded in the logs, have not been thoroughly studied, or cannot be explicitly described in words or formulas for the time being, such as policies, biases, regulations, laws, and ethics that affect the operation of processes.

As shown in Figure 1, this section explores the potential latent factors in the historical event logs to perform real-time prediction of running cases. We divide the algorithm into two phases, offline and online, which are described in four subsections.

1. **Preprocess logs in the offline phase.**
   In Section 4.1, we preprocess the historical event logs and construct the case–event matrix. Then, in Section 4.2, we use the self-encoder to reduce its dimensionality.
2. **Predict event flow in the online phase.**
   In Section 4.3, we construct and train latent factor models based on matrix decomposition techniques and gradient descent methods.
   Finally, in Section 4.4, the model is used to run the real-time prediction of event streams.

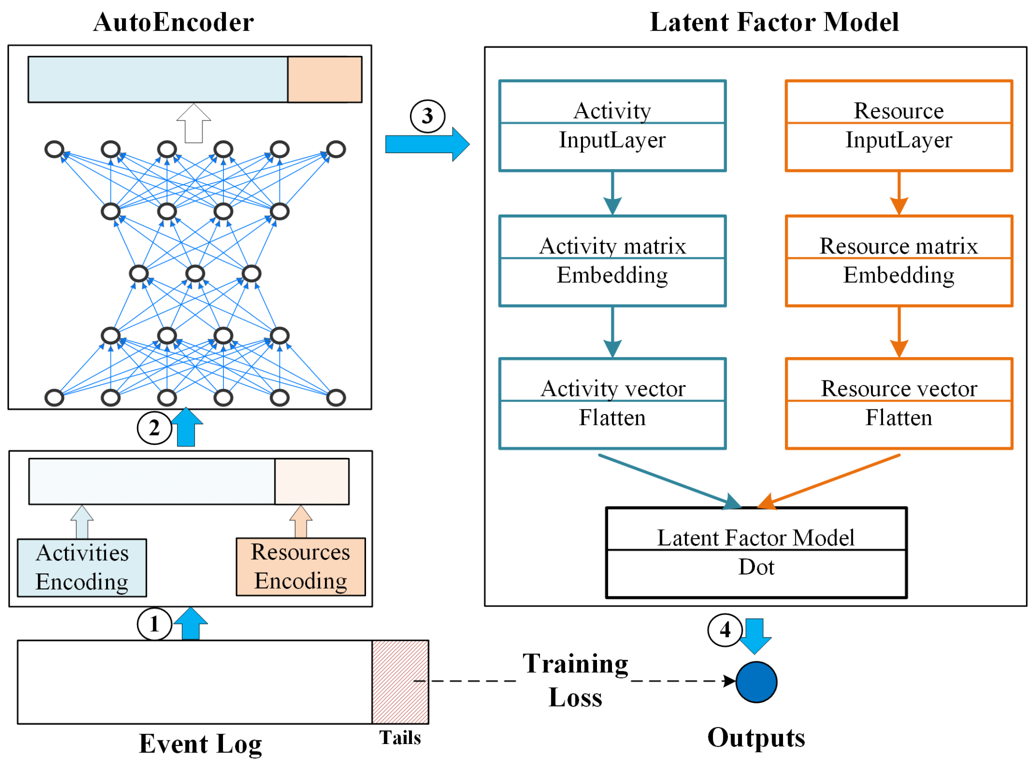

Figure 1. Business Process Outcome Prediction Framework.

4.1. Constructing the Input Log Vector

Predictive models gain insights into the future operational state by analyzing the training set. A good training set is a basis for obtaining an accurate training model. Therefore, we first preprocess the null values in the event log, the training set of this paper, to improve its quality. Subsequently, the event logs are constructed into a case–event matrix to extract the data's essential features.

Since our study aims to predict the run results as early as possible, it is necessary to consider each run trace's activity and attribute information in log L. However, there may be misrecording and omission when recording the logs, so we use Lagrange interpolation to preprocess the numerical-type attributes before constructing the case-activity/attribute matrices.

Numerical missing value processing: Many scenarios contain some kind of intrinsic connection or law between different objects. Lagrangian interpolation can find unknown data by known discrete data when a physical quantity is observed. It is one of the standard algorithms for numerical analysis in mathematics and has been widely used in the data preprocessing stage (e.g., filling missing values) in data science in recent years. Given a log L, for each trace t, x denotes the attribute's location, which is reflected in the log as the ordinal number of the trace, and y denotes the attribute value corresponding to the current x. The filling of missing values is done by applying a Lagrangian interpolation polynomial.

Definition 4. *Lagrangian interpolation formula*

Given a polynomial function with known $k+1$ points: $(x_0, y_0), \ldots, (x_k, y_k)$. Assuming that any two x are not identical to each other, the Lagrangian interpolation polynomial obtained by applying the Lagrangian interpolation formula is

$$\mathcal{L}(x) = \sum_{j=0}^{k} y_j \mathfrak{L}_j(x)$$

In the above equation,

$$\mathfrak{L}_j(x) = \prod_{i=0, i\neq j}^{k} \frac{x-x_i}{x_j-x_i} = \frac{(x-x_0)}{(x_j-x_0)} \cdots \frac{(x-x_{j-1})}{(x_j-x_{j-1})} \frac{(x-x_{j+1})}{(x_j-x_{j+1})} \cdots \frac{(x-x_k)}{(x_j-x_k)}$$

For missing values, the calculation results can be obtained directly by substituting them into the Lagrangian interpolation polynomial.

Resource information processing in logs: We consider different text contents as different data classes for character-based attributes. In order to input them into the algorithm for quantitative analysis, they need to be mapped to numeric type data. Directly mapping character-based data into the corresponding indexes in the order of occurrence can achieve our goal. However, it is not the best solution because there is no size relationship between categories. In natural language processing and machine learning, a common means is to use one-hot encoding to map character-based data into vectors with the same size and only one non-zero value.

The definition of the encoding function is first introduced to map characters to unique integers, creating a bijection between characters and integers.

