Encoding high-cardinality string categorical variables

Patricio Cerda and Gaël Varoquaux

Abstract—Statistical analysis usually requires a vector representation of categorical variables, using for instance *one-hot encoding*. This encoding strategy is not practical when the number of different categories grows, as it creates high-dimensional feature vectors. Additionally, the corresponding entries in the raw data are often represented as strings, that have additional information not captured by one-hot encoding.

Here, we seek low-dimensional vectorial encoding of string categorical variables with high-cardinality. Ideally, these should *i)* be scalable to a very large number of categories, *ii)* be interpretable to the end user, and *iii)* facilitate statistical analysis. We introduce two new encoding approaches for string categories: a *Gamma-Poisson matrix factorization* on character-level substring counts, and a *min-hash encoder*, based on min-wise random permutations for fast approximation of the Jaccard similarity between strings. Both approaches are scalable and are suitable for streaming settings. Extensive experiments on real and simulated data show that these encoding methods improve prediction performance for real-life supervised-learning problems with high-cardinality string categorical variables and works as well as standard approaches with clean, low-cardinality ones. We recommend the following: *i)* if scalability is the main concern, the min-hash encoder is the best option as it does not require any fitting to the data; *ii)* if interpretability is important, the Gamma-Poisson factorization is a good alternative, as it can be interpreted as one-hot encoding, giving each encoding dimension a feature name that summarizes the substrings captured. Both models remove the need for hand-crafting features and data cleaning of string columns in databases and can be used for feature engineering in online autoML settings.

1 Introduction

Common tabular datasets often contain columns with string entries. However, fitting statistical models on such data generally requires a numerical representation of all entries, which calls for building an *encoding*, or vector representation of the entries. Considering string entries as nominal –unordered– categories is convenient for their statistical analysis. In such situations, categories are supposed to be mutually exclusive and unrelated, with a fixed known set of possible values. Yet, in many real-world datasets, the entries of string columns are not standardized in a small number of categories. This poses challenges for statistical learning. First, the set of all possible categories may be huge and not known a priori, as the number of different strings in the column can indefinitely increase with the number of samples. Second, categories may not be unrelated: they often carry some morphological or semantical links.

The classic approach to encode categorical variables in a statistical analysis is *one-hot encoding*: creating vectors that agree with the general intuition of nominal categories: orthogonal and equidistant [1]. For high-cardinality categories, one-hot encoding leads to feature vectors of high dimensionality. This is especially problematic in big data settings, which can lead to a very large number of categories, posing computational and statistical problems.

Data engineering practices typically tackle these issues with data-cleaning techniques [2], [3]. In particular, deduplication tries to merge different variants of the same entity [4], [5]. However, data cleaning often requires human intervention, and is one of the major hurdles to data analysis[1]. To avoid the cleaning step, *Similarity encoding* [6] relaxes one-hot encoding by using *string similarities* [7]. Hence, it addresses the problem of related categories and has been shown to improve statistical analysis upon one-hot encoding [6]. Yet, it does not tackle the problem of high cardinality, and the data analyst much resort to heuristics such as choosing a subset of the training categories [6].

Here, we seek an encoding approach for statistical analysis on string categorical data that is suited to a very large number of categories without any human intervention. Our goals are: *i)* to provide feature vectors of limited dimensionality without any cleaning step, even for very large datasets; *ii)* to improve statistical analysis tasks such as supervised learning; and *iii)* to preserve the intuitions behind categories: entries can be arranged in natural groups that can be easily interpreted. We study two novel encoding methods that both address scalability and statistical performance: a *min-hash encoder*, based on locality-sensitive hashing (LSH) [8], and a low-rank model of co-occurrences in character n-grams: a *Gamma-Poisson matrix factorization*, suited to counting statistics. Both models scale linearly with the number of samples and are suitable for statistical analysis in streaming settings. Moreover, we show that the Gamma-Poisson factorization model enable interpretability by with a sparse encoding that expresses the entries of the data as linear combinations of a small number of latent categories, built from their substring information. This last point is very important as lack of interpretability of black-box machine learning slows their adoption in real-world applications. Often, practitioners resort to manual data cleaning to regain interpretability of the model. An empirical study on 17 real-life datasets demonstrates that our contributed encoding methods enable analysis of non curated data with better

- *Inria, Parietal team*
  E-mail: patricio.cerda@inria.fr

[1] https://www.kaggle.com/surveys/2017

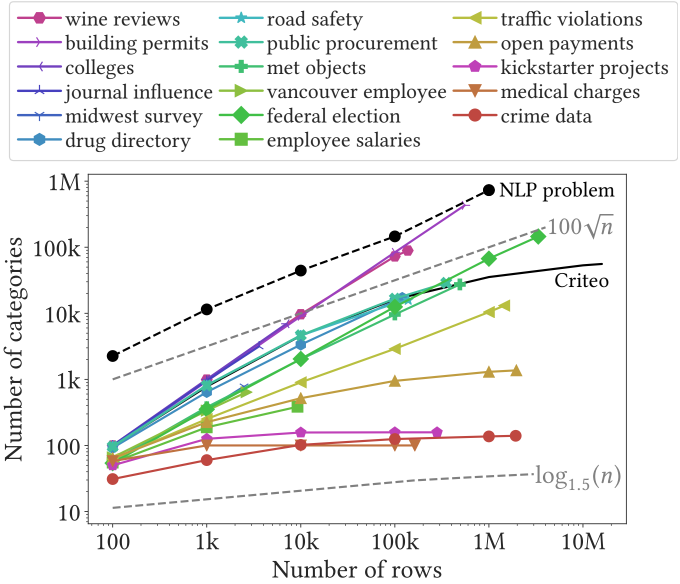


Fig. 1: **Number of categories versus number of samples**. In general, a higher number of samples implies a higher number of categories for the respective variable. In general, the cardinality of categories evolve slower than words in a typical NLP problem (Wikipedia articles in this case).

statistical performance and easier interpretation compared to direct adaptation of natural language processing, such as character-level word vectors [9]. As such, these encodings provide a scalable and automated replacement to data cleaning or feature engineering, and restore the benefits of a low-dimensional categorical encoding, as one-hot encoding.

The paper is organized as follows. Section 2 states the problem in detail and the prior art on creating feature vectors from categorical variables. Section 3 introduces and studies our two encoding approaches. In section 4, we present our experimental study with an emphasis on interpretation and on statistical learning on 17 non-curated datasets. Section 5 discusses these results, after which appendices provide information on the datasets and the experiments to facilitate the reproduction of our findings.

2 Problem Setting and Prior Art

Most academic machine-learning studies focus on datasets that contain only categorical variables with a low cardinality, as datasets[2] in the UCI repository [10]. In such settings, the popular *one-hot encoding* is a suitable solution for supervised learning [1]. It models categories as mutually exclusive and, as categories are known a priory, new categories are not expected to appear in the test set. With enough data, supervised learning can then be used to link each category to a target variable.

2.1 High-cardinality categorical variables

However, in many real-world problems, the number of different string entries in a column is very large, often growing with the number of observations (Figure 1). This leads to a very large number of categories. Consider for instance the *Drug Directory* dataset[3]: one of the variables is a categorical column with *non proprietary names* of drugs. As this column has not been normalized, it displays many different entries that are likely related, *i.e.*, categories that share a common ingredient (see Table 1a). Another example is the *Employee Salaries* dataset[4]. Here, a relevant variable is the position title of employees. As shown in Table 1b, here there is also overlap in the different occupations.

TABLE 1: Examples of high-cardinality categorical variables.

