e-ISSN 2757-5195



Çanakkale Onsekiz Mart University
Journal of Advanced Research in Natural and Applied Sciences
Open Access

doi.org/10.28979/jarnas.1014469 2022, Vol 8, Issue 2, Pages: 222-236 dergipark.org.tr/tr/pub/jarnas

Learning From High-Cardinality Categorical Features in Deep Neural Networks

Mustafa Murat Arat[1*]

[1]Department of Statistics, Faculty of Science, Hacettepe University, Ankara, Türkiye

Article History
Received: 25.10.2021
Accepted: 18.01.2022
Published: 10.06.2022
Research Article

Abstract − Some machine learning algorithms expect the input variables and the output variables to be numeric. Therefore, in an early stage of modelling, feature engineering is required when categorical variables present in the dataset. As a result, we must encode those attributes into an appropriate feature vector. However, categorical variables having more than 100 unique values are considered to be high-cardinality and there exists no straightforward methods to handle them. Besides, the majority of the work on categorical variable encoding in the literature assumes that the categories is limited, known beforehand, and made up of mutually-exclusive elements, independently from the data, which is not necessarily true for real-world applications. Feature engineering typically practices to tackle the high cardinality issues with data-cleaning techniques which they are time-consuming and often needs human intervention and domain expertise which are major costs in data science projects The most common methods of transform categorical variables is one-hot encoding and target encoding. To address the issue of encoding categorical variables in environments with a high cardinality, we also seek a general-purpose approach for statistical analysis of categorical entries that is capable of handling a very large number of categories, while avoiding computational and statistical difficulties. Our proposed approach is low dimensional; thus, it is very efficient in processing time and memory, it can be computed in an online learning setting. Even though for this paper, we opt to utilize it in the input layer, dictionaries are typically architecture-independent and may be moved between different architectures or layers.

Keywords − *Categorical variable, deep neural networks, high cardinality, mean target encoding, one hot encoding*

1. Introduction

In order to transform raw categorical variables before feeding them to the model, in an early stage of modelling, feature engineering is required when there are categorical variables in the dataset because some algorithms expect all the input variables and the output variables to be numeric. Thereby, as a result, we must encode those attributes into an appropriate feature vector. Encoding in this context refers to a parametric map from inputs to their representations. A categorical variable can be easily defined as a discrete collection of values (categories) that might be ordinal or nominal in form. Furthermore, categorical variables can be used as either a dependent variable or an independent variable in a prediction problem. However, this study focuses exclusively on the problem of nominal categorical input variables.

A categorical attribute's cardinality is defined as the number of distinct values that this attribute can take (Perlich & Provost, 2006). Traditional nominal attributes are those with a low cardinality. On the other hand, attributes having a very high cardinality are mainly referred in the literature to as high-cardinality attributes (Moeyersoms & Martens, 2015). Especially in the e-commerce, retail, and banking industries, some categorial predictors like user ID, item category, zip code, and user city might result in hundreds or even thousands of distinct levels. Hence, Moeyersoms & Martens (2015) propose a threshold and consider

[1] muratarat@hacettepe.edu.tr
*Corresponding Author

that nominal variables having more than 100 unique values are examples of "high-cardinality attributes". High-cardinality categorical columns are often critical features in machine learning tasks, representing serious challenges in cases such as classification and regression. Despite their promise, such variables are frequently overlooked in predictive modelling, due to the fact that there exists no straightforward engineering or statistical methods to handle them. The most common methods of transform categorical variables for machine learning algorithms is one-hot encoding or transformation to a continuous feature with prime example being mean target encoding (Micci-Barreca, 2001), which is especially most common technique in Kaggle competitions. The majority of the work on categorical variable encoding in the literature assumes that the categories is limited, known beforehand, and made up of mutually-exclusive elements-independently from the data, which is not necessarily true for real-world applications.

Although, they are time-consuming and often needs human intervention and domain expertise which are major costs in data science projects, typically, feature engineering employs data-cleaning techniques to address issues of high cardinality. (Rahm & Do, 2000) and thus reducing the number of theoretically relevant levels, finding strategies that work well across a wide variety of problems is critical for a wide variety of applications, including automatic machine learning (Feurer, et al., 2019; Thomas, Coors, & Bischl, 2018; Thornton, Hutter, Hoos, & Leyton-Brown, 2013). Optimally, strategies should be model agnostic. However, since here we will be dealing with deep learning architectures, our main focus is to handle this problem in the domain of neural network architectures with online learning.

In order to avoid this step, similarity encoding, which builds feature vectors from cross-category similarities, relaxes one-hot encoding by using string similarities. As a consequence, when used in conjunction with one-hot encoding, it resolves the problem of related categories and has been shown to enhance statistical analysis. It does not, however, solve the problem of high cardinality, and important information may be lost if the merging methodology is not exposed to statistical methods.

Feature hashing, commonly known as the hashing technique, is mostly used to minimize the number of categories (Weinberger, Dasgupta, Langford, Smola, & Attenberg, 2009). It is a technique to represent categories in a “one hot encoding style” as a sparse matrix but with a much lower dimension. Rather than keeping a one-to-one mapping between categorical feature values and feature vector positions, we utilize a hash function to identify the feature’s location in a lower-dimensional vector. Therefore, a large number of values is mapped into a small finite set of values. In practice we are free to specify the dimensionality of the vector space (output). This has the desirable effect of not only reducing dimensionality and therefore computation. It also provides an elegant way of dealing with new or missing entries. However, because its size is arbitrarily chosen, this technique may blindly assign radically different categories to the same embedding vector (multiple distinct values may produce the same hash), a phenomenon known as collision, which results in information loss and model quality degradation. In other words, the smaller the hash size, the more collapsed levels there are. An advantage of this approach leads to a disadvantage. We do not need to keep the exact one-to-one mapping of features to indices that other encoding methods demand. That is why, exactly because we do not save the explicit feature mapping during feature extraction, we lose the ability to execute the inverse mapping from feature indices to feature values.

Guo and Berkhahn (2016) present a neural network-based encoding technique, which maps high-dimensional sparse vectors to a dense representation in a lower-dimensional continuous space that preserves semantic relationships by grouping commonly co-occurring items together in the representation space and further build neural networks to learn the latent patterns. This technique is intriguing since the mapping is learned during the supervised training phase by a neural network. To put it another way, the embeddings serve as the network’s parameters - or weights - and are modified iteratively to minimize performance loss. The resultant embedded vectors are representations of categories in which comparable categories are more closely related to one another in relation to the goal of the analysis.