Definition 5. *Encoding function* [22]

Let X denote all the values that x may pick, i.e., $x \in X, y \in 1, 2, \ldots, |X|$, *then the encoding function can be defined as*

$Map(x) : x \mapsto y$

Definition 6. *One-hot encoding function*

Let Y denote all the values that y may pick, and the One-hot encoding function is $Onehot(y) : y \mapsto \vec{V}$, *which encodes y as a vector of length* $|Y|$, *denoted as*

$$Onehot(y) = \vec{V} = \{v_1, ..., v_i, ..., v_{\max(Y)}\} \tag{7}$$

$$v_i = \begin{cases} 1, & iff\ i = y \\ 0, & otherwise \end{cases}$$

For attribute set $Attr = \{a_1, a_2, \ldots, a_n\}$, after One-hot encoding, each element (character type of attributes) in the set is represented as $a_1 = (1, 0, \ldots, 0),\ |a_1| = n,\ a_2 = (0, 1, \ldots, 0),\ |a_2| = n,\ a_n = (0, 0, \ldots, 1),\ |a_n| = n.$

Event information processing in logs: In the study of business process monitoring, it is common to analogize logs to text in natural language processing, traces to statements, and activities to words. Therefore, it can use the statement processing techniques in natural language processing to analyze log activities.

Here we detail how to use activity vectors to extract feature vectors for event logs.

Definition 7. *n-gram bag of words Given a trace* $T =< a_1, a_2, \ldots, a_{N+n-1} >$ *of an event log L, the n-gram form of T is defined as follows.*

$$T_{n-gram} = (\sigma_1, \sigma_2, \ldots, \sigma_N) \tag{8}$$

where $\sigma_i =< a_i, a_{i+1}, \cdots, a_{i+n} >$, $|\sigma_i| = n$. *The symbol n is the coefficient in the n-gram.*

For each n, T is divided into N n-grams. Assuming that T has a total of φ unique activities, the number of all possible n-grams of T is $(|\varphi|^n)$ [23].

When $n - gram$ encoding is used, the order relationship between elements is ignored. For example, when $n = 1$, the value in the vector-only indicates whether the element occurs and how often it occurs. However, sequential, cyclic, and concurrent relationships exist between each event in the log. We define the activity vector based on $n - gram$ encoding to compensate for this representation bias.

Definition 8. *Activity vector*

Based on the n-gram encoding, setting n to 2, the set of all direct dependency pairs appearing in the log L is denoted as DS. For the direct dependency between activities a and b in the trace, denoted as $ds_{(x,y)}$, its activity vector is defined as follows.

$$V_\sigma = \left[\ldots, \Pi_{DS}(ds_{(x,y)}), \ldots\right] \tag{9}$$

$$\text{where } \Pi_{DS}(ds_{(x,y)}) = \begin{cases} 1, \text{ if } ds_{(x,y)} \in DS \\ 0, \text{ if } ds_{(x,y)} \notin DS \end{cases}, 0 \le i \le \sum_{i=1}^{\sigma}(|\sigma| - 1)$$

When the elements of the vector are greater than 0, it has the physical meaning that there is a direct dependency between two activities; when the elements are equal to 0, there is no direct dependency between two activities.

For example, given an event log $L = \{< a, b, d >, < a, b, c, e >\}$, we can obtain $DS = < a, b >, < b, c >, < b, d >, < c, e >$, then $V_{\sigma_1} = [1, 0, 1, 0]$, $V_{\sigma_2} = [1, 1, 0, 1]$.

Result processing of logs: We consider the last activity of the trace where the log is located as the prediction target, thus attributing the prediction problem of the log to a classification task, and the name of the activity can be considered as a label in the classification task. Therefore, the ending activity needs to be treated differently from the regular activity.

Definition 9. *Log result set*

By noting the last activity of trace σ_i as $Tail(\sigma_i) = o_i$, the set of ending activities of the log is noted as $END = \{o_1, o_2, \ldots\}$, where $o_i = Tail(\sigma_i)$. Then, each ending activity is encoded as one-hot based on the log result set, noted as $onehot(o_i)$.

Constructing the input log vector: After the above step processing, we map the activities and attributes in the event log into vector form. Subsequently, each trace's activity and attribute vectors are concatenated according to the trace identifier and then stacked into a **Case-Event** matrix, called the behavior matrix.

Definition 10. *Behavior vector M_{CE} of log L.*

$$M_{CE} = \begin{bmatrix} onehot(attr_{11}) & \cdots & onehot(attr_{1m}) & 2-gram(a_{11}) & \cdots & 2-gram(a_{1n}) & onehot(o_1) \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ onehot(attr_{j1}) & \cdots & onehot(attr_{jm}) & 2-gram(a_{j1}) & \cdots & 2-gram(a_{jn}) & onehot(o_i) \end{bmatrix} \tag{10}$$

Where

$$a_{jn} \in A \wedge a_{jn} \notin END, j \in [1, |L|], n \in [1, |A|], m \in [1, |Attr|] i \in [1, |END|]$$

Since the behavior vector construction process involves the processing strategies of control flow and data flow, we show the process in Algorithm 1 to make it more transparent.

In Algorithm 1, we start from the event log L and expect to obtain a vector that characterizes the behavioral and attribute dependencies between business activities. First, two empty lists are initialized in **Line 2–3** to store the behavior vectors and interdependencies. Then, we extract the set of activity sequences and the set of attribute sequences from the event log in the form of traces (**Line 4–5**). If there are missing values for numerical attributes, they are repaired by Lagrange interpolation (Line 6–8). After that, the attribute traces and activity traces are encoded using the One-hot function and the 2-gram function, respectively (**Line 9–14**). Finally, the two encoded vectors are connected as a behavior vector (**Line 15**).