Count	Non Proprietary Name
1736	**alcohol**
1089	ethyl **alcohol**
556	isopropyl **alcohol**
16	polyvinyl **alcohol**
12	isopropyl **alcohol** swab
12	62% ethyl **alcohol**
6	**alcohol** 68%
6	**alcohol** denat
5	dehydrated **alcohol**

(a) Count for some of the categories containing the word *alcohol* in the *Drug Directory* dataset. The dataset contains more than 120k samples.

Employee Position Title
Police Aide
Master **Police** Officer
Mechanic **Technician** II
Police Officer III
Senior Architect
Senior Engineer **Technician**
Social Worker III
Bus Operator

(b) Some categories in the *Employee Salaries* dataset. For 10 000 employees, there are almost 400 different occupations. Yet, they share relevant substrings.

Data with high-cardinality categorical variables may arise from variability in their string representations, such as abbreviations, special characters, typos, etc[5]. Such non-normalized data often leads to many rare categories. Yet, these categories tend to have common morphological information. For instance, there is an implicit taxonomy in both examples above, drug names and position titles of employees. Hence, crafting feature engineering or data-cleaning rules may extract a small number of relevant categories, but it is time consuming and often needs domain expertise.

Notation

We write sets of elements with capital curly fonts, as $\mathcal{X}$. Elements of a vector space are written in bold $\mathbf{x}$ with the i-th entry denoted by x_i, and matrices are in capital and bold $\mathbf{X}$, with x_{ij} the entry on the i-th row and j-th column.

Let C be a categorical variable such that $\mathrm{dom}(C) \subseteq \mathcal{S}$, the set of finite length strings. We call *categories* the elements of $\mathrm{dom}(C)$. Let $s_i{\in}\mathcal{S}, i{=}1...n$, be the category corresponding to the i-th sample of a dataset. For statistical learning, we want to find an encoding function $\mathrm{enc}: \mathcal{S} \rightarrow \mathbb{R}^d$, such as $\mathrm{enc}(s_i) = \mathbf{x}_i$. We call $\mathbf{x}_i$ the *feature map* of s_i.

2.2 One-hot encoding, limitations and extensions

From a statistical-analysis standpoint, the multiplication of entries that relate to the same entity is challenging for two

[2] See for example, the Adult dataset (https://archive.ics.uci.edu/ml/datasets/adult)

[3] Product listing data for all unfinished, unapproved drugs. Source: U.S. Food and Drug Administration (FDA)

[4] Annual salary information for employees of the Montgomery County, MD, U.S.A. Source: https://data.montgomerycountymd.gov/

[5] A taxonomy of different sources of *dirty* categorical variables can be found on [11], and a formal description of data quality problems is proposed by [12].

TABLE 2: Summary of notations

Symbol	Definition
$\mathcal{S}$	Set of all finite-length strings.
$\mathcal{G}(s) \subseteq \mathcal{S}$	Set of all consecutive n-grams in $s \in \mathcal{S}$.
$\mathcal{V} = \bigcup_{i=1}^{n} \mathcal{G}(s_i)$	Vocabulary of n-grams in the train set.
C	Categorical variable.
n	Number of samples.
d	Dimension of the categorical encoder.
$m = \lvert\mathcal{V}\rvert$	Cardinality of the vocabulary.
$\mathbf{F} \in \mathbb{R}^{n \times m}$	Count matrix of n-grams.
$\mathbf{X} \in \mathbb{R}^{n \times d}$	Feature matrix of C.
sim : $\mathcal{S} \times \mathcal{S} \to [0, 1]$	String similarity.
$h_k : \mathcal{S} \to [0, 1]$	Hash function with salt value equal to k.
$Z_k : \mathcal{S} \to [0, 1]$	Min-hash function with salt value equal to k.

reasons. First, it dilutes the information: learning on rare categories is hard. Second, with one-hot encoding, representing these as separate entities creates high-dimension feature vectors. This high dimensionality entails large computational and memory costs; it increases the complexity of the associated learning problem, resulting in a poor statistical estimation. Dimensionality reduction of the one-hot encoded matrix can help with this issue, but at the risk of loosing information.

Moreover, encoding all entries with orthogonal vectors discards the overlap information visible in the string representations. Also, one-hot encoding cannot assign a feature vector to new categories that may appear in the testing set, even if its representation is close to one in the training set. Heuristics, such as assigning the zero vector to new categories, create collisions if more than one new category appears. As a result, one-hot encoding is ill suited to online learning settings: if new categories arrive, the entire encoding of the dataset has to be recomputed and the dimensionality of the feature vector becomes unbounded.

2.2.1 *Similarity encoding for string categorical variables*

For categorical variables represented by strings, *similarity encoding* extends one-hot encoding by taking into account a measure of string similarity between pairs of categories [6].

Let $s_i \in \mathcal{S}, i=1...n$, the category corresponding to the i-th sample of a given training dataset. Given a string similarity $\text{sim}(s_i, s_j)$, similarity encoding builds a feature map $\mathbf{x}^{\text{sim}} \in \mathbb{R}^k$ as:

$$\mathbf{x}_i^{\text{sim}} \overset{\text{def}}{=} [\text{sim}(s_i, s^{(1)}), \text{sim}(s_i, s^{(2)}), \ldots, \text{sim}(s_i, s^{(k)})] \in \mathbb{R}^k, \tag{1}$$

where $\{s^{(l)}, l=1 \ldots k\} \subseteq \mathcal{S}$ is the set of all unique categories in the training set—or a subset of prototype categories chosen heuristically[6]. With the previous definition, one-hot encoding corresponds to taking the discrete string similarity:

$$\text{sim}_{\text{one-hot}}(s_i, s_j) = \mathbb{1}[s_i = s_j], \tag{2}$$

where $\mathbb{1}[\cdot]$ is the indicator function.

Empirical work on databases with categorical columns containing non-normalized data showed that similarity encoding with a continuous string similarity brings significant benefits upon one-hot encoding [6]. Indeed, it relates rare categories to similar, more frequent ones. In columns with typos or morphological variants of the same information, a simple string similarity is often enough to capture additional information. Similarity encoding outperforms a bag-of-n-grams representation of the input string, as well as methods that encode high-cardinality categorical variables without capturing information in the strings representations [6], such as *target encoding* [13] or *hash encoding* [14].

A variety of string similarities can be considered for similarity encoding, but [6] found that a good performer was a similarity based on n-grams of consecutive characters. This n-gram similarity is based on splitting the two strings to compare in their character n-grams and calculating the Jaccard coefficient between these two sets [15]:

$$\text{sim}_{\text{n-gram}}(s_i, s_j) = J(\mathcal{G}(s_i), \mathcal{G}(s_j)) = \frac{|\mathcal{G}(s_i) \cap \mathcal{G}(s_j)|}{|\mathcal{G}(s_i) \cup \mathcal{G}(s_j)|} \tag{3}$$

where $\mathcal{G}(s)$, is the set of consecutive character n-grams for the string s.

Beyond the use of string similarity, an important aspect of similarity encoding is that it is a prototype method, using a subset of the categories defined in the train set as prototypes.

2.3 Related solutions for encoding string categories

2.3.1 *Bag of n-grams*

A simple way to capture morphology in a string is to characterize it by the count of its character n-grams. This is sometimes called a *bag-of-n-grams* characterization of strings. Such representation has been shown to be efficient for spelling correction [15], or to learn natural-language semantics while leveraging morphological structure of words [9].

For high-cardinality categorical variables, the number of different n-grams tends to increase with the number of samples. Yet, this number increases slowly, slower than in a typical NLP problem (see Figure 2). Indeed, categorical variables have less entropy than free text: they are usually repeated, often have subparts in common, and refer to a particular, more restrictive subject.

