Exploiting Event Log Event Attributes in RNN Based Prediction

Markku Hinkka[1,3](✉), Teemu Lehto[1,3], and Keijo Heljanko[2,4]

[1] Department of Computer Science, School of Science, Aalto University, Espoo, Finland
[2] Department of Computer Science, University of Helsinki, Helsinki, Finland
[3] QPR Software Plc, Helsinki, Finland
[4] HIIT Helsinki Institute for Information Technology, Espoo, Finland
markku.hinkka@aalto.fi, teemu.lehto@qpr.com, keijo.heljanko@helsinki.fi

Abstract. In predictive process analytics, current and historical process data in event logs are used to predict future. E.g., to predict the next activity or how long a process will still require to complete. Recurrent neural networks (RNN) and its subclasses have been demonstrated to be well suited for creating prediction models. Thus far, event attributes have not been fully utilized in these models. The biggest challenge in exploiting them in prediction models is the potentially large amount of event attributes and attribute values. We present a novel clustering technique which allows for trade-offs between prediction accuracy and the time needed for model training and prediction. As an additional finding, we also find that this clustering method combined with having raw event attribute values in some cases provides even better prediction accuracy at the cost of additional time required for training and prediction.

Keywords: Process mining · Predictive process analytics · Prediction · Recurrent neural networks · Gated Recurrent Unit

1 Introduction

Event logs generated by systems in business processes are used in Process Mining to automatically build real-life process definitions and as-is models behind those event logs. There is a growing number of applications for predicting the properties of newly added event log cases, or process instances, based on case data imported earlier into the system [2,3,8,13]. The more the users start to understand their own processes, the more they want to optimize them. This optimization can be facilitated by performing predictions. In order to be able to predict properties of new and ongoing cases, as much information as possible should be collected that is related to the event log traces and relevant to the properties to be predicted. Based on this information, a model of the system creating the event logs can be created. In our approach, the model creation is performed using supervised machine learning techniques.

In our previous work [5] we have explored the possibility to use machine learning techniques for classification and root cause analysis for a process mining related classification task. In the paper, experiments were performed on the efficiency of several feature selection techniques and sets of structural features (a.k.a. activity patterns) based on process paths in process mining models in the context of a classification task. One of the biggest problems with the approach is that finding of the structural features having the most impact on the classification result. E.g., whether to use only activity occurrences, transitions between two activities, activity orders, or other even more complicated types of structural features such as detecting subprocesses or repeats. For this purpose, we have proposed another approach in [6], where we examined the use of recurrent neural network techniques for classification and prediction. These techniques are capable of automatically learning more complicated causal relationships between activity occurrences in *activity sequences*. We have evaluated several different approaches and parameters for the recurrent neural network techniques and have compared the results with the results we collected in our work. In both the previous publications [5,6], focusing on boolean-type classification tasks based on the *activity sequences* only.

In this work we build on our previous work to further improve the prediction accuracy of prediction models by exploiting additional event attributes often available in the event logs while also taking into account the scalability of the approach to allow users to precisely specify the event attribute detail level suitable for the prediction task ahead. Our goal is to develop a technique that would allow the creation of a tool that is, based on a relatively simple set of parameters and training data, able to efficiently produce a prediction model for any case-level prediction task, such as predicting the next activity or the final duration of a running case. Fast model rebuilding is also required in order for a tool to be able to also support, e.g., interactive event and case filtering capabilities.

To answer these requirements, we introduce a novel method of exploiting event attributes into RNN prediction models by clustering events by their event attribute values, and using the cluster labels in the RNN input vectors instead of the raw event data. This makes it easy to manage the input RNN vector size no matter how many event attributes there are in the data set. E.g., users can configure the absolute maximum length of the one-hot vector used for the event attribute data which will not be exceeded, no matter how many actual attributes the dataset has.

Our prediction engine source code is available in GitHub[1].