Algorithm 1: Construction of behavior vector M_{CE}

```
Input: Event log $L$
Result: Behavior vector $M_{CE}$
begin
    $M_{CE} \leftarrow [\,]$;
    $DS \leftarrow [\,]$; // Set of direct dependencies
    $Acts \leftarrow$ All activities organized by trace in $L$;
    $Attrs \leftarrow$ All attributes organized by trace in $L$;
    if $Attrs_i$ is missing and $Attrs_i$ is numerical then
        $Attrs_i \leftarrow$ Repairing by Lagrangian Interpolation $\mathcal{L}$;
    end
    for trace in Attrs do
        $V_{attr} \leftarrow$ Function Onehot for attributes encoding
    end
    for trace $\sigma$ in $Act$ do
        $V_{act} \leftarrow$ Function 2-gram for direct dependencies encoding
    end
    $M_{CE} \leftarrow V_{attr} \cdot V_{act}$
end
```

4.2. Vector Dimensionality Reduction Based on Stacked Autoencoder

Since the resource encoding takes the One-hot form, there are many 0 elements in the resource vector. If the activity vector is in the worst case, i.e., all activities have direct dependencies on each other when the encoding length of each activity reaches $|\varphi|^2$. This encoding leads to a very sparse vector, which can cause unnecessary space costs. Therefore, we perform feature extraction on the log vector based on the autoencoder model (as shown in the AutoEncoder section of Figure 1).

The autoencoder model can be used for data downscaling and feature learning. The basic idea is to represent the original data with low-dimensional data, similar to principal component analysis (PCA) [24]. In contrast to PCA, the autoencoder model analyzes the nonlinear relationships between features by minimizing reconstruction errors in an unsupervised learning approach. The two main components are the encoder and the decoder. The former encodes the samples into dimension-specific vectors, and the decoder reconstructs the vectors into their original dimensions. The log vector M_{CE} obtained in the previous section is expanded into a set of n-dimensional vectors x, at which point the autoencoder model can be represented as follows [25].

Encoder:

$$h = f(x) = \sigma(Wx + b) \tag{11}$$

The encoder h is the hidden layer of the neural network, which can produce reduced-dimensional encoding.

Decoder

$$r = g(h) = \sigma(W'h + b') \tag{12}$$

Loss function: Similar to other neural network models, the latent layer of the autoencoder model performs a weighted sum of inputs and deviations, which is then processed by an activation function to achieve a nonlinear transformation [26]. For optimizing the parameter settings, the loss function is usually set to the squared error of the reconstructed vector with respect to the original vector.

$$L = \sum_{i=1}^{n} ||x - g(f(x))||^2 \tag{13}$$

Optimization strategy: We then train the autoencoder model using a gradient descent optimization algorithm. It is difficult to obtain more desirable abstraction results due to the weak autoencoder expression capability of the two layers. Therefore, we construct a layer-by-layer stacked deep autoencoder to extract the trained encoder and use it for the deep latent factor model.

4.3. Constructing a Deep Latent Factor Model

The current scholars mainly consider the system's direct influences and analyze the system's occurrence pattern from the dependency relationship between activities. In this section, we construct the Deep Latent Factor Model (as shown in Figure 1) to analyze the implied characteristics behind the occurrence of events through the control and resource flow perspectives of logs.

Deep Latent Factor Model: In this section, the middle part of the autoencoder is used as the model's input data, and the data are re-divided into a behavior vector P_C and a resource vector Q_E.

Definition 11. *Based on the idea of the LFM model, the product $\hat{M}_{CE}$ of P_C and Q_E can be used to fit the latent factor model and decompose it further [18].*

$$\hat{M}_{CE} = P_C Q_E = P_{C,K} Q_{K,E} \tag{14}$$

where each item of M (the two-colored squares in Figure 2) can be obtained by multiplying the corresponding case matrix row vector (the blue squares of $P_{C,K}$ in Figure 2) and the event matrix column vector (the pink squares of $Q_{K,E}$ in Figure 2).

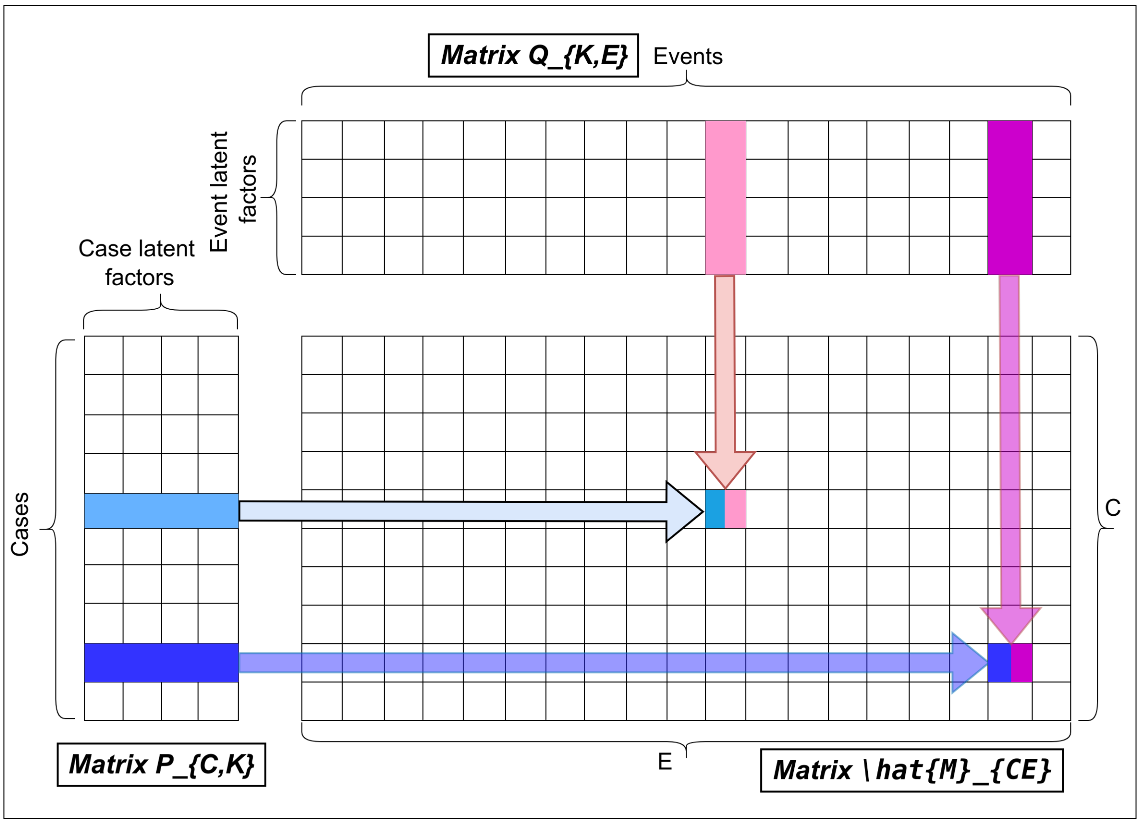

Figure 2. Decomposition diagram of the case–event matrix.