The concept of embeddings has been expanded and developed in several ways (Bengio, Schwenk, Senécal, Morin, & Gauvain, 2006; Mikolov, Chen, Corrado, & Dean, 2013; Mikolov, Sutskever, Chen, Corrado, & Dean, 2013; Pennington, Socher, & Manning, 2014; Levy & Goldberg, 2014; Mnih & Kavukcuoglu, 2013) but each reflects the same main ideas. They have proven critical in a variety of natural language processing applications (Li & Yang, 2018), and additionally, they have been extended to non-textual datasets (Rudolph, Ruiz, Mandt, & Blei, 2016). One can use either pre-trained embeddings for that particular categorical

variable, if exists, as in (Mikolov, Chen, Corrado, & Dean, 2013; Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), or learn the embeddings as part of the training process.

The intuition behind embeddings is that - instead of using one-hot encoded variables, which implies that each input level is orthogonal to every other - we allow for dependence (overlap information) between the levels of categorical variables. Using an embedding layer instead of one-hot encoding significantly lowers the number of parameters and leads to improved predictions, but at the price of higher training and inference time. Most importantly, learning the embeddings as part of the network increases the model's complexity by adding many weights to the model, which means you will need much more labelled data and repeated occurrences of individual exemplars in order to learn. Additionally, it incurs computational and architectural selection costs associated with Deep Learning techniques. It also relies on single-hot encoding, which has some of the same issues (Cerda, Varoquaux, & Kégl, 2018).

Additionally, dimensionality selection for embedding is a well-known research problem. The impact of dimensionality on an embedding layer has not yet been fully understood. In practice it is chosen based on experiments (either ad-hoc or by grid search) (Yin & Shen, 2018). As a critical hyperparameter, the choice of embedding dimensionality has huge influence on the performance.

A low-dimensional embedding layer is generally insufficiently expressive to capture all potential relations, but a high-dimensional embedding layer suffers from over-fitting. With increasing dimensionality, the quality of the embedding improves. However, at a certain point, marginal benefit diminishes (Mikolov, Chen, Corrado, & Dean, 2013). Typically, the vectors have a dimension of between 100 and 1,000. However, there is not a single general rule. In a Google Developers' blogpost (https://tinyurl.com/gglTFDev), it has been suggested to take the 4th root of the number of categories. Another recommended rule of thumb for finding the embedding size is to divide the cardinality size by two, but not to exceed 50 (https://tinyurl.com/mediumEmb).

To address the issue of encoding categorical variables in environments with a high cardinality, we seek a general-purpose approach for statistical analysis of categorical entries that is capable of handling a very large number of categories without requiring human intervention, while also avoiding computational and statistical difficulties.

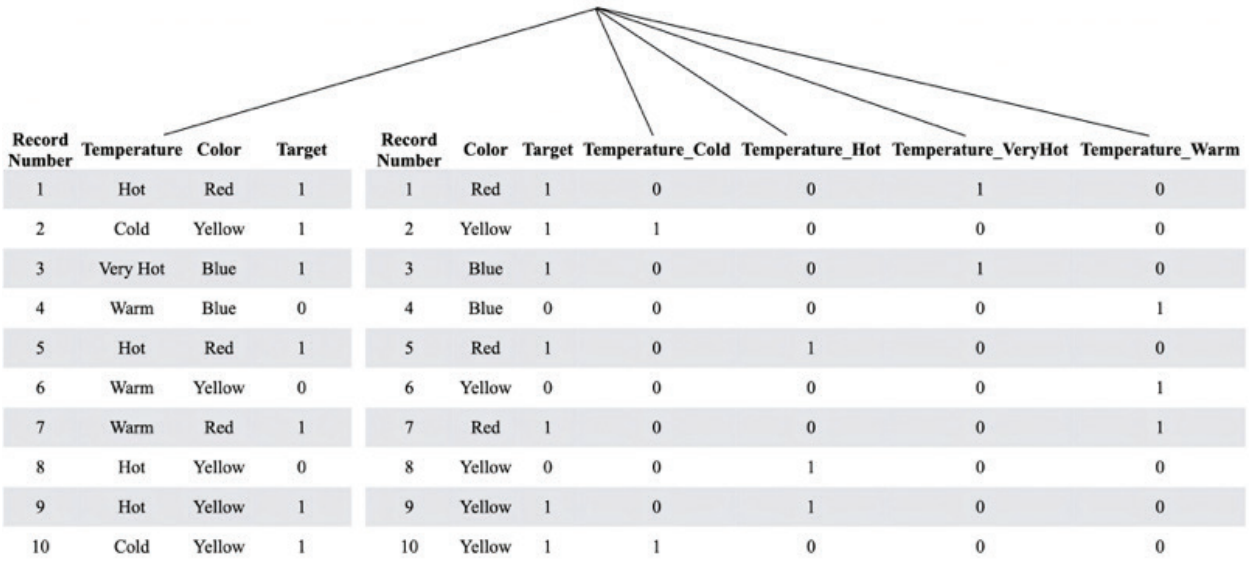
2. Materials and Methods

Before we mention about a novel applied technique for categorical attributes with high cardinality, let us first consider the objectives and some relevant guidelines for methodology development.

2.1. One-Hot Encoding

In supervised learning, we are given N data points, (x_i, y_i), $i = 1,2, \ldots, N$ with a task to learn a mapping from input variable x to the target variable y. Let's assume that x is a categorical variable which has l non-numeric levels, $\{x \in x^{(l)}, x^{(2)}), \ldots, x^{(l)}\}$. Traditionally, handling a categorical variable is a method known as dummy encoding. It's a straightforward and frequently-used encoding technique. This approach has been published for the first time in 1957 and was mostly used in regression analysis (Suits, 1957). Therefore, dummying is implemented by introducing l-1 binary dummy variables into the feature set for a categorical variable with l levels.

The popular one-hot coding scheme is an extension of this method and is the de-facto standard for machine learners on categorical variables. Even though numerous alternative implementations of the one-hot encoding have been presented in the literature., the most common variation basically creates a binary column for each unique levels of the categorical column (Cohen, Cohen, West, & Aiken, 2002). Similar to dummying, this approach is applied to dichotomous variables where each level is represented by a vector of zeros with an entry of 1 each time that particular level appears in the data. In the vector space that we will have in the end, each category is orthogonal and equidistant to each other. It has been added to many software packages as a very early stage before applying any machine learning algorithm. Figure 1 shows a visual example where this type of encoding is applied on an example variable. In this example, we have one categorical variable, *Temperature*, with four different levels, i.e., Hot, Cold, Very Hot, and Warm. Using one-hot encoding approach, we can easily process it numerically and feed into any algorithm.



