Embeddings of Categorical Variables for Sequential Data in Fraud Context

Yoan Russac[1], Olivier Caelen[2], and Liyun He-Guelton[2]

[1] ENSAE ParisTech, Paris, France
`yoan.russac@ensae-paristech.com`,
[2] R&D Worldline Brussels, Belgium and Lyon, France
`firstname.lastname@worldline.com`,

Abstract. In this paper we propose a new generic method to work with categorical variables in case of sequential data. Our main contributions are: (1) The use of *unsupervised* methods to extract sequential information, (2) The generation of embeddings including this information for categorical variables using the well-known Word2Vec neural network. The use of embeddings not only reduced the memory usage but also improved the machine learning algorithms learning capacity from data compared with commonly used One-Hot encoding for example. We implemented those processes on a real world credit card fraud dataset, which represents more than 400 million transactions over a one year time window. We demonstrated that we were able to reduce the memory usage by 50% and to improve performance by 3% while using only a small subset of features.

Keywords: Categorical Variable, Word2Vec, Embeddings, Credit Card Fraud detection

1 Introduction

In a steadily growing e-commerce market with an overall loss due to fraudsters of 6.97 cents per every \$100 in 2016, detecting fraudulent pattern is of great importance. Fraud data possesses particularities: unbalanced class, concept drift, large amount of data, difficulty to define a cost function, overlapping, etc. [1]. For those reasons, fraud detection has gained popularity amongst the machine learning community during the past decade [1, 9].

Another difficulty in fraud detection is that categorical variables are over-represented with more than 80% of our total variables. These variables vary amongst others, including the country of the transaction, the merchant type, the currency used, etc. If we do not find a correct representation of these variables, the machine learning algorithms will not be effective. A standard way to deal with a categorical variable is One-Hot encoding. However, One-Hot vectors have two main shortcomings: (1) They are high-dimensional and sparse. (2) The relations between different values of categorical variables is ignored [2]. It was demonstrated in [2] that using a *supervised* method to create embeddings for categorical variables reduced the memory usage and improved the performance of the neural network as it gave a better data representation. Here, we propose an *unsupervised* method for our sequential fraud data. The advantage of *unsupervised* based methods is their flexibility for injecting different semantic knowledges such as external sources [11]. In our case, we used this approach to inject the sequential pattern of the transactions.

We proceeded the following way: we created sequences centered on the different cardholders. Once we had a set of sequences for different cardholders, we used an internal process for generating the embeddings with Word2Vec [5, 6]. Word2Vec is a neural network frequently used for Natural Language Processing (NLP) tasks [7, 10].

This paper is organized as follows: in section 2 we describe our approach, in section 3 we describe the credit card dataset and the experiment we did. Finally section 4 conclude with an outlook and our future work.

2 Approach

The Word2Vec neural network is often used to learn linguistic regularities by representing words with vectors. The ordinary usage of this neural net is to feed it with sentences from a corpus of documents and to collect the embedding vectors after a training phase. Our process is very similar. The equivalent of the sentences will be variable sequences from our fraud dataset. These sequences will then be used to train the Word2Vec neural network. In this section, we will describe our approach: how we obtained the sequences and how we generated the embeddings using Word2Vec.

2.1 Sequences Generation

There are two ways to generate the sequences. We can either generate sequences for each categorical variable or for joint variables. The joint variable can in addition of the sequential information take into account the interactions between different categorical variables.

Univariate Sequences Generation: In the first case, to create our sequences we decided to group the entire training set by cardholders. For example if the cardholder A bought during the days corresponding to the training set, in France, then in Belgium, and after that back in France, the associated sentence will be 'France Belgium France'. We proceeded the same way with all the cardholders in our training set. After this process we had a collection of sentences where each sentence was linked to the list of transactions of a particular cardholder. This process was repeated for every categorical variable.

Multi-variate Sequences Generation: In a fraud context the joint information of the country of the transaction and the merchant type really mattered. Variables used to monitor suspicious activity included this information. When using One-Hot encoding there was no easy way to have this sequential information. With two categorical variables we could decide to create their product, where for example 'FRA' and merchant_type = '1010' will give us 'FRA1010'. Once this new interaction variable was created, it was possible to use the previous procedure to create the embeddings for the joint variable. By doing so we could create any interaction variables between categorical variables and extract its sequential information through the use of our *unsupervised* embeddings.