To train the deep latent factor model using historical data, we decompose and map the traditional latent factor model matrix into a neuron and weight matrix in a neural network (refer to [18] for the specific mapping method).

We expect the deep latent factor model to produce a probability value, and the current output is a real number with a wide range of values. To obtain a probability number between 0 and 1, the algorithm processes each prediction using the softmax technique, which is common in deep learning for multi-classification problems. After processing, the initial prediction results become a decimal number between 0 and 1, and the sum of all values is 1.

Definition 12. *softmax [27]*

$$softmax(O^p) = \frac{e^{(O_i^p)}}{\sum_{c=1}^{C} e^{(O_c^p)}}, C = |O^p| \tag{15}$$

Example: Suppose the log's set of terminating activities is $\{o_1, o_2, o_3, o_4\}$, and the algorithm's actual output is $O^i = (0.7, 0.6, 1, 2)$. After the softmax transformation, a set of prediction results about different categories, i.e., the prediction probabilities, is obtained, which is noted as $P^i = (0.14442535, 0.13068146, 0.19495381, 0.5299394)$. The subscript with the highest probability value of 0.5299394 is chosen as the prediction classification, and its coordinate $index(max(P^i)$ corresponding to the activity name o_4 is the final result of the prediction process.

The model obtains the probability of each process output result by weighted aggregation of the latent factor model in the last layer and calculates the final output result activity using the SoftMax activation function.

$$O' = softmax(W'' \cdot \hat{M}_{CE} + b'') \tag{16}$$

Loss function: The goal of deep latent factor model training is to minimize the difference between the prediction result O and the actual category label, which can be classified as a multi-classification problem, and usually uses cross-entropy loss.

$$J = -\sum_{i}^{m} y_i \log({o_i}') \tag{17}$$

Where m is the number of resultant activities, $y_i = 1$, when the label is the ith activity, and 0 otherwise; ${o_i}'$ is the probability of the ith category obtained by softmax.

4.4. Predicting Process Results

This section describes how to predict the process outcome using the latent factor model obtained from training.

We consider predicting the running outcome of a process as a multi-categorization problem, such as accept, reject, ignore, or return. Predicting the running instances makes it possible to predict the outcome before the system has finished. This stage generates prediction results by constructing a prefix sequence of running instances as input to the LFM model. For this purpose, the definition of sequence prefixes needs to be given first.

Definition 1 in the previous section only describes the prefix relationship at the log control flow level, and we need to extend it to meet the information needs of the prediction model in this paper.

Definition 13. *Expanded prefixes*

The expanded prefixes are written as $epre^p(\sigma)$, where $\sigma =< a_1, a_2, \ldots, a_n >$, attr denotes the relevant attributes involved in σ, p denotes the number of prefixes, and $fr(\sigma_i)$ refers to the frequency of the subsequence σ_i.

$$epre^p(\sigma) =< fr(a_1, a_2), \ldots, fr(a_{k-1}, a_k), attr_1, \ldots > \tag{18}$$

In the running phase of the system, events belonging to the same trace are associated according to the case identifier to form an expanded prefix of event length $p \in [1, n]$, and then the trained model is used to achieve process outcome prediction in online scenarios.

5. Experiments

We construct the proposed algorithm as a deep latent factor prediction model (DLFM) based on Python, implemented using the PM4Py, Scikit-learn, and TensorFlow frameworks, and evaluate their performance using three real logs. Next, the experimental setup, metrics, experimental results, and discussion are elaborated.

5.1. Experimental Setup

We use logs from five different scenarios to evaluate DLFM. Since process outcome prediction is considered a multi-classification problem, the log with more output results (greater than or equal to 3) is chosen in this paper.

Receipt phase of an environmental permit application process: This log is the execution record of the status of each phase of the building permit application, abbreviated as 'receive' in the following.

Sepsis Cases: This log is derived from sepsis cases recorded by the hospital, where the data attributes are anonymized to protect patient privacy, but the sequence of events is retained in the log and hereafter abbreviated as 'sepsis'.

BPI Challenge 2012: This log records the process activities required to apply for a loan, abbreviated as 'bpic2012'.

Road Traffic Fine Management Process: This log records the actual logs generated by the road traffic fine management system, abbreviated as 'fine' below.

Hospital Billing: This log is a regional hospital's ERP financial execution information. The number of logs is significant, but the information is limited (it does not contain the required resource logs), so it is sliced and diced in this experiment, abbreviated as 'Billing' below.

It is worth noting that the outcome activities in each log are not uniformly distributed. If we consider only the prediction accuracy, the model may obtain spuriously high accuracy by predicting all the samples as the classifications with the highest number due to the unbalanced samples. Therefore, we choose the metric ROC curve (Receiver Operating Characteristic) that is adapted to the unbalanced data scenario.

The ROC curve is a graph with FPR as the x-axis and TPR as the y-axis. FPR indicates the proportion of negative samples incorrectly judged as positive samples, and TPR is the proportion of correctly predicted positive samples among all positive samples. When the area covered by the ROC curve is larger, the model reflects better performance. For quantifying the ROC curve, the area between the curve and the x-axis and y-axis is usually denoted as AUC (Area Under ROC curve). That is, the larger the AUC, the better the model performance.

Figure 3 shows the length distribution of the traces in the selected logs. The average lengths of the traces are all within 50, with more complex trace distributions in 'Sepsis', 'BPIC2012', and 'Billing'. The traces in the 'Receipt' and 'Fine' logs are relatively short and straightforward in structure.

