UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS
INTELIGENCIA ARTIFICIAL

On the encoding of categorical variables for machine learning applications

TESIS

QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
ERICK GARCÍA RAMÍREZ

Director de tesis:
Dr. Ángel Fernando Kuri Morales
ITAM

Ciudad Universitaria, CD. MX. Enero 2021

Universidad Nacional
Autónoma de México

Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

El candidato confirma que el presente trabajo es de su autoría y que el crédito debido se ha extendido en cada referencia al trabajo de otros. La presente copia se ofrece bajo el entendimiento de que está protegida por derechos de autor y que citas a su contenido no pueden ser publicadas sin el reconocimiento apropiado.

©2021 Universidad Nacional Autónoma de México & Erick García Ramírez

Abstract

The present work provides a survey and a comparison study on several methods to numerically encode categorical data. We consider simple and well-known methods, such as One-Hot-Encoding, but also novel methods based on techniques from Natural Language Processing and on investigations about preservation of patterns in data.

Our main experiment compares thirteen methods to encode categorical data on ubiquitous machine learning tasks and real-life datasets. We offer insight into the conditions—dataset size, number and properties of categorical values, presence of a dependent variable, etc—under which methods allow for high performance of machine learning models. Also, we show that the combination of at least a few of these methods and ordinary machine learning models outperforms models designed to deal with categorical data directly.

Keywords: Categorical data, numerical encoding, categorical encoders, machine learning

Resumen

El presente trabajo ofrece una descripción conceptual y un estudio comparativo de diversos métodos para codificar numéricamente datos categóricos. Consideramos métodos simples y comunes, por ejemplo *One-Hot Encoding*, pero también incluimos métodos nuevos basados en técnicas de Procesamiento de Lenguaje Natural y en investigaciones sobre la preservación de patrones en datos.

Nuestro experimento principal compara trece métodos para codificar datos categóricos en tareas de aprendizaje automatizado de uso extendido y datos de la vida real. Ofrecemos un análisis de las condiciones—cantidad de datos, número y propiedades de los valores categóricos, presencia de una variable dependiente, etc.—bajo las cuales los métodos contribuyen al desempeño alto de modelos de aprendizaje automatizado. Además, mostramos que la combinación de al menos algunos de estos métodos y modelos de aprendizaje automatizado comunes superan el desempeño de modelos populares diseñados para procesar datos categóricos directamente.

Palabras clave: Datos categóricos, codificación numérica, codificación categórica, aprendizaje automatizado

Acknowledgements

I wish to thank the community of the *Posgrado en Ciencia e Ingeniería de la Computación* at IIMAS-UNAM for providing an excellent enviroment for the development of my studies. Thanks to the examiners of this work, for their time and suggestions, and to Dr Ángel Fernando Kuri Morales for his support during the development of this work.

Contents

1 Introduction **1**

1.1 Numerical and categorical variables . . . 1

1.2 The problem with categorical variables . . . 2

1.3 Encoding of categorical variables . . . 3

1.4 Alternatives to encoding . . . 5

1.5 The present work . . . 6

1.5.1 Objectives . . . 6

1.5.2 Organisation . . . 7

2 A review of known categorical encoders **9**

2.1 Types of categorical encoders . . . 10

2.2 Example dataset . . . 12

2.3 Encoders . . . 13

2.3.1 Ordinal Encoder . . . 13

2.3.2 Polynomial Encoder . . . 14

2.3.3 One Hot Encoder . . . 14

2.3.4 Backward Difference Encoder 15
2.3.5 Helmert Encoder 17
2.3.6 Entity Embedding Encoder 18
2.3.7 Target Encoder 21
2.3.8 Weight of Evidence Encoder 22
2.4 Target leakage 23

3 Pattern-Preserving Encoders 25

3.1 Preserving patterns 25
3.1.1 The Preservation of Patterns Principle 26
3.2 Pattern-preserving encoders 27
3.2.1 CENG Encoder 28
3.2.2 Genetic Pattern-Preserving Encoder 30
3.2.3 Aging Pattern-Preserving Encoder 31
3.2.4 Simple Pattern-Preserving Encoder 32
3.2.5 CESAMO Encoder 33

4 Experimental comparison of categorical encoders 35

4.1 On the evaluation and comparison of encoders 35
4.1.1 Definition of evaluation of encoders 36
4.1.2 The choice of K 37
4.1.3 Models 37
4.2 Breast Cancer Dataset 41
4.3 Automobile MPG Dataset 44

4.4 Credit Card Dataset 47

4.5 German Credit Dataset 50

4.6 Congress Votes Dataset 53

4.7 Sales Dataset 56

4.8 Mushrooms Dataset 59

5 The power of categorical encoding 63

5.1 On a previous experiment on entity embeddings 63

5.2 Categorical Feature Encoding Challenge 68

6 Conclusions 73

6.1 Guide to the usage of encoders 75

Appendix A: On implementations 77

References 83

Index 89

Chapter 1

Introduction

In this chapter we discuss the nature of variables in structured data, dividing them into *categorical* and *numerical* variables. We bring forward the difficulties that categorical variables pose in machine learning tasks, and proceed to explain ways to alleviate those difficulties.

1.1 Numerical and categorical variables

We consider two sorts of variables in structured data: numerical and categorical.

A *numerical variable* is one that takes real numbers as values. Examples are the temperature at a certain location or a person's age. If a numerical variable takes values within a continuous range we say that it is a *continuous numerical variable* and if it takes integer numbers as values we say that it is a *discrete numerical variable*. For instance, temperature is a continuous numerical variable and a person's age is a discrete numerical variable.

A *categorical variable* is one that can only take values from a finite set (of numbers or of more general names). Examples are employees' gender and genres of music. A further distinction among categorical variables is frequently made based on whether their values are/can be ordered. For example, the values of variables such as a person's level of education (Primary, Secondary, High-School,

Bachelors, Postgraduate) or the degree of injury after an accident (None, Moderate, Severe, Fatality) have a natural ordering. In contrast, categorical variables like a person's race or religion have no reasonable ordering between their values. Categorical variables whose values can, in principle, be ordered are called *ordinal variables*; any other categorical variables are called *nominal variables*.

Since the values of an ordinal variable have an ordering, these sort of variables are likely to be represented directly in a numerical scale. For instance, the degree of injury after an accident with values None, Moderate, Severe and Fatality could instead be represented by the integers 0, 1, 2 and 3, respectively. For reasons that will be made clear in Section 1.3, our position is to avoid this automatic treatment, holding to the convention that, without proper preprocessing, an ordinal variable is purely categorical (nominal).

Structured data and datasets containing only numerical variables are called *(purely) numerical*, while those that contain both numerical and categorical variables are called *mixed*. For our purposes, data and datasets with only categorical variables will be considered as mixed.

1.2 The problem with categorical variables

In machine learning tasks, where many of the methods are based on mathematical models that rely on numerical calculations, mixed datasets pose a fundamental difficulty. It is obvious that, with no preprocessing, a mixed dataset cannot be used to train a model that requires all data to be numerical. This applies to popular machine learning models such as linear and logistic regression, support vector machines and neural networks.

It goes without saying that with the prevalence of categorical data in all areas where machine learning is applied we cannot afford to ignore this sort of data. There are two ways to enable the use of categorical data in machine learning projects.

One way is to resort to the numerical encoding of categorical variables, which is the main topic of this work and we introduce properly in the next section.

The other way is to only employ machine learning models that can deal with categorical variables directly, e.g. decision trees and the Naïve Bayes classifier. We explain more on this in Section 1.4.

1.3 Encoding of categorical variables

The *numerical encoding of a categorical variable* consists of replacing each value of the variable by a number or a tuple of numbers. We of course aim to do this in a meaningful manner, so that the relevant features of the variable are preserved.

We also speak of the *numerical encoding of a mixed dataset* to mean that we replace by numbers or tuples of numbers all the values of all of the categorical variables in the dataset. The goal is to do this in a way that preserves as best as possible the features of the dataset. We often call the process of numerical encoding of categorical variables/mixed datasets *categorical encoding*, and even only *encoding*.

In the rest of this subsection we discuss some considerations regarding categorical encoding.

We wish to draw attention to the widespread but troublesome practice of careless and ill-founded categorical encoding. Take for example a case study where a dataset with medical patients records contains a categorical variable for the county of residence. Suppose that we are asked to develop a classification model to determine whether patients will develop a certain disease. To this effect, we need to have a numerical dataset. In presence of K different possible counties we, without thinking twice about it, may proceed to do one of the following:

(a) Order in some way the K counties (for example, alphabetically or by order of appearance) and then encode each county with an integer from 0 to $K-1$ corresponding to its place in the ordering.

(b) Encode each county with a number from 0 to $K-1$ in a completely random way.

Doing (a) or (b) manages to turn a categorical variable into a numerical one at the cost of probably inducing patterns not originally in the data and/or eliminating patterns that were in the data. For

instance, that two specific counties get encoded with consecutive numbers would suggest that those counties are somehow more similar to each other than to other counties, and a model trained with these data will likely reflect that relationship. In general, the encoding methods suggested in (a) and (b) create hierarchies on the values of categorical variables that the original dataset may not really feature.

Even when dealing with ordinal variables a “reasonable” approach to encoding based on the ordering might be problematic. Encoding, for example, the education levels Primary, Secondary, High-School, Bachelors and Postgraduate with the numbers 0, 1, 2, 3 and 4 has no better justification than encoding the levels with, say, the numbers -3, 0 , 1, 10 and 100. Encoding with 0 through 4 suggests a regular increase between levels of education whilst encoding with -3, 0, 1, 10 and 100 suggests that Secondary and High-School levels are much more related than Bachelors and Postgraduate. The data might not support such premise. This is our main objection to the automatic association of numbers to the values of ordinal variables.

Since careless encoding could have taken place right from the data collection step, we reccommend a vigilant lookout when exploring a dataset, so that careless encoding (including the automatic numeration of values of ordinal variables) is spotted and corrected.

Leaving behind the naïve approaches, we find several methods for encoding that are far better founded. From now on we will shift our attention from methods of encoding to their realizations as *encoders*. An algorithm/method to perform categorical encoding will be called a *categorical encoder*, or simply an *encoder*. The encoders considered in this work are the following.

(1) Ordinal Encoder

(2) Polynomial Encoder

(3) One Hot Encoder

(4) Backward Difference Encoder

(5) Helmert Encoder

(6) Entity Embedding Encoder

(7) Target Encoder

(8) Weight of Evidence Encoder

(9) Categorical Encoding via Neural Networks and Genetic Algorithms(CENG)

(10) Genetic Pattern-Preserving Encoder

(11) Aging Pattern-Preserving Encoder

(12) Simple Pattern-Preserving Encoder

(13) Categorical Encoding by Statistical Applied Modelling (CESAMO)

Chapters 2 and 3 contain reviews of the above categorical encoders whilst Chapter 4 consists of a comparison study.

1.4 Alternatives to encoding

Each of the better-founded categorical encoders mentioned above have nevertheless some drawbacks. Details on these drawbacks will be exposed in Chapters 2 and 3, but for now we can mention that some encoders modify the representation of data so drastically that conceptual questions about the data turn more difficult to answer and other encoders introduce probabilistic uncertainty about the goodness of the encoding. It is therefore valuable to consider alternatives to encoding.

As we mentioned earlier, another way to enable the use of categorical data is to only employ machine learning models whose training allows the direct use of mixed datasets. For the sake of fair and wide exploration, we will include some of these models in our comparison study in Chapter 4. The models that we will include are the following.

1. Naïve Bayes Classifier. Some implementations of this algorithm allow the presence of categorical variables. The typical performance of this classifier is superseded by that of other numerically-based ones, but the Naïve Bayes Classifier has some valuable explicative features.

2. Decision Trees, and by extension, Random (Decision) Forests. Popular implementations of these algorithms, like CART and C5.0, explicitly aim at allowing the presence of categorical variables.

3. CatBoost. This algorithm allows training with mixed datasets and has a version for both supervised and unsupervised learning. CatBoost is based on boosting of tree models, see [20].

The specific implementations of these algorithms used for our comparison are reported in Appendix A.

1.5 The present work

In this brief section we explain the ideas, objectives and organisation of this work.

1.5.1 Objectives

The motivation for this work came from two sources: first, exploring the numerous methods for categorical encoding and their diverse nature; second, compare the impact of using different encoding methods on machine learning applications. In consequence, our objectives for this work are the following.

1. Show that categorical encoding has a strong positive impact on the performance of machine learning models.

2. Show that common encoding methods, particularly One-Hot encoding, are readily overcome by other methods that are not necesarily more complex.

3. Identify the encoders that perform better under specific conditions (type of learning, data size, etc), providing a practical guide to their usage.

4. Show that the newer Entity-Embedding and the Pattern-Preserving encoders are sound and competitive categorical encoders.

Since some of the encoding methods we consider are rather new, general implementations do not exists for them, so a side objective is to develop general implementations for those methods.

1.5.2 Organisation

In Chapter 2 we offer a review of several encoding methods that we consider classical/well established and a newer method based on neural networks. Chapter 3 starts by explaining a newer idea behind how to approach encoding and proceeds to give details on five encoders based on that idea.

The first part of Chapter 4 sets the experimental framework of our main comparison study. The second part of Chapter 4 contains the results of our study in full detail. Chapter 5 consists of two further comparison studies which do not fit the framework and purpose in Chapter 4 because in Chapter 5 we wish to test encoders under more challenging conditions.

Towards the end of this work we offer our conclusions on encoding and on our comparison studies. We also include a brief practical guide to the selection of encoders. Appendix A collects specifics on the implementations of the encoders and other algorithms used in this work.

Chapter 2

A review of known categorical encoders

This chapter contains explanations and references for several categorical encoders that are popular in machine learning. We first establish some concepts and notational conventions used later.

Let D be a mixed dataset. If C is a categorical variable in D, any of the distinct values that C takes will be called a *categorical instance* of/in C. By extension, any of the distinct values of a categorical variable in D will be called a categorical instance of/in D.

A *(numerical) code* for a given categorical instance is a number or a tuple of numbers. Hence, encoding D consists of finding codes for each categorical instance in D, and subsequently replacing all the categorical values in D by those codes consistently.

A collection of codes that contains a code for each categorical instance in D will be called a *(complete) set of codes*[1]. Thus, the cardinality of a set of codes is the total number of categorical instances in D. Finally, if S is a set of codes, $D(S)$ will denote the numerical dataset that results from consistently replacing each categorical value in D by the corresponding code in S. We say that $D(S)$ is the result of encoding D by/using S.

[1] It is useful to think of a set of codes as a dictionary with elements of the form “categorical instance : code”.

2.1 Types of categorical encoders

Depending on the method they rely on to perform the encoding, most categorical encoders can be classified into the following types.

- **Ordinal encoders**. These assume that the values of a categorical variable have an inherent ordering. For a categorical variable with k values an ordinal encoder uses k consecutive integers as codes. Implementations usually expect as argument the ordering of the values (in the form of a dictionary, for example); when this argument is not passed the encoding consists of random integers assigned to the categories.

  An obvious drawback of this type of encoders is that they require us to have solid knowledge about the ordering of the categories. Another drawback is that ordinal encoders use equally spaced numbers as codes, something that we wish to avoid for the reason exposed in page 4.

- **Contrast encoders**. In statistics, if $X_1, \ldots, X_k$ are random variables and $a_1, \ldots, a_k$ are real numbers such that not all of them are zero and $\sum_{i=1}^{k} a_i = 0$, then the linear combination

$$a_1X_1 + a_2X_2 + \cdots + a_kX_k = \sum_{i=1}^{k} a_iX_i$$

  is called a *contrast* [45]. Often—and we will do so in this work—the vector $\gamma = (a_1, \ldots, a_k)$ is itself called a contrast, and we say that two contrasts $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ are *orthogonal* if $\sum_{i=1}^{k} a_ib_i = 0$.

  For a categorical variable a contrast encoder uses one or more constrasts as codes. It is typical that if the variable has k categorical instances, the encoder uses $k - 1$ constrasts that are pairwise orthogonal. Also, the constrasts only depend on k and not on the particular variable or dataset. The main drawback of this type of encoders is that they increase substancially the number of features in datasets. Another issue is that these encoders place data points in N-dimensional space equally distanced from each other, disregarding any degree of proximity among categories.

Below we briefly explain the use of constrasts in statistics as this will help describe particular contrast encoders in Section 2.3.

