Methods for numerical substitution of categorical data

Márk Dániel Szalai

Supervisor:
Dr. Gábor Horváth

Budapest University of Technology and Economics
Department of Networked Systems and Services
Budapest, Hungary
October 2021

Contents

1 Introduction

The handling, clustering and usage of categorical data in machine learning models is a surprisingly hard problem, for which many companies are looking for solutions. The traditional, well proven concepts and metrics are hard or outright impossible to use with them and most of the existing machine learning tools can not interpret them without a numerical substitution. The reason for that is that most of our methods are using some kind of distance metric, which is not necessarily interpretable between categorical variables. A common approach of the problem is to only use the numerical variables with the automated learning processes, while the categorical ones are processed separately with specialized tools or not processed at all. These methods can have huge information losses, which should be avoided, because these variables often hide interesting or critical information. There may also be situations where the valuable information is hidden behind a mixture of categorical and numerical variables. Another approach is to use numerical substitution. A common way to do that is one-hot encoding, which has the downside of significantly increasing the modell's size.

Our goal is to provide an overview and comparison of the existing methods, and develop a new, fast and efficient method, that enables the clustering of categorical variables and its joint handling with numerical data. Since Tensorflow is one of the most widespread tools in machine learning, we are also providing an implementation in Tensorflow, capable of utilizing the extra performance of the GPUs, resulting in an even faster produced and easier to use results. There are already many successful and well-proven algorithms for the supervised learning case, therefore our focus will be on the unsupervised learning applications, which have much less proposals in the literature.

1.1 About categorical data

Categorical data (see also [1]) can be interpreted as information aggregated into groups, rather then being in numerical form. For example let us consider the colors. When we refer to them as "blue", "red" etc. we use a categorical representation. We could also use their wavelength or RGB code representation, in which case we would use a numeric representation, which is more precise (there are dozens of "blues"), but also more difficult for humans to understand. Please note, that the color categories can not be ordered by themselves: we can not say, that "blue" is bigger than "red", despite that their corresponding wavelengths could easily be ordered.

There are two types of categorical data: ordinal and non-ordinal. The distinction between the two is that ordinal is a data type, that has natural, ordered categories, but the distances between the categories are not known. For example: "bad, average, good", it is clear, that the category "bad" is worse than the category "good", therefore we can order them.

1.2 The challenges of categorical data

Most of the problems arise from their above mentioned property: distances are basically non-measurable between the categories and therefore it is incredibly hard to define a good distance metric on them (especially when we have to deal with non-ordinal categorical data), mostly because we can not recover the underlying numerical information. Despite that, they can still hold crucial information, therefore we should take them into consideration when building models and analyzing data sets.

2 Numerical substitution of categorical data

As established, using categorical data in machine learning models without some kind of prepossessing is rather unfortunate, but with proper numerical substitution most of these problems can be eradicated. Now we would like to present some well-known and widely used methods for handling these variables.

2.1 Label encoding

Label encoding simply assigns integers to each category, for example take a look at Table 1 .

Categorical Feature	**Label**
Dog	1
Cat	2
Duck	3
Cow	4
Cat	2
Panda	5

Table 1: Label encoding example

At first glance it seems to be an easy and tempting way of encoding, however it has serious drawbacks. The main problem with this kind of numerical substitution is the unintentional introduction of relationships to the data. For example $1 + 2 = 3$ and $2 * 2 = 4$, but does this also mean, that a Dog and a Cat equals a Duck? Or Cat times Cat equals Cow? Of course not. This is especially an issue when it is used with algorithms, that uses/calculates some kind of distance metric (for example K-Means [2]).

Label encoding is quite popular among beginner data analysts, therefore we felt like it should be included here, but we obviously can not recommend it and this type of encoding should be avoided.

2.2 One-hot encoding

One-hot encoding [3] is the most common approach to handling the problem. It works by replacing each categorical feature with a binary vector, that has exactly one 1 in the position that corresponds to the replaced categorical feature. For example refer to Table 2 .

Categorical Feature	**One-hot encoding**				
-	Dog	Cat	Duck	Cow	Panda
Dog	1	0	0	0	0
Cat	0	1	0	0	0
Duck	0	0	1	0	0
Cow	0	0	0	1	0
Cat	0	1	0	0	0
Panda	0	0	0	0	1

Table 2: One-hot encoding example

One-hot encoding is generally considered a good approach, however it has its own drawbacks. It adds many binary variables to the system and considerably increases the dimensionality of our data set, amplifying the curse of dimensionality, therefore making distance calculation and clustering harder. In machine learning it is good practice to have at least a few samples for each feature combination. Increasing the number of variables can easily lead to situations where we don't have enough training samples. Another possible weakness of One-hot encoding is that it considers each category to be equally similar to each other, this may be an undesired behaviour. (For example consider alcohol consumption over the weak. The weekend and Friday are expected to be closer to each other, than the rest of the weekdays.) A huge positive for the encoding is that it can be used in unsupervised learning.