Representing strings by character-level n-grams is related to vectorizing texts by their tokens or words. Common practice uses *term-frequency inverse-document-frequency* (*tf-idf*) reweighting: dividing a token's count in a sample by its count in the whole document. Dimensionality reduction by a singular value decomposition (SVD) on this matrix leads to a simple topic extraction, latent semantic analysis (LSA) [16]. Another, more scalable, solution for dimensionality reduction are random projections [17].

2.3.2 *Word embeddings*

Another approach to create vector representations for strings is to leverage word embeddings developed in natural language processing [18], [19]. These find a vector representation of words such that Euclidean similarity of vectors captures related semantic meaning in words. Multiple words can be easily represented as a weighted sum of their vectors, or with more complex approaches [20]. These NLP approaches can be readily used to encode string categories in tables if they are made of common words. To cater for out-of-vocabulary strings, FastText [9] considers subword

[6] In this work, we use as dimensionality reduction technique the k-means strategy explained in [6].

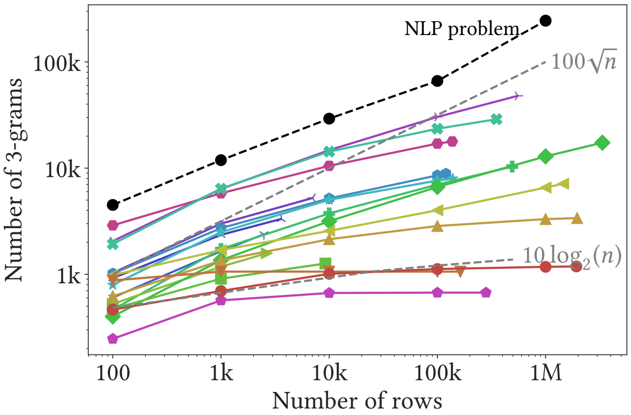


Fig. 2: **Number of 3-gram versus number of samples** (colors as in Figure 1). The number of different n-grams tends to increase slower than in a typical NLP problem (Wikipedia articles in this case).

information of words, *i.e.*, character-level n-grams. Hence, it can encode strings even in the presence of typos. Word vectors computed on very large corpora are available for download. These have captured fine semantic links between words. However, to analyze a given database, the danger of such approach is that the semantic of categories may differ from that in the pretrained model. These encodings do not capture the information specific to the categorical data at hand.

3 Scalable encoding of string categories

In this section we describe two novel alternatives for encoding categorical variables. Both of them are based on the character-level structure of categories. The first one, that we call *min-hash encoding*, comes from the document indexation literature, and in particular from the idea of locality-sensitive hashing (LSH) [8]. The method is stateless and it has been shown to approximate the Jaccard coefficient between two strings [21]. The second one is the *Gamma-Poisson factorization* [22], a matrix factorization technique—originally used in the probabilistic topic modeling literature—that assumes a Poisson distribution on the n-gram counts of categories, with a Gamma prior on the activations. An online algorithm of the factorization matrix allows to scale the method with a linear complexity on the number of samples. Both methods are intended to capture the morphological similarity of categories in a reduced dimensionality.

3.1 Min-hash encoding

3.1.1 Background: min-hash

Locality-sensitive hashing (LSH) [8] has been extensively used in the problem of approximate nearest neighbor search as an efficient way of finding similar objects (documents, pictures, etc.) in high-dimensional settings. One of the most famous functions in the LSH family is the min-hash function [21], [23], originally designed to retrieve similar documents in terms of the Jaccard coefficient of the word counts of documents (see [24], chapter 3, for a primer).

Let $\mathcal{X}^\star$ be a totally ordered set and π a random permutation of the order in $\mathcal{X}$. For any non-empty $\mathcal{X} \subseteq \mathcal{X}^\star$ with finite cardinality, the min-hash function $Z(\mathcal{X})$ can be defined as:

$$Z(\mathcal{X}) \overset{\text{def}}{=} \min_{x \in \mathcal{X}} \pi(x) \tag{4}$$

Note that $Z(\mathcal{X})$ can be also seen as a random variable. As shown in [21], for $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{X}^\star$, the min-hash function has the following property:

$$\mathbb{P}\left(Z(\mathcal{X}){=}Z(\mathcal{Y})\right) = J(\mathcal{X}, \mathcal{Y}) = \frac{|\mathcal{X} \cap \mathcal{Y}|}{|\mathcal{X} \cup \mathcal{Y}|} \tag{5}$$

Where J is the Jaccard coefficient between the two sets. For a controlled approximation, several random permutations can be taken, which defines a min-hash signature. For d permutations π_i drawn *i.i.d.*, Equation 5 leads to:

$$\sum_{i=1}^{d} \mathbb{1}[Z_i(\mathcal{X}) = Z_i(\mathcal{Y})] \sim \mathcal{B}(d, J(\mathcal{X}, \mathcal{Y})). \tag{6}$$

where $\mathcal{B}$ denotes the Binomial distribution. Dividing the above quantity by d thus gives a consistent estimate of the Jaccard coefficient $J(\mathcal{X}, \mathcal{Y})$[7].

Without loss of generality, we can consider the case of $\mathcal{X}^\star$ being equal to the real interval $[0, 1]$, so now for any $x \in [0, 1]$, $\pi_j(x) \sim \mathcal{U}(0, 1)$.

Proposition 3.1. Marginal distribution. *If* $\pi_j(x) \sim \mathcal{U}(0, 1)$, *and* $\mathcal{X} {\subset} [0, 1]$ *such that* $|\mathcal{X}| {=} k$, *then* $Z(\mathcal{X}) \sim Dir(k, 1)$.

Proof. It comes directly from considering that $\mathbb{P}(Z(\mathcal{X}){\leq}z) = 1{-}\mathbb{P}(Z(\mathcal{X}){>}z) = 1{-}\prod_{i=1}^{k}\mathbb{P}(\pi_j(x_i) > z) = 1{-}(1{-}z)^k$. □

Now that we know the distribution of the min-hash random variable, we will show how each dimension of a min-hash signature maps inclusion of sets to simple arithmetic comparisons.

Proposition 3.2. Inclusion. *Let* $\mathcal{X}, \mathcal{Y} {\subset} [0, 1]$ *such that* $|\mathcal{X}| {=} k_x$ *and* $|\mathcal{Y}| {=} k_y$.

(i) If $\mathcal{X} \subset \mathcal{Y}$, *then* $Z(\mathcal{Y}) \leq Z(\mathcal{X})$.

(ii) $\mathbb{P}\left(Z(\mathcal{Y}){\leq}Z(\mathcal{X}) \,|\, \mathcal{X}{\cap}\mathcal{Y}{=}\emptyset\right) = k_y/(k_x + k_y)$

Proof. (i) is trivial and *(ii)* comes directly from Prop. 3.1:

$$\begin{aligned}
&\mathbb{P}\left(Z(\mathcal{Y}){-}Z(\mathcal{X}) \leq 0 \,|\, \mathcal{X}{\cap}\mathcal{Y}{=}\emptyset\right) \\
&\quad = \int_0^1 \int_0^x f_{Z(\mathcal{Y})}(y) f_{Z(\mathcal{X})}(y)\, dy\, dx \\
&\quad = \int_0^1 \left(1 - (1-x)^{k_y}\right) k_x (1-x)^{k_x - 1} dx \\
&\quad = k_y/(k_x + k_y)
\end{aligned}$$

□

At this point, we do not know anything about the case when $\mathcal{X} \not\subset \mathcal{Y}$, so for a fixed $(Z(\mathcal{X})$, we can not ensure that any set with lower min-hash value has $\mathcal{X}$ as inclusion. The following theorem allows us to define regions in the vector

[7] Variations of the min-hash algorithm, as the min-max hash method [25] have proven to reduce the variance of the Jaccard similarity approximation.

space generated by the min-hash signature that, with high probability, are associated to inclusion rules.