Record Number	Temperature	Color	Target
1	Hot	Red	1
2	Cold	Yellow	1
3	Very Hot	Blue	1
4	Warm	Blue	0
5	Hot	Red	1
6	Warm	Yellow	0
7	Warm	Red	1
8	Hot	Yellow	0
9	Hot	Yellow	1
10	Cold	Yellow	1

Record Number	Color	Target	Temperature_Cold	Temperature_Hot	Temperature_VeryHot	Temperature_Warm
1	Red	1	0	0	1	0
2	Yellow	1	1	0	0	0
3	Blue	1	0	0	1	0
4	Blue	0	0	0	0	1
5	Red	1	0	1	0	0
6	Yellow	0	0	0	0	1
7	Red	1	0	0	0	1
8	Yellow	0	0	1	0	0
9	Yellow	1	0	1	0	0
10	Yellow	1	1	0	0	0

Figure 1. An example on creating dummy variables

When the number of categorical features and the number of possible levels is modest, this transformation produces acceptable performance. However, when one-hot encoding is used, high-cardinality categorical variables provide a variety of difficulties.

First of all, when we are dealing with high-cardinality categorical variables, this method can end up being intractable and introduce sparsity due to the expansion of the feature matrix because the majority of predictive modelling approaches cannot handle such dimensions. It is highly prone to over-fitting, resulting in a poor statistical estimation because it may be plagued by the curse of dimensionality, particularly in deep learning applications. Previous research has shown that eliminating unnecessarily large feature input space increases the robustness of neural networks (Xu, Evans, & Qi, 2018).

We can perform dimension reduction on the one-hot encoded matrix, albeit at the expense of information loss. It also often results in an unrealistic amount of computational and memory requirements. As the number of distinct categories increases, so does the number of model inputs, and therefore the number of parameters to estimate. Secondly, while encoding all unique items using orthogonal vectors, we completely disregard the overlap information which can be present in the representations, thus we might ignore the rich relationships existing among categories that can be exploited during training. Thirdly, the resulting binary features can be strongly correlated.

2.2. Mean Target Encoding

Another popular approach to incorporate the information from categorical variables is called mean target encoding, which creates features that include information about the target variable. It is also known as likelihood encoding. It is one of the most powerful techniques in feature engineering which is reported to be useful by many practitioners on the influential machine learning competition platform, Kaggle (https://www.kaggle.com), where many challenging datasets contain high cardinality features. It is also built into popular machine learning libraries, which use tree-based Boosting algorithms, such as LightGBM (Ke, et al., 2017) and CatBoost (Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2018).

Target encoding method is overall continuousification of categorical variables via target using a statistical aggregate function, e.g., mean, median, variance. The fundamental transformation in this technique is one that translates each value of a categorical variable to the target variable's probability estimate. The categorical variable is replaced with a single new numerical variable, and each category of the categorical variable is replaced with its corresponding probability of the target (if categorical) or mean of the target (if numerical), a process known as mean encoding.

Main purpose of this technique is to deal categorical features without exploding dimensionality of the dataset, which its one of the strategy's early formal descriptions is given in Micci-Barreca (2001). There are also similar ideas existed in the credit scoring literature (Hand & Henley, 1997).

Following the previous example, mean target encoding maps each level of into a feature, say, $\kappa \in \{\kappa^{(1)}, \kappa^{(2)}, \ldots, \kappa^{(l)}\}$ as given in Equation 2.1.

$$\kappa^{(j)} = \frac{1}{N^{(j)}} \sum_{i=1}^{N} y_i \, I\{x_i = x^{(j)}\} \tag{2.1}$$

where $I\{\bullet\}$ is the indicator function and $N^{(j)} = \sum_{i=1}^{N} I\{x_i = x^{(j)}\}$. That is to say $K^{(j)}$ is the average -value in level j and $N^{(j)}$ is the total number of observations in level j. Figure 2 shows a visual example where mean target encoding is applied on an exemplary *Temperature* variable.

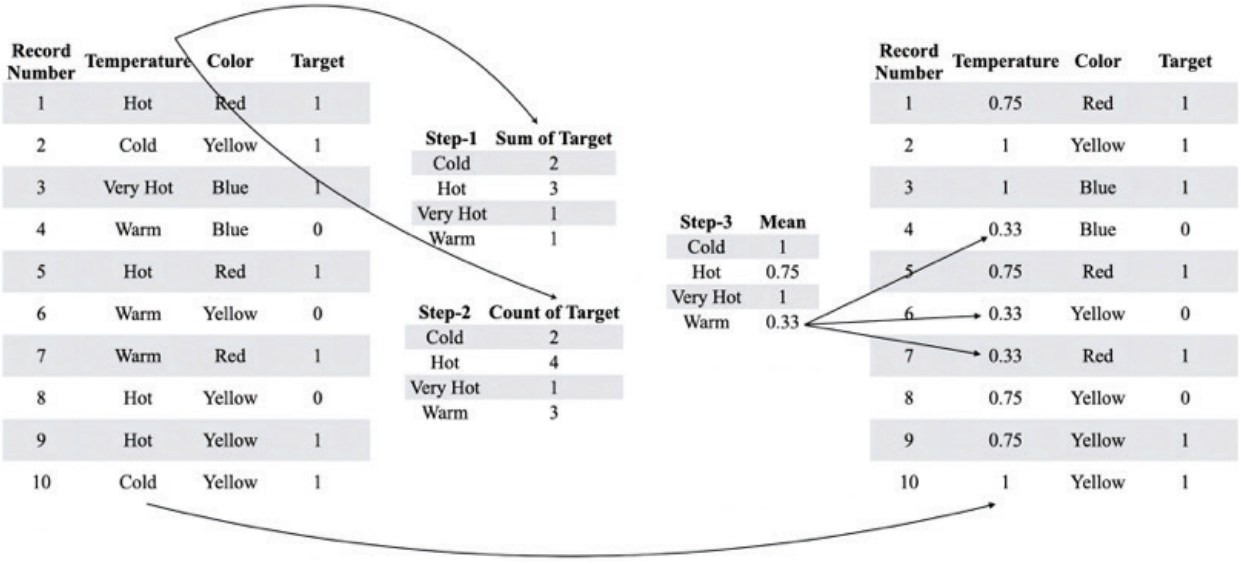


Figure 2. An example of how to use mean target encoding approach

One issue with target-based encoding is that some of the categories would have a very small number of samples in the training data, e.g., zip-codes with small populations, since calculated values are based on the frequency of the levels. In the extreme situation of a categorical feature having only unique values, the mean target for each level of this feature will be similar to the true target value of a single observation. For example, when $N^{(j)} = 1$. When $x_i = x^{(j)}$, the summation in Equation 2.1 includes just one non-zero term. As a result, we get $\kappa^{(i)} = y_i$; that is, $\kappa^{(i)}$ is encoded with the exact value we are attempting to predict. This would make the average target (label) values for those small categories (or only unique categories) unstable as this will cause overfitting of the training set and will lead to poor generalization of the model. This issue is exacerbated when working with high-cardinality categorical variables. In order to avoid this kind of overfitting, we may introduce regularization, or add random noise to the representation of the category in train dataset or use Double Validation. However, all these strategies have important disadvantages. First, they introduce additional hyperparameters for regularization and noise variance, which can be expected to be crucial for the effectiveness of the encoding. Second, they can tremendously increase the amount of computation especially for large samples.