Generating the sequences this way was meaningful because unlike the One-Hot encoding which only included the information of the current transaction, with this process the embeddings would extract some of the information of the sequences of transactions which can help detecting uncommon behaviors.

2.2 Word2Vec

The embeddings extracted from Word2Vec reflect the linguistic regularities and semantic information of the input word sequences. The outputs of the Word2Vec tool are the embedding vectors. A fundamental notion to comprehend the network is the context window. The context window represents the words surrounding a given word in a sentence. It contains past words but also future words. When using Word2Vec one has to set the size of the context window and the dimension of the embeddings to create the right neural architecture.

The neural network: Word2Vec contains 3 layers. The input layer, one hidden layer and the output layer. It is worth mentioning that two predictive methods are possible when using Word2Vec. The first one is skip-gram architecture, where given an input word context words are predicted. The second one is the CBOW (continuous bags-of-words) where given a context window we try to predict the appropriate word. We will focus on the CBOW predictive method. The significant part of the network is the weights between the hidden layer and the output layer. These weights are updated during the back propagation. At the end of the update the embedding vectors will be the weight matrix. The structure of the neural network is represented in the Fig. 1 where V denotes the cardinal of the vocabulary (i.e. the different words in the corpus) and N denotes the dimension of the embedding vectors (i.e. the hidden layer has N dimensions). With a CBOW structure the matrix of interest is $\widetilde{W}$. In the input layer, which contains C parts, where C is the context window's size, each one of the input words is one-hot encoded, thus each input block has a V-dimension. During the training phase the weights of the matrix $\widetilde{W}$ will be updated and after a sufficient number of iterations over the corpus, the embedding vectors are available.

The neurons colored in blue in the Fig. 1 are 0 with a unique 1 per block because the input words are one-hot encoded. They are C blocks corresponding to a C-size context window. The output of neural net is probabilities this is why the output layer is not colored in blue.

Subsampling frequent words: In a large corpus stop-words can be really frequent. Observing co-occurrence of a frequent word and another word will bring less information. To counter this imbalance between occurrences of words in the corpus, a discard probability taking into account the frequency of the word was introduced [6]. The more frequent a word in the sentences is the higher the

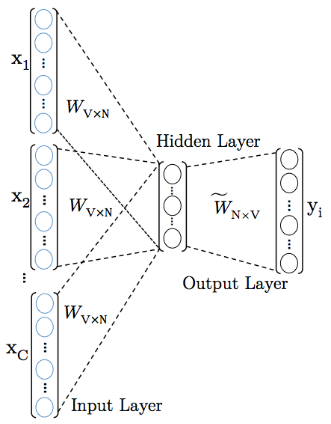

Fig. 1. Architecture of Word2Vec with CBOW technique

probability of being discarded will be. In our context because our data are coming from a Belgian payment processor, the country Belgium was overrepresented in the dataset. This is why we used undersampling technique.

Negative sampling: The idea behind negative sampling is to update only a sample of the output vectors at each iteration instead of updating the entire weight matrix.

These two extensions not only significantly reduced the computational requirements of the training process but also improved the quality of the embedding vectors [6].

3 Experiment

In this section we present our experiments. We compared the prediction's quality of several data structures: (1) One-Hot for every categorical variable, (2) Embedding for every categorical variable, (3) One-Hot and Embedding for every categorical variable (OHE in the different figures). We used either Logistic Regression or Random Forests to compare the precisions.

3.1 Data and Experimental Design

Data: We worked on a labeled dataset provided by Worldline company. This set contained one year of transactions. Each day was composed of approximately 500 000 transactions. We decided to keep 3 categorical variables: merchant type, country of the transaction, local currency and a quantitative variable which is the amount of the transaction. Specialists in the fraud detection domain use much more variables [1] but our aim was to compare the quality of the prediction. Please note that transactions must be separated in two categories: e-commerce transactions which take place on the Internet and physical transactions in other words face-to-face ones. For example if a cardholder A makes face-to-face transactions in America and in Belgium in a short time period there is probably something uncommon happening. However if they are e-commerce transactions it is more likely. We generated different embedding vectors in function of the e-commercial status of the transaction.