Contrasts are chiefly used to test more specific hypotheses than the typical null hypothesis in an analysis of variance (ANOVA). In a typical ANOVA we are interested in testing whether the means of k population groups are the same. If those means are $\mu_1, \ldots, \mu_k$, the *null hypothesis* of the ANOVA is $H_0 : \mu_1 = \mu_2 = \cdots = \mu_k$. If the results of the ANOVA suggest us to reject H_0, we have evidence towards the alternative hypothesis of at least two of the means being different. At this point, contrasts can be added to test for more precise hypotheses like $\mu_1 = \mu_2$ or $\mu_1 = \mu_3$. Formally, adding the contrast $\gamma = (a_1, \ldots, a_k)$ to the ANOVA would test for the hypothesis $H_\gamma : a_1\mu_1 + \cdots + a_k\mu_k = 0$. Thus, the contrast $(-1, 1, 0, \ldots, 0)$ corresponds to testing $\mu_2 - \mu_1 = 0$ or $\mu_1 = \mu_2$, and the contrast $(-1, \frac{1}{2}, \frac{1}{2}, 0, \ldots, 0)$ corresponds to testing $\mu_1 = \frac{\mu_2+\mu_3}{2}$. See [38, Chapter 8], [45, Chapter 13], [37, Section 17.3] and [1] for further details on the use of contrasts in statistics.

- **Latent space encoders**. For these encoders the dataset is first encoded in a simple way—for example, by the One Hot Encoder (Subsection 2.3.3) or by randomly assigning integers to categories—and then used to train a neural network. The neural network must have a few initial layers dedicated to reduce the dimension of the entries and the output of these initial layers will be the final encoded dataset. Through this process the original dataset is mapped to a set of latent variables in a space of small dimension. It is suggested that a high performance of the neural network implies that proximities and patterns in the original dataset are preserved.

  Inspiration for this type of encoders is drawn from the *word-embeddings* used in Natural Language Processing, which aim at offering a continuous representation of words based on semantics rather than a sparse representation that treats all words as independent. See [2, 3].

  The main drawback of the latent space encoders is that the target variable in the dataset is used for the training of the neural network. This introduces the problem of *target leakage*, which affects the generalisation performance of models trained with the encoded dataset. We will

explain more about target leakage in Section 2.4. Notice too that it is hard to interpret the variables in the encoded dataset.

- **Pattern-preserving encoders**. This type of encoders will be addressed in Chapter 3
- **Target encoders**. Their main characteristic is that they use the target (dependent) variable in the dataset to perform the encoding. Besides this, they might also fit into one of the previous types.

  Besides their obvious inadequacy for datasets lacking a target variable (unsupervised learning), the main drawback of target encoders is target leakage, see Section 2.4.

2.2 Example dataset

Along with the description of categorical encoders in Sections 2.3 and 3.2 we will show the result of applying each encoder to a specific dataset.

The dataset *Individual Company Sales Data* available at [7] contains sales data for a product a company offers; it is a mixed dataset with fourteen features and its target (dependent) variable is a flag marking whether the customer bought the product[2]. The specific example dataset used in this chapter is a preprocessed extract of *sales_data.csv*. Our dataset is split into two CSV files, Sales-Features containing only four of the feature variables and Sales-Target containing the target variable. Both CSV files are available in the repository for this project [15].

Table 2.1 resumes the structure of Sales-Features and Table 2.2 is a sample of its contents.

Variable	Type	No of values	Values
gender	Categorical	2	F, M
education	Categorical (ordinal)	5	lessHS, HS, Some College, Bach, Grad
region	Categorical	5	West, South, Midwest, Northeast, Rest
house_val	Numerical	—	range: 5000 to 9999999

Table 2.1: Variables in the Sales-Features dataset

[2]The target variable is 1 if the customer bought the product and 0 otherwise.

gender	education	region	house_val
F	Bach	West	248694
F	Bach	South	416925
F	Bach	South	245686
F	Some College	Midwest	360587
M	lessHS	South	162884
F	Some College	Midwest	239560
M	lessHS	Midwest	60822
F	Some College	Midwest	136729

Table 2.2: First 8 rows of the Sales-Features dataset

2.3 Encoders

2.3.1 Ordinal Encoder

This is the main encoder of ordinal type. If the categorical variable has an ordering, this ordering should be passed to the Ordinal Encoder. If no ordering is known or suggested the categories are encoded by order of appearance (top to bottom) in the dataset. The codes are integers from 0 to the number of categories minus 1.

Our reference implementation of the Ordinal Encoder is from the Scikit-Learn 0.22.2 library [40]. A sample of the encoding the Ordinal Encoder performs on the Sales-Features is Table 2.3.

gender	education	region	house_val
0	3	0	248694
0	3	1	416925
0	3	1	245686
0	2	2	360587
1	0	1	162884
0	2	2	239560
1	0	2	60822
0	2	2	136729

Table 2.3: Encoding of Sales-Features by the Ordinal Encoder. Note that only “education” has a natural ordered, the other categorical variables are encoded by order of appearance.

2.3.2 Polynomial Encoder

The Polynomial Encoder is considered an ordinal and constrast encoder. It works best with equally spaced, ordered categorical variables. In statistics, the (orthogonal) polynomial constrasts are used to test hypotheses about linear, quadratic, cubic, etc, trends in data. These contrasts are chiefly applied when a continuous independent variable is cut into ranges and made thus into a categorical variable. For their use in statistics see [12] and [6, Section 12.5.3.3].

For a categorical variable with k categorical instances the polynomial encoder uses $k-1$ polynomial contrasts to replace the column of the variable with $k-1$ numerical columns. The contrasts only depend on the number k. Table 2.4 shows the codes used for $k = 5$. The calculation of these contrasts is involved but a good explanation is in [39]. In this example, the code for the value A is the tuple (-0.632456, 0.534522, -0.316228, 0.119523).

Value	0_0	0_1	0_2	0_3
A	-0.632456	0.534522	-0.316228	0.119523
B	-0.316228	-0.267261	0.632456	-0.478091
C	0.000000	-0.534522	0.000000	0.717137
D	0.316228	-0.267261	-0.632456	-0.478091
E	0.632456	0.534522	0.316228	0.119523

Table 2.4: Codes used by the Polynomial Encoder for a variable with $k = 5$ values. Each numerical column is a constrast.

Our reference implementation of this encoder comes from the Category Encoders 2.1.0 library[3]. Table 2.5 is a sample of the encoding produced by the Polynomial Encoder on the Sales-Features dataset.

2.3.3 One Hot Encoder

This is likely the most popular encoder and virtually every software package for analysis and preprocessing of data implements it. It is also called *Dummy Encoder*. This encoder does not fit into any of

[3]A project by W. McGinnis that implements several categorical encoders and is highly compatible with Scikit-Learn and Pandas. Available at `https://contrib.scikit-learn.org/categorical-encoding/`.

gender_0	education_0	education_1	education_2	education_3	region_0	region_1	region_2	region_3	house_val
-0.707107	-0.632456	0.534522	-0.316228	0.119523	-0.632456	0.534522	-0.316228	0.119523	248694
-0.707107	-0.632456	0.534522	-0.316228	0.119523	-0.316228	-0.267261	0.632456	-0.478091	416925
-0.707107	-0.632456	0.534522	-0.316228	0.119523	-0.316228	-0.267261	0.632456	-0.478091	245686
-0.707107	-0.316228	-0.267261	0.632456	-0.478091	0.000000	-0.534522	0.000000	0.717137	360587
0.707107	0.000000	-0.534522	0.000000	0.717137	-0.316228	-0.267261	0.632456	-0.478091	162884
-0.707107	-0.316228	-0.267261	0.632456	-0.478091	0.000000	-0.534522	0.000000	0.717137	239560
0.707107	0.000000	-0.534522	0.000000	0.717137	0.000000	-0.534522	0.000000	0.717137	60822
-0.707107	-0.316228	-0.267261	0.632456	-0.478091	0.000000	-0.534522	0.000000	0.717137	136729

Table 2.5: Encoding of Sales-Features by the Polynomial Encoder.

the types discussed in Section 2.1.

For a categorical variable with k instances the One Hot Encoder replaces the column of the variable with k columns of 0's and 1's. One of the instances is chosen and associated to the code $e_1 = (1, 0, \dots, 0)$ and from then on other values get codes $e_2 = (0, 1, 0, \dots, 0)$, $e_3 = (0, 0, 1, 0, \dots, 0)$, etc, until $e_k = (0, \dots, 0, 1)$. Some implementations with a more pronounced statistical point of view instead use vectors of length $k-1$, starting with $(0, 0, \dots, 0)$ and continue with analogues to $e_1, e_2, \dots, e_{k-1}$, as this corresponds conceptually to comparing a control group (the one that gets the all-zeroes code) to $k-1$ treatments in an experiment. In this work we follow the approach with codes e_1 to e_k as this is rather more common in machine learning.

Our reference implementation of this encoder comes from the Category Encoders 2.1.0 library. Table 2.6 is a sample of the encoding produced by the One Hot Encoder on the Sales-Features dataset.

Disadvantages of the One Hot Encoder are that the proximities between categories are entirely ignored and that datasets could easily become so large and sparse that the training of models could become inefficient due to implementation and memory constrains.

2.3.4 Backward Difference Encoder

This is a constrast encoder suitable for any categorical variable. The constrasts come from a specific statistical design where we want to point out where a response (a particular/interesting change in the values of the target variable) starts; see [12].

gender_1	gender_2	education_1	education_2	education_3	education_4	education_5	region_1	region_2	region_3	region_4	region_5	house_val
1	0	1	0	0	0	0	1	0	0	0	0	248694
1	0	1	0	0	0	0	0	1	0	0	0	416925
1	0	1	0	0	0	0	0	1	0	0	0	245686
1	0	0	1	0	0	0	0	0	1	0	0	360587
0	1	0	0	1	0	0	0	1	0	0	0	162884
1	0	0	1	0	0	0	0	0	1	0	0	239560
0	1	0	0	1	0	0	0	0	1	0	0	60822
1	0	0	1	0	0	0	0	0	1	0	0	136729
0	1	0	0	1	0	0	0	0	0	1	0	128402
0	1	1	0	0	0	0	0	1	0	0	0	308817

Table 2.6: Encoding of Sales-Features by the One Hot Encoder.

For a categorical variable with k instances the Backward Difference Encoder uses $k-1$ contrasts to replace the column of the variable by $k-1$ numerical columns. Conceptually, in an ANOVA, the $j-th$ constrast compares the means $\mu_1, \ldots, \mu_j$ of the first j groups to the means $\mu_{j+1}, \ldots, \mu_k$ of the subsequent groups. The first contrast is $\left(-\frac{k-1}{k}, \frac{1}{k}, \ldots, \frac{1}{k}\right)$, the second is $\left(-\frac{k-2}{k}, -\frac{k-2}{k}, \frac{2}{k}, \ldots, \frac{2}{k}\right)$, the third is $\left(-\frac{k-3}{k}, -\frac{k-3}{k}, -\frac{k-3}{k}, \frac{3}{k}, \ldots, \frac{3}{k}\right)$, so on, until the $(k-1)$-th which is $\left(-\frac{1}{k}, \ldots, -\frac{1}{k}, \frac{k-1}{k}\right)$.

Our reference implementation of this encoder comes from the Category Encoders 2.1.0 library. Table 2.7 is the encoding for a variable with $k = 5$, where, for example, the value A gets the tuple $(-4/5, -3/5, -2/5, -1/5)$ as code. Table 2.8 is a sample of the encoding performed by the Backward Difference Encoder on the Sales-Features dataset.

Value	0_1	0_2	0_3	0_4
A	-4/5	-3/5	-2/15	-1/5
B	1/5	-3/5	-2/15	-1/5
C	1/5	2/5	-2/15	-1/5
D	1/5	2/5	3/5	-1/5
E	1/5	2/5	3/5	4/5

Table 2.7: Encoding by the Backward Difference Encoder for a categorical variable with $k = 5$. Each numerical column is a constrast.

The related *Forward Difference Encoder* instead proposes contrasts used to point out the end of a response. We decided not to include this encoder in this work as it clearly is a dual to the Backward Difference Encoder.

gender_0	education_0	education_1	education_2	education_3	region_0	region_1	region_2	region_3	house_val
-0.5	-0.8	-0.6	-0.4	-0.2	-0.8	-0.6	-0.4	-0.2	248694
-0.5	-0.8	-0.6	-0.4	-0.2	0.2	-0.6	-0.4	-0.2	416925
-0.5	-0.8	-0.6	-0.4	-0.2	0.2	-0.6	-0.4	-0.2	245686
-0.5	0.2	-0.6	-0.4	-0.2	0.2	0.4	-0.4	-0.2	360587
0.5	0.2	0.4	-0.4	-0.2	0.2	-0.6	-0.4	-0.2	162884
-0.5	0.2	-0.6	-0.4	-0.2	0.2	0.4	-0.4	-0.2	239560
0.5	0.2	0.4	-0.4	-0.2	0.2	0.4	-0.4	-0.2	60822
-0.5	0.2	-0.6	-0.4	-0.2	0.2	0.4	-0.4	-0.2	136729
0.5	0.2	0.4	-0.4	-0.2	0.2	0.4	0.6	-0.2	128402
0.5	-0.8	-0.6	-0.4	-0.2	0.2	-0.6	-0.4	-0.2	308817

Table 2.8: Encoding of Sales-Features by the Backward Difference Encoder.

2.3.5 Helmert Encoder

This is a constrast encoder suitable for any categorical variable. The version of this encoder that we use in this work is called *Reverse Helmert Encoder* in [12]; the Reverse and Forward Helmert Encoders are dual to each other and is enough to pick one of the two.

For a categorical variable with k instances, the Helmert Encoder uses $k-1$ contrasts to replace the column of the variable with $k-1$ numerical columns. Added to an ANOVA, the $j-$th contrast compares the mean μ_{j+1} of the $(j+1)-$th group to the means $\mu_1, \ldots, \mu_j$ of the prior j groups. The first contrast is $\left(-\frac{1}{2}, \frac{1}{2}, 0, \ldots, 0\right)$, the second is $\left(-\frac{1}{3}, -\frac{1}{3}, \frac{2}{3}, 0, \ldots, 0\right)$, so on, until the $(k-1)-$th which is $\left(-\frac{1}{k}, \ldots, -\frac{1}{k}, \frac{k-1}{k}\right)$.

Our reference implementation of this encoder is from the Category Encoders 2.1.0 library. Table 2.9 is the encoding for a variable with $k=5$, and Table 2.10 is a sample of the encoding performed by the Helmert Encoder on the Sales-Features dataset.

Category	0_1	0_2	0_3	0_4
A	-1/2	-1/3	-1/4	-1/5
B	1/2	-1/3	-1/4	-1/5
C	0	2/3	-1/4	-1/5
D	0	0	3/4	-1/5
E	0	0	0	4/5

Table 2.9: Encoding by the Helmert Encoder for a categorical variable with $k=5$ values. Each numerical column is a constrast.

gender_0	education_0	education_1	education_2	education_3	region_0	region_1	region_2	region_3	house_val
-0.5	-0.5	-0.333333	-0.25	-0.2	-0.5	-0.333333	-0.25	-0.2	248694
-0.5	-0.5	-0.333333	-0.25	-0.2	0.5	-0.333333	-0.25	-0.2	416925
-0.5	-0.5	-0.333333	-0.25	-0.2	0.5	-0.333333	-0.25	-0.2	245686
-0.5	0.5	-0.333333	-0.25	-0.2	0.0	0.666667	-0.25	-0.2	360587
0.5	0.0	0.666667	-0.25	-0.2	0.5	-0.333333	-0.25	-0.2	162884
-0.5	0.5	-0.333333	-0.25	-0.2	0.0	0.666667	-0.25	-0.2	239560
0.5	0.0	0.666667	-0.25	-0.2	0.0	0.666667	-0.25	-0.2	60822
-0.5	0.5	-0.333333	-0.25	-0.2	0.0	0.666667	-0.25	-0.2	136729
0.5	0.0	0.666667	-0.25	-0.2	0.0	0.000000	0.75	-0.2	128402
0.5	-0.5	-0.333333	-0.25	-0.2	0.5	-0.333333	-0.25	-0.2	308817

Table 2.10: Encoding of Sales-Features by the Helmert Encoder.

2.3.6 Entity Embedding Encoder

This is the main latent space encoder. Among all the encoders in this section is probably the least known as it is comparatively recent. It was implicitly proposed in [21] and addresses the main problems of the One Hot Encoder: it aims at delivering a numerical representation of data that (a) does not increase the dimensions of datasets excessively and (b) reflects/preserves actual proximities among categories.

A latent space encoder is determined by the architecture of a neural network. In the case of the Entity Embedding Encoder, the first part of its neural network consists of *embedding layers*, which are layers used in Natural Language Processing to represent words as numerical vectors. The crucial observation here is that the values of categorical variables are regarded and processed as words in a vocabulary. The embedding layers we use in the Entity Embedding Encoder will be called *EE layers* and, in terms of their implementation, they are the `Embedding` layers from the Keras API[4].