2.3. Deep Learning

Classical machine learning algorithms are somewhat limited in their capabilities to learn from raw data and how the data is fed to the model has a huge influence on their performances (LeCun, Bengio, &

Hinton, 2015). Thus, feature engineering focuses on converting data into meaningful features in the goal of accurately capturing the fundamental structures (Domingos, 2012). It is an essential step of any kind of data analysis but can be tedious, time-consuming, domain-specific and requires substantial human-effort.

Deep neural networks are a subset of machine learning models that have been effectively applied to a variety of tasks. Through a hierarchical learning process, they automatically extract high-level, complex abstractions as data representations. At each level of the hierarchy, complex abstractions are learned based on comparatively smaller abstractions defined in the preceding level (LeCun, Bengio, & Hinton, 2015). This technique enables neural network models to efficiently learn high-order features from raw input.

Deep learning methods are used to discover the parameters of a nested parametrized non-linear function by utilizing gradient descent algorithm to minimize an example-based differentiable cost function. Deep architectures of consecutive layers create aforementioned nestedness. Each layer performs a nonlinear transformation on its input and outputs a representation. The output of this algorithm is determined by characteristics of the features and the weights associated with the interconnections among them.

Let's assume that we have a data set where we have n observations and p features, which can be shown mathematically as $x^{(i)} \in \mathbb{R}^p$, $y^{(i)} \in \mathbb{R}$, $i = 1, 2, 3, \ldots, n$. A feature vector $x^{(i)}$ represents a record, an item, a picture, or a vector expressing a word, for example. Its output instance $y^{(i)}$ is either a real-valued vector or a scalar, as in regression, or an integer- or scalar-valued vector or scalar, as in classification. We want the machine to learn from the training set and then to predict $y^{(i)}$ using the information given by $x^{(i)}$, so that we can extract the underlying mapping function that maps $x^{(i)}$ to $y^{(i)}$. We want to build a neural network $f_\theta: \mathbb{R}^p \rightarrow \mathbb{R}$ which maps an input x to a predicted output ŷ, where θ is model parameters, i.e., weights and biases in the network. The forward propagation equation for this particular sample is then given Equation 2.2.

$$f_\theta(x) = W^{(L)T}\, \phi_{L-1}\left(W^{(L-1)T}\left(\ldots \phi_2\left(W^{(2)T}\, \phi_1\left(W^{(1)T}x + b^{(1)T}\right) + b^{(2)T}\right)\ldots\right) + b^{(L-1)T}\right) + b^{(L)T} \quad (2.2)$$

where $\theta = [W^{(1)}, W^{(2)}, \ldots, W^{(L)}, b^{(1)}, b^{(2)}, \ldots, b^{(L)}]$. We begin by enumerating the layers using the index , where each layer has a weight matrix and a bias vector $b^{(l)}$, $l = 1, 2, 3, \ldots, L$. $\phi_l(\cdot)$ which is a function from a real space to another real space is called an activation function and it can be different for each layer and may or may not exist for the final layer based on the task we try to solve. We stack multiple layers, here. Each layer consists of M_l hidden units, where the dimension of which might vary depending on the layer. The very first hidden layer, which corresponds to $l = 1$, has a weight matrix $W^{(i)}$ of the dimension $p \times M_1$, where p is the number of neurons in the input layer and a bias vector of the dimension $1 \times M_1$. For the final layer, if it is a regression problem, $W^{(L)}$ has the dimension $M_{(L-1)} \times 1$ and bias vector $b^{(L)}$ is a scalar. For applications of multi-class classification with K classes, each neuron in the output layer will represent a class, therefore, $W^{(L)}$ has the dimension $M_{L-1} \times$ K and bias vector b^((L)) has the dimension $1 \times K$. For all the intermediate layers, $l = 2,3,\ldots,L-1$, $W^{(l)}$ has dimension $M_{l-1} \times M_l$ and the corresponding bias vector is of size $1 \times M_l$ Figure 3 shows a schematic representation of an artificial neural network structure for multi-class classification that includes an input layer, $L - 1$ hidden layer, and an output layer (layer L).

Final layer, which is also known as output layer, provides the output of the model. Depending on the problem, we decide whether to use an activation function or not. If it is a regression problem, the output layer neurons often do not have an activation function (or it can also be thought as having an identity activation function) since the values are unbounded. If it is a binary or multi-label classification problem, sigmoid function will provide the probabilities of the input belonging to each of two categories. If it is a multi-class classification, a Softmax layer is appended to the model for the same reasoning (Russell & Norvig, 2020).

Algorithms for deep learning are not new. However, as volume of data has increased and computational capacity have become more accessible, these networks have garnered more interest in recent years. The models can be enormous in size – often millions of parameters - call this the complexity of the model since each layer to be added carries a weight matrices and bias vector with it. Therefore, in order to train these models to get the most precise and accurate results, a vast amount of labelled input data is required.

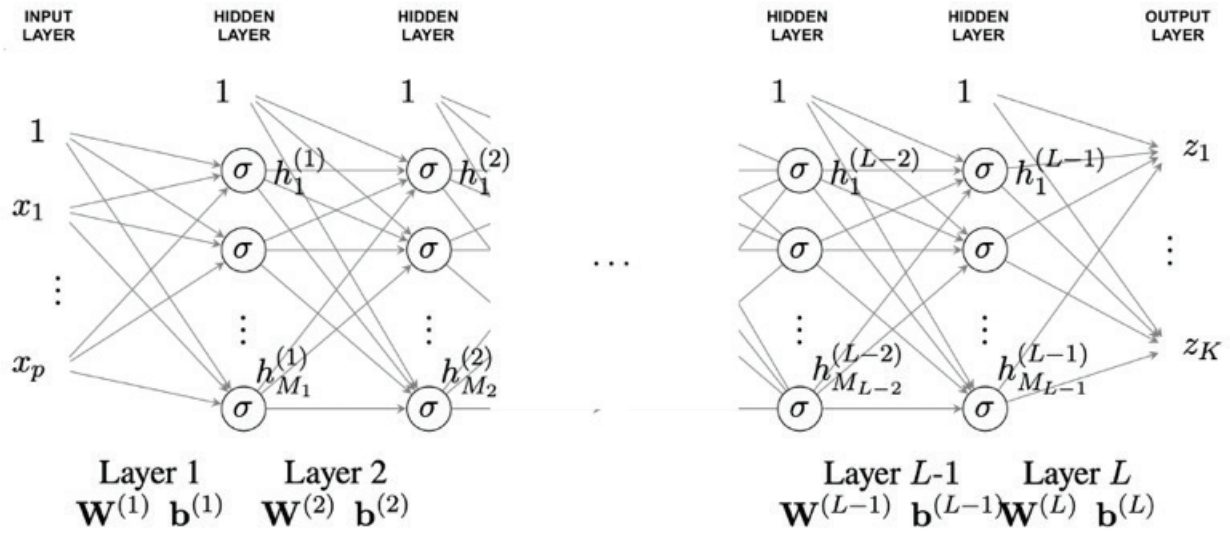


Figure 3. A schematic of the deep learning architecture.