2.3 Target encoding

Target encoding (also referred as mean encoding) [4] is arguably one of the most effective supervised way of the numerical substitution of categorical variables. It substitutes each group in a categorical feature with the groups' average response in the target variable. For an example

Categorical Feature	**Target variable**	**Target encoding**
Dog	1	0.67
Cat	1	0.33
Duck	1	1
Dog	1	0.67
Dog	0	0.67
Cow	1	0.5
Cat	0	0.33
Cow	0	0.5
Cat	0	0.33
Panda	0	0

Table 3: Target encoding example

Target encoding solves the high dimensional data problem of One-hot encoding and is being applied successfully in many machine learning models.

Unfortunately the above described version of mean encoding is quite capable of overfitting (in case of a categorical value having a lack of samples the modell will find a direct link between the variable and target, see Table 3 Panda and Duck), therefore we need some kind of regularization. There are two popular and powerful way to do this (often used in Kaggle [5] competitions): additive smoothing [6] and K-Fold target encoding [7].

Additive smoothing is a very simple regularization technique, that takes into consideration the average value of the target variable. The intuition here is that in case we lack a certain category, we should rely on the global mean of the target variable. In case we have enough samples of that category, we can rely on the local mean (x). This method introduces a hyper-parameter, m, the weight we want to assign to the global mean ($\mathcal{X}$). If we denote n the number of occurrences of that category, then the modified encoding (μ) can be calculated as described by Equation 1,

$$\mu = \frac{n * x + m * \mathcal{X}}{n + m}. \tag{1}$$

An example of this type of encoding, with $m = 3$ and the same data set as before, can be seen on Table 4 . Notice that the outliers have disappeared, which is a desired behaviour, because thanks to the lack of samples we could not have been sure whether they assumed the extreme values by chance or because most of the corresponding target variables are actually extreme. This uncertainty applies to all training data sets as it is always just a subset of all the possible combinations of variables. Regularization with smoothing can ease concerns about the quality of our training data set, that are arising from the above described uncertainty.

Categorical Feature	**Target variable**	**Target encoding**	**Target encoding with smoothing**
Dog	1	0.67	0.583
Cat	1	0.33	0.417
Duck	1	1	0.625
Dog	1	0.67	0.583
Dog	0	0.67	0.583
Cow	1	0.5	0.5
Cat	0	0.33	0.417
Cow	0	0.5	0.5
Cat	0	0.33	0.417
Panda	0	0	0.375

Table 4: Target encoding example with smoothing, $m = 3$

K-Folding [7] is another powerful regularization technique and it is easy to un-

derstand. The idea is to divide our data into folds (groups) and for each group's variables we assign the average responses in the target variable according to the ***rest*** of the data. This means that the same categorical value can be encoded slightly differently in each fold. As a final step we take the average of the encoded values for each corresponding categorical value and calculate their mean. This mean will be the encoding of the variable. The number of folds, K, is a hyperparameter of the process and needs to be adjusted depending on our data set, typically $K \in [5; 10]$. In case a certain value of the categorical variable is only present in one fold, we can simply encode it with its global mean. An example with a different data set (better illustrates this encoding scheme) can be seen on Table 5. Notice that in extreme cases the outliers can remain outliers.

Categorical Feature	**Target variable**	**Target encoding**	**Encoded values on each fold**	**K-Fold encoding**
Dog	1	0.75	0.5	0.667
Cat	1	0.5	0.333	0.5
Dog	1	0.75	0.5	0.667
Dog	1	0.75	0.667	0.667
Cow	1	1	1	1
Cat	0	0.5	0.667	0.5
Cat	0	0.5	0.5	0.5
Cat	1	0.5	0.5	0.5
Dog	0	0.75	1	0.667

Table 5: Target encoding example with K-Folding, folds are color-coded

2.4 Bayesian target encoding

There is an enhanced version of target encoding, called Bayesian target encoding, published in [8] and [9]. It fits distributions to the target variable distinctly for each category and learns the distribution's parameters, which is used to represent the category. Bayesian target encoding uses prior (the expected behaviour of data before seeing any) and posterior distributions (the expected behaviour of data after inspecting all of them). It is often assumed, that the prior and posterior distributions are in the same probability distribution family, thus they can be treated as conjugate distributions (see also [10]). The beauty of them is that they have a rather simple update process, with which we can update the parameters of the prior distribution as we read the data and with sufficient training samples we can approximate the posterior distribution. The posterior distribution is then used to calculate the variance, the skewness and potentially even higher moments of the distribution. It is important that we choose an appropriate prior distribution:

- For binary-classification the conjugate prior is beta distribution
- For multi-class problems it is Dirichlet distribution

- For regression problems it is normal-inverse-gamma distribution.