Theorem 3.1. Identifiability of inclusion rules. *Let $\mathcal{X},\mathcal{Y}\subset[0,1]$ be two finite sets such that $|\mathcal{X}|{=}k_x$ and $|\mathcal{Y}|{=}k_y$. $\forall\,\epsilon{>}0$, if $d\geq\lceil-\log(\epsilon)/\log(1{+}\frac{k_x}{k_y})\rceil$, then*

$$\mathcal{X}\not\subseteq\mathcal{Y}\Rightarrow\mathbb{P}\left(\sum_{i=1}^{d}\mathbb{1}[Z_i(\mathcal{Y}){\leq}Z_i(\mathcal{X})]=d\right)\leq\epsilon. \tag{7}$$

Proof. First, notice that:

$$\mathcal{X}\not\subseteq\mathcal{Y} \iff \exists\, k\in\mathbb{N}, 0\leq k\leq k_x \text{ such that } |\mathcal{X}\cap\mathcal{Y}|=k$$

Then, defining $\mathcal{Y}'\overset{\text{def}}{=}\mathcal{Y}\setminus(\mathcal{X}\cap\mathcal{Y})$, with $|\mathcal{Y}'|=k_y-k$:

$$\begin{aligned}\mathbb{P}\left(Z(\mathcal{Y}){\leq}Z(\mathcal{X})\,|\,\mathcal{X}\not\subseteq\mathcal{Y}\right)&=\mathbb{P}\left(Z(\mathcal{Y}'){\leq}Z(\mathcal{X})\,|\,\mathcal{X}\cap\mathcal{Y}'=\emptyset\right)\\&=(k_y-k)/(k_x+k_y-k)\\&\leq k_y/(k_x+k_y)\\&=\mathbb{P}\left(Z(\mathcal{Y}){\leq}Z(\mathcal{X})\,|\,\mathcal{X}\cap\mathcal{Y}=\emptyset\right)\end{aligned}$$

Finally:

$$\begin{aligned}&\mathbb{P}\left(\sum\nolimits_{i=1}^{d}\mathbb{1}[Z_i(\mathcal{Y}){\leq}Z_i(\mathcal{X})]=d\,|\,\mathcal{X}\not\subseteq\mathcal{Y}\right)\\&\quad=\mathbb{P}\left(Z(\mathcal{Y}){\leq}Z(\mathcal{X})\,|\,\mathcal{X}\not\subseteq\mathcal{Y}\right)^d\\&\quad\leq\mathbb{P}\left(Z(\mathcal{X}){\leq}Z(\mathcal{Y})\,|\,\mathcal{X}\cap\mathcal{Y}{=}\emptyset\right)^d=(k_y/(k_x+k_y))^d\end{aligned}$$

□

Theorem 3.1 tells us that, by taking enough random permutations, the probability of finding false positives for the null hypothesis $\mathcal{X}\not\subseteq\mathcal{Y}$, it is bounded by ϵ. This result is very important, as it shows a global property of the min-hash representation when using several *i.i.d.* random permutations, going beyond the well-known local properties of the min-hash signature.

3.1.2 The min-hash encoder

A practical way to build a computationally efficient implementation of min-hash is to use a hash function with different salt numbers instead of random permutations. Indeed, hash functions can be built with suitable *i.i.d.* random-process properties [23]. Thus, the min-hash function can be constructed as follows:

$$Z_j(\mathcal{X})=\min_{x\in\mathcal{X}}h_j(x), \tag{8}$$

where h_j is a hash function[8] on $\mathcal{X}$ with salt value j.

For the specific problem of categorical data, we are interested in a fast approximation of $J(\mathcal{G}(s_i),\mathcal{G}(s_j))$, where $\mathcal{G}(s)$ is the set of all consecutive character n-grams for the string s. We define the min-hash encoder as:

$$\mathbf{x}^{\text{min}}(s)\overset{\text{def}}{=}[Z_1(\mathcal{G}(s)),\ldots,Z_d(\mathcal{G}(s))]\in\mathbb{R}^d, \tag{9}$$

Considering the hash functions as random processes, Equation 6 implies that this encoder has the following property:

$$\mathbb{E}\left[\|\mathbf{x}^{\text{min}}(s_i)-\mathbf{x}^{\text{min}}(s_j)\|_{\ell_0}\right]=J(\mathcal{G}(s_i),\mathcal{G}(s_j)) \tag{10}$$

Proposition 3.2 tells us that the min-hash encoder transforms, the containment operation in the string space into an order relation in the feature space. This is especially relevant for learning tree-based methods, as theorem 3.1 proofs that by performing a reduced number of splits in the min-hash dimensions, the space can be divided between the elements that contain and do not contain a given substring s_i. As an example, Figure 3 shows this global property of the min-hash encoder for the case of the employe salaries dataset with $d{=}2$. The substrings `Senior`, `Supply` and `Technician` are all included in the category `Senior Supply Technician` , and as consequence, the position for this category in the encoding space will be always in the bottom-left region generated by the substrings.

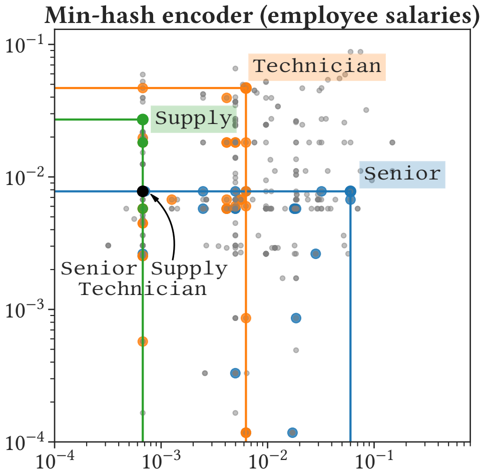


Fig. 3: **Interpretation of min-hash encodings.** Color dots are categories that contain the corresponding colored substrings and gray dots are categories that do not contain any of them. The category `Senior Supply Technician` (black dot) contains all substrings.

Finally, this encoder is specially suitable for very large scale settings, as it is completely stateless and very fast to compute.

3.2 Gamma-Poisson factorization

3.2.1 Model

The Gamma-Poisson model [22] is a matrix factorization technique that assumes a generative model for counting statistics. The idea was originally developed for finding low-dimensional representations, known as topics, of documents given its word count representation. As categories are usually composed of much shorter strings than documents, and can also contain typos, we consider the character-level structure of n-grams instead. Each observation, a category described by the count vector of its character n-grams $\mathbf{f}\in\mathbb{N}^m$, is modeled as a linear combination of d unknown *prototypes* or *topics*, $\mathbf{\Lambda}\in\mathbb{R}^{d\times m}$:

$$\mathbf{f}\approx\mathbf{x}\,\mathbf{\Lambda}, \tag{11}$$

Here, $\mathbf{x}\in\mathbb{R}^d$ are the activations that decompose the observation $\mathbf{f}$ in the prototypes $\mathbf{\Lambda}$ in the count space. As we will see later, these prototypes can be seen as latent categories.