2.4. Online learning

Deep Neural Networks are typically trained by back-propagation in a batch setting, requiring the entire static training data to be first acquired prior to the learning task. This type of learning is called offline learning, also sometimes known as batch learning. While running a deep neural network model, dataset to be processed is potentially of a very large size, which might not fit in the memory, thus can raise the issue of storage and memory management. With high volume incoming data, oftentimes, reading in the data all at once is not feasible. It is not scalable for many real-world settings where we mostly work with streaming data. Because online learning updates its model parameters progressively using just the most recent data points, the system does not need to keep a huge quantity of data in memory; moreover, after data has been consumed, it is no longer necessary. In comparison to offline learning, the model may change on the go to keep up with real-time trends.

If the data is too big to fit in the RAM, we can read it in chunks or one observation at a time. Since we do not see the entire data set, we cannot not know number of unique values in a categorical variable. Therefore, in online learning fashion, an inherent limitation called "absent levels" problem, whose consequences have never been carefully explored in the literature (Au, 2018), might arise for both of these encoding approaches because of the fact that we need to have the same levels of a categorical variable in three sets. In real-life problems, the categorical variable is not static and therefore, through the time, new levels may emerge. In other words, new, previously unseen unique values might be discovered in "unseen data" (test/validation sets) and the encoder does not know how to deal with these values. The transformed training, validation, and test sets must all have the same number of columns in order to be used in any machine learning method (Géron, 2019).

Thus, in this type of learning scheme, one-hot encoding does not fare well in the presence of new and unseen levels, which are encoded in the training set because we need to go over all the data to do so. One possible approach to handle this while using One-hot encoding approach, is to train an explicit "Unknown" level (like "other" or "missing") based on some prepared training data, where it will contain all 0 values for all the cells in the row. In the validation and testing phases, all new levels are subsequently recognized and mapped to this "Unknown" level. And then, we periodically refresh the model to include the recent appeared levels into this "Unknown" level, which will create collisions. Consequently, one-hot encoding is unsuitable not just for online learning but also for offline versions: if additional (new) categories are added, the whole dataset's encoding must be recalculated, and thus, the feature vector's dimension becomes unbounded.

Similar case is also valid for target encoding. For each level of a categorical variable, computing the summary statistics of corresponding values of target variable is more challenging. Additionally, where some new category is found, which are not available in training data, we substitute values of that category with the global mean of the target variable. This works fine as long as we have large amount of training data and a categorical feature with low cardinality. But this fails to work in the other cases.

However, the procedures mentioned above are not elegant at all and can cause a lot of loss of information. For the proposed method, this does not pose a problem because we can dynamically initialize a new key-value pair, where key represent the new category and value is a randomly initialized weight for this new category and then we keep on training.

2.5. The proposed method

So, we introduce a look-up-based approach arranged as key-value pairs, where the categorical levels are the keys and their corresponding float-numbers weights are the values. A hash table provide a constant time lookup and update for a mapping from a key to a value.

A dictionary, which has been created using unique levels of categorical variable, is initialized randomly or from zero. Here, we add an additional bias term to Equation 2.2. Training involves jointly learning a dictionary of weights of levels of categorical variables with high-cardinality and a small set of affine transformations using the concatenation operator. Weight updates will be done using backpropagation algorithm (Rumelhart, Hinton, & Williams, 1986).

Our proposed approach is low dimensional; thus, it is very efficient in processing time and memory, it can be computed in an online learning setting. Even though for this paper, we opt to utilize it in the input layer, dictionaries are typically architecture-independent and may be moved between different architectures or layers.

2.6. Data

An American direct-response television shopping network serves as the context of this study. It is a network which does broadcasting live shows 24 hours a day and 7 days a week on its own channel and sells exclusive hedonistic products in multiple product categories. The dataset used is obtained from this company, contains 362,235 data points. It consists of marketing and show related information, including gross margin as the dependent variable. Table 1 summarizes the variables in the dataset.

The data set includes features that are both continuous and nominal. We differentiate categorical features into two kinds of nominal attributes: traditional features and features with a high degree of cardinality. The attribute Master ID is indicated as high-cardinality data, and this attribute has more than 100 distinct categories.

Identifier characteristics, such as product identification numbers or personal names, which might contain a high number of categories, are rarely used in statistical modeling (Perlich & Provost, 2006). Therefore, in this study, we use Master ID variable, which is a unique ten-digit number identifier for every unit in the dataset. In the dataset, there exists 6944 unique IDs, where we are going to use as a categorical variable with high cardinality.

The salespersons variable (i.e., show host), where we use person-specific dummy variables to represent it, carries the name of professional individual who pitches this particular product and there exists 53 different people hosting the shows, in which the product was shown. Show type variable indicates general business unit associated with a show where the product was shown. It is one of the two types of products whose names are not disclosed for confidentiality. Merchandise Department variable describes the features of the product and consists of 24 unique levels. Country of Origin variable represents the country of manufacture and/or production where the product comes from. In the dataset, there are 40 different countries. All the traditional nominal variables are dummy encoded.

Table 1.

Attributes included in the data

Continuous	Traditional Nominal	High-cardinality Nominal
Unit Offer Price	Salesperson	Master ID
Unit Cost	Show Type	
Gross Margin	Merchandise Department	
Duration in Seconds	Country of Origin	
Showing Start Time in Minutes		
Showing End Time in Minute		

Descriptions of variables, unit cost and unit offer price are self-explanatory. Unit cost is the cost and unit offer price is the price of one unit of a particular product being sold. Duration is adjusted display time on-air in seconds for a given presentation of the product. Showing Start Time in Minutes and Showing End Time in Minutes represent the start and end time of the product in minutes.

2.7. Model Architecture

In this work, our main focus will be on generic non-linear feed-forward neural network, where layers are fully-connected to each other and each layer has number of hidden units that are of the same order of magnitude, as this is the most commonly implemented network architecture and fairly easy to train. We have a regression problem where we try to predict gross margin of the product. We highlight that our goal here is not to continuously employ various tricks or fine-tune model parameters in order to evaluate the performance of different machine learning algorithms at prediction. Rather, the major goal of this work is to test and assess the usefulness of the suggested approach in a circumstance when additional tricks are not present.

Since we compare three different encoding schemes based on some factors, we keep exactly the same neural network architecture across all three methods. All the models are trained independently. As previously mentioned, we treat only one categorical variable with high cardinality, i.e., Master ID variable.