This type of encoding does increase the number of features of the model, but it might be necessary in case of certain data sets, where the mean of the target variable does not produce enough information. The higher moments of the distribution allow us to add details to our model, about the intra-category distribution of the target variable. This property can be considered both a strength and a weakness: if the same distribution is true for the unseen data it can increase the performance of the model, otherwise it will most likely overfit on the higher moments, in which case Target encoding provides better results.

This method heavily relies on descriptive statistics, which have certain limitation. A great and famous example for numerical cases is Anscombe's quartet, see Figure 1. These are four significantly different data sets, designed to showcase the limitations of descriptive statistical analysis. The four data sets have the same mean, variance and correlation between X and Y, even the linear regression line is almost the same.

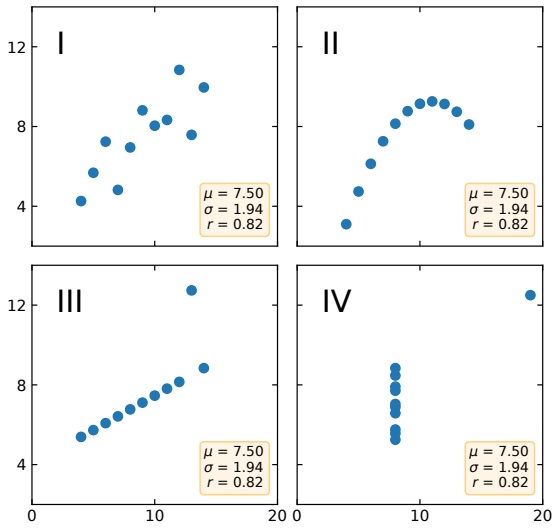


Figure 1: Anscombe's quartet

Regardless of the possible overfitting and the limitations of descriptive statistics Bayesian target encoding is getting more and more popular amongst the best data analysts and kagglers, but unfortunately it only works in the case of supervised learning.

2.5 Sampling Bayesian encoding

In a recent article [11] the author suggests an improvement of the Bayesian target encoding. The idea is the following: instead of using the moments of the posterior distribution as the new features, we could sample the distribution and use the sampled values for the numeric representation of categories. We can apply an arbitrary

number of different functions on the sampled values, including: identity, polynomials of the value and even more complex functions. The output of these functions forms a vector which will be the encoded, numerical representation of the categorical value. The idea relies on the machine learning model to figure out how to weight these function outputs during the training phase. Sampling Bayesian encoding could help avoid both target leakage and overfitting. Just like the regular Bayesian target encoding, it only works in the case of supervised learning.

2.6 Spectral encoding

Spectral encoding is focusing on providing a low dimensional, numerical representation of the categorical variables. A simple way of doing so is to first use One-hot encoding on our data and then use PCA (Principal Component Analysis [12], a well known data transformation technique, which is commonly used for dimension reduction purposes). If the number of dimensions reduced is the same as the number of dimension introduced by One-hot encoding, the result is a data-set with numeric values and with the same dimensionality as our original data.

A more sophisticated approach is set around the idea that if we can define a similarity/kernel function between the different values of our categorical features, we can use it with the methods of spectral analysis to find a low dimensional representation. This kernel function can be used to construct an adjacency matrix (symmetric) for the categories of a feature. (The adjacency matrix of data-points can be constructed from the individual kernel functions.) This matrix can be used as the basis of constructing other matrices (normalized Laplacian matrix), whose PCA-like dimension reduction yields our low dimensionality representation. An important question is how to find the kernel function? Luckily the algorithm can learn it several ways, one of which utilizes the Kullback-Leibler Divergence.

In [13] is a detailed overview of the spectral encoding methods and [14] is a great tutorial for a kernel based spectral encoding.

2.7 Entity embedding layers

Entity embedding layers originates from the field of Natural Language Processing. As it can be expected this method is designed for neural networks. The idea is to dedicate certain layers just to transform the input into a lower (than the number of categories, i.e. the number of dimensions resulting from one-hot encoding) dimensional space. The target dimension is a hyper-parameter that we must experiment with for each categorical feature, but as a rule of thumb we can use the square-root of the number of unique categories for each feature.

The premise of the method is that it will group the categories that have similar output. For example consider the prediction of the daily alcoholic beverage sales. We can expect that on Friday, Saturday and Sunday people consume more alcohol than on weekdays. The layer would group them accordingly, which means there is no need for feature engineering (i.e. manually force certain categories to be closer)

and it also results in a performance boost, because we are not (directly) teaching the neural network with the large dimensional one-hot encoded variables.

The main drawback of this method is that it requires the neural network to encode the variables, which in cases, where we do not want to feed our data to a neural network (for example we want to use a tree based learning algorithm), is undesirable. In [15] there is a short but instructive description of the method.

2.8 Usupervised Feature Transformation

Unsupervised Feature Transformation (UFT) introduced in [16] is an interesting way of encoding categorical data. The method tries to find a numeric substitution ($\tilde{X}$) for a categorical feature (X) such that:

$$I(X, \tilde{X}) = H(X), \tag{2}$$

where $I(A, B)$ is the mutual information and $H(A)$ is the entropy. In other words: we want to find a numeric substitution $\tilde{X}$ that has the same mutual information with X as X has with itself ($H(A) = I(A, A)$), which is an intuitive way to describe that we want the numeric substitution to contain the same information as the original data.