Given a training dataset with n samples, the model estimates the unknown prototypes $\mathbf{\Lambda}$ by factorizing the

[8] Here we use a 32bit version of the MurmurHash3 function [26].

data's bag-of-n-grams representation $\mathbf{F} \in \mathbb{N}^{n\times m}$, where m is the number of different n-grams in the data:

$$\mathbf{F} \approx \mathbf{X}\,\mathbf{\Lambda}, \quad \text{with } \mathbf{X} \in \mathbb{R}^{n\times d}, \mathbf{\Lambda} \in \mathbb{R}^{d\times m} \tag{12}$$

As $\mathbf{f}$ is a count vector, it is natural to consider a Poisson distribution for each of its elements:

$$p\big(f_j|(\mathbf{x}\,\mathbf{\Lambda})_j\big) = \frac{1}{f_j!}(\mathbf{x}\,\mathbf{\Lambda})_j^{f_j}\,e^{-(\mathbf{x}\,\mathbf{\Lambda})_j}, \quad j = 1, ..., m. \tag{13}$$

For a prior on the elements of $\mathbf{x} \in \mathbb{R}^d$, we use a Gamma distribution, as it is the conjugate prior of the Poisson distribution:

$$p(x_i) = \frac{x_i^{\alpha_i-1}e^{-x_i/\beta_i}}{\beta_i^{\alpha_i}\,\Gamma(\alpha_i)}, \quad i = 1, ..., d. \tag{14}$$

Where $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \in \mathbb{R}^d$ are the shape and scale parameters of the Gamma distribution for each one of the d topics.

3.2.2 Estimation strategy

To fit the model to the input data, we maximize the likelihood $\mathcal{L}$ of the model, denoted by:

$$\mathcal{L} = \prod_{j=1}^{m} \frac{(\mathbf{x}\,\mathbf{\Lambda})_j^{f_j}\,e^{-(\mathbf{x}\,\mathbf{\Lambda})_j}}{f_j!} \prod_{i=1}^{d} \frac{x_i^{\alpha_i-1}e^{-x_i/\beta_i}}{\beta_i^{\alpha_i}\,\Gamma(\alpha_i)} \tag{15}$$

Maximizing the log-likelihood with respect to the parameters gives:

$$\frac{\partial}{\partial \Lambda_{ij}} \log \mathcal{L} = \frac{f_j}{(\mathbf{x}\,\mathbf{\Lambda})_j} x_i - x_i \tag{16}$$

$$\frac{\partial}{\partial x_i} \log \mathcal{L} = \sum_{j=1}^{m} \frac{f_j}{(\mathbf{x}\,\mathbf{\Lambda})_j} \Lambda_{ij} - \Lambda_{ij} + \frac{\alpha_i - 1}{x_i} - \frac{1}{\beta_i} \tag{17}$$

As explained in [22], these expressions are analogous to solving the following non-negative matrix factorization (NMF) with the generalized Kullback-Leibler divergence[9] as loss:

$$\begin{pmatrix} \mathbf{F} \\ \text{diag}(\boldsymbol{\beta})^{-1} \end{pmatrix} = \mathbf{X} \begin{pmatrix} \mathbf{\Lambda} \\ \text{diag}(\boldsymbol{\alpha}) - I_d \end{pmatrix} \tag{18}$$

In other words, the Gamma-Poisson model can be interpreted as a constrained non-negative matrix factorization in which the generalized Kullback-Leibler divergence is minimized between $\mathbf{F}$ and $\mathbf{X\Lambda}$, subject to a Gamma prior in the distribution of the elements of $\mathbf{X}$. The Gamma prior induces sparsity in the activations $\mathbf{x}$ of the model.

To solve the NMF problem above, [27] proposes the following recurrences:

$$\Lambda_{ij} \leftarrow \Lambda_{ij} \left(\sum_{\ell=1}^{n} \frac{f_{\ell j}}{(\mathbf{X\Lambda})_{\ell j}} x_{\ell i}\right) \left(\sum_{\ell=1}^{n} x_{\ell i}\right)^{-1} \tag{19}$$

$$x_{\ell i} \leftarrow x_{\ell i} \left(\sum_{j=1}^{m} \frac{f_{\ell j}}{(\mathbf{X\Lambda})_{\ell j}} \Lambda_{ij} + \frac{\alpha_i - 1}{x_{\ell i}}\right) \left(\sum_{j=1}^{m} \Lambda_{ij} + \beta_i^{-1}\right)^{-1} \tag{20}$$

As $\mathbf{F}$ is a sparse matrix, the summations above need to be computed only on the non-zero elements of $\mathbf{F}$. This fact considerably decreases the complexity of the algorithm.

Algorithm 1: Online Gamma-Poisson factorization

```
   Input  : $\mathbf{F}$, $\mathbf{\Lambda}^{(0)}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\rho$, $q$, $\eta$, $\epsilon$
   Output: $\mathbf{X}$, $\mathbf{\Lambda}$
1  while $\|\mathbf{\Lambda}^{(t)} - \mathbf{\Lambda}^{(t-1)}\|_F > \eta$ do
2      draw $\mathbf{f}_t$ from the training set $\mathbf{F}$.
3      while $\|\mathbf{x}_t - \mathbf{x}_t^{old}\|_2 > \epsilon$ do
4          $\mathbf{x}_t \leftarrow$
             $\left[\mathbf{x}_t \left(\frac{\mathbf{f}_t}{\mathbf{x}_t \mathbf{\Lambda}^{(t)}}\right) \mathbf{\Lambda}^{(t)\mathsf{T}} + \boldsymbol{\alpha} - 1\right] . \left[\mathbf{1}\,\mathbf{\Lambda}^{(t)\mathsf{T}} + \boldsymbol{\beta}^{-1}\right]^{.-1}$
5      end
6      $\tilde{\mathbf{A}}_t \leftarrow \mathbf{\Lambda}^{(t)} . \left[\mathbf{x}_t^{\mathsf{T}} \left(\frac{\mathbf{f}_t}{\mathbf{x}_t \mathbf{\Lambda}^{(t)}}\right)\right]$
7      $\tilde{\mathbf{B}}_t \leftarrow \mathbf{x}_t^{\mathsf{T}} \mathbf{1}$
8      if $t \equiv 0 \bmod q$, then
9          $\mathbf{A}^{(t)} \leftarrow \rho\,\mathbf{A}^{(t-q)} + \sum_{s=t-q+1}^{t} \tilde{\mathbf{A}}^{(s)}$
10         $\mathbf{B}^{(t)} \leftarrow \rho\,\mathbf{B}^{(t-q)} + \sum_{s=t-q+1}^{t} \tilde{\mathbf{B}}^{(s)}$
11         $\mathbf{\Lambda}^{(t)} \leftarrow \mathbf{A}^{(t)} ./\, \mathbf{B}^{(t)}$
12     end
13     $t \leftarrow t + 1$
14 end
```

Following [28], we present an online version of the Gamma-Poisson (algorithm 1) for the Generalized KL divergence. The basic idea of the algorithm is to exploit the fact that in the recursion for $\mathbf{\Lambda}$, the summations are done with respect to the training samples. Instead of computing the numerator and denominator in the entire training set at each update, one can update this values only with mini-batches of data, which considerably decreases the memory usage and time of the computations.

We carefully adapt the implementation of this solver to the specificities of our problem—factorizing substring counts across entries of a categorical variable. We improve the efficiency of the algorithm by saving a dictionary of the activations for each category in the convergence of the previous mini-batches (algorithm 1, line 4) and use them as an initial guess for the same category in a future mini-batch. This is a warm restart and is especially important in the case of categorical variables because for most datasets, the number of unique categories is much lower than the number of samples.