We use a neural network architecture with four dense layers, which consist of decreasing number of neurons at each layer, i.e., 1024, 512, 256, 128, respectively. Since this is a regression problem, the output layer has only one neuron. To introduce non-linearity into the system, we focus on the popular Rectified Linear Unit (ReLU) function which can be mathematically shown as $\sigma(x) = \max(x, 0)$ for all the hidden layers, though our results can be extended (at notational cost) to cover other activation functions. Because of its superior gradient propagation and efficiency in calculation, the ReLU activation function is the most commonly used in practical applications (Glorot, Border, & Bengio, 2011). It often achieves better performance. Since this is a regression problem, linear activation function is used in the output layer, where no transform is applied at all.

All the weights in the models were initialized randomly using Glorot (also known as Xavier) Initialization schema in order to prevent gradients from exploding or vanishing (Glorot & Bengio, 2010). The idea is to initialize each weight with a small Gaussian value with zero mean and variance based on the number of input and output units of a particular layer. Biases were typically initialized as 0. Similarly, the dictionary which carries the weights for each of the product IDs has been initialized randomly.

All the models are trained from scratch using the entire training data set to learn the parameters. While training networks, we follow the standard practices (Bengio, 2012). Trainings are performed with one of the adaptive gradient methods, Adam optimizer (Kingma & Ba, 2015), which minimizes Mean Squared Error loss function using a batch size of size 32 which is selected through the preliminary experimentation, and learning rate of 0.0001 in all cases. Since we use Adam optimizer, we do not decay the learning rate. The other parameters were maintained at their default values $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times e^{-7}$.

Given that the number of epochs for which the algorithms were trained might affect the final results, all models were trained for 200 epochs.

Implementation is done using Python and one of most well-known open-source Deep Learning framework, low-level Tensorflow. All of our experiments run on 4 identical Nvidia GeForce GTX 1080 Ti GPUs, each one has 3584 CuDA cores and 11 GB GDDR5X VRAM memory and Neural Networks were constructed using Google's famous Python library, Tensorflow 1.15.0.

3. Results and Discussion

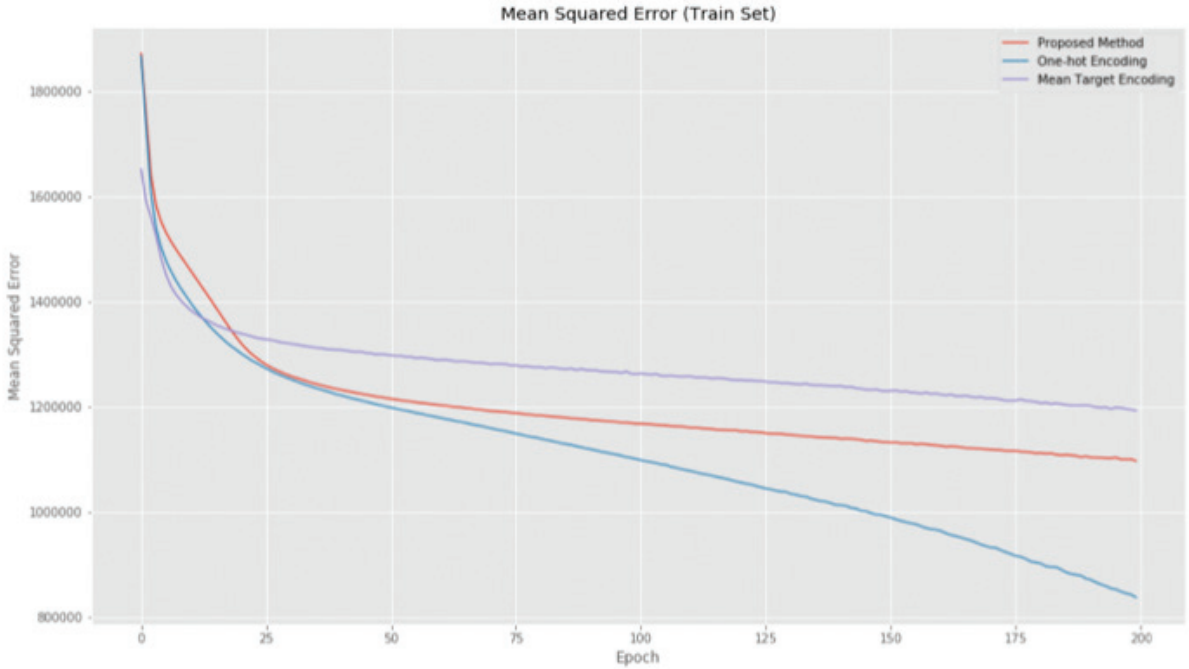
Preprocessing is needed before to applying the various encoding methods to the data in order to make it suitable for analysis. Therefore, we perform normalization on continuous features, and we use dummy encoding on traditional nominal features because of the reasons we mentioned above.

When developing predictive models, it is crucial not to leak any information from the training dataset into the validation set and testing set, which are created to simulate real-world, unseen data because of the generalization being the goal. Data preprocessing (including creation of dummy variables and computing mean values for the levels of categorical features from target variable) needs to be done after splitting the data. Each such preprocessing should be first done on the training set before being combined or concatenated with the validation and test subsets. Therefore, first, we split the data in to a training set and test set with the ratio 9:1. Following that, the remaining portion of training set is split into a training portion (3/4) for training the model and a validation portion (1/4) for identifying the best model with the lowest loss value (Géron, 2019). Since our purpose for this study is to compare proposed method with two existing encoding approaches, we keep the same levels of high-cardinality feature balanced out for each of the sets that have been created, meaning that, we will not be dealing with unseen category problem.