The rest of this paper is structured as follows: Sect. 2 is a short summary of the latest developments around the subject. In Sect. 3, we present the problem statement and the related concepts. Section 4 presents our solution for the problem. In Sect. 5 we present our test framework used to test our solution. Section 6 describes the used datasets as well as performed prediction scenarios. Section 7 presents the experiments and their results validating our solution. Finally Sect. 8 draws the final conclusions.

[1] https://github.com/mhinkka/articles.

2 Related Work

Lately there has been a lot of interest in the academic world on predictive process monitoring which can clearly be seen, e.g., in [4] where the authors have collected a survey of 55 accepted academic papers on the subject. In [12], the authors have compared several approaches spanning three different research fields: Machine learning, process mining and grammar inference. As result, they have found that overall, the techniques from machine learning field generate more accurate predictions than grammar inference and process mining fields.

In [13] the authors used Long Short-Term Memory (LSTM) recurrent neural networks to predict the next activity and its timestamp. They use one-hot encoded activity labels and three numerical time-based features: duration between the current activity and the previous activity, time within the day and time within the week. Event attributes were not considered at all. In [2] the authors trained LSTM networks to predict the next activity. In this case however, network inputs are created by concatenating categorical, character string valued event attributes and then encoding these attributes via an embedding space. They also note that this approach is feasible only because of the small number of unique values each attribute had in their test datasets. Similarly, in [11], the authors take a very similar approach based on LSTM networks, but this time also incorporate both discrete and continuous event attribute values. Discrete values are one-hot encoded, whereas continuous values are normalized using min-max normalization and added to the input vectors as single values.

In [9] the authors use Gated Recurrent Unit (GRU) recurrent neural networks to detect *anomalies* in event logs. One one-hot encoded vector is created for activity labels and one for each of the included string valued event attributes. These vectors are then concatenated in similar fashion to our solution into one vector representing one event, that is then given as input to the network. We use this approach for benchmarking our own clustering based approach (labeled as *Raw* feature in the text below). The system proposed in their paper is able to predict both the next activity and the next values of event attributes. Specifically, it does not take case attributes and temporal attributes into account.

In [14] the authors trained a RNN to predict the most likely future activity sequence of a running process based only on the sequence of activity labels. Similarly our earlier publication [5] used sequences of activity labels to train a LSTM network to perform a boolean classification of cases. None of the mentioned earlier works present a solution that is scalable for datasets having lots of event- or case attributes and unique attribute values.

3 Problem

Using RNN to perform case-level predictions on event logs has lately been studied a lot. However, there has not been any scalable approach on handling event attributes in RNN setting. Instead, e.g., in [9] authors used separate one-hot encoded vector for each attribute value. Having this kind of an approach when you have, e.g., 10 different attributes, each having 10 unique values would already require a vector of 100 elements to be added as input for every event. The longer the input vectors become, the more time and memory it gets for the model to create accurate models from them. This increases the time and memory required to use the model for predictions.

Table 1. Feature input vector structure

f_{11}	f_{12}	...	f_{1m_1}	f_{21}	...	f_{2m_2}	...	f_{n1}	...	f_{nm_n}

Table 2. Feature input vector example content

row	$activity_{eat}$	$activity_{drink}$	$food_{salad}$	$food_{pizza}$	$food_{water}$	$food_{soda}$	$cluster_1$	$cluster_2$
1	1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	1	0
3	1	0	0	1	0	0	0	1
4	1	0	0	1	0	0	0	1
5	0	1	0	0	0	1	0	1

4 Solution

We decided to include several feature types into the input vectors of the RNN. Input vectors are formatted as shown in Table 1, where each column represents one feature vector element f_{ab}, where a is the index of the feature and b is the index of the element of that feature. In the table, n represents the number of feature types used in the feature vector and m_k represents the number of elements required in the input vector for feature type k. Thus, each feature type produces one or more numeric elements into the input vector, which are then concatenated together into one actual input vector passed to RNN both in training and in prediction phases. Table 2 shows an example input vector having three different feature types: activity label, raw event attribute values (only single event attribute named *food* having four unique values) and the event attribute cluster where clustering has been performed separately for each unique activity.