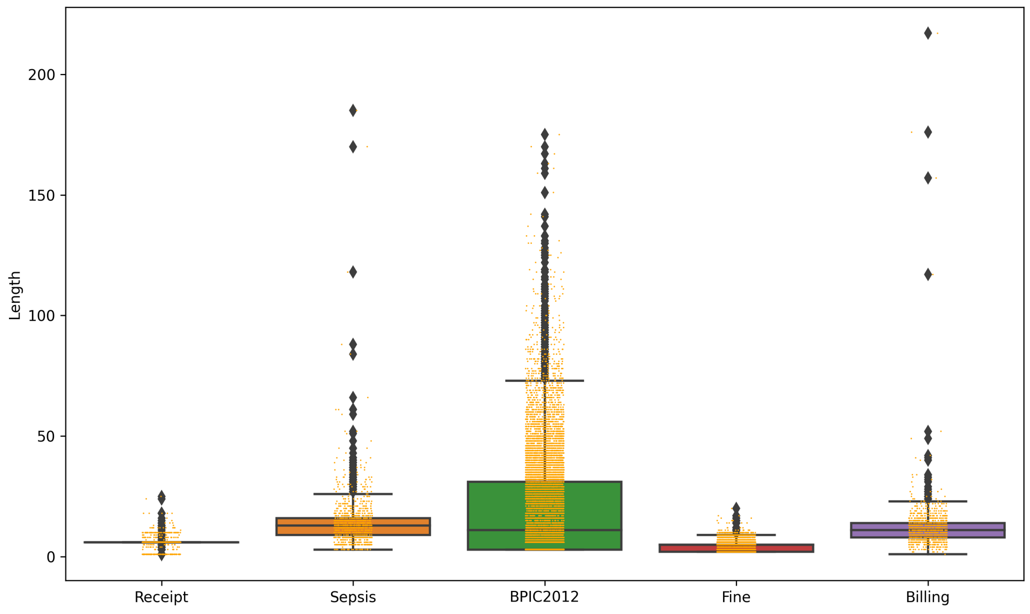

Figure 3. Distribution of trace lengths in the logs.

Combining the statistical information of the logs in Table 1, we can find that the selected logs all contain multiple resultant activities (consider the activity names as classification tags). Moreover, the first three logs contain more information with an adequate number of activities and resources, and the last two logs contain only activity information and lack resource attributes.

Table 1. Statistical information of logs.

Number	Receipt	Sepsis	BPIC2012	Fine	Billing
Traces	1318	1049	13,087	150,370	1013
Events	8577	15,214	262,200	561,470	12,506
Activities	27	16	36	11	18
Resources	48	26	69	\	\
Results	13	14	11	7	14
Unique direct dependency	99	115	125	70	142

We conducted multiple comparison experiments on the selected logs.

1. **We set different numbers of latent factors.** Multiple sets of experiments analyze the prediction effects of the proposed prediction method under five logs.
2. **Analysis based on different data objects.** Compare the prediction results of the model when only control flow or data flow is considered with those when both control flow and data flow are considered.
3. **Comparison with other outcome prediction methods.** We compare DLFM with the baseline method provided by Teinemaa et al. [3] and the prediction method based on Attention-Based Bidirectional LSTM Neural Networks proposed by Wang et al. [4] to evaluate the performance of our algorithm.

5.2. Results

We take Receipt logs as an example to show the relevant experimental results in detail.

The best performance of the model arises from the bounds of underfitting and overfitting. Figure 4 shows the trend of the training-validation loss of the Receipt log where the number of latent factors k is set between [1, 120] and the number of iterations is between [1, 200]. We can find that the starting value of the validation loss of the model is lower than the starting value of the training error, which is the underfitting state at this point.

The overall trend of the validation error is increasing, while the training error is gradually decreasing. As the number of iterations increases, the validation error gradually intersects with the training error and is greater than the latter, becoming an overfitting state.