Based on ideas in [21], given a mixed dataset, the Entity Embedding Encoder works as described below.

- Each numerical variable is scaled to $[0, 1]$.
- Each categorical variable is encoded with integers (in no specific way or order, as we would do with words).

[4]See `https://keras.io/api/layers/core_layers/embedding/`

- A neural network is built as follows (see Figure 2.1 for a graphical example).
  - Start by adding an input layer for each variable.
  - Right after the input layer, for each categorical variable we add an EE layer. A parameter of the EE layer is the desired length of the ouput vectors. This parameter is called *the size or dimension of* the EE layer. If the variable has k instances, the corresponding EE layer should have size larger than 1 to allow for enough trainable weights but—because we want compact numerical datasets—the size should be significantly smaller than k. There is no blanket guideline for the size of EE layers. For the experiment in [21, Section VI] the size was aggresively small for large k (e.g. for $k = 1115$ a size 10 EE layer was used) and $k/2$ or $k - 1$ for small k (< 50). All EE layers use the identity as activation function (linear activation). Note that this step is skipped for numerical variables.
  - The outputs of all the EE layers and the numerical variables are concatenated into a single numerical vector.
  - The vector from the previous step is sequentially fed to two dense layers with 1000 and 500 neurons, respectively; each of these dense layers uses ReLU as activaction function.
  - The final layer, the output layer, should be the natural depending on the target variable: if categorical, a softmax layer with as many neurons as target categories; if numerical, a dense layer with one neuron and sigmoid activation function.
  - No dropout or other regularisation considered.
- The neural network built above is trained.
- A categorical variable C is encoded with the outputs of its corresponding EE layer.

The reference implementation for this encoder is our own and is available as the standalone module [17]. In this implementation we default to $\min(30, \ulcorner k/3 \urcorner)$ for the size of the EE layer of a variable with k unique values[5], see Appendix A.3.

[5] Where $\ulcorner x \urcorner$ stands for the ceiling of x, the smallest integer equal to or larger than x.

Table 2.11 is a sample of the encoding of the Sales-Features dataset by the Entity Embedding Encoder. Column 5 is the numerical house_val variable scaled to $[0, 1]$. Figure 2.1 is the Neural Network used for the encoding.

region_0	region_1	education_0	education_1	genre	house_val
0.412381	0.639337	0.404513	0.248812	0.429172	0.024382
0.412381	0.639337	0.404513	0.569963	0.349609	0.041213
0.412381	0.639337	0.404513	0.569963	0.349609	0.024081
0.412381	0.435858	0.375698	0.434408	0.298243	0.035576
0.196514	0.429078	0.639524	0.569963	0.349609	0.015796
0.412381	0.435858	0.375698	0.434408	0.298243	0.023468
0.196514	0.429078	0.639524	0.434408	0.298243	0.005585
0.412381	0.435858	0.375698	0.434408	0.298243	0.013179

Table 2.11: Encoding of Sales-Features by the Entity Embedding Encoder.

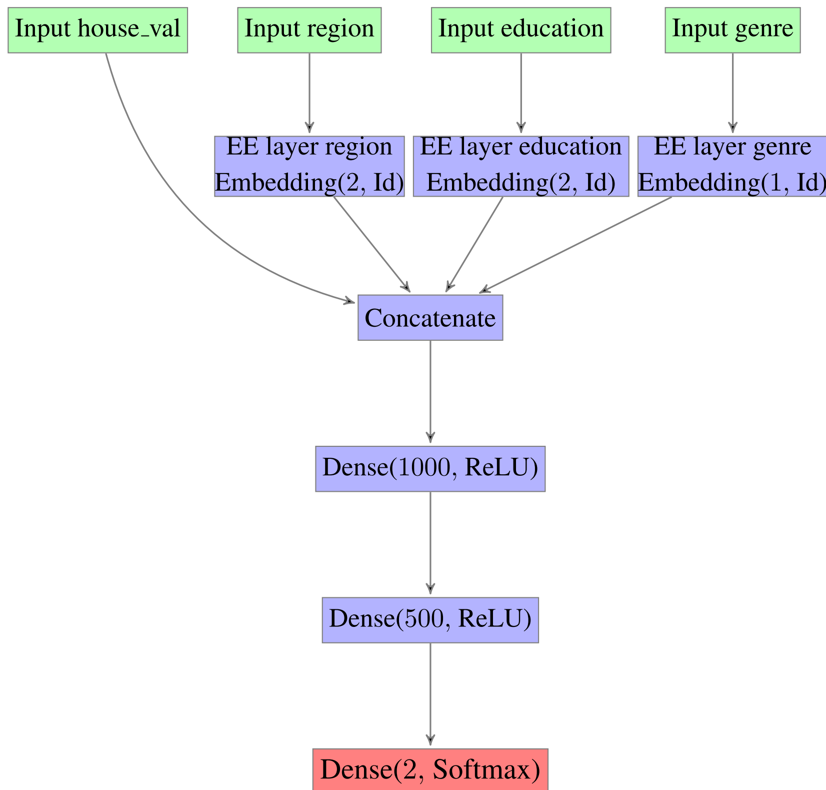

Fig. 2.1: Neural network architecture used to encode the Sales-Features dataset with the Entity Embedding Encoder. After training, the different outputs of an EE layer are the codes for the different values of its corresponding variable.

2.3.7 Target Encoder

This encoder was developed in [36] with the aim of encoding categorical variables with a large number of unique values. Its statistical foundation attempts to ensure that most of the predictive power of a categorical variable is preserved after encoding [*ibid.*].

For a categorical variable with k instances the Target Encoder uses k numbers in $[0,1]$ as codes. Let C be one of the k instances. If the target variable Y is itself categorical, the code of C is $\lambda\mathbb{P}(Y|C) + (1-\lambda)\mathbb{P}(Y)$, where $\mathbb{P}(Y|C)$ is the probability of the target given the category C, $\mathbb{P}(Y)$ is the overall probability of the target in the whole dataset and λ is a number in $(0,1)$ that depends on the size of C. Similarly, if the target variable is continuous, the code of C is $\lambda\mathbb{E}(Y|C)+(1-\lambda)\mathbb{E}(Y)$, where probabilities have been replaced by expected values (scaled to $[0,1]$). See [36, Section 3] for guidelines on the selection of λ.

Note that when two categories C and D have similar target statistics, depending on the type of Y, $\mathbb{P}(Y|C) \approx \mathbb{P}(Y|D)$ or $\mathbb{E}(Y|C)) \approx \mathbb{E}(Y|D)$, so C and D get similar codes under the Target Encoder. This is the basis for how this encoder deals with a large number of unique categories.

Our reference implementation of this encoder is from the Category Encoders 2.1.0 library. Table 2.12 is a sample of the encoding performed by the Target Encoder on the Sales-Features dataset using Sales-Target as target.

gender	education	region	house_val
0.473677	0.702636	0.625768	248694
0.473677	0.702636	0.612936	416925
0.473677	0.702636	0.612936	245686
0.473677	0.593878	0.564654	360587
0.683841	0.395194	0.612936	162884
0.473677	0.593878	0.564654	239560
0.683841	0.395194	0.564654	60822
0.473677	0.593878	0.564654	136729

Table 2.12: Encoding of Sales-Features by the Target Encoder.

2.3.8 Weight of Evidence Encoder

This encoder is based on a technique for measuring the predictive power that a variable has on the target variable. [4] is an excellent reference on such technique and its use in predictive modelling. The Weight of Evidence Encoder is a target encoder and it is suitable for any categorical variable.

Briefly, if the target variable is binary with 0 (negative event) and 1 (positive event) as values, the code associated to a categorical instance C is given by:

$$\begin{aligned}\text{Code of } C &= \log\left(\frac{\%\text{ of negative events in } C}{\%\text{ of positive events in } C}\right)\\ &= \log\left(\frac{\#\text{ of elements with value 0 in } C \times \#\text{ of elements with value 1}}{\#\text{ of elements with value 0} \times \#\text{ of elements with value 1 in } C}\right)\end{aligned} \tag{2.1}$$

The formula 2.1 is readily generalised to the situation where the target variable is no longer binary but still discrete. When the target variable is continuous, it is first binned and summarised, which puts the problem back in the case of a discrete target variable. See [5] for details.

Our reference implementation of this encoder is from the Category Encoders 2.1.0 library[6]. Table 2.13 is a sample of the encoding performed by the Weight of Evidence Encoder on the Sales-Features dataset using Sales-Target as target.

gender	education	region	house_val
-0.531498	0.433372	0.087714	248694
-0.531498	0.433372	0.033412	416925
-0.531498	0.433372	0.033412	245686
-0.531498	-0.046261	-0.166202	360587
0.345203	-0.851086	0.033412	162884
-0.531498	-0.046261	-0.166202	239560
0.345203	-0.851086	-0.166202	60822
-0.531498	-0.046261	-0.166202	136729

Table 2.13: Encoding of Sales-Features by the Weight of Evidence Encoder.

[6]This implementation does not work when the target variable is discrete but not binary.

2.4 Target leakage

Target leakage, or *data leakage*, occurs if before or during the training of a model we use information about the target variable that either does not generalise to or is unavailable for the test data [35, 26]. The main problem that target leakage introduces is that models generalise poorly, i.e., have poor performance on test data despite performing well on training data. categorical

In the encoding of categorical variables target leakage is present when the encoding method uses the target variable. Target encoders may provide codes that already contain useful information towards the prediction of the target variable on the training data. Of the encoders reviewed in Section 2.3, the Entity Embedding, the Target and the Weight of Evidence encoders are target encoders and caution should be exercised when any of them is employed.

There are no definitive guidelines on how to reduce the impact of target leakage on encoding but the following can be considered:

- Mostly pay attention to the test performance of models.
- Be suspicious and investigate further if the encoding produces almost perfect prediction on the training data for more than one model.
- To report the performance of the combination of a target encoder E and a model M, consider a method similar to K*-fold cross validation*: we partition the data into $K \geq 3$ disjoint sets $D_1, \ldots, D_K$ and then for each D_i, (1) apply E to the union U_i of all D_j's with $j \neq i$, (2) train M with the encoded U_i and let t_i be the performance of M on D_i. Finally, the performance of the combination of E and M is the average of the t_i's. A disadvantage here is that some encoders would require an excessive amount of time to complete this process; so, K should not be large.

Chapter 3

Pattern-Preserving Encoders

In this chapter we present and discuss a more novel type of encoders. These are based in a proposed meaning to preserving patterns in data.

3.1 Preserving patterns

Machine learning is primarily concerned with *pattern recognition*; that is, identifying regularities and non-accidental features of data. The purpose of this is to obtain generative or predictive models, whose performance serves as an indicator of the success in the recognition of patterns.

Whilst pattern recognition is well and objectively understood in machine learning, the same cannot be said about pattern preservation. For the author of this work, *pattern preservation after the modification of data means that such modification has no impact on the results of pattern recognition tasks*. As natural as it may seem, this idea has flaws and is practically impossible to apply regarding the encoding of categorical variables. As pointed out in Section 1.2, the main problem with mixed datasets is that most of the well-established models in machine learning cannot be used with them. Hence, the evaluation of the pattern preservation after categorical encoding is limited, because we cannot apply the same models to a mixed dataset and its encoded version.

Below we present a proposal based on the work of A. F. Kuri-Morales [29, 30] on what pattern preservation means, and on which some novel methods of encoding are inspired.

3.1.1 The Preservation of Patterns Principle

What we discuss in this section is a formalisation of the ideas that previously appeared in [29, 30]. The driving idea is that for pattern preservation the categorical encoding must reflect the relationships that each variable in the dataset has with all (or some) of the others. For the rest of the chapter, we will asumme that numerical variables in mixed datasets are all scaled to $[0, 1]$.

Let $m \geq 2$. Fix $1 \leq l \leq m - 1$ and let $\mathcal{F} := \{f \mid f \text{ is a function from } [0, 1]^l \text{ to } [0, 1]\}$. Let $\mathcal{A}$ be a family of functions from $\mathcal{F}$. Ideally, $\mathcal{A}$ should consists of universal approximators for continuous functions in $\mathcal{F}$. For example, $\mathcal{A}$ can be the set of all the polynomials in l variables (Stone-Weierstrass Theorem, mathematical folklore) or the output functions of fully-connected neural networks with l input neurons, at least one hidden layer with sigmoidal activations and an output neuron with linear/sigmoidal activation (Universal Approximation Theorem for neural networks [9, 13]).

Let D be a mixed dataset with m independent variables (there might be a target variable but it will not be used).

Definition 3.1.1. Let S be a set of codes in $[0, 1]$ for all the categorical instances in D. The *fitness of* S is calculated as follows. Encode D with S to obtain $D(S)$ and enumerate the variables in $D(S)$ as $v_1, \ldots, v_m$. For each $i \in \{1, \ldots, m\}$ find (fit the parameters of) a function $f_i \in \mathcal{A}$ that takes l of the variables $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m$ as arguments and approximates v_i over $D(S)$. Let e_i be some measure of the approximation error, e.g. the maximum absolute error or the mean squared error. We set the fitness of S as

$$\text{fitness}(S) := \max_{i \leq m} e_i.$$

The fitness of a set of codes is an indicator of how well each variable in the encoded dataset relates to all ($l = m - 1$) or some ($l < m - 1$) of the other variables.

The following is based upon the ideas in [29] and [30, Section 2.2] on what pattern preservation means.

> **Preservation of Patterns Principle**: Encoding with a set of codes of minimum fitness preserves the patterns in D.[1]

This principle suggests that a way to encode mixed datasets is to search for sets of codes of minimum fitness. Encoders that have this idea at their core will be called *pattern-preserving encoders* and some will be presented in the following section.

3.2 Pattern-preserving encoders

Succintly, a *Pattern-Preserving Encoder* is one that aims to find a set of codes of minimum fitness with which to encode. Therefore,

Definition 3.2.1. According to what we discussed in the previous section and its definition, a pattern-preserving encoder is determined by

1. an integer $l \geq 1$ that will state how many variables are used to approximate another,
2. a family of functions $\mathcal{A}$ whose elements will be used to approximate continuous functions in l variables,
3. a method to fit[2] the elements of $\mathcal{A}$ along with a way to measure the approximation errors,
4. a sampling method for sets of codes.

[1] There is a difference here with [29, 30]. As explained in [29, Section 4.3] and [30, Section 2.2], a set of codes that preserves the patterns in D is one in the Pareto front of the multi-objective optimisation problem $\min\{e_0, e_1, \ldots, e_m\}$. We have explicitly swapped this multi-objective problem for the single-objetive problem $\min(\max\{e_0, e_1, \ldots, e_m\})$ (minimise the fitness). The problems are not mathematically equivalent, but we found that the main algorithm (CENG below and in [29, 30, 34]) actually solves our single-objective problem. We believe that taking the steps to solve the more complex multi-objective problem would bring no significant benefit to encoding and would certainly worsen the already computationally taxing nature of the methods.

[2] That is, given any continuous function f the method must find an approximator of f in $\mathcal{A}$.

In consequence, there is an unlimited number of possible pattern-preserving encoders. In the following subsections we explore some of these.

Before moving to the technical details of some pattern-preserving encoders it is worth pointing out that in practice finding a set of codes of minimum fitness is unlikely. The main reason is that the number of possible sets of codes grows exponentially with respect to the number of categorical instances. Another reason is that the choices for 1–4 in Definition 3.2.1 typically introduce some level of probabilistic uncertainty to the evaluation of fitness. Consequential issues are that all the pattern-preserving encoders require large amounts of computational resources and that they return different sets of codes at each run.

3.2.1 CENG Encoder

CENG stands for *Categorical Encoding via Neural Networks and Genetic Algorithms*. This method of encoding was introduced in [29] and it was the first pattern-preserving encoder.

Definition 3.2.2. The CENG Encoder is determined as follows.

1. l is the maximum number possible, i.e. the number of independent variables in the dataset minus 1,
2. $\mathcal{A}$ is the family of all the output functions of fully-connected neural networks with l input neurons, two hidden layers with sigmoidal activations and a single output neuron with sigmoidal activation,
3. naturally, the method to fit (train) the elements of $\mathcal{A}$ is *back-propagation*; the error measure is the standard mean squared error,[3]
4. the sampling of sets of codes is directed by the *Eclectic Genetic Algorithm*.

Perhaps the most remarkable aspect of the CENG Encoder is its use of a genetic algorithm to traverse the search-space of sets of codes. The Eclectic Genetic Algorithm is an elitist genetic algorithm

[3] This implicity assumes that classification tasks are seen as regression tasks, as is sometimes done in machine learning.

that has been shown to have an above-average performance compared to other well-known genetic algorithms; see [32, 33] for the details and advantages of this algorithm.