For this paper, we encoded only event activity labels and event attributes into the input vectors. However, this mechanism can easily incorporate also other types of features not described here. The only requirement for added features is that it needs to be able to be encoded into a numeric vector as shown in Table 1 whose length must be the same for each event.

4.1 Event Attributes

Our primary solution for incorporating information in event attributes into input vectors is to cluster all the event attribute values in the training set and then use a one-hot encoded cluster identifier to represent all the attribute values of the element. The used clustering algorithm must be such that it tries to automatically find the optimal number of clusters for the given data set within the range of 0 to N clusters, where N can be configured by the user. By changing N, the user can easily configure the maximum length of the one-hot-vector as well as the precision of how detailed attribute information will be tracked. For this paper, we experimented with slightly modified version of Xmeans-algorithm [10].

It is very common that different activities get processed by different resources yielding a completely different set of possible attribute values. E.g., different departments in a hospital have different people, materials and processes. Also in the example feature vector shown in Table 2, *food*-event attribute has completely different set of possible values depending on the *activity* since it is forbidden by, e.g., the external system to not allow activity of type *eat* to have *food* event attribute value of *water*. If we cluster all the event attributes using single clustering, we would easily lose this activity type specific information.

In order to retain this activity specific information, we used separate clustering for each unique activity type. All the event attribute clusters are encoded into one one-hot encoded vector representing only the resulting cluster label for that event, no matter what its activity is. This is shown in the example table as $cluster_N$, which represents the row having N as clustering label. E.g., in the example case, $cluster_1$ is 1 in both rows 1 and 2. However, row 1 is in that cluster because it is in the 1st cluster of the $activity_{eat}$ activity, whereas row 2 is in that cluster because it is in the 1st cluster of the $activity_{drink}$ activity. Thus, in order to identify the actual cluster, one would require both the activity label and the cluster label. For RNN to be able to properly learn about the actual event attribute values, it needs to be given both the activity label and the cluster label in input vector. Below, this approach is labeled as *ClustN*, where N is the maximum cluster count.

For benchmarking, we also experimented with a *raw* implementation where event attributes were used so that every event attribute is encoded into its own one-hot encoded vector and then concatenated into the actual input vectors. This method is lossless, since every unique event attribute value has its own indicator in the input vector. Below, this approach is referred to as *Raw*. Finally, we experimented also using both *Raw* and *Clustered* event attribute values. Below, this approach is referred to as *BothN*, where N is the maximum cluster count.

5 Test Framework

We have performed our test runs using an extended Python-based prediction engine that was used in our earlier work [5]. The engine is still capable of supporting most of the hyperparameters that we experimented with in our earlier work, such as used RNN unit type, number of RNN layers and the used batch size. The prediction engine we built for this work takes a single JSON configuration file as input and outputs test result rows into a CSV file.

Tests were performed using a commonly used 3-fold cross-validation technique to measure the generalization characteristics of the trained models. In

3-fold cross-validation the input data is split into three subsets of equal size. Each of the subsets is tested one by one against models trained using the other two subsets.

5.1 Training

Training begins by loading the event log data contained in the two of the three event log subsections. After this, the event log is split into actual training data and validation data that used to find the best performing model out of all the model states during all the test iterations. For this, we picked 75% of the cases for the training and the rest for the validation dataset. After the this, we initialize event attribute clusters as described in Sect. 4.1.

The actual prediction model and the data used to generate the actual input vectors is performed next. This data initialization involves splitting cases into prefixes and also taking a random sample of the actual available data if the amount of data exceeds the configured maximum amount of prefixes. In order to avoid running out of memory during any of our tests, these limits were set to 75000 for training data and 25000 for validation data. We also had to filter out all the cases having more than 100 events.