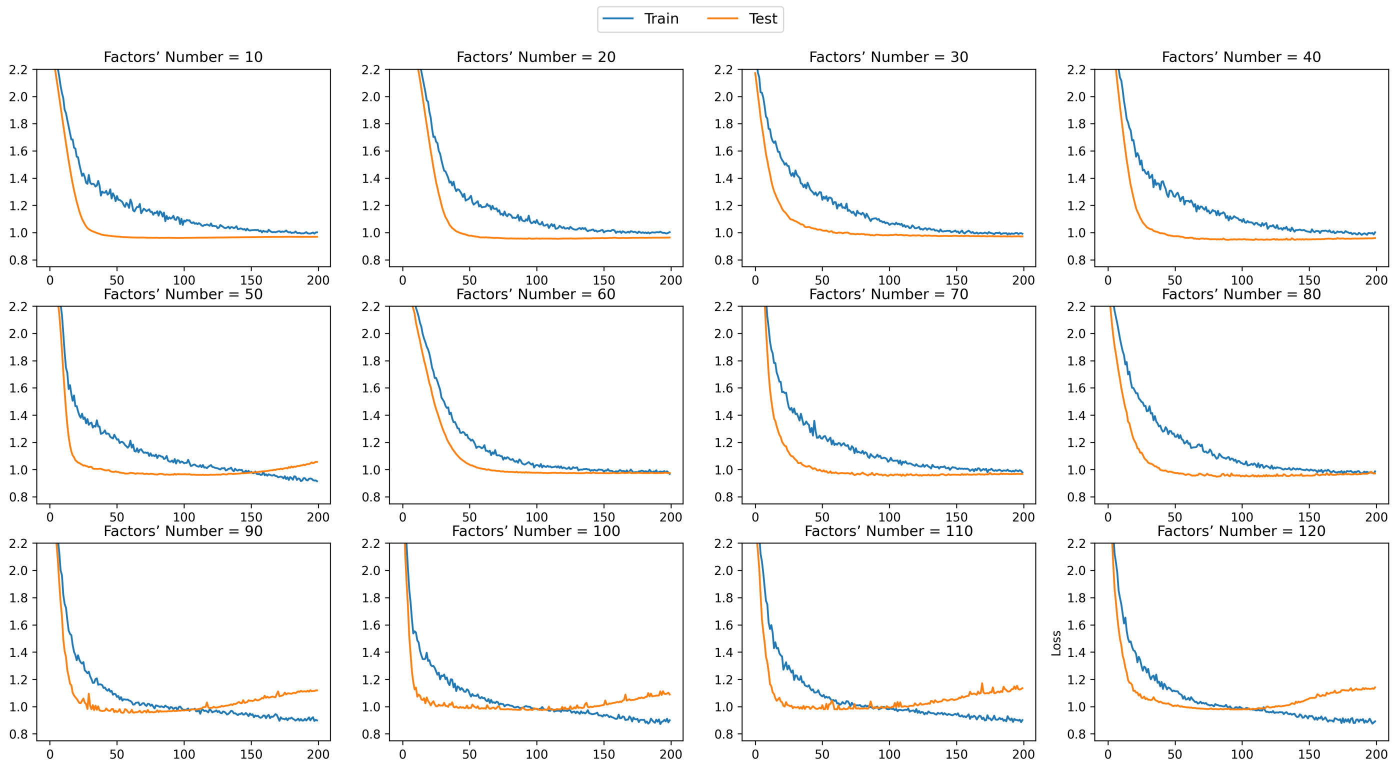

Figure 4. Loss values for Receipt log.

Thus, Figure 4 illustrates two conclusions obtained with the increase of the number of latent factors:

(1) The number of iterations for the model to reach the best performance decreases. When the number of hidden factors is 1, the model is not overfitted even at the training stop (at the end of 200 iterations). In contrast, when the number of hidden factors is 90, the model reaches the optimal point at the 100th iteration.

(2) The rate of decrease of the loss value increases. Comparing the graphs in rows 1 and 3, we can find that the gradient of the loss function in row 3 is significantly larger than that in row 1. This gradient indicates that the training and validation loss can achieve fast optimality when the number of hidden factors is more significant.

Table 2 shows the prediction performance of the five logs under different parameter settings.

(1) The effect of the number of latent factors is first analyzed. Observing the corresponding three rows of data in each log, from top to bottom, as the number of latent factors becomes larger, the AUC values of most of the model predictions increase continuously. For example, in Sepsis, the AUC of the model under control flow constraint grows from 0.7829 to 0.8479, an increase of 0.0650, while the AUC under control flow and data flow constraints grow from 0.8411 to 0.8548. The situation in the BPIC2012 log differs from the overall pattern. Its overall decreasing trend as the latent factor can be explained according to the model structure of DLFM. This situation is due to a large number of direct dependencies in the logs, and the feature vector becomes very sparse. Additionally, during data preprocessing, the descriptive power of some features is lost, which reduces the prediction results.

Table 2. The highest AUC values of the 5 logs at different settings.

	RF [3]	**XGBoost** [3]	**Att-Bi-LSTM** [4]	**DLFM**			
				Factors' Number	**AUC under Control Flow**	**AUC under Data Flow**	**AUC under Double Flows**
Receipt	0.8235	0.8316	\	10	0.8611	0.6123	0.9452
				50	0.8823	0.8082	0.9554
				100	0.9051	0.8125	**0.9581**
Sepsis	0.8104	0.8412	**0.92**	10	0.7829	0.8374	0.8411
				50	0.8469	0.8368	0.8429
				100	0.8479	0.8527	0.8548
BPIC2012	0.7126	0.7121	**0.85**	10	0.7930	0.7876	0.7978
				50	0.7782	0.7848	0.7928
				100	0.7523	0.7606	0.7783
Fine	0.8163	0.8232	0.85	10	0.9388	\	0.9388
				50	**0.9391**	\	**0.9391**
				100	0.9388	\	0.9388
Billing	0.7761	0.8109	0.82	10	0.8856	\	0.8856
				50	0.8957	\	0.8957
				100	**0.9077**	\	**0.9077**

Next, the effect of control flow or data flow on the model's predictive power is analyzed. We now need to study Table 2 from left to right for an overall view. When there are no resource attributes in the logs, the model can make predictions under the constraints of control flow. When resource attributes are present in the logs, the AUC under the control flow level constraint is higher than the result under the data flow constraint, and the AUC is higher in the case of the combination of control flow and data flow.

Finally, we compare the DLFM algorithm with other algorithms. The random forest (RF) and gradient boosted trees (XGBoost) methods are reported to outperform other baseline algorithms [3]. Att-BiLSTM also performs well in terms of time and performance [4]. We reran the code of Teinemaa et al. to obtain values for both columns of Table 2 (RF) and (XGBoost). Since Wang's code is not yet available, we list the optimal values of their experimental results in Table 2. The results show that DLFM outperforms both RF and XGBoost on the logs used. While it performs better on log Sepsis and Billing compared to Att-BiLSTM, the results are lower on log Sepsis and BPIC2012 compared to Att-BiLSTM.

We still use the Receipt log as an example, and Figure 5 visualizes the resource relationship in the Receipt log, which is generated by the Mine for a Working-Together Social Network plugin in the ProM framework. This figure shows that resources act as additional data attributes in the log messages, acting as constraints and limitations during the event occurrence. Moreover, the use of these relationships in prediction can assist the control flow in decision making and improve the prediction results of the model.