The algorithm also requires some input parameters and initializations that can affect convergence. One important parameter is ρ, the discount factor for the previous iterations of the topic matrix $\mathbf{\Lambda}^{(t)}$ (algorithm 1, line 9-10). Figure 10 in the appendix shows that choosing ρ=0.95 gives a good compromise between stability of the convergence and data fitting in term of the Generalized KL divergence. With respect to the initialization of the topic matrix $\mathbf{\Lambda}^{(0)}$, a good option is to choose the centroids of a k-means clustering (Figure 11) in a hashed version of the n-gram count matrix $\mathbf{F}$ (in order to speed-up the k-means algorithm) and then project back to the n-gram space with a nearest neighbors algorithm.

[9] In the sense of the NMF literature. See for instance [27].

3.2.3 Inferring feature names

Besides prediction performance, a good categorical encoder needs to be easily interpretable. A straightforward strategy for interpretation is to identify each encoding dimension as a latent category. To accomplish this, one alternative is to observe the feature maps of each category, and assign labels based on the categories that activate the most in a given dimensionality. Another option is to observe the same phenomena but for each word contained in the categories. For the experimental section, we followed the last approach because a lot of datasets are composed of categories that are overlapped, so individual words could carry more information for interpretability than the entire string categories.

This method can be applied to any encoder, but it is expected to work well if the encodings are sparse and composed of only positive values with a meaningful magnitude.

4 Experimental study

In this section, we show empirical results for different encoding methods in terms of interpretability and prediction performance. For this purpose, we used three different types of data: simulated categorical data, and real data with curated and non-curated categories.

We benchmarked the following strategies: one-hot, tf-idf, fastText [29], similarity encoding [6], and the Gamma-Poisson factorization on character n-grams. For all the strategies based on a n-gram representation, we used the set of 2-4 character grams[10]. For a fair comparison across encoding strategies, we used the same dimensionality d in all approaches. To set the dimensionality of one-hot encoding, tf-idf and fastText, we used a truncated SVD (implemented efficiently following [30]). Note that dimensionality reduction improves one-hot encoding with tree-based learners for data with rare categories [6]. For similarity encoding, we selected prototypes with a *k-means* strategy, following [6], as it gives slightly better prediction results than the *most frequent categories*[11].

4.1 Datasets with high-cardinality categories

In order to evaluate the different encoding strategies presented in the previous sections, we collected 17 real-world datasets containing a prediction task and at least one relevant high-cardinality categorical variable as feature[12]. Table 3 shows a quick description of the datasets and the corresponding categorical variables (see Section subsection A.1 for a description of datasets and the related learning tasks).

4.2 Recovering latent categories

Given Table 3, we notice that the most typical sources of high cardinality are the presence of *multi-label* categories, which represents all forms of concatenation of information. The second most common problem is the presence of *typos* (or any source of morphological variation of the same idea). To analyze this two cases in a more controlled setting, we created two simulated categorical variables. In Table 4 we can observe examples of the created categories taking as a base 8 ”ground truth” categories of animals: `chicken`, `eagle`, `giraffe`, `horse`, `leopard`, `lion`, `tiger` and `turtle`.

The multi-label data was created by concatenating k ground truth categories, with $k-2$ following a Poisson distribution (all categories have at least two labels). For the generation of data with typos, we added 10% of typos to the original ground truth categories by randomly replacing one character by another one (`x`, `y`, or `z`).

To test the ability of an encoder to recover a feature matrix close to a one-hot encoding matrix of ground-truth categories in these simulated settings, we use the Normalized Mutual Information (NMI) as metric. Given two random variables X_1 and X_2, the NMI is defined as:

$$\mathrm{NMI} = 2\,\frac{I(X_1; X_2)}{H(X_1) + H(X_2)} \tag{21}$$

Where $I(\cdot\,;\cdot)$ is the mutual information and $H(\cdot)$ the entropy. To apply this metric to the feature matrix $\mathbf{X}$ generated by the encoding of all ground truth categories, we consider that $\mathbf{X}$, after a simple rescaling, can be seen as a two dimensional probability distribution. As for some encoders the encoding matrix can contain negative values, we take in these cases the element-wise absolute value of $\mathbf{X}$. The NMI of any permutation of the identity matrix is equal to 1 and the NMI of any constant matrix is equal to 0. Thus, the NMI in this case is interpreted as a recovering metric of a one-hot encoded matrix of latent, ground truth, categories.

Table 5 shows the NMI values for both simulated datasets. The Gamma-Poisson factorization obtained the highest values in both cases and for different dimensionalities of the encoders, with the maximum at the dimension equal to the number of ground-truth categories, *i.e.*, k=8.

4.2.1 Results for real non-curated data

To evaluate the prediction behavior of encoders when the categorical variables have already been curated (usually, standardization of categories also generates low cardinality), we collected seven datasets of this kind.

4.2.2 Results for real curated data

In the case of curated data, Table 7 shows the NMI values for different datasets. In 4 out 5 datasets, the Gamma-Poisson factorization gives the highest score.

4.3 Encoding for supervised learning

For supervised learning, we report prediction results with gradient boosted trees, as implemented in XGBoost[13]. [31]. Note that trees can be implemented on categorical variables. However, this encounter the same problems as one-hot encoding: the number of comparisons grows with the number of categories. Hence, the best trees approaches for categorical data use target encoding to impose an order on categories [32].

[10] In addition to the word as tokens, pretrained versions of fastText also use the set of 3-6 character n-grams.

[11] We do not test the *random projections* strategy for similarity encoding as it is not scalable.

[12] If a dataset has more than one categorical variable, only one selected variable was encoded with the proposed approaches, while the rest of them were one-hot encoded.

[13] XGBoost does not support categorical features. The recommended option is to use one-hot encoding (https://xgboost.readthedocs.io).

TABLE 3: **Non-curated datasets.** Description for the corresponding high-cardinality categorical variable.

Dataset	#samples	#categories	#categories per 1000 samples	Gini coefficient	Mean category length (#chars)	Source of high cardinality
Crime Data	1.5M	135	**64.5**	0.85	30.6	Multi-label
Medical Charges	163k	100	**99.9**	0.23	41.1	Taxonomy; Multi-label
Kickstarter Projects	281k	158	**123.8**	0.64	11.0	Morphological overlap
Employee Salaries	9.2k	385	**186.3**	0.79	24.9	Multi-label
Open Payments	2.0M	1.4k	**231.9**	0.90	24.7	Multi-label
Traffic Violations	1.2M	11.3k	**243.5**	0.97	62.1	Typos; Multi-label
Vancouver Employees	2.6k	640	**341.8**	0.67	21.5	Multi-label
Federal Election	3.3M	145.3k	**361.7**	0.76	13.0	Typos; Multi-label
Midwest Survey	2.8k	844	**371.9**	0.67	15.0	Typos
Met Objects	469k	26.8k	**386.1**	0.88	12.2	Typos; Multi-label
Drug Directory	120k	17.1k	**641.9**	0.81	31.3	Multi-label
Road Safety	139k	15.8k	**790.1**	0.65	29.0	Multi-label
Public Procurement	352k	28.9k	**804.6**	0.82	46.8	Multi-label; Multi-language
Journal Influence	3.6k	3.2k	**956.9**	0.10	30.0	Multi-label; Multi-language
Building Permits	554k	430.6k	**940.0**	0.48	94.0	Typos; Description
Wine Reviews	138k	89.1k	**997.7**	0.23	245.0	Description
Colleges	7.8k	6.9k	**998.0**	0.02	32.1	Multi-label

TABLE 4: Examples of simulated categorical variables.