To achieve this and to narrow down the list of possible solutions to a single one UFT makes several assumptions:

- Feature independence
- Gaussian distribution of variables
- Gaussian distribution of the numeric substitution
- Equidistant expected values of these distributions
- Monotonocity of seemingly randomly chosen expected values

Despite these assumptions based on the test results in [16] UFT performed quite well on many data-sets. The performance of the method was measured by the average accuracy of several K-Means clusterings [2] on the transformed data.

2.9 Summary

So far we have examined several different encoding schemes, each of them can be incredibly powerful in the right hands, under the right circumstances. In supervised learning the best methods are arguably Target encoding and Bayesian target encoding. But what are we to do, when it comes to unsupervised learning? In this case we have no target variable, no expected output, only the data-set we need to extract information from. Well, the good news is, that some of the methods described so far can definitely work with these models, just as long as they can be performed on

the unlabeled data-set. Unfortunately these methods usually do not perform as well as the ones with target variables. So what else can be done?

One way is to use dimension reducing techniques: auto-encoders, principal component analysis (which is basically a base transformation, but it is used to drop unnecessary features) etc. These methods, although useful and they certainly have their place in the literature, all have the same side-effect: the relationships between the categories and categorical features disappear in the low dimensional space, which can be problematic.

Our idea is a new way to encode the variables, we assume that our variables correlate with each other (which is highly likely, as they all used to describe the same system) and do a numeric substitution such that it maximises the correlation between the variables. In the next section we would like to present our method.

3 CORRANS-CAT

CORRelation As Numeric Substitution for CATegorical variables is a new way to encode categorical variables which heavily relies on the correlation of categorical features in the data-set. It is intended to be a preprocessing step: take the categorical features of the data-set, encode them re-add the numeric features and continue processing the data with some kind of machine learning process.

3.1 The idea

As we have mentioned earlier the idea is that we assume that our variables correlate with each other (which is a reasonable assumption, considering they describe the same system) and we find the numerical substitution, that maximises the sum of the pairwise correlations between the variables. Let us denote our data matrix, where each row contains a sample and the columns are the (strictly) categorical features, with M. If n is the number of variables (i.e. the number of columns in M) and C is the correlation matrix, then we would minimise:

$$-\sum_{i=1}^{n}\sum_{j=1}^{n} C(i,j). \tag{3}$$

More precisely, we do not want negative correlations to act as punishments, thus we take the squares of the correlations:

$$-\sum_{i=1}^{n}\sum_{j=1}^{n} C(i,j)^2. \tag{4}$$

Now, certain constraints must be applied, we can not just change the values in M as we please, because we must ensure that the same categorical value encodes to the same numerical value. We need to make our optimizer take this into consideration, otherwise we would loose the relationships between the categorical features, it would be possible to substitute the same value to every element in M, which would maximize the correlation, but would not be very useful.

The solution is to find the unique values in M and change them all to the same (but preferably still unique) value. We have paid special attention to make the unique values appear only in one column. This is an important pre-processing step, because the same categorical value might appear in multiple features, meaning it has different context. Take a look at Table 6 for an example.

EyeColor	CarColor
blue	brown
brown	black
green	red
brow	brown

(a) Original data

EyeColor	CarColor
EyeColor.blue	CarColor.brown
EyeColor.brown	CarColor.black
EyeColor.green	CarColor.red
EyeColor.brow	CarColor.brown

(b) Pre-processed data

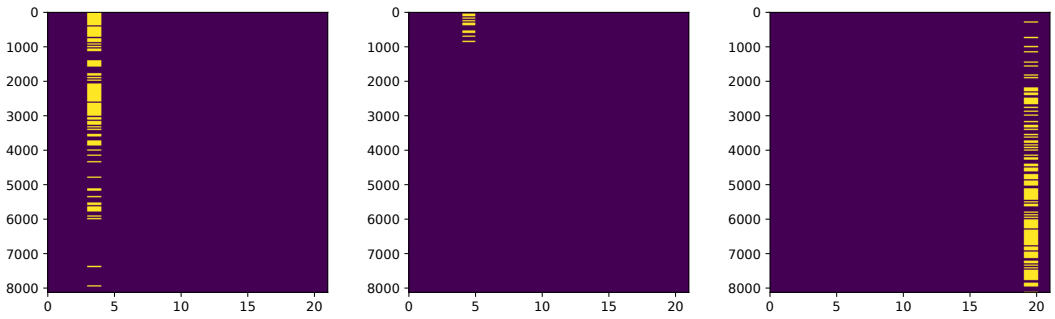
Table 6: An example of the pre-processing step

This way we do not force the values of seemingly unrelated features to assume the same encoding. It is a degree of freedom: the optimizer may "choose" different numeric substitutions for these values to achieve higher overall correlation.