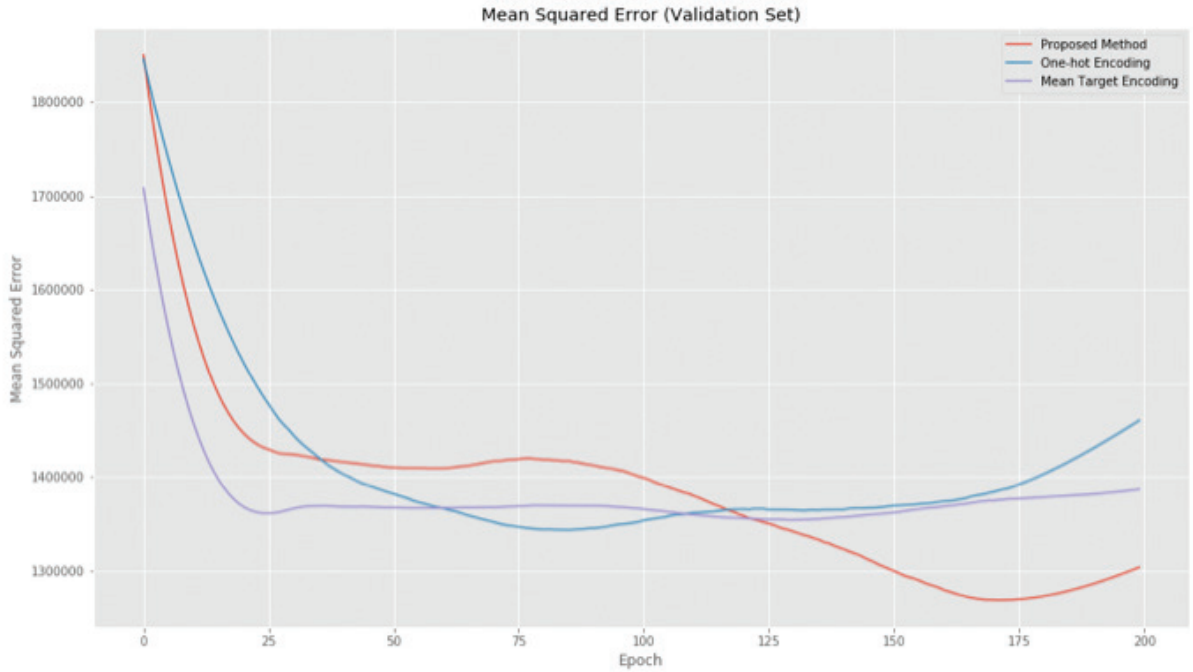
After the preparation of the datasets, we then follow the instructions given in the previous section to run three different neural network models, where the high-cardinality categorical feature, Master ID, has been encoded using three different approaches. Therefore, if we compute the number of neurons in the input layer, we have 7,068 neurons for one-hot encoding, 125 neurons for mean target encoding and 124 neurons for proposed method. The total number of trainable parameters for those models are 7,895,041 and 785,409 and 791,329, respectively.

Now, we can take a closer look at the performance of all the encoding methods.Figure 4 shows the predictive performance of models, in terms of mean squared error, for both training and validation sets. A first observation is that model with mean target encoding under-performs for both training and validation datasets. For model with one-hot encoding approach, the training error goes down sharply. At first, it seems like doing a good job, however, after a point, it starts to overfit the training set and not to generalize well to unseen data. For the proposed technique, the training error slowly decreases and so does validation error. In other words, without memorizing the training set, using this encoding technique, the neural network can learn unknown examples much better. We also track the training time of models, which is composed of two main components: the time required to obtain the data and the time required to process (learn from) the data, as shown in Figure 5. While training, the average time per epoch took roughly 821 seconds (approximately 14 minutes) for one-hot encoding, 289 seconds (approximately 5 minutes) for mean target encoding and 275 seconds (approximately 5 minutes) for proposed method. Overall training time was 45 hours, 16 hours and 15 hours, respectively. We may deduce from the graph that our suggested approach is considerably quicker in terms of training time per epoch and overall.

Based on these findings, we can confidently assert that our suggested approach reduces the human cost of encoding categorical variables with high cardinality, as well as the training time and cost of employing neural networks for applications.



(a) Model performance on training set



(b) Model performance on testing set

Figure 4. Comparing performances of three encoding approaches

4. Conclusion

We have presented a simple but efficient method for dealing with categorical attributes with a high degree of cardinality and compared its predictive performance with two well-known strategies that are used to encode categorical features with a high number of unordered levels. In summary, the empirical analysis shows promising results. In the proposed method, we can create an efficient data representation, without consuming computational resources. The proposed methodology is model agnostic, can be combined with