Experimental Design: Our experiment was structured the following way: (1) Randomly select an initial date. Create the corresponding training and testing set. (2) Files processing along the required structure (One-Hot, embeddings). (3) Apply the algorithm on this processed training and testing files. The process was repeated 100 times and is illustrated with algorithm 1

Algorithm 1 Experimental Design

```
Require: D: dataset, Processing method, L: learning algorithm
1: $Prediction\_list \leftarrow [\,]$
2: $d_1, .., d_{100} \sim Sample(100, length(D))$          ▷ Select 100 initial date
3: for $d_n \in \{d_1, ..., d_{100}\}$ do
4:     $(Training\_set, Testing\_set) \leftarrow ((d_n, d_{n+1}, d_{n+2}), d_{n+3})$
5:     $Training\_set \leftarrow Processing\ method(Training\_set)$
6:     $Model \leftarrow L(Training\_set)$
7:     $Prediction\_list \leftarrow Prediction\_list + [Model.fit(Testing\_set)]$
8: end for
9: return $Prediction\_list$          ▷ Predictions for the 100 testing files
```

3.2 Treatment of Categorical Variable

We used the Python's *gensim*[3] package [8] to generate the embeddings. It has a very efficient implementation of Word2Vec. We created a list of list containing all the sentences from users in the 3 days training set. The sequences were obtained following the method presented in subsection 2.1. For every categorical variable, we then had to train the Word2Vec neural network feeding it the associated sequence. For a given categorical variable we built a dictionary where every level of the variable was mapped to the associated embedding vector. We were then able to integrate these embedding vectors to a *Pandas*[4] dataframe.

As explained in the subsection 2.2 several parameters could be set for the configuration of Word2Vec. In our experiment we have chosen a context window of size 5, which mean that in best case scenario the five previous words and the five following words would be considered. We chose the CBOW predictive technique because we empirically noticed that it outperformed the Skip-Gram architecture for the experiment. Negative sampling was used with 5 *noise words* per training. The threshold for subsampling frequent words was set to 10^{-3}. The

[3]https://radimrehurek.com/gensim/
[4]http://pandas.pydata.org/

embeddings were generated with this configuration. Several embeddings dimensions were tested (see Table 1).

We also created an extended model where we computed the embeddings of the variable Country $\times$ Merchant Type. It is another advantage of embeddings which allows to create the interaction between several categorical features.

3.3 Resampling method

Before applying a Logistic Regression or Random Forests on the different training sets we had to deal with this unbalanced distribution issue (less than 0.2% of our entire dataset are fraudulent transactions). Machine learning algorithms usually perform poorly on such a training set [3]. We adopted a strategy similar to the *EasyEnsemble* strategy [4]. *EasyEnsemble* learns different aspects of the original majority class in an unsupervised manner. This is done by creating different balanced training sets by *undersampling*, learning a model for each dataset and then combining all predictions as in bagging. For a given training set our method was the following: (1) Collect every fraudulent transaction. (2) Select randomly a given number of non fraudulent transactions in the training set so that the proportion of non fraudulent transaction vs. fraudulent transaction in the new artificially created dataset will be 80%. (3) Create a model with this new dataset. We had to repeat (1-3) at least 100 times. To obtain a prediction on a new testing example, we took the average of the predicted probabilities of all the models created.

3.4 Performance measure

With fraud classification problems we have very unbalanced classes. Therefore classical performance measures are not suitable. With an overall proportion of 0.2% of frauds, classifying every transaction as a legitimate one gave an accuracy (i.e. proportion of correct classification) of 99.8%, even if the model was absolutely naive. We also aimed to use measures which made sense for real world fraud detection. Fraud experts check manually only hundreds of transactions (in function of the size of the team). Therefore we used the precision at k ($p@k$) which is the proportion of real frauds among the k riskiest transactions according to the chosen algorithm [1]. For example if $p@100 = 20\%$ then among the 100 riskiest transaction for the model, 20 of them were truly frauds.

The $p@k$ measure was somewhat variable, thus to compare the models we wanted to use a more global metric. Using the average of the K first transactions $p@k$ made sense in order to measure the global performance. Therefore we used the following metric:

$$Average\ p@K = \frac{1}{K} \sum_{i=1}^{K} p@i \qquad (1)$$

As already discussed in [1] there was no need to observe the quality of the model beyond K transactions with this metric because fraud experts were unlikely to manually check more than K transactions in a single day. In our experiment, we fixed K at 500.

3.5 Results

To sum up the experiment, we ran the model on 100 different 4-day-periods. For each one of those periods, we used a resampling method. This resampling method consisted in building 100 new datasets which were extracted from the training set and where classes are much more balanced.