As a prototype we used various optimizers and tested our method by clustering on our modified data and inspected the model's accuracy. We have achieved remarkable results on various data-sets, with accuracies well over 90%. Satisfied with the prototype's results, we decided to reproduce our method in TensorFlow [17] (with GPU support), because it is capable of optimizing orders of magnitudes faster than regular optimizers.

3.2 TensorFlow

Most of TensorFlow's data types are immutable, so building an appropriate model is by no means an obvious task. Our solution utilizes a set of constant tensor masks, for each unique value. The masks has the same size as M and they contain ones exactly where their unique value is present in M and they contain zeros everywhere else, see Figure 2 for an example.



(a) (b) (c)

Figure 2: Examples of masks (X axis is stretched for better visibility)

The masks are multiplied by their corresponding unique value and the sum of the resulting tensors yields the matrix for which we want to maximise the correlation. The variables that TensorFlow can change are the unique values. Since these values

influence the matrix, they also influence the correlation and TensorFlow will change them accordingly.

All of this can be written in just a few lines of code:

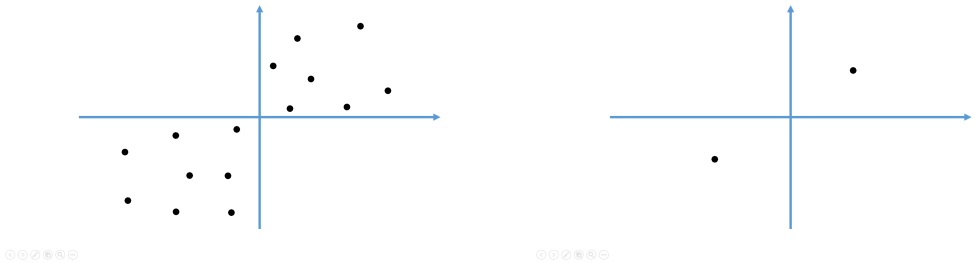
```
import tensorflow as tf
import tensorflow_probability as tfp
import numpy as np

config = tf.ConfigProto()
config.gpu_options.allow_growth = True

M_unique = np.unique(M)
masks = []
tfvars = []
for i in range(M_unique.shape[0]):
    masks.append(np.ma.filled(np.ma.masked_equal(np.ma.filled(
        np.ma.masked_not_equal(M, M_unique[i]), 0), M_unique[i]), 1)
    )

    tfvars.append(tf.Variable(initial_value = M_unique[i], trainable=True, ))

tensormaskswithvalues = []
for i in range(len(tensormasks)):
    tensormaskswithvalues.append(tf.math.multiply(tensormasks[i], tfvars[i]))

M_tensor = tf.math.add_n(tensormaskswithvalues, name="results_M_tensor")

corr = tfp.stats.correlation(M_tensor, sample_axis=0,
                             event_axis=-1, keepdims=False)

loss = -tf.math.reduce_sum(tf.math.multiply(corr2,corr2))
train =  tf.train.GradientDescentOptimizer(0.1).minimize(loss)
init = tf.global_variables_initializer()
sess = tf.Session(config=config)
sess.run(init)
for i in range(1000):   #arbitrary number of iterations
    sess.run(train)
sess.close()
```

3.3 The problem

TensorFlow (with GPU support) optimised faster so we let it take more steps. We observed that after the initial substantial increases in accuracy, it started to diverge: the correlation between our loss function and the accuracy of the clustering

diminished. We also noticed that the optimizers tended to generate a deformed solution, where one category assumed significantly higher numerical values than the others, which remained quite close to each other. With these findings it has become apparent that we either need some kind of regularization member in our objective function or we need early stopping.

As an illustration consider the following example. On Figure 3 we can see a scenario, where the optimizer correctly maximised the correlation by condensing the data points in each quarter into 1-1 point. This means that substitution only produces two unique values for our input which results in a maximal correlation: 1. Unfortunately this is a real possibility, as the optimizer can find this deformed solution thus we need to prepare the algorithm to avoid such solutions.



(a) The categorical data points (b) A possible numerical substitution

Figure 3: Example of a not desired substitution

3.3.1 Early stopping

Here we do not mean early stopping in the deeplearning context, we mean literally stopping early, after just a few iterations. We have seen that the accuracy of the clustering increases in the first few iteration, thus it is reasonable to only take a few iterations. We decided to follow the adding a regularization member to the loss function path.

3.3.2 Regularization

We have tried adding several different regularization terms to our loss function, including variance, forcing a few of the eigenvalues of the correlation matrix to be bigger and even some non-derivable as well (just out of curiosity, it is usually not a good idea to make a gradient based optimizer optimize a non-derivable function).