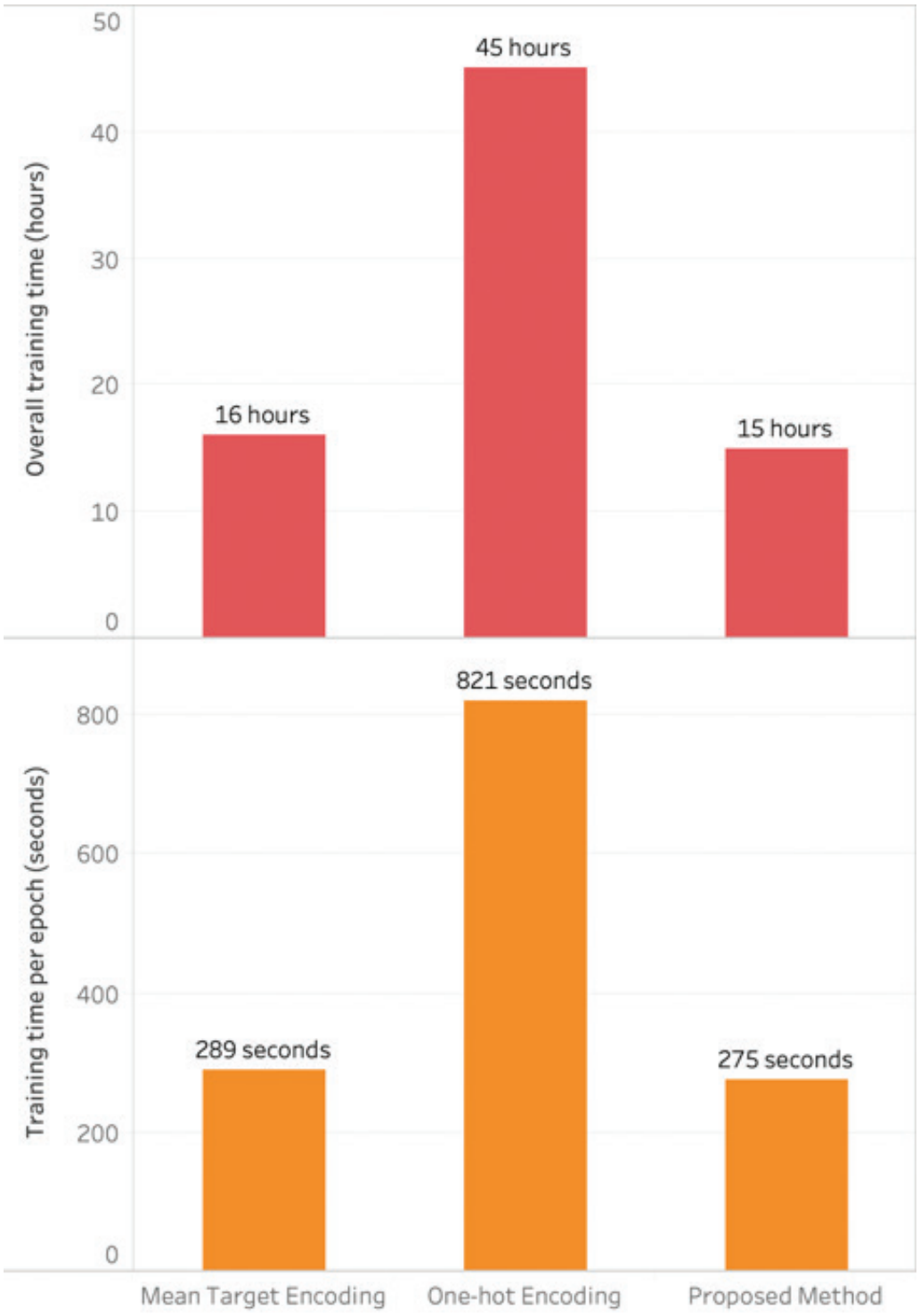


Figure 5. Comparison of training time of encoding methods

any supervised deep learning algorithm, and it may be utilized in online-learning environments, allowing for manageable analysis of extremely big datasets without the need for data cleaning. This article barely scrapes the surface of learning from non-curated tables with high-cardinality categorical features, a topic that is seldom addressed in machine learning. We hope to inspire further research into encoding of categorical features in the context of identifier attributes to be included in any deep neural network model via such a technique which is relatively straightforward to implement because the most of the papers in the scientific literature seems to circle around string attributes. We expect that the benchmark datasets will stimulate further research on this topic.

Acknowledgement

The author would like to thank his PhD advisor, Dr. Michel Ballings, for the all support that he provided.

Author Contributions

Mustafa Murat Arat: Planned the study, designed the paper, performed data analysis, produced the statistical results and outputs, made the visualization and wrote everything

Conflicts of Interest

The authors declare no conflict of interest.

References

Au, T. C. (2018). Random Forests, Decision Trees, and Categorical Predictors: The "Absent Levels" Problem. *Journal of Machine Learning Research, 19*, 1-30. Retrieved From: https://www.jmlr.org/papers/v19

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. In G. Montavon, G. B. Orr, & K. R. Müller (Eds.), *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science* (Vol. 7700). Berlin, Heidelberg: Springer. DOI: https://doi.org/10.1007/978-3-642-35289-8_26

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., & Gauvain, J.-L. (2006). Neural Probabilistic Language Models. In Holmes D.E., Jain L.C. (Eds.), *Innovations in Machine Learning* (Vol. 194, pp. 137-186). Berlin, Heidelberg: Springer. DOI: https://doi.org/10.1007/3-540-33486-6_6

Cerda, P., Varoquaux, G., & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. *Machine Learning*, 1477-1494. DOI: https://doi.org/10.1007/s10994-018-5724-2

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2002). *Applied multiple regression/correlation analysis for the behavioral sciences* (3rd ed.). Lawrence Erlbaum Associates Publishers. ISBN: 9780203774441

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM, 55*(10), 78-87. DOI: https://doi.org/10.1145/2347736.2347755

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). Auto-sklearn: Efficient and Robust Automated Machine Learning. In F. Hutter , L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning. The Springer Series on Challenges in Machine Learning.* (pp.113-134). Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-05318-5_6

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media, Inc. ISBN: 9781492032649

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, *9*, 249-256. Retrieved From: https://proceedings.mlr.press/v9/glorot10a.html

Glorot, X., Border, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, *15*, 315-323. Retrieved From: https://proceedings.mlr.press/v15/glorot11a.html

Guo, C., & Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. *arXiv*. Retrieved From: https://arxiv.org/abs/1604.06737

Hand, D. J., & Henley, W. E. (1997). Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statistical Society. Series A (Statistics in Society), 160*(3), 523-541. DOI: https://doi.org/10.1111/j.1467-985X.1997.00078.x

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Proceedings of the 31st International Conference on Neural Information Processing*

Systems, 3149–3157. Retrieved From: https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *The Proceedings of 3rd International Conference on Learning Representation.* San Diego, CA, USA. Retrieved From: https://arxiv.org/abs/1412.6980

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*, 436–444. DOI: https://doi.org/10.1038/nature14539

Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2177-2185. Retrieved From: https://papers.nips.cc/paper/2014/hash/feab05aa91085b7a8012516bc3533958-Abstract.html

Li, Y., & Yang, T. (2018). Word Embedding for Understanding Natural Language: A Survey. In S. Srinivasan (Ed.), *Guide to Big Data Applications* (pp. 83-104). Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-53817-4_4

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl., 3*(1), 27-32. DOI: https://doi.org/10.1145/507533.507538

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Proceedings of 1st International Conference on Learning Representations.* Scottsdale, Arizona, USA. Retrieved From: https://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 3111-3119. Retrieved From: https://papers.nips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

Mnih, A., & Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2265-2273. Retrieved From: https://proceedings.neurips.cc/paper/2013/hash/db2b4182156b2f1f817860ac9f409ad7-Abstract.html

Moeyersoms, J., & Martens, D. (2015). Including high-cardinality attributes in predictive models: A case study in churn prediction in the energy sector. *Decision Support Systems, 72*, 72-81. DOI: https://doi.org/10.1016/j.dss.2015.02.007

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 1532-1543. DOI: https://doi.org/10.3115/v1/D14-1162

Perlich, C., & Provost, F. (2006). Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning, 62*(1), 65-105. DOI: https://doi.org/10.1007/s10994-006-6064-1

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 6639–6649. Retrieved From: https://arxiv.org/abs/1706.09516

Rahm, E., & Do, H.-H. (2000). Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin, 23*, 3-13. Retrieved From: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.8661

Rudolph, M., Ruiz, F. J., Mandt, S., & Blei, D. M. (2016). Exponential family embeddings. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 478-486. Retrieved From: https://papers.nips.cc/paper/2016/hash/06138bc5af6023646ede0e1f7c1eac75-Abstract.html

Rumelhart, D., Hinton, G. & Williams, R. (1986) Learning representations by back-propagating errors. Nature, 323, 533-536. DOI: https://doi.org/10.1038/323533a0

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th Edition ed.). Pearson. ISBN: 0134610997

Suits, D. B. (1957). Use of Dummy Variables in Regression Equations. *Journal of the American Statistical Association, 52*(280), 548-551. DOI: https://doi.org/10.2307/2281705

Thomas, J., Coors, S., & Bischl, B. (2018). Automatic Gradient Boosting. *ArXiv. Retrieved From: https://arxiv.org/abs/1807.03873*

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855. DOI: https://doi.org/10.1145/2487575.2487629

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., & Attenberg, J. (2009). Feature hashing for large scale multitask learning. *Proceedings of the 26th Annual International Conference on Machine Learning*, 1113–1120. DOI: https://doi.org/10.1145/1553374.1553516

Xu, W., Evans, D., & Qi, Y. (2018). Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *Proceedings of 2018 Network and Distributed System Security Symposium.* Retrieved From: https://arxiv.org/abs/1704.01155

Yin, Z., & Shen, Y. (2018). On the dimensionality of word embedding. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 895-906. Retrieved From: https://arxiv.org/abs/1812.04224