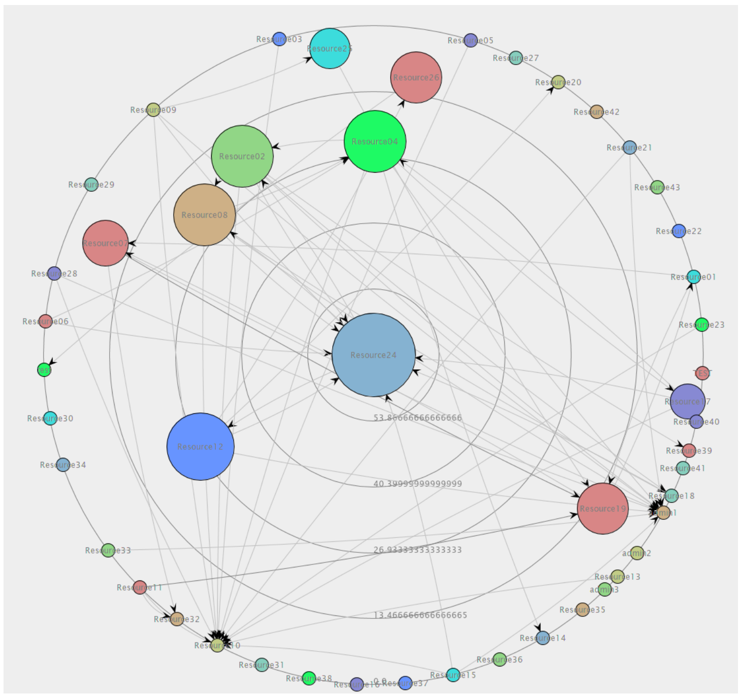

Figure 5. Analysis of the relationship between various resources in the Receipt log.

5.3. Discussions

The experimental results show that considering the hidden factor can improve the prediction. When the number of selected hidden factors is appropriate, it can enhance the model's prediction ability. However, the larger the number of hidden factors is, the better. When it reaches a critical value, a further increase will lead to a decrease in prediction ability. This is because the dimensions of activities and resources are then multiplied by the dimensions of the hidden factor matrix to obtain a vast sparse matrix, leading to a decrease in the expressiveness of the model.

In addition, resources are interrelated and can influence the direction of the control flow to some extent. Unfortunately, the exploration of resource perspectives in this paper is limited. In the experimental results of the previous section, we found that the prediction model had mediocre performance with some resources alone, yet, when combined with the control flow, it was able to improve the prediction results significantly. Moreover, the addition of some resources produced the opposite result. We infer that not all resources are beneficial to the prediction model. Some useless resource constraints may limit the model's performance, leading to the so-called feature explosion. Therefore, selective resource enrichment should be further explored in future work.

6. Conclusions

This paper proposes a deep latent factor model to predict the process outcome while the system process is still in progress using potential factors affecting the system operation. The contributions of this paper can be summarized in the following two aspects:

(1) A deep latent factor model is constructed and optimized offline based on activity and resource dependencies. We first build a composite vector by incorporating activity and resource sequences and then stack the auto-encoder model to reduce the dimensionality of the data to extract the core factors. The encoder part of this model is then used to construct the deep latent factor model, and the model optimization is performed on the historical logs;

(2) The results are predicted for the running event stream online. We construct the online event stream as a sequence of event streams to allow for prediction in the operational state of the system. To validate the prediction results of the proposed model, we implement the algorithm based on an open-source framework and evaluate it using several real logs.

Experiments show that better results are obtained when the model considers both control and resource flow.

We additionally considered the resource perspective other than the control flow, however, limited by the nature of the One-hot coding approach, we only considered the question of whether resources exist. Do resources affect each other? Do resources have an impact on the control flow? Which resources have a more significant impact on the control flow? In addition, the impact of data from other perspectives on control flow needs to be further explored, such as the activity execution role perspective and the organizer perspective. These exciting questions have not yet been definitively answered, and in future work, we will further explore these factors by deepening theoretical studies and expanding the scale of experiments.

Author Contributions: Methodology, K.L., X.F. (Xinjian Fang) and X.F. (Xianwen Fang); Software, K.L.; Supervision, X.F. (Xianwen Fang); Writing—original draft, K.L.; Writing—review & editing, X.F. (Xinjian Fang) and X.F. (Xianwen Fang). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation, China (No.61572035, 61402011), the Leading Backbone Talent Project in Anhui Province, China(2020-1-12), the Natural Science Foundation of Anhui Province, China (No. 2008085QD178), Anhui Province Academic and Technical Leader Foundation (No. 2019H239), and the Open Project Program of the Key Laboratory of Embedded System and Service Computing of Ministry of Education (No.ESSCKF2021-05).

Data Availability Statement: All the data in this paper were obtained from the 4tu database, please visit https://data.4tu.nl . The code is implemented based on PM4Py, Scikit-learn, and TensorFlow APIs. Experimental data and code related to this paper can be obtained by contacting the corresponding author.

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

References

1. van der Aalst, W.M.P. *Process Mining: Data Science in Action*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016; doi:10.1007/978-3-662-49851-4.
2. van der Aalst, W.; Adriansyah, A.; de Medeiros, A.K.A.; Arcieri, F.; Baier, T.; Blickle, T.; Bose, J.C.; van den Brand, P.; Brandtjen, R.; Buijs, J.; et al. Process Mining Manifesto. In *Proceedings of the Business Process Management Workshops*; Daniel, F., Barkaoui, K., Dustdar, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 99, pp. 169–194, doi:10.1007/978-3-642-28108-2_19.
3. Teinemaa, I.; Dumas, M.; Rosa, M.L.; Maggi, F.M. Outcome-Oriented Predictive Process Monitoring: Review and Benchmark. *ACM Trans. Knowl. Discov. Data* **2019**, *13*, 1–57, doi:10.1145/3301300.
4. Wang, J.; Yu, D.; Liu, C.; Sun, X. Outcome-Oriented Predictive Process Monitoring with Attention-Based Bidirectional LSTM Neural Networks. In Proceedings of the 2019 IEEE International Conference on Web Services (ICWS), Milan, Italy, 8–13 July 2019; doi:10.1109/icws.2019.00065.
5. Andreas, M.; Adrian, N. Considering Non-sequential Control Flows for Process Prediction with Recurrent Neural Networks. In Proceedings of the 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech Republic, 29–31 August 2018; doi:10.1109/seaa.2018.00051.
6. Hinkka, M.; Lehto, T.; Heljanko, K.; Jung, A. Classifying Process Instances Using Recurrent Neural Networks. In *Proceedings of the Business Process Management Workshops*; Daniel, F., Sheng, Q.Z., Motahari, H., Eds.; Lecture Notes in Business Information Processing; Springer International Publishing: Cham, Switzerland, 2019; pp. 313–324, doi:10.1007/978-3-030-11641-5_25.
7. Folino, F.; Folino, G.; Guarascio, M.; Pontieri, L. Learning Effective Neural Nets for Outcome Prediction from Partially Labelled Log Data. In Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, 4–6 November 2019; Volume 11; pp. 1396–1400, doi:10.1109/ICTAI.2019.00196.
8. Mongia, A.; Majumdar, A. Matrix Completion on Learnt Graphs: Application to Collaborative Filtering. *Expert Syst. Appl.* **2021**, *185*, 115652, doi:10.1016/j.eswa.2021.115652.
9. Maggi, F.M.; Di Francescomarino, C.; Dumas, M.; Ghidini, C. Predictive Monitoring of Business Processes. In *Proceedings of the Advanced Information Systems Engineering*; Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Kobsa, A., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2014; Volume 8484, pp. 457–472, doi:10.1007/978-3-319-07881-6_31.
10. de Leoni, M.; van der Aalst, W.M.; Dees, M. A General Process Mining Framework for Correlating, Predicting and Clustering Dynamic Behavior Based on Event Logs. *Inf. Syst.* **2016**, *56*, 235–257, doi:10.1016/j.is.2015.07.003.