We ended up with the following goals in mind:

- Punish the extreme values in each feature
- Do so with a derivable function

We are looking for an objective function that fits the following pattern:

$$-\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}C(i,j)^2+\frac{1}{n}\sum_{i=1}^{n}r_i, \tag{5}$$

where the role of r_i is to punish the extreme values of the ith feature. (The divisions by n is to ensure that the correlation does not dominate the objective function is high dimensions.) To achieve this we first need to find the most extreme values. Unfortunately a simple max function is not derivable, thus we decided to use softmax. Suppose the ith feature assumes the following unique values after their normalization (normalization is done on the unique set, please note that correlation is scale and shift invariant, meaning it is unaffected by normalization): $[v_1, v_2, ..., v_D]$. Define m_i as:

$$m_i=\sum_{k=1}^{D}|v_k|\frac{e^{\alpha|v_k|}}{\sum_{j=1}^{D}e^{\alpha|v_j|}} \tag{6}$$

Now what does m_i exactly mean? The right side of the product (the fraction) is the softmax function. The fraction is very close to 1 when v_k assumes extreme values and close to 0 otherwise, meaning that if we multiply it with v_k ($|v_k|$, because an extreme value could also be negative) and take the sum of them we get m_i which is really close to just be the result of max. The parameter α is a hyper-parameter it can be adjusted arbitrarily.

We are almost done, but we do not want to punish the ideal (ideal: does not have extreme values, the regularization might work against the correlation) substitution. It is easy to see that an equidistant substitution would be the most ideal, where $-v_1 = v_D, -v_2 = v_{D-1}$ etc. This means that the values of an ideal substitution in the function of their index forms a line, see Figure 4.

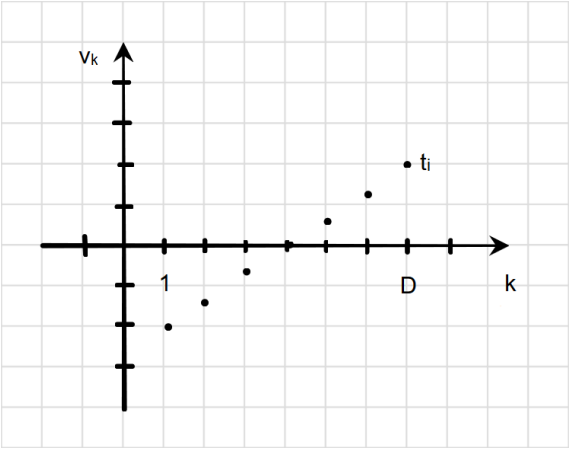


Figure 4: The values forming the line

It would not be fair to punish the extreme value (t_i) of the ideal substitution, thus we need to figure out this extreme value and subtract it from m_i, so that it

would not contribute to the regularization. The equation of the line is:

$$f(k) = a(k - b). \tag{7}$$

We need to find a and b. Luckily we have 2 equation to solve it, because from the nature of the ideal substitution we can see that the mean of the values must be 0 (Equation 8) and the standard deviation must be 1 (Equation: 9):

$$\frac{1}{D}\sum_{i=1}^{D} f(k) = 0, \tag{8}$$

$$\frac{1}{D}\sum_{i=1}^{D} f(k)^2 = 1. \tag{9}$$

The solution for this system is: $a = \frac{2\sqrt{3}}{\sqrt{(D-1)(D+1)}}$ and $b = \frac{D+1}{2}$, so $f(k)$ is:

$$f(k) = \frac{2\sqrt{3}}{\sqrt{D^2-1}}(k - \frac{D+1}{2}) \tag{10}$$

Now to get the value of t_i we need to calculate either $f(1)$ or $f(D)$ (notice that $-f(1) = f(D)$). From (10) we can get t_i:

$$t_i = f(D) = \frac{\sqrt{3}}{\sqrt{D^2-1}}(D-1). \tag{11}$$

Finally we can define $r_i = (m_i - t_i)^2$. The complete objective function can be seen on Equation 12:

$$-\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} C(i,j)^2 + \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k=1}^{D}|v_k|\frac{e^{\alpha|v_k|}}{\sum_{j=1}^{D}e^{\alpha|v_j|}} - \frac{\sqrt{3}}{\sqrt{D^2-1}}(D-1)\right)^2. \tag{12}$$

We also implemented this type of regularization in TensorFlow:

```
colvars = []
for i in range(masks[0].shape[1]):
    colvars.append([])
for i in range(x_unique.shape[0]):
    colvars[np.where(M==M_unique[i])[1][0]].append(tfvars[i])

a=5
colvars_normalized = []
for i in range(len(colvars)):
    colvars_normalized.append([])
for i in range(len(colvars_normalized)):
```

```
    meanscol = tf.math.reduce_mean(colvars[i])
    stdcol = tf.math.reduce_std(colvars[i])
    for k in range(len(colvars[i])):
        colvars_normalized[i].append(
            tf.math.divide(
                tf.subtract(colvars[i][k],meanscol), stdcol
            )
        )

reg = tf.math.reduce_sum([tf.math.divide(
    tf.tensordot(tf.math.abs(i), tf.math.exp(a*tf.math.abs(i)), 1),
    tf.math.reduce_sum(tf.math.exp(a*tf.math.abs(i))))
    - (tf.size(i, out_type=tf.float32)-1.0)*tf.math.sqrt(
        tf.math.divide(tf.identity(tf.constant(3, dtype=tf.float32)),
        tf.size(i, out_type=tf.float32)*
            tf.size(i, out_type=tf.float32)-1.0))
        for i in colvars_normalized])
N = M.shape[1]
loss = -(1/(N*N))*lossCorrOnNormed+(1/N)*reg
```