Finally after the model is initialized, we start the actual training in which we concatenate all the requested feature vectors as well as the expected outcome into the RNN model repeatedly for the whole training set until 100 test iterations have passed. The number of actual epochs trained in each iteration is configurable. In our experiments, the total number of epochs was set to be 10. After every test iteration the model is validated against the validation set. In order to improve validation performance, if the size of the validation set is larger than separately specified limit (10000), a random sample of the whole validation set is used. These test results, including additional status and timing related information, is written into resulting test result CSV file. If the prediction accuracy of the model against the validation set is found to be better than the accuracy of any of the models found thus far, then the network state is stored for that model. Finally after all the training, the model having the best validation test accuracy is picked as the actual result of the model training.

5.2 Testing

In the testing phase, the third subset of cross-validation folding is tested against the model built in the previous step. After initializing the event log following similar steps as in the training phase, the model is asked for a prediction for each input vector built from the test data. In order to prevent running out of memory and to ensure tests are not taking exceedingly long time to run, we limited the number of final test traces to 100000 traces and used random sampling when needed. The prediction result accuracy, as well as other required statistics are written to the resulting CSV file.

6 Test Setup

We performed our tests using several different data sets. Some details of the used data sets can be found in the Table 3.

Table 3. Used event logs and their relevant statistics

Event log	# Cases	# Activities	# Events	# Attributes	# Unique values
BPIC12[a]	13087	24	262200	1	3
BPIC13, incidents[b]	7554	13	65533	8	2890
BPIC14[c]	46616	39	466737	1	242
BPIC17[d]	31509	26	1202267	4	164
BPIC18[e]	43809	41	2514266	5	360

[a] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[b] https://doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee
[c] https://doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35
[d] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
[e] https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

For each dataset, we performed next activity prediction where we wanted to predict the next activity of any ongoing case. In this case, we split every input case into possibly multiple *virtual* cases depending on the number of events the case had. If the length of the case was shorter than 4, the whole case was ignored. If the length was equal or higher, then a separate *virtual* case was created for all prefixes at least of length 4. Thus, for a case of length 6, 3 cases were created: One with length 4, one with 5 and one with 6. For all these prefixes, the next activity label was used as the expected outcome. For the full length case, the expected outcome was a special *finished*-token.

7 Experiments

For experiments, we have used the same system that we used already in our previous work [5]. The system had Windows 10 operating system and its hardware consisted of 3.5 GHz Intel Core i5-6600K CPU with 32 GB of main memory and NVIDIA GeForce GTX 960 GPU having 4 GB of memory. Out of those 4 GB, we reserved 3 GB for the tests. The testing framework was built on the test system using Python programming language. The actual recurrent neural networks were built using Lasagne[2] library that works on top of Theano[3]. Theano was configured to use GPU via CUDA for expression evaluation.

[2] https://lasagne.readthedocs.io/.
[3] http://deeplearning.net/software/theano/.

We used one layer GRU [1] as the RNN type. Adam [7] was used as gradient descent optimizer. 256 was used as hidden dimension size and 0.01 as learning rate even though it is quite probable that more accurate results could have been achieved by selecting, e.g., different hidden dimension sizes depending on the size of the input vectors. However, since that would have made the interpretation of the test results more complicated, we decided to use the constant hidden dimension size on all the tests.

We performed next activity predictions using all the four combinations of features, five data sets and three different maximum cluster counts: 20, 40, and 80 clusters. The results of these runs are shown in Table 4. In the table, *Features*-column shows the used set of features. *S.rate* shows the achieved prediction success rate. *In.v.s.* shows the size of the input vector. This column can be used to give some kind of indication on the memory usage of using that configuration. Finally, *Tra.t.* and *Pred.t.* columns tell us the time required for performing the training and the prediction for all the cases in the test dataset. In both the cases, this time includes the time for setting up the neural network, clusterings and preparing the dataset from JSON format. Sample standard deviation has been included into both *S.rate* and *Tra.t* in parentheses to indicate how spread out the measurements are within all the three test runs. Each row in the table represents three cross validation runs with unique combination of dataset and feature that was tested. Rows having a best prediction accuracy within a dataset are shown using bold font. *None*-feature represents the case in which there were no event attribute information at all in the input vector, *ClustN* represents a test with one-hot encoded cluster labels of event attributes clustered into maximum of *N* clusters, *Raw* represents having all one-hot encoded attribute values individually in the input vector, and finally *BothN* represents having both one-hot encoded attribute values and one-hot encoded cluster labels in the input vector.