11. Francescomarino, C.D.; Dumas, M.; Maggi, F.M.; Teinemaa, I. Clustering-Based Predictive Process Monitoring. *IEEE Trans. Serv. Comput.* **2019**, *12*, 896–909, doi:10.1109/TSC.2016.2645153.
12. Leontjeva, A.; Conforti, R.; Francescomarino, C.D.; Dumas, M.; Maggi, F.M. Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In *Business Process Management*; Proceedings; Motahari-Nezhad, H.R., Recker, J., Weidlich, M., Eds.; Lecture Notes in Computer Science; Springer: Innsbruck, Austria, 2015; Volume 9253, pp. 297–313, doi:10.1007/978-3-319-23063-4_21.
13. Pasquadibisceglie, V.; Appice, A.; Castellano, G.; Malerba, D.; Modugno, G. Orange: Outcome-oriented Predictive Process Monitoring Based on Image Encoding and CNNs. *IEEE Access* **2020**, *8*, 184073–184086, doi:10.1109/ACCESS.2020.3029323.
14. Cheng, Z.; Ding, Y.; Zhu, L.; Kankanhalli, M. Aspect-Aware Latent Factor Model: Rating Prediction with Ratings and Reviews. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*; ACM Press: Lyon, France, 2018; pp. 639–648, doi:10.1145/3178876.3186145.
15. Wang, Q.; Chen, S.; Luo, X. An Adaptive Latent Factor Model via Particle Swarm Optimization. *Neurocomputing* **2019**, *369*, 176–184, doi:10.1016/j.neucom.2019.08.052.
16. Luo, X.; Zhou, M.; Li, S.; Shang, M. An Inherently Nonnegative Latent Factor Model for High-Dimensional and Sparse Matrices from Industrial Applications. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2011–2022, doi:10.1109/TII.2017.2766528.
17. Wu, D.; Luo, X.; Shang, M.; He, Y.; Wang, G.; Wu, X. A Data-Characteristic-Aware Latent Factor Model for Web Services QoS Prediction. *IEEE Trans. Knowl. Data Eng.* **2020**, 1, doi:10.1109/TKDE.2020.3014302.
18. Mongia, A.; Jhamb, N.; Chouzenoux, E.; Majumdar, A. Deep Latent Factor Model for Collaborative Filtering. *Signal Process.* **2020**, *169*, 107366, doi:10.1016/j.sigpro.2019.107366.
19. Wu, D.; Luo, X.; Shang, M.; He, Y.; Wang, G.; Zhou, M. A Deep Latent Factor Model for High-Dimensional and Sparse Matrices in Recommender Systems. *IEEE Trans. Syst. Man, Cybern. Syst.* **2021**, *51*, 4285–4296, doi:10.1109/TSMC.2019.2931393.
20. Weinzierl, S. Exploring Gated Graph Sequence Neural Networks for Predicting Next Process Activities. In *Proceedings of the Business Process Management Workshops*; Marrella, A., Weber, B., Eds.; Lecture Notes in Business Information Processing; Springer International Publishing: Cham, Switzerland, 2022; pp. 30–42, doi:10.1007/978-3-030-94343-1_3.
21. Santoso, A.; Felderer, M. Specification-Driven Predictive Business Process Monitoring. *Softw. Syst. Model.* **2020**, *19*, 1307–1343, doi:10.1007/s10270-019-00761-w.
22. Hinkka, M.; Lehto, T.; Heljanko, K. Exploiting Event Log Event Attributes in RNN Based Prediction. In *Proceedings of the New Trends in Databases and Information Systems*; Welzer, T., Eder, J., Podgorelec, V., Wrembel, R., Ivanović, M., Gamper, J., Morzy, M., Tzouramanis, T., Darmont, J., Kamišalić Latifić, A., Eds.; Springer International Publishing: Cham, Switzerland, 2019; Volume 1064, pp. 405–416, doi:10.1007/978-3-030-46633-6_4.
23. Mehdiyev, N.; Evermann, J.; Fettke, P. A Novel Business Process Prediction Model Using a Deep Learning Method. *Bus. Inf. Syst. Eng.* **2020**, *62*, 143–157, doi:10.1007/s12599-018-0551-3.
24. Abdi, H.; Williams, L.J. Principal Component Analysis. *Wiley Interdiscipl. Rev. Comput. Stat.* **2010,7**, *2*, 433–459, doi:10.1002/wics.101.
25. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–444, doi:10.1038/nature14539.
26. Hoang, T.C.N.; Lee, S.; Kim, J.; Ko, J.; Marco, C. Autoencoders for Improving Quality of Process Event Logs. *Expert Syst. Appl.* **2019**, *131*, 132–147, doi:10.1016/j.eswa.2019.04.052.
27. Horiguchi, S.; Ikami, D.; Aizawa, K. Significance of Softmax-based Features in Comparison to Distance Metric Learning-based Features. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *42*, 1279–1285, doi:10.1109/TPAMI.2019.2911075.