For a dataset D with m independent variables, for each element of each generation of the population in the genetic algorithm, the CENG Encoder requires fitting m neural networks. If m is large and D has a large number of entries, the whole procces becomes overwhelmingly cpu-expensive. To addressed this, [34] introduced the parallelisation of the process. Also, some of the other pattern-preserving encoders, in fact, were developed to improve the processing time of the CENG Encoder.

The stocasthic nature of several steps in the process indicate that two runs of the CENG Encoder will almost surely produce different encodings. In [30], *validation of clusters* is offered as evidence towards the equivalence between encodings produced by this encoder.

Our reference implementation for the CENG Encoder is due to A. F. Kuri-Morales and some of his collaborators. The binary files of this implementation are included in the repository for the present work [15]. In this implementation the number of neurons in the hidden layers of the neural networks in item 2 of Definition 3.2.1 is calculated from a measure of the information in the dataset, following the indications in [28]. Table 3.1 is a sample of the encoding of the Sales-Features dataset performed by this implementation of the CENG Encoder.

gender	education	region	house_val
.277814388	.507709026	.687397003	0.024382
.277814388	.507709026	.501939296	0.041213
.277814388	.507709026	.501939296	0.024081
.277814388	.374680757	.715367317	0.035576
.343182802	.556364059	.501939296	0.015796
.277814388	.374680757	.715367317	0.023468
.343182802	.556364059	.715367317	0.005585
.277814388	.374680757	.715367317	0.013179

Table 3.1: Encoding of Sales-Features by the CENG Encoder.

3.2.2 Genetic Pattern-Preserving Encoder

This encoder aggressively weakens some of the requirements of the CENG Encoder, aiming at reducing the computational requirements.

Definition 3.2.3. The Genetic Pattern-Preserving Encoder is determined as follows.

1. $l = 2$,
2. $\mathcal{A}$ is the family of all polynomials in two variables with no interactions and of degree at most three,
3. the method to fit the elements of $\mathcal{A}$ is the typical Ordinary Least Squares algorithm; the error measure is the standard mean squared error,
4. the sampling of codes is done using the Eclectic Genetic Algorithm.

Compared to the CENG Encoder, the crucial changes are that only two variables are used for each approximation ($l = 2$) instead of the maximum possible and the approximation functions are no longer universal approximators but are much simpler and much faster to fit.

Our reference implementation for this encoder is our own[4]; it is available in [15] and in the standalone module [19]. Table 3.2 is a sample of the encoding of the Sales-Features dataset performed by the Genetic Pattern-Preserving Encoder.

gender	education	region	house_val
0.884982	0.387128	0.340757	0.024382
0.884982	0.387128	0.336506	0.041213
0.884982	0.387128	0.336506	0.024081
0.884982	0.636561	0.172570	0.035576
0.632512	0.642117	0.336506	0.015796
0.884982	0.636561	0.172570	0.023468
0.632512	0.642117	0.172570	0.005585
0.884982	0.636561	0.172570	0.013179

Table 3.2: Encoding of Sales-Features by the Genetic Pattern-Preserving Encoder.

[4] It is worth pointing out that we employ a direct representation of sets of codes as arrays of real numbers, while in the CENG Encoder a more costly binary representation is used.

3.2.3 Aging Pattern-Preserving Encoder

The only difference with the previous Genetic Pattern-Preserving Encoder is that the sampling method is replaced by a lighter genetic algorithm.

Definition 3.2.4. The Aging Pattern-Preserving Encoder is determined as follows.

1. $l = 2$,
2. $\mathcal{A}$ is the family of all polynomials in two variables with no interactions and of degree at most three,
3. the method to fit the elements of $\mathcal{A}$ is the typical Ordinary Least Squares algorithm; the error measure is the standard mean squared error,
4. the sampling of codes is via the *Aging Evolution Algorithm*.

The Aging Evolution Algorithm is a simplified, non-elitist, genetic algorithm that only performs mutation on selected individuals and removes the oldest individual at each cycle. The original use of this algorithm was in the automatic design of deep neural networks, see [42].

Our reference implementation for this encoder is our own. It is included in [15] and in the standalone module [19]. Table 3.3 is a sample of the encoding of the Sales-Features dataset performed by the Aging Pattern-Preserving Encoder.

gender	education	region	house_val
0.497922	0.611118	0.384466	0.024382
0.497922	0.611118	0.034238	0.041213
0.497922	0.611118	0.034238	0.024081
0.497922	0.703164	0.080297	0.035576
0.651360	0.509853	0.034238	0.015796
0.497922	0.703164	0.080297	0.023468
0.651360	0.509853	0.080297	0.005585
0.497922	0.703164	0.080297	0.013179

Table 3.3: Encoding of Sales-Features by the Aging Pattern-Preserving Encoder.

3.2.4 Simple Pattern-Preserving Encoder

This encoder is mainly included as a performance reference for all the other pattern-preserving encoders.

Definition 3.2.5. The Simple Pattern-Preserving Encoder is determined as follows.

1. $l = 2$,
2. $\mathcal{A}$ is the family of all polynomials in two variables with no interactions and of degree at most three,
3. the method to fit the elements of $\mathcal{A}$ is the typical Ordinary Least Squares algorithm; the error measure is the standard mean squared error,
4. the way to sample sets of codes is simply random-search.

Our reference implementation for the Simple Pattern-Preserving Encoder is our own. It is included in the repository for this work [15] and as part of the standalone module [19]. This implementation samples a total of 600 sets of codes. Table 3.4 is a sample of the encoding of the Sales-Features dataset performed by the Simple Pattern-Preserving Encoder.

gender	education	region	house_val
0.352530	0.379774	0.263527	0.024382
0.352530	0.379774	0.430866	0.041213
0.352530	0.379774	0.430866	0.024081
0.352530	0.459335	0.303602	0.035576
0.299552	0.148200	0.430866	0.015796
0.352530	0.459335	0.303602	0.023468
0.299552	0.148200	0.303602	0.005585
0.352530	0.459335	0.303602	0.013179

Table 3.4: Encoding of Sales-Features by the Simple Pattern-Preserving Encoder.

3.2.5 CESAMO Encoder

CESAMO stands for *Categorical Encoding by Statistical Applied Modelling*. This encoder was introduced in [30, 31] as a reformulation of the CENG Encoder to reduce the resource requirements. A thorough exposition is [14].

This encoder incorporates rather different elements compared to the previous encoders.

Definition 3.2.6. The CESAMO Encoder is determined as follows.

1. $l = 1$,
2. $\mathcal{A}$ is the family of all polynomials of the form

$$p(x) = \beta_0 + \beta_1 x^1 + \beta_2 x^3 + \beta_3 x^5 + \beta_4 x^7 + \beta_5 x^9 + \beta_6 x^{11}, \tag{3.1}$$

3. the method to fit the elements of $\mathcal{A}$ is the standard Ordinary Least Square with constrains; the error measure is the mean squared error,
4. the way to sample sets of codes is the following,

   • For each categorical variable C in the dataset:

   (i) Let E be an empty set.

   (ii) Assign random numbers in $[0, 1]$ as codes for the distinct values of C.

   (iii) Choose randomly and with replacement a variable V different from C. If V is itself categorical, assign random numbers in $[0, 1]$ as codes for its values.

   (iv) Find a polynomial p of the form in 3.1 such that p has V as argument and approximates C.

   (v) Set e to be the mean squared error of the approximation of C by p, and put e in E.

   (vi) If the distribution of the elements in E calculated so far is not normal, re-

turn to step (ii). Otherwise, encode C with the codes that correspond to the minimum error in E.

That the polynomials in (3.1) are univariate and of fixed maximum degree makes the CESAMO Encoder require less computational resources than the CENG Encoder, at the cost of no longer being universal approximators. In step (vi) above, to test the normality of the distribution of the elements in E we use the *D'Agostino-Pearson omnibus test* [10]. This test has similar power to the popular *Saphiro-Wilk test*[5].

Our reference implementation for the CESAMO Encoder is our own. It is included in the repository for this work [15] and it is also available as the stand-alone module [16]. Table 3.5 is a sample of the encoding of the Sales-Features dataset performed by this implementation of the CESAMO Pattern-Preserving Encoder.

gender	education	region	house_val
0.358548	0.456529	0.740997	0.024382
0.358548	0.456529	0.822969	0.041213
0.358548	0.456529	0.822969	0.024081
0.358548	0.440519	0.652679	0.035576
0.353254	0.415911	0.822969	0.015796
0.358548	0.440519	0.652679	0.023468
0.353254	0.415911	0.652679	0.005585
0.358548	0.440519	0.652679	0.013179

Table 3.5: Encoding of Sales-Features by the CESAMO Pattern-Preserving Encoder.

It is worth mentioning that the first version of the CESAMO Encoder and its implementation disagree on two technical points with our version above. First, the algorithm to fit the polynomials of the form in (3.1) is the *Ascent Algorithm* [8, Chapter 2]. Second, the normality test is a custom test detailed in [30, 31].

[5]We prefer the D'Agostino-Pearson test because it asks for a meaningful minimum number of samples and it is the normality test prefered by the *Scipy Statistical Package*.

Chapter 4

Experimental comparison of categorical encoders

4.1 On the evaluation and comparison of encoders

Our proposed way to evaluate categorical encoders aligns itself to our assertion that the need to encode mixed data at all is to improve the outcomes of pattern recognition tasks, that is, improve the performance of machine learning models.

We will evaluate and compare encoders by looking at the improvement they bring to machine learning models. So, for the encoders in Chapters 2 and 3 we need to set a number of datasets and models and a framework for a fair comparison that takes into account (A) the impact of target leakage for target encoders and (B) the chance of reporting biased performance due to the stochastical nature of the training of models.

The most important design decision is on the features of the framework mentioned above. Our proposal is encapsulated in the following definitions.

4.1.1 Definition of evaluation of encoders

Definition 4.1.1. (Supervised learning case) Fix an integer $K \geq 1$. Given a mixed dataset D with a target variable, an encoder E, a model M for supervised learning[1] and a performance metric for M, the *evaluation of* E *on* D *and* M, denoted by $v_E(D, M)$, is calculated as follows:

For $i = 1$ to K:

(i) Split D randomly into datasets D_1 and D_2 in the proportion 70% and 30%.

(ii) Apply the encoder E to D_1 and let S be the associated set of codes.

(iii) Train M with $D_1(S)$.

(iv) Encode D_2 with S, giving $D_2(S)$.

(v) Using the perfomance metric, let v_i be the performance of M on $D_2(S)$.

Set $v_E(D, M)$ to be the average of the v_i's, that is, $v_E(D, M) := \frac{1}{K}\sum_{i=1}^{K} v_i$

This way of evaluating encoders in supervised learning deals with (A) above by only looking at test perfomance of models, and deals with (B) by providing an average (if we set $K > 1$) for the performance of models (which should neutralise the chance of reporting atypical values).

In the absense of a target variable the risk of target leakage is absent, so for unsupervised learning there is no need to tackle (A).

Definition 4.1.2. (Unsupervised learning case) Fix an integer $K \geq 1$. Given a mixed dataset D with no target variable, an encoder E, a model M for unsupervised learning and a performance metric for M, the *evaluation of* E *on* D *and* M, denoted by $v_E(D, M)$, is calculated as follows:

For $i = 1$ to K:

(i) Apply the encoder E to D and let S be the associated set of codes.

(ii) Train M with $D(S)$.

(iii) Using the perfomance metric, let v_i be the performance of M on $D(S)$.

[1] e.g. if the target variable is binary, M can be a model for binary classification.

Set $v_E(D, M)$ to be the average of the v_i's, that is, $v_E(D, M) := \frac{1}{K}\sum_{i=1}^{K} v_i$

This definition deals with (B) the same way Definition 4.1.1 does in supervised learning.

Note that if we calculate $v_E(D, M)$ for all M in a family $\mathcal{M}$ of representative models of the same kind, we could obtain an overall measure of how good the encoder E is on the dataset D. If we wish to give a single number for that we can use the average

$$v_E(D) := \frac{1}{|\mathcal{M}|}\sum_{M\in\mathcal{M}} v_E(D, M),$$

4.1.2 The choice of K

The integer K in Definitions 4.1.1 and 4.1.2 is a meta-parameter in the evaluation of encoders. It is important for K to be bigger than 1, as that deals with (B) above. On the other hand, the fact that some encoders (particularly, the CENG Encoder and the GeneticPPEncoder) take a considerable amount of time to encode datasets, even if small, implies that K should not be too large. We thus set $K := 3$ for our study below.

4.1.3 Models

We wish to compare encoders on both supervised and unsupervised learning tasks. So we will include models and datasets for classification, regression and clustering. We first give details on the models considered for each of these three tasks.

Classification

Table 4.1 lists the models we consider for classification tasks. Their implementations all come from the package Scikit-Learn 0.22.2. The performance metric for these models is the *area under the ROC curve* [25, pag. 147], usually denoted as `auc`.

[2] Although it is natively a regression model, we use Linear Regression for classification regarding the predictions as approximations to the labels 0-1.

Model	Implementation
Naïve Bayes	`GaussianNB()`
Linear Regression[2]	`LinearRegression()`
Logistic Regression	`LogisticRegression()`
Linear Support Vector Machine	`LinearSVC()`
Radial Support Vector Machine	`SVC(kernel='rbf')`
K-Neighbours	`KNeighborsClassifier(K=7)`
Random Forest	`RandomForestClassifier(n_estimators=50)`
Neural Network	`MLPClassifier(hidden_layer_sizes=(50,20))`

Table 4.1: Models considered for classification

Regression

Table 4.2 lists the models we consider for regression tasks. Their implementations all come from the package Scikit-Learn 0.22.2. The performance metric for these models is the typical mean squared error, which we abreviate as `mse`.

Model	Implementation
Linear Regression	`LinearRegression()`
Linear Support Vector Machine	`LinearSVR()`
Radial Support Vector Machine	`SVR(kernel='rbf')`
K-Neighbours	`KNeighborsRegressor(K=7)`
Random Forest	`RandomForestRegressor(n_estimators=50)`
Neural Network	`MLPRegressor(hidden_layer_sizes=(50,20))`

Table 4.2: Models considered for regression

Clustering

Table 4.3 lists the models/algorithms we consider for clustering tasks. Their implementations all come from the package Scikit-Learn 0.22.2. The choice of performance metric in this case is not as straigth-forward as in the previous cases. Ours will be based on *validation indexes* [24, Section 4.2]

and is important to have in mind that each of these look for different characteristics of clusters (e.g. intra vs inter cluster distances or variances).

The index we have chosen for its simplicity and stability is the *Silhouette Coefficient* [43], which we abbreviate as `sil`. This index takes values in $[-1, 1]$, and higher values indicate better clusterings. For a single sample (entry), this index measures how well the sample is matched to its assigned cluster and not to other cluster, with the value being -1 if the sample would be better matched in a different cluster, +1 if the assigned cluster is actually the best cluster for the sample and 0 if the sample is well matched in more than one cluster (overlapping). For a dataset the silhouette coefficient is simply the average over all samples.

For reasons of wide exploration, we will actually perform clustering on datasets that have known labels. This gets rid of the issue of deciding how many clusters should be considered and also allows us to employ one more way to evaluate models and encoders. By the latter we mean that knowing the ground-truth labels we can use *external* validation indexes, which compare the ground-truth labels to the labels induced by the clustering. The external validation index we chose is the *Adjusted Mutual Information index* [47], denoted by `ami`. This is a measure of the agreement between the true labels and the labels given by the clusters corrected for agreement due to chance; a higher value indicates better clustering.

Note too that having ground-truth labels available enables the use of target encoders. However, we decided not to consider these encoders in the comparison because for unsupervised learning we do not include a mechanism for neutralising target leakage, see Definition 4.1.2.

Model	Implementation
K–Means	`Kmeans()`
Spectral Clustering	`SpectralClustering()`
Agglomerative Clustering	`AgglomerativeClustering()`

Table 4.3: Models considered for clustering

Models that do not require encoding

Table 4.4 gives the details of models/algorithms that can handle mixed data directly, without encoding. As explained in Chapter 1, we include these in our study with the purpose of showing that encoding leads to superior learning outcomes.

Name	Task	Package
Mixed Naïve Bayes	Classification	From remykarem[3]
CatBoost Classifier	Classification	Catboost 0.16
CatBoost Regressor	Regression	Catboost 0.16
C5.0	Classification	C50 R Library 0.1.3
K-Modes	Clustering	Kmodes 0.10.2[4]

Table 4.4: Models that do not require encoding.

For the first four models in Table 4.4 we report the average test performance (`auc` or mean squared error) out of ten independent training sessions. In each run the dataset is randomly split in the proportion 70% and 30%, to align the process with Definition 4.1.1. For K-Modes we cannot report the Silhouette Coefficient, because its calculation requires all features to be numerical, so we only report the `ami`.