When we created embedding vectors we manually chose their dimensions (N). As far as we know, there is no theory to know which dimension one should consider. In practice we chose them empirically. On average, the more levels a categorical variable had the larger the dimension we took because we assumed that higher dimensions would allow us to gather more precise information on the sequences of transaction. With our data the country variable had 180 levels, the merchant type had more than 600 levels and the local currency had 150 levels. We created four different sizes which are gathered in the Table 1. The interaction variable is only included when it is specified that the model is *extended.*

Table 1. Embedding configurations (N) of the experiment

Variables	Country	Merchant Type	Local Currency	Country $\times$ Merchant Type
Small dim	10	25	25	50
Med. dim	25	50	50	NA
Large dim	50	75	75	NA
Large High dim	80	150	150	NA

Logistic Regression: We implemented a logistic regression using the Python's package *scikit-learn*[5] and compared the results with One-Hot encoding versus other configurations. The results are reported on the Fig. 2 and Fig. 3. On the Fig. 2 we represented the Average p@500 and their $t-based$ confidence intervals ($\alpha = 0.05$). It shows that with small or medium dimensions using only embeddings was less effective than the classic One-Hot encoding but higher dimension improves the performance. Combining One-Hot encoding and embeddings gave an Average p@500 of 8.2% which must be compared to the 5.8% Average p@500 of the One-Hot encoding. Our method improved from 2.4% the results in a Logistic Regression configuration and was statistically better than a plain One-Hot.

Three main conclusions to be drawn were: (1) On average, the higher the embedding dimension was, the better the results were. But the difference was not significant with only 3 variables. (2) Embedding vectors gave slightly better results than One-Hot encoding but with a use of far less memory. In our example

[5]http://scikit-learn.org/stable/

the memory usage of the dataframe containing 3 days of data with embedding vectors with small dimensions was 2 times less than with One-Hot. (3) Combining One-Hot and embedding techniques gave us better results than using One-Hot only. The p@100 gained 3% to 4% when we combined the techniques and thus enriched the data.

Random Forests: We implemented a Random Forests algorithm using *scikit-learn* and compared the results with One-Hot encoding versus other configurations. We reported the Average p@500 for the different methods on the Fig. 4. With a non-linear algorithm the results were different. (1) The Average p@500 was much higher than with a Logistic Regression. (2) The previous remark concerning the dimensions of the embeddings seemed infringed with this algorithm. (3) Adding One-Hot to the embeddings was less effective, but embeddings alone performed significantly better than a raw One-Hot encoding according to this metric. The Average p@500 when using One-Hot encoding was less than 37.5% and it was around 40% when using embeddings. (4) When we observed the 95% confidence intervals (Fig. 4) we could assure that we have a significant improvement when using embeddings.
The best model for the different configurations are represented in the Fig. 5.

Remark: The confidence intervals are larger than with the Logistic Regression algorithm because the p@k vary much more with Random Forests than with Logistic Regression (as can be seen on Fig. 3 and Fig. 5)

On the Table. 2 we reported p@k for different values of k when using Random Forest with small embeddings configuration on the *extended* model. The *extended* model slightly outperformed the plain one. Different k values in the table demonstrated that the integration of the joint variable was a moderate success.

Table 2. Comparison between *extended* model and non *extended* one

	Small Embeddings	Small Extended Embeddings
p@50	55.31%	55.33%
p@150	43.27%	43.63%
p@250	37.48 %	37.73%
p@500	28.5%	28.72%

The conclusions for the Random Forest algorithm were: (1) Using non linear algorithm to detect fraudulent pattern performed well because p@100 rose from 9% in the best case scenario when using Logistic Regression to almost 48 % with Random Forest. (2) Replacing the One-Hot encoding worked: e.g. the p@100 performance increase from more than 3% by using them. (3) As can be seen in the Table. 2 using the extended model with the joint variable improved slightly the performance for different p@k measures. In average we achieved a 0.2% improvement by adding the joint variable.