3.4 Evaluation

We took 1000 optimizing steps (after that the changes in loss were negligible), used a regular gradient descent optimizer and with a learning rate of 0.1 for each data-set and performed a K-Means clustering [2] at the end of optimization. Since K-Means initialization is random, we ran K-Means 100 times and took the average accuracy of the clustering.

3.4.1 Used data-sets

The data-sets are from the UCI Machine Learning Repository [18], all of them contain several categorical features. The descriptors of these data-sets can be seen on Table 7. Soybean contains the label-encoded characteristics of soybean plants that are suffering from some sickness. German and Australian contain credit approval data from Germany and Australia respectively. Zoo contains mostly boolean data regarding the characteristics of animals, the target variable is the taxonomic classification of the animals (e.g.: mammal, fish etc.). Dermatology contains dental records, most features are characteristics of deformities and their value (0-3) represent the severity of the deformity. Lastly, the Mushroom data-set contains characteristics of mushrooms (e.g.: shape/surface of cap) and the target variable is whether or not the mushroom is edible. (We would like to point out that originally there were 3 target categories: edible, poisonous and unknown. Unfortunately this last category was also labeled as poisonous, which may distort our results.)

Data-set	**Samples**	**Classes**	**Numerical features**	**Categorical features**	**Total features**
Soybean	47	4	0	35	35
German	1000	2	3	17	20
Australia	690	2	6	8	14
Zoo	101	7	1	15	16
Dermatology	366	6	1	33	34
Mushroom	8124	2	0	22	22

Table 7: Data-set descriptors

3.4.2 Comparison and results

We compare our algorithm with same state of the art encoding schemes as UFT (which has been discussed in detail in Section 2.8), namely: K-Prototypes [19], Improved K-Prototypes [20] and KL-FCM-GM (Kullback–Leibler information fuzzy c-means combined with Gauss-multinomial distribution) [21]. K-Prototypes uses both Hamming- and Euclidean distances for categorical and numerical data respectively, which has two major drawbacks: the weight of Hamming distances must be set and modified manually; and the combination of these distances is linearly problematic, since they have different physical meanings. Improved K-Prototypes aims to solve the former, while KL-FCM-GM the latter problem. UFT was chosen because our algorithm also works by separating the categorical and numerical features, encoding the categorical and then recombine with the numerical features before proceeding with the clustering. The results can be seen on Table 8 which is almost the same table as in [16] but with our measurements added.

Clustering algorithms	**Accuracy (mean of 100) %**				
-	*Soybean*	*German*	*Australian*	*Zoo*	*Dermatology*
CORRANS-CAT	97.87	70.00	56.21	41.83	34.91
UFT-k-means	96.17	75.14	91.19	84.85	63.02
UFT-GMM	78.51	70.07	72.04	74.80	52.69
k-prototypes	87.45	70.00	74.67	77.95	52.47
improved k-prototypes	90.85	70.00	78.72	81.98	52.40
KL-FCM-GM	57.45	70.00	68.71	40.59	33.99

Table 8: The performance of CORRANS-CAT and other clustering processes

We can see that our algorithm with the previously described regularization member shines and outperforms others (including UFT) on highly correlated data-sets,

such as Soybean, but performs rather poorly on data-sets with many non-correlated features.

3.4.3 Comparison of different regularisation techniques

In the following tables (Table 9 shows the clustering accuracy only on the categorical data transformed by CORRANS-CAT. Table 10 shows the clustering accuracy with the numerical features reintroduced for the clustering.) we compare different kinds of regulation techniques and their combinations. We also experiment a little with their weight, which can be seen directly after the name of the technique. We can choose the best individual regularisations from Table 9 and 10 based on majority voting (how many data-sets "prefers" a certain regularization). these are the following: Our-0.1, L1-0.01, L2-0.01 and Our-0.03, L1-0.1, L2-0.03. The combined entries show the performance of CORRANS-CAT with the combination of these regularisation techniques.