Reported time

For each encoder we also report the accummulated time it took to complete its evaluation on all the models. For a sensible comparison, all encoders were evaluated on the same system, running on an AMD Ryzen 5-3500U CPU and 12GB of DDR4 RAM. The time was reported by the `perf_counter()` function from the `time` Python module.

[3] Available at https://github.com/remykarem/mixed-naive-bayes
[4] Available as the Pypi module `kmodes`

4.2 Breast Cancer Dataset

The dataset contains details about the recurrence or no recurrence of breast cancerous tumors. Details on the meaning of the variables and the original data are available in [48] at the UCI Datasets Archive.

The preprocessed[5] CSV file we use for this work is 'breast_cancer.csv', included in the repository for this work [15]. **This file consists of 277 entries (rows), a 0-1 target variable and 9 predictor variables. Of the 9 predictors 8 are categorical variables. There are a total of 38 categorical instances**.

Discussion

Table 4.6 contains the results of the comparison of encoders on this dataset. Recall that each numerical cell contains the `auc` for the corresponding encoder and model. For each row we have shaded the best value, which indicates the best encoder for the given model. We do this in all our future examples.

In general, we can notice that the linear regression model is best for this dataset, and the best combination of encoder and model is the AgingPP Encoder with the Linear Regression model (`auc`$=0.7626$). No encoder was best across all or most models. Also we can notice that, barring the CESAMO Encoder, all the pattern-preserving encoders have similar perfomance for each model.

In Table 4.5 we can see that the performance of models that do not require encoding is well below several of the combinations encoder–model in Table 4.6. The WOE Encoder has the best average performance ($v_E(D) = 0.6396$) over all models followed by the Target ($v_E(D) = 0.6365$) and the Simple Pattern-Preserving ($v_E(D) = 0.6338$) Encoders.

On this rather small dataset we can already notice that the Pattern-Preserving encoders take much

[5] e.g. we removed entries with unknown values

longer than the rest of encoders, with the CENG Encoder taking several, even hundreds of, times longer than the other encoders. This difference will only become larger with bigger datasets.

Model	Performance (auc)
Mixed Naïve Bayes	0.65813
CatBoost Classifier	0.61253
C5.0	0.67972

Table 4.5: Performance of models that do not require encoding on the Breast Cancer dataset.

Model\Encoder	Ordinal	WOE	Target	BwDifference	Polynomial	Helmert	OneHot	EntityEmb	CESAMO	AgingPP	SimplePP	GeneticPP	CENG
Naïve Bayes	0.6880	0.6801	0.6839	0.5689	0.6747	0.5792	0.5936	0.6882	0.6494	0.6854	0.7188	0.7112	0.6783
Linear Regression	0.6965	0.7363	0.7559	0.6967	0.6967	0.6820	0.6480	0.7196	0.6864	0.7626	0.7481	0.7457	0.7553
Logistic Regression	0.5909	0.6372	0.6150	0.6110	0.6109	0.6326	0.6202	0.6043	0.5000	0.6116	0.6343	0.6073	0.5674
Linear SVM	0.5784	0.6553	0.6166	0.6253	0.6036	0.6292	0.5856	0.5971	0.5395	0.6574	0.6371	0.6389	0.5858
Radial SVM	0.5095	0.6229	0.6553	0.5437	0.6131	0.5252	0.6224	0.4891	0.5678	0.6082	0.6206	0.5696	0.5868
K-Neighbours	0.5336	0.6330	0.6444	0.5503	0.6339	0.5256	0.6133	0.5841	0.5863	0.6215	0.6035	0.5865	0.6095
Random Forest	0.6496	0.6332	0.6209	0.6121	0.6023	0.6132	0.6374	0.6631	0.6073	0.6282	0.6219	0.6451	0.6176
Neural Network	0.5000	0.5190	0.5000	0.5049	0.4889	0.5321	0.4953	0.5744	0.5450	0.4892	0.4863	0.5013	0.5007
Average ($v_E(D)$)	0.5933	0.6396	0.6365	0.5891	0.6155	0.5899	0.6020	0.6150	0.5852	0.6330	0.6338	0.6257	0.6127
Time	4.7s	6.5s	6.8s	7.5s	7.6s	7.7s	7.8s	45.1s	2m 49s	2m 54s	4m 29s	8m 30s	27m 46s

Table 4.6: Comparison of encoders on the Breast Cancer Dataset; values are `auc`

4.3 Automobile MPG Dataset

The main task for this dataset is to predict the continuous attribute ‘mpg’ (miles-per-hour) of cars from numerical and categorical predictors. Details on the original data can be accessed in [41] at the UCI Datasets Archive.

The file we use here, a preprocessed version of the original, is the CSV file ‘mpg.csv’ included in the repository for this work [15]. **The file consists of 392 entries, a continuous target variable (mpg) and as predictors we have 5 numerical variables and 3 categorical variables. The total number of categorical instances is 317**, but there is an important observation here. One of the categorical variables stands for the car model, which is almost unique for each entry, accounting for 301 of the categorical instances. It seems that such variable is an unimportant predictor for the mpg, but we decided against removing it as this dataset provides a good example of a small dataset with a large amount of categorical instances.

The Weight of Evidence Encoder does not allow a continuous target variable and the used implementation of the Polynomial Encoder cannot handle a large number of categories for a single variable, so these two encoders were not evaluated on this dataset.

Discussion

The results of the comparison on this dataset are in Table 4.8[6]. In this case the performance metric is the `mse`, so lower values are better.

In each row we have shaded the value of the best encoder for the given model. Among all combinations of encoder and model, the one of the Random Forest model and the Target Encoder is the best (`mse` $= 0.0058$). It is also clear that for this dataset the Target Encoder ($v_E(D) = 0.0068$) was consistently the best encoder for all models. The second and third best encoders for all models were, respectively, the Entity Embedding ($v_E(D) = 0.0086$) and the CESAMO ($v_E(D) = 0.0096$) encoders.

[6] Some values are missing due to being out of reasonable ranges; this was likely due to numerical instabilities

This dataset supports regression, so the only applicable model that does not require encoding is the CatBoost Regressor; the result is in Table 4.7. In this case the CatBoost Regressor was worse only than the best and second-best combination of encoder and model (Target Encoder with Random Forest and Radial SVM, respectively) in Table 4.8. It is worth pointing out that the CatBoost algorithm already includes model selection, so with further tuning we could much further separate the performance of encoding plus a Random Forest from that of the CatBoost Regressor.

Model	Performance (mse)
CatBoost Regressor	0.00651

Table 4.7: Performance of models that do not require encoding on the Automobile MPG Dataset.

Model\Encoder	Target	Ordinal	Helmert	BwDifference	OneHot	CESAMO	EntityEmb	AgingPP	SimplePP	GeneticPP	CENG
Linear Regression	0.0072	0.0125	—	—	—	0.0107	0.0092	0.0124	0.0124	0.0130	0.0124
Linear SVM	0.0075	0.0448	0.0331	0.0100	0.0097	0.0105	0.0095	0.0132	0.0131	0.0139	0.0128
Radial SVM	0.0062	0.0423	0.0344	0.0124	0.0092	0.0097	0.0083	0.0114	0.0124	0.0122	0.0127
K-Neighbours	0.0071	0.0461	0.0382	0.0219	0.0108	0.0095	0.0075	0.0129	0.0128	0.0126	0.0128
Random Forest	0.0058	0.0099	0.0086	0.0119	0.0085	0.0071	0.0092	0.0094	0.0103	0.0105	0.0102
Neural Network	0.0071	0.0982	2.5560	0.0821	0.0129	0.0101	0.0077	0.0132	0.0118	0.0119	0.0140
Average ($v_E(D)$)	0.0068	0.0423	0.0534	0.0277	0.0102	0.0096	0.0086	0.0121	0.0121	0.0124	0.0125
Time	6.6s	9.5s	10.5s	13.9s	15.8s	23s	38.9s	3m 1s	5m 57s	13m 7s	35m 49s

Table 4.8: Comparison of encoders on the Automobile MPG Dataset; values are `mse`

4.4 Credit Card Dataset

This dataset contains details on credit card applications and it is suitable for classification (approval vs rejection). Details on the original data can be accessed in [46] at the UCI Datasets Archive.

The file we use is a preprocessed version of the original and it is included in the repository for this work [15] as the file 'credit.data'. **The file consists of 653 entries, a 0–1 target variable (credit card approval) and as predictors there are 6 numerical variables and 9 categorical variables. There are a total of 40 categorical instances**.

Discussion

The results of the evaluation of encoders on this dataset are in Table 4.10. In general, we obtained good test performance, with the best combination of encoder and model being that of the Weight of Evidence Encoder and Linear Regression (a certainly good $\texttt{auc} = 0.9449$). Accross all encoders except for the Entity Embedding Encoder, the Linear Regression model reached average $\texttt{auc}$ of at least 0.9019.

Just by looking at the shaded boxes marking the best encoder for the given model in each row, we can see that the best overall encoders are the CENG and the Weight of Evidence Encoders. However, in absolute average the best encoder over all models was the CENG Encoder ($v_E(D) = 0.8437$) followed by the Aging ($v_E(D) = 0.8242$) and Genetic ($v_E(D) = 0.8164$) Pattern-Preserving Encoders.

In Table 4.9 we can see the performance of models that do not require encoding on this dataset. Although the CatBoost Classifier and C5.0 have reasonably good performance, none of them is better than the combination of Linear Regression with any encoder, barring the Entity Embedding Encoder. Incidentally, the CatBoost Classifier and C5.0 have very similar performance to that of any encoder combined with Random Forest, which should not be a surprise knowing that both are tree-based algorithms.

The place of the CENG Encoder as the best over all the models should be contrasted with its time requirement. It took about 1 hour and 54 minutes to complete its evaluation while the second best encoder over all models, the Aging Pattern-Preserving Encoder, took less than 5 minutes. At this point and looking towards scalability we can start to weight in the time requirement on the choice of encoder.

Model	Performance (auc)
Mixed Naïve Bayes	0.76760
CatBoost Classifier	0.87559
C5.0	0.85721

Table 4.9: Performance of models that do not require encoding on the Credit Card dataset.

Model\Encoder	Ordinal	Target	WOE	OneHot	BwDifference	Polynomial	Helmert	EntityEmb	AgingPP	CESAMO	SimplePP	GeneticPP	CENG
Naïve Bayes	0.7397	0.7304	0.7873	0.8114	0.7769	0.7898	0.7776	0.6341	0.7916	0.7449	0.7393	0.7692	0.7765
Linear Regression	0.9231	0.9247	0.9449	0.9224	0.9233	0.9204	0.9083	0.7022	0.9283	0.9241	0.9019	0.9193	0.9222
Logistic Regression	0.7949	0.8529	0.8873	0.8578	0.8526	0.8631	0.8353	0.6637	0.8203	0.7263	0.7795	0.7908	0.8631
Linear SVM	0.6670	0.6905	0.6817	0.8007	0.7974	0.6647	0.7315	0.6655	0.8254	0.7359	0.8276	0.8146	0.8606
Radial SVM	0.6033	0.6183	0.6136	0.6118	0.6161	0.6230	0.6206	0.5000	0.8220	0.7373	0.8129	0.8159	0.8624
K-Neighbours	0.6051	0.6451	0.6532	0.6294	0.6615	0.6545	0.6622	0.6545	0.7778	0.6810	0.6790	0.7799	0.8384
Random Forest	0.8856	0.8750	0.8948	0.8699	0.8808	0.8607	0.8522	0.8249	0.8735	0.8563	0.8371	0.8647	0.8441
Neural Network	0.6550	0.6267	0.7041	0.6870	0.6701	0.6685	0.6741	0.5074	0.7546	0.6522	0.7626	0.7767	0.7826
Average ($v_E(D)$)	0.7342	0.7454	0.7709	0.7738	0.7723	0.7556	0.7577	0.6440	0.8242	0.7572	0.7925	0.8164	0.8437
Time	8.6s	10.2s	11.2s	12.9s	13s	13.3s	13.6s	1m 7s	4m 47s	5m 18s	7m 41s	14m 21s	1h 54m 50s

Table 4.10: Comparison of encoders on the Credit Card Dataset; values are `auc`

4.5 German Credit Dataset

The purpose of this dataset is to classify customers as *good* or *bad* credit risks. For the particular meaning of the variables in this dataset see [22] at the UCI Datasets Archive. This dataset is known for being challenging, due to it containing several predictors with many of them categorical.

The file we use is a preprocessed version of the original and it is included in the repository for this work [15] as the file 'germancredit.csv'. **The file consists of 1000 entries, a 0–1 target variable** (good vs bad credit risk). **There are 20 predictor variables, 8 are numerical and 13 are categorical**. Several of the latter have only two values (e.g. 'yes' and 'no' for being a foreign worker) so at the end **there are a total of 54 categorical instances**.

Discussion

The results of the evaluation of encoders on the German Credit Dataset are in Table 4.12. The overall performance of encoders and models is lower than it was for the previous dataset, which was expected knowing this dataset is challenging.

The best combination of encoder and model is that of the One Hot Encoder and Linear Regression ($\texttt{auc} = 0.7924$). Furthermore, the One Hot Encoder was better across all models ($v_E(D) = 0.6190$) than several of the other encoders; only the Simple and the Genetic Pattern-Preserving and the CENG encoders performed overall better, if only by little. We put forward the hypothesis/explanation that the relative success of the One Hot Encoder in this case is due to 12 of the 13 categorical predictors having at most 5 unique values.

Note that the Entity Embedding Encoder had poor performance overall on this dataset [7],

The results of applying models that do not require encoding on this dataset are in Table 4.11. All of these models had comparable performance to that of several good combinations of encoders and

[7] We suspect either a bug or inadecuacy of the current design of the Entity Embedding Encoder for this dataset, as it had such a poor performance, actually comparable to random uniform classification. We addressed this issue repeating the experiment and trying other tests, but got essentially the same results.

models in Table 4.12. However models in Table 4.11 have much lower performance than the combination of any encoder with Linear Regression.

Model	Performance (auc)
Mixed Naïve Bayes	0.68073
CatBoost Classifier	0.66397
C5.0	0.64715

Table 4.11: Performance of models that do not require encoding on the German Credit Dataset.

Model\Encoder	Ordinal	WOE	Target	OneHot	Helmert	Polynomial	BwDifference	EntityEmb	CESAMO	AgingPP	SimplePP	GeneticPP	CENG
Naïve Bayes	0.6199	0.6985	0.6322	0.7078	0.6971	0.7235	0.6709	0.5000	0.6417	0.6369	0.6807	0.6648	0.6635
Linear Regression	0.7200	0.7842	0.7832	0.7924	0.7760	0.7691	0.7691	0.5702	0.7716	0.7227	0.7593	0.7287	0.7600
Logistic Regression	0.5985	0.6685	0.5690	0.6846	0.6612	0.6548	0.6574	0.5000	0.5502	0.5759	0.6190	0.6154	0.6062
Linear SVM	0.5069	0.5425	0.4995	0.5288	0.5383	0.5058	0.5058	0.5000	0.5725	0.5832	0.6249	0.6138	0.6157
Radial SVM	0.5212	0.5212	0.5212	0.5347	0.5341	0.5329	0.5329	0.5000	0.5060	0.5397	0.6102	0.5791	0.5663
K-Neighbours	0.5232	0.5213	0.5194	0.5387	0.5523	0.5449	0.5449	0.5039	0.5400	0.5364	0.5870	0.6009	0.5744
Random Forest	0.6310	0.6385	0.6441	0.6653	0.6398	0.6326	0.6506	0.68521	0.6821	0.6491	0.6475	0.6438	0.6634
Neural Network	0.5054	0.5000	0.5199	0.5000	0.5056	0.5000	0.5012	0.5000	0.5000	0.5111	0.5036	0.5238	0.5263
Average ($v_E(D)$)	0.5783	0.6093	0.5861	0.6190	0.6130	0.6080	0.6041	0.5307	0.5955	0.5944	0.6290	0.6213	0.6220
Time	8.5s	9.9s	11.2s	12s	13.1s	13.2s	14.6s	1m 16s	2m 41s	6m 6s	10m 16s	20m 24s	4h 36m 14s

Table 4.12: Comparison of encoders on the German Credit Dataset; values are `auc`

4.6 Congress Votes Dataset

This dataset contains votes by US congress members on 16 issues. The task we wish to perform on this dataset is to split the members into two clusters corresponding to the political factions ‘Democrat’ and ‘Republican’. The original labels in the dataset stand exactly for this division. For further information and details on the actual issues voted on see [44] at the UCI Datasets Archive.