We also aggregated some of these results over all the datasets using maximum cluster size of 80 clusters. Figure 1 shows average success rates of different event attribute encoding techniques over all the tested datasets. Figure 2 shows the average input vector lengths. Figures 3 and 4 shows the averaged training and prediction times respectively.

Based on all of these results, we can see that having event attribute values included clearly improved the prediction accuracy over not having them included at all in all datasets. The effect ranged from 0.5% in BPIC12 model to 8.5% in BPIC18. As shown in Fig. 1, very similar success rates were achieved using *ClustN* features as with *Raw*. However, model training and the actual prediction can be performed clearly faster using *ClustN* approaches than either *Raw* or *BothN*. This effect is the most prominently visible in BPIC13 results, where, due to the model having large amount of unique attribute values, the size of the input vector is almost 68 times bigger and the training time almost 14 times longer using *Raw* feature than *Clust20*. At the same time, the accuracy is still clearly better than not having event attributes at all (about 3.9% better) and only slightly worse (about 1.4%) than when using *Raw* feature. This clearly indicates that clustering can be really powerful technique for minimizing the time required for training

Table 4. Statistics of next activity prediction using different sets of input features

Dataset	Features	S.rate (σ)	In.v.s.	Tra.t. (σ)	Pred.t.
BPIC12	None	85.8% (0.3%)	25.7	489.0 s (7.0 s)	35.1 s
	Clust20	86.0% (0.4%)	30.0	500.6 s (2.5 s)	31.6 s
	Clust40	85.8% (0.3%)	30.0	499.7 s (1.3 s)	31.9 s
	Clust80	86.2% (0.1%)	30.0	502.1 s (2.3 s)	7.5 s
	Raw	85.9% (0.3%)	29	504.3 s (0.5 s)	38.9 s
	Both20	86.0% (0.2%)	33	515.3 s (2.6 s)	40.4 s
	Both40	86.0% (0.4%)	33	517.7 s (3.6 s)	40.4 s
	Both80	**86.3% (0.1%)**	**33**	**518.2 s (4.0 s)**	**40.7 s**
BPIC13	None	62.9% (0.3%)	13.7	165.6 s (21.2 s)	3.5 s
	Clust20	66.8% (0.3%)	34.7	188.0 s (22.4 s)	4.7 s
	Clust40	67.2% (0.7%)	54.7	214.8 s (3.1 s)	5.4 s
	Clust80	67.0% (0.6%)	94.7	258.4 s (4.7 s)	6.0 s
	Raw	68.2% (1.1%)	2353.7	2611.7 s (44.7 s)	74.8 s
	Both20	**69.1% (0.6%)**	**2359.3**	**2464.6 s (309.0 s)**	**94.4 s**
	Both40	68.9% (0.5%)	2395.7	2687.1 s (227.3 s)	106.6 s
	Both80	68.4% (0.7%)	2429.3	2821.8 s (33.5 s)	194.3 s
BPIC14	None	37.8% (1.5%)	40.3	488.1 s (5.3 s)	36.1 s
	Clust20	39.9% (0.5%)	61.7	523.3 s (3.5 s)	40.4 s
	Clust40	40.0% (0.3%)	80.3	553.5 s (3.8 s)	43.6 s
	Clust80	40.2% (0.1%)	84.7	556.8 s (10.5 s)	43.6 s
	Raw	39.7% (1.4%)	272.0	825.7 s (2.8 s)	68.0 s
	Both20	40.6% (0.6%)	292.3	907.1 s (7.5 s)	78.6 s
	Both40	**40.6% (0.6%)**	**309.3**	**943.3 s (10.6 s)**	**82.0 s**
	Both80	37.3% (4.2%)	305.0	935.1 s (26.9 s)	156.7 s
BPIC17	None	86.4% (0.4%)	27.7	518.7 s (2.8 s)	107.7 s
	Clust20	**90.8% (0.3%)**	**48.7**	**556.3 s (3.7 s)**	**132.4 s**
	Clust40	90.2% (1.4%)	68.3	637.5 s (58.3 s)	143.9 s
	Clust80	90.2% (0.4%)	108.7	647.3 s (3.7 s)	142.8 s
	Raw	89.9% (0.5%)	190	816.4 s (5.2 s)	164.9 s
	Both20	89.9% (0.5%)	211.0	867.8 s (3.5 s)	188.0 s
	Both40	90.2% (0.3%)	230.3	910.9 s (19.3 s)	193.7 s
	Both80	89.6% (0.6%)	271.3	986.5 (4.4 s)	197.7 s
BPIC18	None	71.3% (9.3%)	43	516.0 s (9.5 s)	197.0 s
	Clust20	79.0% (0.9%)	64.0	588.7 s (13.7 s)	268.7 s
	Clust40	**79.9% (0.2%)**	**84.0**	**628.4 s (2.8 s)**	**286.1 s**
	Clust80	79.5% (0.1%)	124.0	701.3 s (7.4 s)	306.9 s
	Raw	79.3% (0.4%)	349.7	1173.7 s (83.1 s)	381.2 s
	Both20	79.7% (0.5%)	377.7	1213.1 s (48.1 s)	463.2 s
	Both40	79.9% (0.5%)	401.0	1301.9 s (82.9 s)	540.3 s
	Both80	79.3% (0.5%)	425.7	1405.4 s (87.2 s)	619.9 s