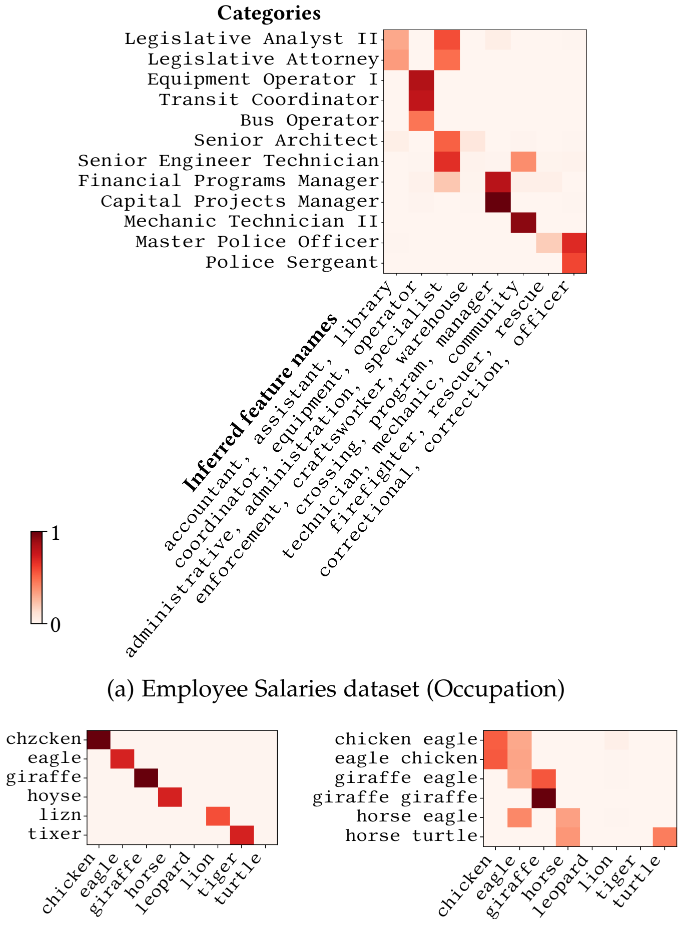
Type	Example categories
Ground truth	`chicken; eagle; giraffe; horse; leopard; lion; tiger; turtle.`
Multi-label	`lion chicken; horse eagle lion; tiger leopard giraffe turtle.`
Typos (10%)	`chxcken; eazle; gixaffe; gizaffe; hoyse; lexpard; lezpard; lixn; tiyer; tuxtle.`

TABLE 5: **Recovery of latent categories**: Normalized mutual information for different encoders.

Encoder	Multilabel			Typos		
	k=6	**k=8**	k=10	k=6	**k=8**	k=10
Tf-idf + SVD	0.16	0.18	0.17	0.17	0.17	0.17
FastText + SVD	0.08	0.09	0.09	0.08	0.08	0.09
Similarity Encoder	0.32	0.25	0.24	0.72	0.82	0.78
Min-hash Encoder	0.14	0.15	0.13	0.14	0.15	0.13
Gamma-Poisson	**0.76**	**0.82**	**0.79**	**0.78**	**0.83**	**0.80**

TABLE 6: Wilcoxon test p-values for different encoders between SVD and Gaussian random projections as a dimensionality reduction technique

Encoder	SVD v/s Random projection
Tf-idf	**0.003**
FastText	**0.001**
One-hot	0.492



(a) Employee Salaries dataset (Occupation)

(b) Simulated typos

(c) Simulated multi-label categories

Fig. 4: **The Gamma-Poisson factorization gives positive and sparse representations that are easily interpretable.** Examples of encoding vectors (d=8) for a real dataset (a) and for simulated data (b and c) obtained with a Gamma-Poisson factorization. The x-axis shows the activations with their respective inferred feature names.

Depending on the dataset, the learning task can be either *regression*, *binary* or *multiclass* classification[14]. As datasets get different prediction scores, we visualize encoders' performance with prediction accuracy results scaled in a *relative score*. It is a data-specific scaling of the original score, in order to bring performance across datasets in the same range. In other words, for a given dataset i:

$$\text{relative score}_j^i = 100 \frac{\text{score}_j^i - \min_j \text{score}_j^i}{\max_j \text{score}_j^i - \min_j \text{score}_j^i} \tag{22}$$

[14] Depending on the type of prediction task, we used different scores to evaluate the performance of the supervised learning problem: for regression, we used the R^2 score; for binary classification, the average precision; and for multiclass classification, the accuracy score.

TABLE 7: **Recovering true categories in low-cardinality case.** Normalized mutual information for different encoders (d=30)

Dataset (cardinality)	Gamma-Poisson	Similarity Encoding	Tf-idf + SVD	FastText + SVD
Adult (5)	**0.75**	0.71	0.54	0.19
California Housing (5)	0.46	0.51	**0.56**	0.20
Dating Profiles (18)	**0.52**	0.24	0.25	0.12
House Prices (15)	**0.83**	0.24	0.31	0.11
House Sales (70)	**0.42**	0.04	0.18	0.06

where score^i_j is the the prediction score for the dataset i with the configuration $j \in \mathcal{J}$, the set of all trained models—in terms of dimensionality, type of encoder and cross-validation split. The relative score is figure-specific and is only indented to be used as a visual comparison of classifiers' performance across multiple datasets. A higher relative score means better results.

For a proper statistical comparison of encoders, we use a ranking test across multiple datasets [33]. Note that in this framework each dataset represents a single sample, and not the cross-validation splits which are not mutually independent. To do so, for a particular dataset, encoders were ranked according to the median score value over cross-validation splits. At the end, the relevant statistics is the average ranking across datasets. This statistic is computed for all encoders at a fixed dimensionality d. This test compares the average ranking using a F-distribution. If the null hypothesis is rejected, we use a Nemenyi post-hoc test to verify whether the difference in performance across pairs of encoders is significant [33].

To do pairwise comparison between two encoders, we use a pairwise Wilcoxon signed rank test and we provide the corresponding p-values given the null hypothesis that the two encoders are equally performing across different datasets.

4.3.1 Prediction with non-curated data

Figure 5 shows the results of the prediction benchmark with 17 datasets. It compares encoders in terms of the relative score of Equation 22. All n-gram based encoders clearly improve upon one-hot encoding, at all dimensions. Min-hash seems to have a slightly better prediction performance for both dimensionalities in terms of the median relative score across categories, despite of being the only studied method that does not require a data fit.

Results of the Nemenyi ranking test for both dimensionalities are similar. It confirms the superior performance of n-gram-based methods in comparison to one-hot encoding.

5 Discussion

One-hot encoding is not well suited for a column of a table containing categories represented with many different strings. A character n-gram count vector can represent strings well. Yet, it dilutes the notion of categories, because it creates extremely high-dimensional vectors. Indeed, strings that represent categories, even when normalized, tend to have many n-grams in common. A good encoding should capture this information from the data and reflect it in a lower dimensional encoding.

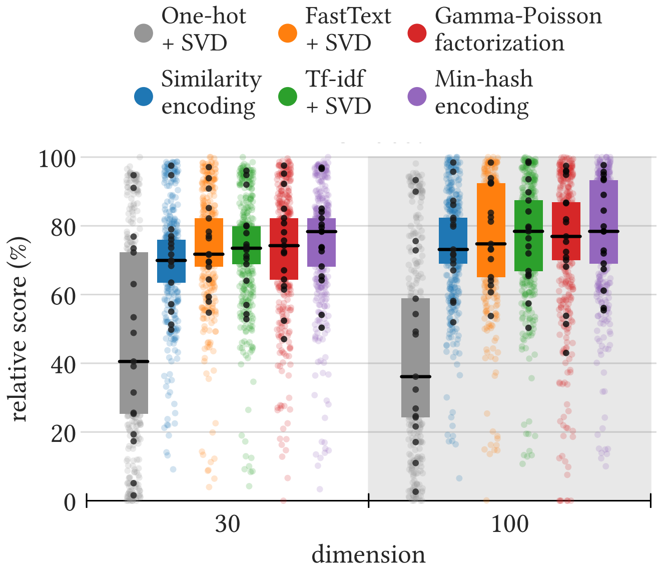


Fig. 5: **Encoders that consider subword information perform significantly better than one-hot.** Classifier: XGBoost. (a) Comparison of encoders in terms of a relative score (the prediction score on the particular dataset, rescaled with respect to the global maximum and minimum score values across dimensions). Color dots indicate the scores for each cross-validation fold, black dots the median score across folds for a dataset, the black line indicates the median score and the box gives the interquartile range. For one-hot encoding, bag-of-n-grams and fastText, the dimensionality was fixed by using a truncated SVD.