The original dataset contains 16 variables corresponding to the 16 votes by 435 congress members, plus one variable for the political party affiliation. There are several unknown values and we decided against removing them as several of them actually point out cases where a member abstained from voting (which we think indicates a political stance too). The preprocessed version we use in this work is available as the file ‘congressvotes.csv’ in the repository for this work [15]. **It consists of 435 rows and 16 categorical variables** (and a binary target variable only used when calculating the Adjusted Mutual Information Index, see page 38). All the 16 variables take only three values, correspoding to‘yes’, ‘no’ and ‘unknown’, **so there are 48 categorical instances in total**.

Its is important to note that all the variables have a few different values and this might preemptively suggest that encoders like the One Hot Encoder work well on this dataset.

Discussion[8]

The results of evaluating encoders on this dataset are in Table 4.14. The missing values correspond to spurious results probably due to instabilities in the process[9]. On each row the best encoder for the given model and metric (Silhouette Coefficient `sil` or Adjusted Mutual Information index `ami`) is shaded.

For the Silhouette Coefficient index: The Aging Pattern-Preserving Encoder consistently obtained

[8] As a demonstration of target leakage, the evaluation of the target encoders on this Congress Votes dataset showed that, as expected, they are frequently better than encoders that do not use the target variable. The target encoder combined with Spectral Clustering reached $\texttt{sil} = 0.5180$ and the Weight of Evidence Encoder combined with Agglomerative Clustering reached $\texttt{ami} = 0.7171$.

[9] This is often present when combining Spectral Clustering with encoders similar to the One Hot Encoder. We believe this occurs because that clustering relies on finding the eigenvalues of an *affinity matrix*, which in presence of sparse, evenly distanced data is itself sparse and also has a large number of repeated entries; under such conditions, finding the eigenvalues is an unstable numerical problem.

the best value across all models (so it got the best overall `sil`, $v_E(D) = 0.4085$). The best combination of encoder and model for this metric was that of this encoder with the K-Means model (`sil`$= 0.4130$). The second best encoder over all models was the CESAMO Encoder (`sil`, $v_E(D) = 0.3710$).

For the Adjusted Mutual Information index: The best combination of encoder and model was that of the Ordinal Encoder and Agglomerative Clustering (`ami`$= 0.6682$), and actually due to this particularly high value that encoder got the best overall `ami` performance ($v_E(D) = 0.5533$). The second best overall encoder was the Simple Pattern-Preserving Encoder (`ami`, $v_E(D) = 0.5405$) and, unlike the Ordinal Encoder, this encoder was similarly good across all the models, .

The result of applying the K-Modes algorithm on this dataset is in Table 4.13, where we can see that this clustering algorithm that does not require encoding had much lower `ami` performance than the combination of any model in Table 4.14 and the Ordinal or the Simple Pattern-Preserving encoders.

Model	Performance (`ami`)
K-Modes	0.4430

Table 4.13: Performance of models that do not require encoding on the Congress Votes dataset.

Model\Encoder		Ordinal	BwDifference	OneHot	Polynomial	Helmert	AgingPP	SimplePP	CESAMO	GeneticPP	CENG
K-Means	sil	0.3536	0.3294	0.3131	0.3131	0.3060	0.4130	0.3457	0.3778	0.3145	0.2881
	ami	0.4985	0.5141	0.4939	0.4939	0.5308	0.4128	0.5403	0.2421	0.4448	0.4506
Spectral	sil	0.3523	0.3287	0.2658	0.2658	0.3646	0.4117	0.3446	0.3662	0.3139	0.2878
	ami	0.4933	0.5141	—	—	—	0.4197	0.5254	0.2769	0.4408	0.4424
Agglomerative	sil	0.3385	0.3205	0.3051	0.3054	0.2920	0.4007	0.3196	0.3689	0.2771	0.2603
	ami	0.6682	0.4134	0.4505	0.4432	0.4773	0.3871	0.5558	0.2125	0.4208	0.3927
Average ($v_E(D)$)	sil	0.3481	0.3262	0.2947	0.2948	0.3209	0.4085	0.3366	0.3710	0.3018	0.2787
	ami	0.5533	0.4805	0.4722	0.4685	0.5040	0.4065	0.5405	0.2438	0.4355	0.4286
Time		2.8s	5s	5s	5.8s	3m	5m 41s	9m 40s	9m 42s	16m 13s	1h 59m 56s

Table 4.14: Comparison of encoders on the Congress Votes dataset

4.7 Sales Dataset

This dataset is another extract of the Individual Company Sales Data [7], used in Chapter 2. This time we picked 8 categorical variables and 1 numerical variable, all standing for socioeconomic information. Our goal is to group entries in 5 clusters that represent levels of education. In the original dataset there is the column 'Education' which has 5 different values and we use it for the ground-truth labels when calculating the Adjusted Mutual Information index.

The extract we use in our experiment is the file 'salesdata_cluster.csv' available in the repository [15]. **It consists of 1447 entries, 8 categorical variables and one numerical variable**. (For the calculation of `ami` we use the 'Education' variable, which has 5 integer values). **There are a total of 23 categorical instances**.

Discussion

Table 4.16 contains the results of the evaluation of encoders on the Sales dataset. As usual, in each row we have shaded the best encoder for the given model.

For the Silhouette Coefficient index: the CESAMO Encoder was consistently the the best for all models ($v_E(D) = 0.4705$, with the second overall best getting a distant $v_E(D) = 0.3193$). This encoder combined with K-Means and Agglomerative Clustering reached considerably higher `sil` (0.5286 and 0.5293, respectively) than any other combination of encoder and model. While lower than the performance of the CESAMO Encoder, all the other Pattern-Preserving Encoder had good and similar performance. All the other encoders had noticeably poor performance.

For the Adjusted Mutual Information index: we straight away can see that across all combinations of encoders and models the clusterings were poor predictors for the 'Education' variable. The maximum `ami` was 0.0517, obtained by the Simple Pattern-Preserving Encoder combined with Spectral Clustering. That all `ami` values are close to zero means that the labels obtained from the clusterings did not match the ground-truth labels better than a random grouping would have done. This suggests that the choice of variables was not adequate to predict the 'Education' variable. Since the `ami`

values are also very close to each other, there is a chance that their differences across encoders are not statistically significant, so we suggest taking the following comparison with caution. The Simple Pattern-Preserving Encoder was the best encoder over all models (ami, $v_E(D) = 0.0452$), followed by the Ordinal Encoder ($v_E(D) = 0.0416$). The Helmert Encoder should also be mentioned because it got the best ami values for K-Means and Agglomerative Clustering (ami $= 0.0504$ and 0.0472, respectively). In Table 4.15 we can see that the K-Modes algorithm got ami $= 0.0293$, which is considerably lower than several of the combinations of encoders and models.

Model	Performance (ami)
K-Modes	0.0293

Table 4.15: Performance of models that do not require encoding on the Sales dataset.

Model\Encoder		Ordinal	BwDifference	Polynomial	OneHot	AgingPP	CESAMO	SimplePP	GeneticPP	Helmert	CENG
K-Means	sil	0.2833	0.1913	0.1625	0.1625	0.3217	0.5286	0.3208	0.3047	0.3017	0.3027
	ami	0.0458	0.0439	0.0279	0.0279	0.0340	0.0390	0.0420	0.0220	0.0504	0.0294
Spectral	sil	0.3303	0.2019	0.1432	0.1432	0.3129	0.3537	0.2592	0.2832	0.2613	0.2736
	ami	0.0418	0.0416	0.0382	0.0382	0.0296	0.0372	0.0517	0.0235	0.0206	0.0274
Agglomerative	sil	0.2157	0.1600	0.1277	0.1277	0.3234	0.5293	0.3076	0.3006	0.2976	0.2822
	ami	0.0373	0.0429	0.0232	0.0232	0.0308	0.0394	0.0419	0.0213	0.0472	0.0386
Average ($v_E(D)$)	sil	0.2764	0.1844	0.1445	0.1445	0.3193	0.4705	0.2959	0.2962	0.2869	0.2862
	ami	0.0416	0.0428	0.0298	0.0298	0.0315	0.0385	0.0452	0.0223	0.0394	0.0318
Time		12.2s	13s	13.5s	13.7s	5m 9s	7m 31s	8m 56s	16m 43s	20m 58s	1h 12m 7s

Table 4.16: Comparison of encoders on the Sales dataset

4.8 Mushrooms Dataset

This dataset contains records for the characteristics of mushrooms and the original task was to predict if a given mushroom is edible or poisonous. We use this dataset for clustering and naturally consider grouping into two clusters. For details on the characteristics of mushrooms collected and the names of variables see [27] at the UCI Dataset Archive.

We employ a slightly smaller and preprocessed version of the original dataset, included in [15] as the file 'mushrooms.csv'. **This file contains 3898 entries and 20 categorical variables, with a total of 97 categorical instances**. When calculating the Adjusted Mutual Information index we also include the binary target variable for 'Edible'. The challenge of this dataset is that it is considerably larger than previous datasets and all variables are categorical.

Discussion

The evaluation of encoders on this dataset gave the results in Table 4.18. As for the Congress Votes dataset some values are missing due to numerical instabilities. Since this dataset is considerably larger than previous ones, we used only 1000 entries when encoding with the CENG Encoder.

For the Silhouette Coefficient index: the best combination of encoder and model was that of the CESAMO Encoder and Agglomerative Clustering ($\texttt{sil} = 0.4582$). The best encoder over all models was in fact the CESAMO Encoder ($v_E(D) = 0.3778$) followed by the CENG Encoder ($v_E(D) = 0.3620$). For this index, the remaining pattern-preserving encoders had very similar performance accross all models. The performance of any of the well-known encoders (first five from left-to-right in Table 4.18 was in general poorer than that of the pattern-preserving encoders.

For the Adjusted Mutual Information index: the best result was obtained by the One Hot and the Polynomial encoders combined with K-Means (a tie at $\texttt{ami} = 0.5224$). It should be noted that the CESAMO Encoder with K-Means reached $\texttt{ami} = 0.5121$ and the Simple, the Genetic Pattern-Preserving encoders and the CENG Encoder obtained $\texttt{ami}$ higher than 0.5100 when coupled with

Spectral Clustering, all values fairly close to the best mentioned above. In fact, the Genetic Pattern-Preserving Encoder was very consistent accross all the models and was hence the best overall encoder ($v_E(D) = 0.5072$). The second best overall encoder was the CESAMO Encoder ($v_E(D) = 0.4816$).

The `ami` performance of the K-Modes algorithm, in Table 4.17, is considerably lower than many combinations of encoders and models in Table 4.18, and particularly lower than that of the Genetic Pattern-Preserving Encoder with any of the models.

The performance of the CENG Encoder, under both indexes, was not particularly poor; however, this case suggests that the benefits of employing this encoder can be debated in sight of its large time requirements. Note that even when we use only 1000 entries out of the 3898 available, the CENG Encoder took several times longer than any other encoder. Perhaps the evaluation of the CENG Encoder could improve if all the 3898 entries are used, but estimates and partial experiments we performed suggest that that would take at least 30 hours to complete[10].

Model	Performance (ami)
K-Modes	0.4185

Table 4.17: Performance of models that do not require encoding on the Mushrooms dataset.

[10]With our available AMD Ryzen-5 U3500 CPU.

Model\Encoder		Helmert	BwDifference	Ordinal	OneHot	Polynomial	CESAMO	AgingPP	SimplePP	GeneticPP	CENG
K-Means	sil	0.1993	0.2073	0.1843	0.2396	0.2396	0.3173	0.3006	0.2857	0.2940	0.3601
	ami	0.5167	0.5064	—	0.5224	0.5224	0.5121	0.3754	0.3904	0.5040	0.4005
Spectral	sil	0.1944	0.3342	0.3732	0.1832	0.1832	0.3580	0.2981	0.2842	0.2938	0.3659
	ami	0.0019	—	—	—	—	0.4234	0.3449	0.5175	0.5106	0.5145
Agglomerative	sil	0.1993	0.2073	0.1693	0.2390	0.2390	0.4582	0.3001	0.2849	0.2935	0.3601
	ami	0.5167	0.5064	0.4629	0.5167	0.5167	0.5093	0.3848	0.3991	0.5071	0.3999
Average ($v_E(D)$)	sil	0.1977	0.2496	0.2423	0.2206	0.2206	0.3778	0.2996	0.2849	0.2938	0.3620
	ami	0.3451	0.3382	—	0.3486	0.3486	0.4816	0.3684	0.4357	0.5072	0.4383
Time		58.4s	1m 3s	1m 13s	4m 7s	4m 8s	7m 18s	10m 17s	22m 38s	35m 34s	6h 20m 32s

Table 4.18: Comparison of encoders on the Mushrooms dataset

Whilst each discussion in this chapter helps pointing out the best encoder for a given task, we will concentrate our conclusions and reccomendations in Chapter 6.

Chapter 5

The power of categorical encoding

This chapter contains a couple of experiments on the application of categorical encoding in machine learning. In both experiments we have a view towards scalability.

5.1 On a previous experiment on entity embeddings

This section is entirely inspired on the work on entity embeddings in [21][1]. In Section 2.3.6 we described how our implementation of the Entity Embedding Encoder is based on ideas from that paper.

Goals

Our aims for this experiment are the following.

(I) Compare some of the encoders from the previous chapters on a large dataset.

(II) Test further the advantages of entity embeddings; particularly, we wish to test the hypothesis that the inclusion of entity embeddings (EE layers in Section 2.3.6 and from now on) improves the results of neural networks.

[1] Preprint as of July 2020

The experiment

We consider the large dataset `train.csv` from the Kaggle Rossmann Store Sales Competition. This dataset contains the details of daily sales of 1115 Rossmann Stores from January 2013 to July 2015. The original data and further details are available in [11]. The main task for this dataset is to predict the daily (future) sales for each store.

Because of (II) above, we preprocessed the original dataset following the indications in [21, Section VI], even including data about the location of stores[2]. Table 5.1 lists the predictor variables we use, which coincide with the ones selected in [21, Table I]; unlike in the latter, we consider that variables like 'Day of the week' and 'Month' are not ordinal but purely nominal categorical variables (since the data ranges across several weeks and months, we believe the real-life ordering of their values might not actually hold for the data or even might not make much sense). There is a total of 1174 categorical instances, with the Store variable having most of them.

Variable	Type	No of values
Store	Categorical	1115
Day of the week	Categorical	7
Day	Categorical	31
Month	Categorical	12
Year	Categorical	3
Promotion	Numerical	2
State	Categorical	6

Table 5.1: Predictor variables chosen from the Rossmann Stores dataset

The authors of [21] considered both shuffled and unshuffled splits for training and test data, showing that the latter led to better results consistently. We only consider an unshuffled split. The original dataset contains 1,017,210 records and following [21, Section VI], we first remove all records for

[2]Available at `https://www.kaggle.com/c/rossmann-store-sales/forums/t/17048/putting-stores-on-the-map`

which the value of sales equals 0, then split into two parts with 90% and 10% of the data, respectively, preserving the cronological order. As training data we take 200,000 records from the 90% part and for test data we take the whole 10% part (84434 records).

The only difference we introduce with the handling of the data in [21] is that instead of using the logarithm of the sales variable, $\log(Sales)$, we use $\log(Sales)/log(max(Sales))$ as target variable. Our choice is in line with the scaling of all variables to the interval $[0, 1]$ needed for some encoders. When comparing our results to the ones in [21, Section VI, B] we should have in mind that they differ in range, but the order comparisons should be unaffected.

The predictive models we consider are listed in Table 5.2. The training parameters are set to be exactly the same as in [21, Table II], except that objective=reg:linear was officially deprecated in favour of objective=reg:squarederror by the Xgboost developers.

For neural networks we consider two architectures. The first, Architecture A, is for already encoded data and consists of and input layer of the required size followed by two dense layers with ReLU activation and 1000 and 500 neurons, respectively, and a final output neuron with sigmoid activation. The second architecture, Architecture B, is for the not encoded data, which starts with an input layer for the integer encoding of the data, followed by EE Layers and completed by concatenating Architecture A as final layers. For Architecture B the final predictions are the average prediction over five networks, as done in [21]. The EE layers we use have the following dimensions: Store: size 30 , Day of the week: size 3, Day: size 11, Month: size 4, Year: size 1, and State: size 2, which come from our design of the Entity Embedding Encoder and differ slightly from the manually chosen in [21, TABLE I].

All models were evaluated on the test data using the mean absolute percentage error (`mape`) as done in [21].