especially when there are a lot of unique event attribute values in the used event log. Even when using the maximum cluster count of 20, prediction results will be either not affected or improved with relatively small impact to the training and prediction time. In all the datasets, the best prediction accuracy is always achieved either by using only clustering, or by using both clustering and raw attributes at the same time.

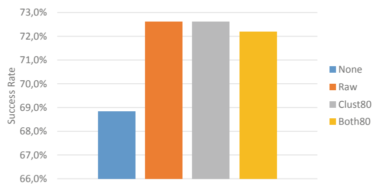

Fig. 1. Average prediction success rate over all the datasets

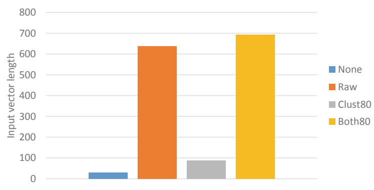

Fig. 2. Average length of the input vector over all the datasets

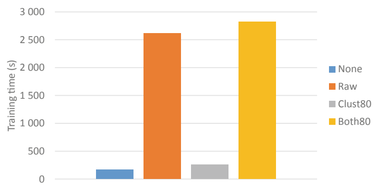

Fig. 3. Average training time over all the datasets

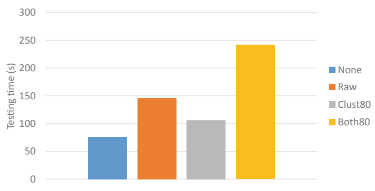

Fig. 4. Average prediction time over all the datasets

7.1 Threats to Validity

As threats to validity of the results in this paper, it is clear that there are a lot of variables involved. As initial set of parameter values, we used parameters that were found good enough in our earlier work and did some improvement attempts based on the results we got. It is most probable that the set of parameters we used were not optimal ones in each test run. We also did not test all the parameter combinations and the ones we did, we tested often only once, even though there was some randomness involved, e.g., selecting the initial cluster centers in the XMeans algorithm. However, we think that since we tested the results in several different datasets using 3-fold cross validation technique, our results can be used at least as a baseline for further studies. All the results generated by the test runs, as well as all the source data and the test framework itself, are available in support materials[4].