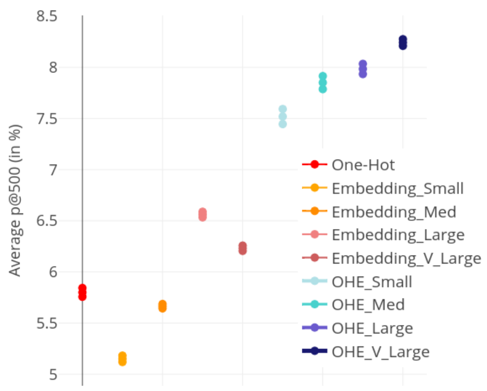

Fig. 2. *Average* $p@500$ and confidence interval using Logistic Regression (plot order follows the legend)

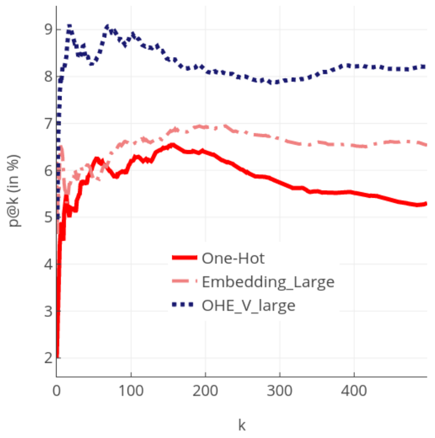

Fig. 3. $p@k$ in function of k with different encoding methods using Logistic Regression

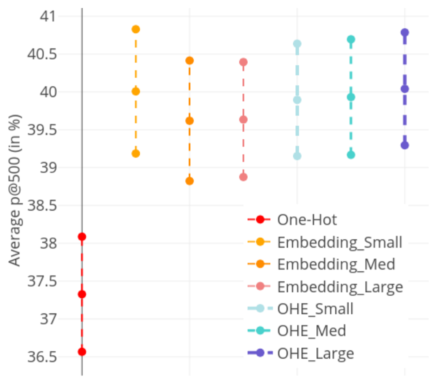

Fig. 4. *Average* $p@500$ and confidence interval using Random Forest (plot order follows the legend)

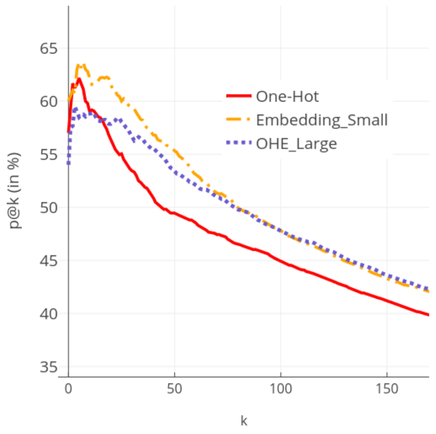

Fig. 5. $p@k$ in function of k with different encoding methods using a Random Forest algorithm

4 Conclusions

In this paper we made the following contributions: (1) We implemented a generic method to generate *unsupervised* embeddings for categorical variables and applied it to credit card fraud detection. This method can be used for any sequential data and allows to inject sequential information through embeddings. (2) We created the embeddings in two different ways: for each categorical variable independently and for joint variables. By doing so we enriched the dataset. (3) Combining One-Hot encoding and embeddings improved the $p@100$ performance, which is one of Worldline's performance metric, by 3% for both algorithms. (4) When using low dimensional embeddings we reduced the memory usage by 50% in comparison to One-Hot encoding. We proved that the fraud detection performance (Average p@500) was enhanced with the support of the 95% confidence level.

Due to the time constraint and heavy computation charge, we only tested our approach with 3 categorical variables and a small time window. In the future, we will extend our experiments with all categorical variables ($\approx$ 20 variables) and a larger time window. We also plan on implementing these methods in a *online learning* context and integrating the embeddings in an advanced machine learning algorithm, such as $LSTM$ neural network, to verify its efficiency.

References

1. Andrea Dal Pozzolo, Olivier Caelen, Yann-Ael Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications*, 41(10):4915–4928, 2014.
2. Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *CoRR*, abs/1604.06737, 2016.
3. Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.
4. Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2009.
5. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
6. T. Mikolov, I. Sutskever, K. Chen, G. S Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.
7. C. Musto, G. Semeraro, M. De Gemmis, and P. Lops. Word embedding techniques for content-based recommender systems: An empirical evaluation. In *RecSys Posters*, 2015.
8. Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.
9. I. Trivedi and M. M. Monika. Credit card fraud detection. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(1), 2016.
10. Yujun Wen, Hui Yuan, and Pengzhou Zhang. Research on keyword extraction based on word2vec weighted textrank. In *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*, pages 2109–2113. IEEE, 2016.
11. K. Ziegler, O. Caelen, M. Garchery, M. Granitzer, L. He-Guelton, J. Jurgovsky, P.-E. Portier, and S. Zwicklbauer. Injecting semantic background knowledge into neural networks using graph embeddings. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017 IEEE 26th International Conference on*, pages 200–205. IEEE, 2017.