Reg.	**Accuracy (mean of 100) %**					
-	*Soybean*	*Mushroom*	*German*	*Australian*	*Zoo*	*Dermatology*
Our-0.01	97.28	70.95	70.00	67.83	41.52	31.47
Our-0.03	97.53	70.95	70.00	67.83	41.43	31.40
Our-0.1	97.72	70.95	70.00	67.83	41.66	31.39
L1-0.01	78.68	89.63	70.00	73.33	41.84	33.71
L1-0.03	76.77	88.82	70.00	73.33	41.75	33.60
L1-0.1	77.36	73.07	70.00	73.33	41.92	33.67
L2-0.01	78.87	89.62	70.00	73.33	41.88	34.31
L2-0.03	78.64	89.62	70.00	73.33	41.75	33.40
L2-0.1	78.63	89.62	70.00	73.33	41.84	33.68
Combined	97.36	70.95	70.00	67.83	41.55	31.49

Table 9: The performance of CORRANS-CAT and using different regularisation techniques and weights

Reg.	**Accuracy (mean of 100) %**					
-	*Soybean*	*Mushroom*	*German*	*Australian*	*Zoo*	*Dermatology*
Our-0.01	97.74	70.95	70.00	56.23	41.51	31.44
Our-0.03	97.43	70.95	70.00	56.23	41.60	31.51
Our-0.1	97.70	70.95	70.00	56.23	41.52	31.44
L1-0.01	97.34	70.95	70.00	56.23	41.50	31.54
L1-0.03	97.53	70.95	70.00	56.22	41.54	31.38
L1-0.1	97.87	70.95	70.00	56.22	41.67	31.53
L2-0.01	97.43	70.95	70.00	56.22	41.61	31.49
L2-0.03	97.79	70.95	70.00	56.23	41.49	31.51
L2-0.1	97.17	70.95	70.00	56.22	41.50	31.45
Combined	97.61	70.95	70.00	56.22	41.54	31.39

Table 10: The performance of CORRANS-CAT and using different regularisation techniques and weights with numerical features reintroduced for K-Means clustering

3.5 Future work and conclusion

Our early testings did show that CORRANS-CAT combined with K-Means could be capable of much more precise clustering than what we have seen on Table 8, therefore we believe that CORRANS-CAT has the potential to become an incredibly precise way of encoding categorical variables. We are planning to revisit the regularization term and trying out different kinds of regularization techniques.

It has become apparent that CORRANS-CAT is capable of outperforming other numerical substitution methods on data-sets that have many correlating features.

4 Closing remarks

Machine learning problems in general and especially unsupervised learning problems are incredibly difficult to solve. The right process to encode categorical variables must be chosen carefully for all data-sets, there is no one solution that works properly in every imaginable case.

When it comes to clustering algorithms there seems to be a lack of algorithms with small number of hyper-parameters that are capable of real-time adaptation and following data trends, for example growing additional clusters. It is an interesting and hard problem which we are yet to solve.

References

[1] Aurelien Geron. *Hands-on machine learning with scikit-learn, keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2 edition, 2019.

[2] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[3] What is one hot encoding and how to do it. https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179. Accessed: 2021-10-20.

[4] Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, 3(1):27–32, July 2001.

[5] Kaggle. https://www.kaggle.com. Accessed: 2021-10-20.

[6] Target encoding done the right way. https://maxhalford.github.io/blog/target-encoding/. Accessed: 2021-10-20.

[7] K-fold target encoding. https://medium.com/@pouryaayria/k-fold-target-encoding-dfe9a594874b. Accessed: 2021-10-20.

[8] Beta target encoding. https://mattmotoki.github.io/beta-target-encoding.html. Accessed: 2020-11-15.

[9] Austin Slakey, Daniel Salas, and Yoni Schamroth. Encoding categorical variables with conjugate bayesian models for wework lead scoring engine, 2019.

[10] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004.

[11] Michael Larionov. Sampling techniques in bayesian target encoding, 2020.

[12] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[13] Lawrence Saul, Kilian Weinberger, Fei Sha, Ji-Hun Ham, and Daniel Lee. Spectral methods for dimensionality reduction. *In B. Schoelkopf, O. C., and Zien, A., eds., Semisupervised Learning. MIT Press.*, 09 2006.

[14] Beta target encoding. https://towardsdatascience.com/spectral-encoding-of-categorical-features-b4faebdf4a. Accessed: 2020-11-15.

[15] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables, 2016.

[16] Min Wei, Tommy Chow, and Rosa Chan. Clustering heterogeneous data with k-means by mutual information-based unsupervised feature transformation. *Entropy*, 17:1535–1548, 03 2015.

[17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[18] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[19] Zhexue Huang. Clustering large data sets with mixed numeric and categorical values. In *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 21–34, 1997.

[20] Jinchao Ji, Tian Bai, Chunguang Zhou, Chao Ma, and Zhe Wang. An improved k-prototypes clustering algorithm for mixed numeric and categorical data. *Neurocomputing*, 120:590–596, 2013.

[21] Sotirios P. Chatzis. A fuzzy c-means-type algorithm for clustering of data with mixed numeric and categorical attributes employing a probabilistic dissimilarity functional. *Expert systems with applications*, 38, 2011.