The training dataset is large (200,000 records), so the CENG and the Genetic Pattern-Preserving encoders would take an unreasonable time to encode it. In this experiment we include the One Hot,

Model	Implemented from	Training parameters
Xgboost	Xgboost Python API	`num_rounds=3000, max_depth=10, eta=0.02, objective=reg:squarederror, colsample_bytree=0.7, subsample=0.7, metric=mean_absolute_error`
Random Forest	Scikit-Learn	`n_estimators=200, max_depth=35, min_sample_split=2, min_sample_leaf=1`
K-Nearest Neighbours	Scikit-Learn	`n_neighbors=10, weights=distance, p=1`
Neural Networks	Tensforflow & Keras	`epochs=10, optimizer=adam, loss=mean_absolute_error`

Table 5.2: Models applied to the Rossmann Stores dataset

the Target, the Aging Pattern-Preserving, the CESAMO and the Entity Embedding encoders (the latter is disguised as Architecture B above).

Results

The results of our experiment are in Table 5.3.

Lets us first address our Goal (II), comparing the inclusion of entity embeddings to one-hot encoding. In [21] the difference between the performance of a neural network that includes entity embeddings (Architecture B) is only slightly better than the performance of a neural network that does not include such embeddings (Architecture A fed directly with the one-hot encoding of the data). In that paper, the former reached $\mathtt{mape} = 0.093$ and the latter $\mathtt{mape} = 0.101$, a difference of only 0.08. To us it was reasonable to suspect that this difference could be due to chance, i.e. that architectures A and

Model\Encoder	One Hot	Target	CESAMO	AgingPP	Entity Emb
Xgboost	0.02046	0.01460	0.01566	0.01540	0.01298
K-Nearest Neighbours	0.02864	0.01806	0.03359	0.03522	0.01564
Random Forest	0.02920	0.01515	0.01497	0.01643	0.01394
Neural Networks	0.01495 (A)	0.01987	0.03645	0.03550	0.01467 (B)

Table 5.3: Results on the Rossman Stores dataset; values are `mape` on test data

B actually perform equally well, thus minimising the importance of including entity embeddings for neural networks.

In Table 5.3 we see that neural networks with entity embeddings, cell marked with (B), reached $\texttt{mape} = 0.01467$ and neural networks without those embeddings, cell marked with (A), reached $\texttt{mape} = 0.01495$, again a rather small difference[3]. This indicates, as we suspected, that we cannot as of now conclude that neural networks with entity embeddings are superior to neural networks without them. Of course, a comprehensive statistical study should be performed to draw the correct conclusion on this.

Despite our suspicion that entity embeddings do not actually improve the performance of neural networks, we believe they have an important place in encoding of categorical variables, through the Entity Embedding Encoder. Evidence to this is that the application of that encoder led to the best `mape` result for every single model. The Entity Embedding Encoder combined with the Xgboost model got a remarkably low `mape` of 0.01298.

With respect to our Goal (I) of comparing some encoders on a large dataset, we can see in Table 5.3 that the Aging Pattern-Preserving and the CESAMO encoders combined with either the Xgboost or the Random Forest models got competitive `mape` values. Those two encoders performed very similar on each model. Also, we should not ignore the good, consistent performance of the Target Encoder accross all the models.

Yet another time we verify that the One Hot Encoder performs poorly compared to other encoders on most models.

Conclusions

In view of Table 5.3, we gather further evidence that the Entity Embedding, the CESAMO, the Aging Pattern-Preserving and the Target encoders are stable and competitive encoding methods.

[3]The difference of 0.0027 here should translate to a difference of no more than 0.03 in the scale of the results in [21] because $log(max(Sales))$ is about 10.3

The Rossmann Sales dataset and the Automobile MPG dataset from Section 4.3 both have a continuous numerical target variable and for both the Entity Embedding Encoder had remarkable performance across all models. We strongly recommend the use of the Entity Embedding and the Target encoders in regression problems with mixed datasets.

5.2 Categorical Feature Encoding Challenge

In this experiment we gauge the performance of some encoders in the presence of a purposely complicated dataset.

The data we consider for this experiment comes from the 'Categorical Features Encoding Challege', a competition set up by Kaggle Inc in [23] and that asks competitors to develop a classification model for a considerably large dataset with categorical inpedendent variables and a large number of categorical instances. The characteristics of this dataset makes it an important use-case for the application of the encoders in previous chapters.

Goal

The main goal in this experiment is to find the best result possible for the Categorical Features Encoding Challege using some of the encoders in the previous chapters.

The experiment

We use the original data from [23], which consists of a training set, which we call the *Kaggle Challenge* dataset and a test set. Note that the fitting of encoders and the training and validation of models will be done exclusively with the Kaggle Challenge dataset. The test dataset will be only use at the very end of the process to be able to submit our predictions to the kaggle website.

The Kaggle Challenge dataset contains a total of 300,000 records(rows), 23 independent categorical variables and a binary (0–1) target variable. Table 5.4 describes the dataset further. Each variable

name hints at the type of the corresponding variable, e.g. nom_0 is a nomimal variable and ord_0 is an ordinal variable. Nevertheless we ignore this information because, as it has been stated throughout this work, the real-life ordering and/or distance between the values of a categorical variable might not hold in the actual data.

Variable	No of values	Variable	No of values	Variable	No of values
bin_0	2	nom_3	6	ord_1	5
bin_1	2	nom_4	4	ord_2	6
bin_2	2	nom_5	222	ord_3	15
bin_3	2	nom_6	522	ord_4	26
bin_4	2	nom_7	1220	ord_5	192
nom_0	3	nom_8	2215	day	7
nom_1	6	nom_9	11981	month	12
nom_2	6	ord_0	3		

Table 5.4: Variables in the Kaggle Challenge dataset

There is a wide range for the number of unique values of variables and there is a large total of 16,461 categorical instances in the Kaggle Challenge dataset[4]. This number of categorical instances diminishes the plausibility of applying the popular One Hot Encoder. Training models with a sparse numerical dataset with 300,000 rows and 16,461 columns puts pressure on computational resources (memory-wise particularly) and leads to poor performance due to numerical instabilities. The alternatives are (1) limit our choice to models that are designed to deal with sparse data and/or to be trained by batches or (2) directly employ other categorical encoders to avoid such a large and sparse dataset. We will adhere to (2).

For this experiment we chose the Aging and the Genetic Pattern-Preserving, the CESAMO, the Entity Embedding, the Target and the Weight of Evidence(WOE) encoders. The first two encoders were fitted with 30% of the data, the third was fitted with 50% of the data and the latter three were

[4]For perspective, the first implementations of the CENG and CESAMO encoders could only handle a maximum of 200 categorical instances.

fitted with the full data. We did so to keep the process under reasonable time and computational resources usage[5].

For each encoder, once the whole Kaggle Challenge dataset was encoded, we proceeded to find the best Xgboost model to predict the target variable. Our aim here was to obtain the best possible area-under-roc-curve(`auc`) value, so for each encoded dataset we perform model selection by optimising the parameters of the Xgboost model. The Xgboost framework used was provided by the Xgboost Python API[6], and the parameter optimisation was performed using the Bayesian Optimisation framework Hyperopt[7]. The code (in Jupyter notebooks) for the encoding of the dataset and the training and selection of models is available in the repository [18].

Results

Table 5.5 contains the results of the best Xgboost model found for each encoder. In the second and third columns we report training and validation performance, respectively, where 75% of the 300,000 records was used for training and the remaining 25% for validation.

The set 'test.csv' mentioned earlier has the same variables as 'train.csv' except that the target values are not included. Only the competition organisers at Kaggle have access to the actual target values for the 'test.csv' dataset. Participants were asked to make predictions on this dataset to submit as competition entries, and the competition website [23] would in turn report the `auc` score reached. In the fourth column of Table 5.5 we report the `auc` score on the 'test.csv' as reported by the competition website.

From the second and third columns in Table 5.5 we can see that the pattern-preserving encoders allowed significant overfitting, with a large gap between training and validation `auc` scores. Among these encoders, while the Aging Pattern-Preserving Encoder had better validation score (`auc` =

[5] Even this way, for instance, encoding 30% of the data with the Aging Pattern-Preserving encoder took about 5 hours, and encoding the full dataset with the Entity Embedding Encoder took about 14 minutes.
[6] Details are available at `https://xgboost.readthedocs.io/en/latest/python/index.html`
[7] Available at `https://github.com/hyperopt/hyperopt`.

Encoder	Training `auc`	Validation `auc`	Test `auc`
AgingPP	0.847602	0.780741	0.78463
CESAMO	0.873787	0.773439	0.77596
Entity Emb	0.866985	0.822120	0.79342
GeneticPP	0.858948	0.774878	0.77889
Target	0.840344	0.830170	0.78677
WOE	0.839698	0.830302	0.79065

Table 5.5: Performance of an optimised Xgboost model for each encoder on the Kaggle Challenge data.

0.780741), the Genetic Pattern-Preserving Encoder actually reached a higher score on the test dataset (`auc` = 0.77889). The CESAMO Encoder had poorer performance overall.

The Target and the Weight of Evidence encoders performed in a very similar way, with a small gap between training and validation `auc` scores, significantly high validation `auc` scores, and very competitive scores on the test data. On test data, the Weight of Evidence Encoder reached an `auc` of 0.79065 beating the 0.78677 score of the Target Encoder.

Although the gap between training and validation `auc` scores for the Entity Embedding Encoder is larger than that for the Target and WOE encoders, it still generalised better to the test dataset, reaching the best overall test `auc` score of 0.79342.

It is important to note that the palpable gap between the validation and test `auc` scores for the Entity Embedding, the Target and the Weight of Evidence encoders is most likely an example of target leakage, since these encoders are all target encoders and used both the training and validation data to be fitted. Further evidence is that all of the other encoders, which are not target encoders, showed very similar `auc` scores in validation and test data.

To put the results in Table 5.5 in perspective, the winning entry of the Kaggle competition reached an `auc` score of 0.80283 on test data. While not all competitors published the code of their work, many shared solutions include feature engineering and/or some crafty model design. Our results were obtained without any of such detailed work, employing only out-of-the-box encoders and some parameter optimisation.

Conclusions

In this particular experiment we had the opportunity of seeing how target encoders like the Target and the Weight of Evidence encoders suffer from target leakage and should be used cautiously. On the other hand, for pattern-preserving encoders we should introduce stronger model selection to avoid overfitting.

We have seen further competitive performance by the Entity Embedding Encoder and we fully endorsed it to be considered as a solid categorical encoder.

We have seen again that pattern-preserving encoders allow for good performace of machine learning models. However, facing this further example, we acknowledge that other encoders perform better and require fractions of the time and computational resources that the pattern-preserving encoders need.

Chapter 6

Conclusions

We offer our conclusions in terms of how our work covered our objectives in Section 1.5.1.

1. *Objective: Show that categorical encoding has a strong positive impact on the performance of machine learning models.*

   The experiments in Chapter 4 show the advantages that categorical encoding offers. For each dataset in Chapter 4 we showed that at least for one categorical encoder, and often for many, the results of classification, regression or clustering were significantly better than that of models that do not require encoding.

2. *Objective: Show that common encoding methods, particularly One-Hot encoding, are readily overcome by other methods that are not necesarily more complex.*

   Only for a couple of datasets and learning algorithms the One-Hot Encoder managed to obtain competitive results. In most cases, popular encoders like the One-Hot, Ordinal and Polynomial encoders were overcome by the Target and the Entity-Embedding encoders (the Pattern-Preserving encoders often performed better too but we consider them to be of a more pronounced complexity).

3. *Objective: Identify the encoders that perform better under specific conditions (type of learning, data size, etc), providing a practical guide to their usage.* See below.

4. *Objective: Show that the newer Entity-Embedding and the Pattern-Preserving encoders are sound and competitive categorical encoders.* See below.

For the objectives 3 and 4 we have the following:

- Target encoders, in particular the Target Encoder, are often superior in supervised learning. Caution should still be exercised around target leakage.

- Pattern-Preserving encoders have typically similar performance accross all types of tasks.

- In cases where the categorical variable take only a few values (upto 5), encoders similar to the One Hot Encoder have good performance.

- The CENG Encoder is stable accross all tasks, reaching sometimes very good results, but its excessive time requirement cannot be ignored.

- If a non-target encoder is to be recommended, we choose the CESAMO Encoder and the Aging Pattern-Preserving Encoder. Both were stable and consistently good accross tasks and datasets, and their time requirements are, we believe, reasonable. The Genetic Pattern-Preserving Encoder could also be recommended here, but offers similar performance at a higher computational cost.

- Our implementation of the Entity Embedding Encoder reaches competitive performance with the exception of some cases. A remarkably bad case was seen in the German Credit dataset; perhaps, further fine-tuning of the neural network behind the encoder might improve its performance. Modifications can be made to the size of EE-layers and, more importantly to avoid overfitting, to the number of neurons in the dense layers. Our implementation [17] allows for those values to be set manually and even to add dropout. We wanted to keep close to the general design from [21] so we did not experiment with those values.

A side objective mentioned in Section 1.5.1 was: *Develop implementations of those encoders that are not available yet*. We implemented the Entity-Embedding Encoder, the CESAMO Encoder and

the other Pattern-Preserving Encoders—bar the CENG Encoder; details on the implementations are available in Appendix A and the implementations themselves can be found as the modules [17], [16] and [19], respectively.

6.1 Guide to the usage of encoders

Table 6.1 is a concentrated version of our observations, and can be used as a guide on how to choose an encoder for a given task and dataset.

In Table 6.1, '# Cat vars' stands for the number of categorical variables in the given dataset, 'Largest k' stands for the largest number of categorical instances (unique values) accross all the categorical variables in the dataset and 'n' is simply the total number of rows in the dataset[1].

Task	# Cat vars	Largest k	n	**Recommended Encoder(s)**
Classification	≤ 20	any	$\leq 10,000$	SimplePP, AgingPP
	≤ 20	≤ 5	$\leq 100,000$	One Hot
	any	any	any	Entity Embedding, Target, WOE
Regression	≤ 20	any	$\leq 1,000$	CENG
	≤ 20	≤ 5	$\leq 100,000$	One Hot
	any	any	any	Entity Embedding, Target
Clustering	≤ 20	any	$\leq 1,000$	CENG
	≤ 20	any	$\leq 10,000$	SimplePP, AgingPP
	any	any	any	CESAMO

Table 6.1: Guide to the usage of encoders

[1]Note that little thought is given to the total number of columns m in the dataset. This is so because, except for the CENG Encoder, the encoders appearing in Table 6.1 are not significantly affected by the size of m under practical circumstances, ie whenever $m \leq 100$ (in our opinion it is atypical to work with structured datasets for which $m > 100$; even then we would suggest applying feature engineering to reduce the number of variables first).

Appendix A: On implementations

A.1. Tables: Implementations used in this work

This project was developed using mostly the Python programming language version 3.7.1, and makes extensive use of the following software packages:

- Pandas 1.0.1
- Tensorflow 2.0.0
- Scikit-Learn 0.22.2
- Category Encoders 2.1.0[2]

Further details on the development enviroment used for this project can be found at its repository [15].

Table A.2 gives details on the implementations of the encoders considered in this work. Table A.3 contains details on the implementations of other algorithms used in this work.

Name	Package	Language
Ordinal Encoder	Scikit-Learn 0.22.2	Python
Polynomial Encoder	Category Encoders 2.1.0	Python
One Hot Encoder	Category Encoders 2.1.0	Python
Backward Difference Encoder	Category Encoders 2.1.0	Python
Helmert Encoder	Category Encoders 2.1.0	Python
Entity Embedding Encoder	Own implementation [17]	Python
Target Encoder	Category Encoders 2.1.0	Python
Weight of Evidence Encoder	Category Encoders 2.1.0	Python
CENG Encoder	By A. F. Kuri-Morales	Java
Genetic PP Encoder	Own implementation [19]	Python
Aging PP Encoder	Own implementation [19]	Python
Simple PP Encoder	Own implementation [19]	Python
CESAMO Encoder	Own implementation[3] [16]	Python

Table A.2: Implementations of categorical encoders.

[2]From `https://contrib.scikit-learn.org/categorical-encoding/`.

Name	Package	Interface or source language
Mixed Naïve Bayes	From remykarem[4]	Python
CatBoost	Catboost 0.16	Python
C5.0	C50 R Library 0.1.3	R
K-Modes	Kmodes 0.10.2	Python
Xgboost	Xgboost 1.1.1 Python API	Python

Table A.3: Implementations of other models/algorithms used in this work.

A.2. Handling previously unseen categorical instances

In general, we do not recommend the reuse of a set of codes calculated from a dataset D_1 to encode a second, similar dataset D_2. Even if D_1 and D_2 have exactly the same variables, the distribution of the data in them might not be sufficiently similar.

Nevertheless, in some situations it is useful to be able to reuse a set of codes. For example, in Chapter 4 we split a dataset, encoded one of the parts and steps later we encoded the other part with the same codes. In this case, as the split was random, it is reasonable to assume that both parts have similar data distributions, so it is sensible to reuse codes.