Also, we did not really test with datasets having really many event attribute values, the maximum amount tested being 2890. However, it can be seen that since the size of the input vectors is completely user configurable when performing event attribute clustering, the user him/herself can easily set limits to the input vector length which should take the burden off from the RNN and move the burden to the clustering algorithms, which are usually more efficient in handling lots of features and feature values. When evaluating the results of the performed tests and comparing them with other similar works, it should be taken into account that data sampling was used in several phases of the testing process.

[4] https://github.com/mhinkka/articles.

8 Conclusions

Clustering can be applied on attribute values to improve accuracy of predictions performed on running cases. In four of the five experimented data sets, having event attribute clusters encoded into the input vectors outperforms having the actual attribute values in the input vector. In addition, due to raw attribute values having direct effect to input vector lengths, the training and prediction time will also be directly affected by the number of unique event attribute values. Clustering does not have this problem: The number of elements reserved in the input vector for clustered event attribute values can be adjusted freely. The memory usage is directly affected by the length of the input vector. In the tested cases, the number of clusters to use to get the best prediction accuracy seemed to depend very much on the used datasets, when the tested cluster sizes were 20, 40 and 80. In some cases, having more clusters improved the performance, whereas in others, it did not have any significant impact, or even made the accuracy worse. We also found out that in some cases, having attribute cluster indicators in the input vectors improved the prediction even if the input vectors also included all the actual attribute values.

As future work, it would be interesting to test this clustering approach also with other machine learning model types such as more traditional random forest and gradient boosting machines. Similarly it could be interesting to first filter out some of the most rarely occurring attribute values before clustering the values. This could potentially reduce the amount of noise added to the clustered data and make it easier for the clustering algorithm to not be affected by noisy data. Finally, more study is required to understand whether similar clustering approach performed for event attributes in this work could be applicable also for encoding case attributes.

Acknowledgments. We want to thank QPR Software Plc for funding our research. Financial support of Academy of Finland project 313469 is acknowledged.

References

1. Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: encoder-decoder approaches. In: Wu, D., Carpuat, M., Carreras, X., Vecchi, E.M. (eds.) Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014, pp. 103–111. Association for Computational Linguistics (2014)
2. Evermann, J., Rehse, J., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
3. Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. CoRR, abs/1506.01428 (2015)
4. Francescomarino, C.D., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: which one suits me best? In: Weske, et al. [15], pp. 462–479

5. Hinkka, M., Lehto, T., Heljanko, K., Jung, A.: Structural feature selection for event logs. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 20–35. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_2
6. Hinkka, M., Lehto, T., Heljanko, K., Jung, A.: Classifying process instances using recurrent neural networks. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 313–324. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_25
7. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR, abs/1412.6980 (2014)
8. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: 2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, 27 November–1 December 2017, pp. 1–7. IEEE (2017)
9. Nolle, T., Seeliger, A., Mühlhäuser, M.: BINet: multivariate business process anomaly detection using deep learning. In: Weske, et al. [15], pp. 271–287
10. Pelleg, D., Moore, A.W.: X-means: extending k-means with efficient estimation of the number of clusters. In: Langley, P. (ed.) Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, 29 June–2 July 2000, pp. 727–734. Morgan Kaufmann (2000)
11. Schönig, S., Jasinski, R., Ackermann, L., Jablonski, S.: Deep learning process prediction with discrete and continuous data features. In: Damiani, E., Spanoudakis, G., Maciaszek, L.A. (eds.) Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2018, Funchal, Madeira, Portugal, 23–24 March 2018, pp. 314–319. SciTePress (2018)
12. Tax, N., Teinemaa, I., van Zelst, S.J.: An interdisciplinary comparison of sequence modeling methods for next-element prediction. CoRR, abs/1811.00062 (2018)
13. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
14. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Chasovskyi, D., Rozumnyi, A.: Tell me what's ahead? Predicting remaining activity sequences of business process instances, June 2016
15. Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.): BPM 2018. LNCS, vol. 11080. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7