Having in mind that codes could be reused, a technical issue raises. Suppose that we have encoded a dataset D_1 with a set of codes S, and that we wish to use S to encode a second, similar dataset D_2. It is possible (even if both datasets come from a random split of a dataset) that D_2 contains categorical instances that were not present in D_1, so S would not be a complete set of codes for D_2.

[3] The first implementation is due to A. F. Kuri-Morales, which we could not use in our enviroment as it was written in FoxPro 2.0.

[4] Available at https://github.com/remykarem/mixed-naive-bayes

Our approach in this situation is to simply *assign a random code to any new categorical instance in* D_2.[5]

All our implementations of encoders for this work, except that of the Entity Embedding Encoder, include this mechanism.

We could not include the mechanism for the Entity Embedding Encoder because the Embedding layers from the Keras API do not allow previously unseen words to be passed after training. To alleviate this, our implementation allows to pass the optional parameter `test` that takes a dataframe from which new categories expected after training are collected. These are added to the vocabulary for the corresponding EE layer before training.

A.3. On the size of entity embedding layers

In Section 2.3.6 we mentioned that the original paper on entity embeddings [21] gives no general rule on the size of EE layers. Here we argue for our specific formula for this number, namely, if a categorical variable C has $k \geq 1$ unique values, its correspoding EE layer should have size $\varphi(k) = \min(30, \ulcorner k/3 \urcorner)$.

Our choice of φ places a hard upper cap of 30 for any categorical variable with more than 90 unique values. This enforces our need to keep the encoded sets compact, a crucial goal of the Entity Embedding Encoder. On the other hand, for a variable with up to 90 unique values the calculation $\ulcorner k/3 \urcorner$ allows for a large enough EE layer size and, therefore, enough trainable parameters.

Other formulas we have seen in other projects use either the floor or the ceiling of the expressions $k/2$ and $\sqrt{k}$. The former allows the generation of fairly large datasets (which becomes worse if a hard upper cap is not set). The latter, $\sqrt{k}$, gives agressively-low values and that does not allow enough trainable weights to encode the variable. For example, a variable with 90 unique values would either get a size of $90/2 = 45$, which achieves little in reducing the number of columns for the encoded

[5] We could encode new categorical instances with the average of the already known codes, but this would give the same code to every unseen categorical instance of a variable, so in general it is more problematic.

variable, or a size of 9 or 10 (floor or ceiling of $\sqrt{90}$) which lands in the extreme opposite. We believe the range 20–30 is appropriate in this case, and $\varphi(k) = 30$.

Figure A.1 gives a graphical idea of some functions based on the numerical expressions mentioned above. In the design of the Entity Embedding Encoder we assume that $\varphi(k) = \min(30, \ulcorner k/3 \urcorner)$ is the most appropriate choice.

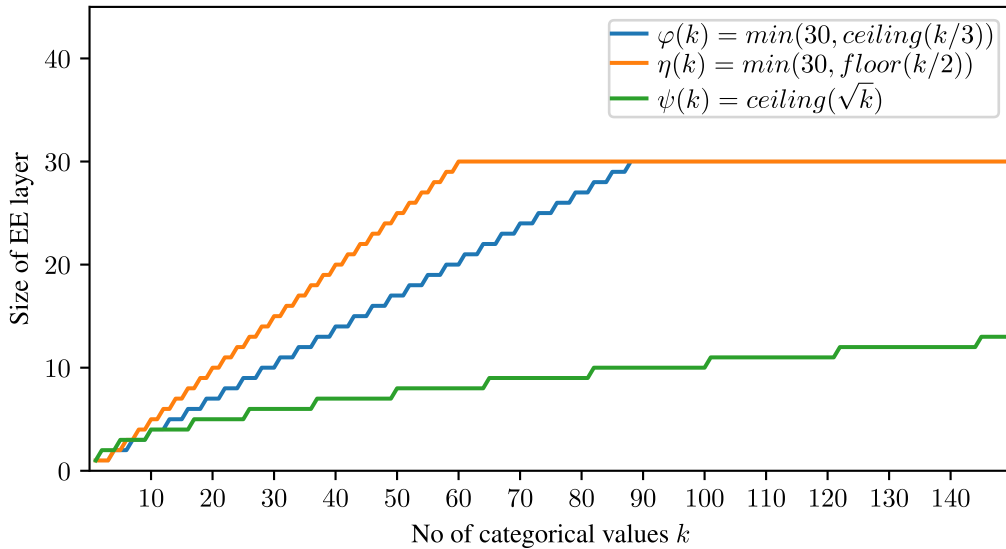

Fig. A.1: Functions considered for the size of EE layers. The Entity Embedding Encoder uses φ.

Nevertheless, our implementation of the Entity Embedding Encoder in [17] allows the parameter `ee_sizes`, which takes a Python dictionary with elements of the form `num_column:number` where `number` stands for the size of the EE layer of the variable in column `num_column`. If passed, the dictionary does not have to set the values for all categorical variables. Those not set by the dictionary are assigned an EE layer of size $\varphi(\text{no of unique values})$. The default value for `ee_sizes` is an empty dictionary and in that case all categorical variable are assigned EE layers of the size given by φ. In all experiments in this work we always used the default value for the `ee_sizes` parameter.

A.4. The genericity of the implementations of the pattern-preserving and the cesamo encoders

Recall that a pattern-preserving encoder is determined by the choices for 1–4 in Definition 3.2.1, and that in consequence there is a multitude of possible encoders of this type. The implementations in [16] and [19] of pattern-preserving encoders cater to this fact. They all have a broad choice for the estimators (Definition 3.2.1 item 2) and all except CESAMOEncoder have a free choice for the number of variables used as estimators (l in Definition 3.2.1 item 1).

Each `Encoder` among `AgingPPEncoder`, `GeneticPPEncoder` and `SimplePPEncoder` has the signature,

```
Encoder(estimator_name=_ , num_predictors=_),
```

and CESAMOEncoder has the signature,

```
CESAMOEncoder(estimator_name=_).
```

For `num_predictors` any integer $l \geq 1$ is allowed (the implementations have a mechanism to take the maximum possible number of predictor variables if l is bigger than the number of independent variables in the dataset minus 1). Table A.4 shows the strings `estimator_name` accepts.

Parameter string	Description
'LinearRegression'	Ordinary linear regression
'LogisticRegression'	Ordinary logistic regression
'SGDRegressor'	Linear regression w/ optimised training
'SVR'	Support vectors machine regression
'PolynomialRegression'	Polynomial regression w/ max degree 3
'Perceptron'	Perceptron w/ max_iter=150, hidden_layers_size = (10,5)
'CESAMORegression'	Polynomial regression with bias $\neq 0$, zero even terms and max degree 11

Table A.4: Strings allowed for `estimator_name`

The default values of `num_predictors` and `estimator_name` correspond exactly to the encoders as described in Chapter 3. For example, `GeneticPPEncoder()` (no parameters passed) is equivalent to

```
GeneticPPEncoder(estimator_name='PolynomialRegression',
                 num_predictors=2),
```

and `CESAMOEncoder()` is equivalent to

```
CESAMOEncoder(estimator_name='CESAMORegression').
```

Note that if the mixed dataset D has m independent variables, then

```
GeneticPPEncoder(estimator_name='Perceptron', num_predictors=m-1)
```

is simply another implementation of the CENG Encoder for D.

References

[1] H. Abdi and L. J. Williams. Contrast analysis. In N. Salkind, editor, *Encylopedia of Reseach Design*. Sage, 2010.

[2] E. Arisoy, T. Sainath, B. Kingsbury, and B. Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-Gram Model? On the Future of Language Modeling for HLT*, WLM '12, pages 20–28, USA, 2012. Association for Computational Linguistics.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[4] D. Bhalla and ListenData.com. *Weight of Evidence (WOE) and Information Value (IV) Explained*. Available at https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html, 2015.

[5] D. Bhalla and ListenData.com. *Weight of Evidence and Information Value for Continuous Dependent Variable*. Available at https://www.listendata.com/2019/08/WOE-IV-Continuous-Dependent.html, 2019.

[6] G. Carey. *Quantitative Methods in Neuroscience*. Disponible en http://psych.colorado.edu/~carey/qmin.php, 2013.

[7] G. Chang and Kaggle user:Mickey1968. *Individual Company Sales Data*. Available at https://kaggle.com/mickey1968/individual-company-sales-data, 2018.

[8] E. Cheney. *Introduction to approximation theory*. AMS Chelsea Publishing, 1996.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems*, 1:303–314, 1989.

[10] R. D'Agostino and E. S. Pearson. Tests for departure from normality. Empirical results for the distributions of b_2 and $\sqrt{b_1}$. *Biometrika*, 60(3):613–622, 1973.

[11] Dirk Rossmann GmbH (donor). Rossmann store sales. Available at https://www.kaggle.com/c/rossmann-store-sales, 2015.

[12] UCLA: Institute for Digital Research & Education. *R Library Contrast Coding Systems for categorical variables*. Available at https://stats.idre.ucla.edu/r/library/r-library-contrast-coding-systems-for-categorical-variables, 2011.

[13] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[14] R. Galindo-Hernández. Determinación estadística de códigos numéricos en bases de datos mixtas. Master's thesis, Universidad Nacional Autónoma de México, 2019.

[15] E. García-Ramírez. *GitHub Repository: categorical_encoding*. Available at https://github.com/erickgrm/categorical_encoding, 2020.

[16] E. García-Ramírez. *GitHub Repository: cesamo-encoder*. Available at https://github.com/erickgrm/cesamo-encoder, 2020.

[17] E. García-Ramírez. *GitHub Repository: entity-embedding-encoder*. Available at https://github.com/erickgrm/entity-embedding-encoder, 2020.

[18] E. García-Ramírez. *GitHub Repository: Kaggle Categorical Features Challenge*. Available at https://github.com/erickgrm/kaggle-categorical-features-challenge, 2020.

[19] E. García-Ramírez. *GitHub Repository: pattern-preserving-encoders*. Available at https://github.com/erickgrm/pattern-preserving-encoders, 2020.

[20] A. Gulin and the CatBoost Team. *Catboost Project*. Available at https://www.catboost.ai, 2020.

[21] C. Guo and F. Berkhahr. Entity embeddings of categorical variables. *Available at: https://arxiv.org/abs/1604.06737*, 2016.

[22] H. Hofmann. German credit data. https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data), 1994. University of California, Irvine, School of Information and Computer Sciences.

[23] Kaggle Inc. Categorical feature encoding challenge (binary classification, with every feature a categorical). Available at https://www.kaggle.com/c/cat-in-the-dat, 2019.

[24] A. Jain and R. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988.

[25] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning. With applications in R*. Springer, 2013.

[26] S. Kaufman, S. Rosset, and C. Perlich. Leakage in data mining: formulation, detection, and avoidance. In *In Proceedings of the 2011 conference on Knowledge Discovery in Data Mining*, pages 556–563, 2011.

[27] A. Knopf and J. Schlimmer(donor). Mushroom data set. https://archive.ics.uci.edu/ml/datasets/Mushroom, 1987. University of California, Irvine, School of Information and Computer Sciences.

[28] A. Kuri-Morales. Closed determination of the number of neurons in the hidden layer of a multilayered perceptron network. *Soft Computing*, 21:597–609, 2017.

[29] A. F. Kuri-Morales. Categorical encoding with neural network and genetic algorithms. In X. Zhuang and C. Guarnaccia, editors, *Proceedings of the 6th International Conference on Applied Information and Computation Theory*, pages 167–175. WSEAS, 2015.

[30] A. F. Kuri-Morales. Pattern discovery in mixed data bases. In J. Martínez-Trinidad and et. al., editors, *Mexican Conference in Pattern Recognition 2018*, Lecture Notes in Computer Science. Springer, 2018.

[31] A. F. Kuri-Morales. Minimum database determination and preprocessing for machine learning. In L. Zhang and Y. Ning, editors, *Innovative Solutions and Applications of Web Services Technology*. IGI Global, 2019.

[32] A. F. Kuri-Morales and E. Aldana-Bobadilla. The best genetic algorithm i: A comparative study of structurally different genetic algorithms. In F. Castro, A. Gelbukh, and M. González-Mendoza, editors, *Advances in Soft Computing and its Applications: Part II*, Lecture Notes in Artificial Intelligence, 8266, pages 580–594. Springer, 2013.

[33] A. F. Kuri-Morales, E. Aldana-Bobadilla, and I. López-Peña. The best genetic algorithm ii: A comparative study of structurally different genetic algorithms. In F. Castro, A. Gelbukh, and M. González-Mendoza, editors, *Advances in Soft Computing and its Applications: Part II*, Lecture Notes in Artificial Intelligence, 8266, pages 594–608. Springer, 2013.

[34] A. F. Kuri-Morales and J. Sagastuy-Breña. A parallel genetic algorithm for pattern recognition in mixed databases. In *Proceedings of the Mexican Conference on Pattern Recognition*, pages 13–21. Springer, 2017.

[35] K. Larsen and D. Becker. Seven types of target leakage in machine learning and an exercise. In *Automated Machine Learning for Business*. Oxford University Press, 2019.

[36] D. Micci-Berreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explorations*, 3(1):27–32, 2001.

[37] J. Neter, M. Kutner, C. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. Irwin, Times Mirror Higher Education Group Inc., 4th edition, 1996.

[38] R. Lyman Ott and M. Longnecker. *Statistical Methods and Data Analysis*. Duxbury, Thomson Learning, 5th edition, 2001.

[39] A. Parellada and the stats.stackexchange Community. *Polynomial contrasts for regression*. Available at `https://stats.stackexchange.com/questions/105115/polynomial-contrasts-for-regression`, 2016.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] E. Ramos and D. Donoho. Auto mpg data set. `https://archive.ics.uci.edu/ml/datasets/auto+mpg`, 1983. University of California, Irvine, School of Information and Computer Sciences.

[42] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789. AAAI Press, 2019.

[43] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[44] J. Schlimmer. Congressional voting records data set. `https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`, 1987. University of California, Irvine, School of Information and Computer Sciences.

[45] H. Seltman. *Experimental Design and Analysis*. Available at `www.stat.cmu.edu/~hseltman/309/Book/`, 2018.

[46] Confidential source (submitted by R. Quinlan). Credit approval dataset. https://archive.ics.uci.edu/ml/datasets/Credit+Approval, 1987. University of California, Irvine, School of Information and Computer Sciences.

[47] N. X. Vinh and J. Epps. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.

[48] M. Zwitter and M. Soklic. Breast cancer data set. https://archive.ics.uci.edu/ml/datasets/breast+cancer, 1988. University of California, Irvine, School of Information and Computer Sciences.

Index

A

`ami` 39
Analysis of variance 11
ANOVA 11
`auc` 37

C

C5.0 40
CatBoost 6
- Classifier 40
- Regressor 40

Categorical
- Encoder 4
- Encoding 3
- Instance 9
- Variable 1

Classification 37
Clustering 38
- Agglomerative 39
- Spectral 39

Code 9
Contrast 10
- Encoders 10
- in ANOVA 11
- orthogonal 10

D

$D(S)$ 9
Dataset
- Automobile MPG 44
- Breast Cancer 41
- Congress Votes 53
- Credit Card 47
- German Credit 50
- Kaggle Challenge 68
- Mixed 2
- Mushrooms 59
- Numerical 2
- Rossmann Stores 64
- Sales 56

Decision Tree 6

E

EE layer 79
Embedding layer 18, 79
Encoder 4
- Aging PP 5, 31
- Backward Difference 4, 15
- CENG 5, 28
- CESAMO 5, 33
- Entity Embedding 4, 18
- Genetic PP 5, 30
- Helmert 4, 17
- One Hot 4, 14
- Ordinal 4, 13
- Pattern-Preserving 27
- performance 36
- Polynomial 4, 14
- Simple PP 5, 32
- Target 5, 21
- Weight of Evidence 5, 22

Encoding 3

H

Hyperopt 70

K

K–Means 39
K-Modes 40

K-Neighbours 37

L

Latent space encoders 11

M

mape 65

mse 38

N

Naïve Bayes 5, 37

- Mixed 40

Neural Network 18, 37

O

Ordinal encoders 10

R

Random Forest 6, 37

Regression 38

- Linear 37
- Logistic 37

S

Sales-Features 12

Sales-Target 12

Set of codes 9

- fitness of 26

sil 39

Support Vector Machine

- Linear 37
- Radial 37

T

Target encoders 12

Target leakage 23

V

Variable

- Categorical 1
- Continuous 1
- Discrete 1
- Nominal 2
- Numerical 1
- Ordinal 2, 4

X

Xgboost 65