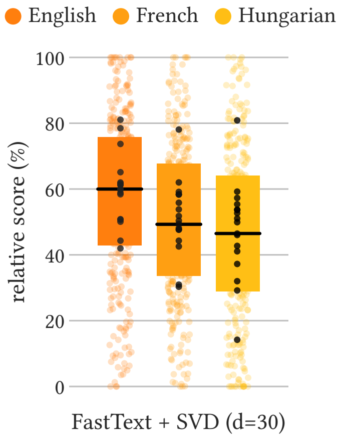


Fig. 6: Relative score

Fig. 7: **FastText prediction performance drops languages other than English.** Relative prediction scores with pretrained fastText vectors in different languages. The dimensionality was set using a truncated SVD. A pairwise Wilcoxon signed rank tests give the following p-values: English-French p=0.056, French-Hungarian p=0.149, English-Hungarian p=0.019.

We have studied several encoding approaches to capture the structural covariation of string categorical variables in tables. We use a low-rank approximation of the multivariate distribution of n-gram with a Gamma-Poisson factorization, which leads to a form of non-negative matrix factorization adapted to count data.

Experiments on 17 real-word dataset show that all encoding techniques that describe well the covariations of character n-grams in the data lead to a good prediction. However, the encodings created by the Gamma-Poisson factorization are much more interpretable: the model estimates

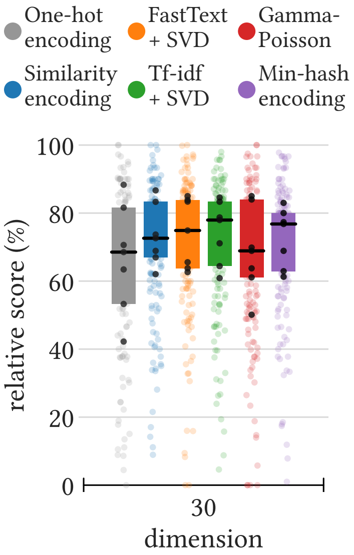


Fig. 8: **All encoders perform well for low-cardinality datasets** (classifier: XGBoost). The relative score denotes the percentage of the corresponding score value with respect to the minimum and maximum scores by dataset. Color dots indicate the scores for each cross-validation fold, black dots the median score across folds, the black line indicates the median score and the box gives the interquartile range.

latent categories from the data, and gives a sparse, non-negative, encoding on these. In addition, the experiments outline the importance of having an encoding adapted to the data. For instance, character-level vectors embeddings from natural language processing give competitive prediction performance only if they are built for the same language as the database studied. In this sense, it may be preferable to fully capture the semantics from the data, as with the Gamma-Poisson factorization.

6 Conclusion

This work shows how to overcome the limitations of one-hot encoding for high-cardinality string categories, as with non-normalized entries. Two aspects are important for an encoding to give a good prediction in supervised learning settings: a character-level representation of the entries, and capturing well the distribution on strings induced by categories. The min-hash encoder seems to have a slightly superior prediction performance (with gradient boosted trees) and is the fastest method, as it is the only one that does not require any fit to the data. A Gamma-Poisson factorization gives a good encoding approach: it can be implemented online with an efficient solver; it leads to good prediction; and it finds latent categories thanks to substrings repeated across the entries. As such, it gives a readily-usable replacement to one-hot encoding for high-cardinality string categorical variables: it enables a statistical analysis on reduced-dimension encoding that still captures meaningful categorical information. It avoids one of the time-consuming steps of a data-science study: normalizing entries of databases via various forms of data cleaning such as deduplication or standardizing rules.S

Acknowledgments

Authors were supported by the DirtyData (ANR-17-CE23-0018-01) project.

References

[1] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[2] D. Pyle, *Data preparation for data mining*. Morgan Kaufmann, 1999, vol. 1.

[3] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 3–13, 2000.

[4] W. E. Winkler, "Overview of record linkage and current research directions," in *Bureau of the Census*. Citeseer, 2006.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 1, pp. 1–16, 2007.

[6] P. Cerda, G. Varoquaux, and B. Kégl, "Similarity encoding for learning with dirty categorical variables," *Machine Learning*, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10994-018-5724-2

[7] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[8] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association of Computational Linguistics*, vol. 5, no. 1, pp. 135–146, 2017.

[10] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu

[11] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee, "A taxonomy of dirty data," *Data mining and knowledge discovery*, vol. 7, no. 1, pp. 81–99, 2003.

[12] P. Oliveira, F. Rodrigues, and P. R. Henriques, "A formal definition of data quality problems," in *Proceedings of the 2005 International Conference on Information Quality (MIT IQ Conference)*, 2005.

[13] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," *ACM SIGKDD Explorations Newsletter*, vol. 3, no. 1, pp. 27–32, 2001.

[14] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1113–1120.

[15] R. C. Angell, G. E. Freund, and P. Willett, "Automatic spelling correction using a trigram similarity measure," *Information Processing & Management*, vol. 19, no. 4, pp. 255–261, 1983.

[16] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[17] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, 2008, p. 1177.

[18] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532–1543.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR Workshop Papers*, 2013.

[20] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," 2016.

[21] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 1997, pp. 21–29.

[22] J. Canny, "Gap: A factor model for discrete data," in *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '04. New York, NY, USA: ACM, 2004, pp. 122–129. [Online]. Available: http://doi.acm.org/10.1145/1008992.1009016

[23] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[24] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge university press, 2014.

[25] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang, "Min-max hash for jaccard similarity," in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 301–309.

[26] A. Appleby, "Murmurhash3 http://code. google. com/p/smhasher/wiki," 2014.

[27] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 556–562. [Online]. Available: http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf

[28] A. Lefevre, F. Bach, and C. Févotte, "Online algorithms for non-negative matrix factorization with the itakura-saito divergence," in *Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2011, p. 313.

[29] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *International Conference on Language Resources and Evaluation (LREC)*, 2018.

[30] N. Halko, P.-G. Martinsson, and J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, p. 217, 2011.

[31] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *SIGKDD*, 2016, pp. 785–794.

[32] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Neural Information Processing Systems*, 2018, p. 6639.

[33] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[34] 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*, vol. 12, pp. 2825–2830, 2011.

Appendix A Reproducibility

A.1 Dataset Description

Building Permits [15] (sample size: 554k). Permits issued by the Chicago Department of Buildings since 2006. Target (regression): *Estimated Cost*. Categorical variable: *Work Description* (cardinality: 430k).

Colleges [16] (7.8k). Information about U.S. colleges and schools. Target (regression): *Percent Pell Grant*. Cat. var.: *School Name* (6.9k).

Crime Data [17] (1.5M). Incidents of crime in the City of Los Angeles since 2010. Target (regression): *Victim Age*. Categorical variable: *Crime Code Description* (135).

Drug Directory [18] (120k). Product listing data submitted to the U.S. FDA for all unfinished, unapproved drugs. Target (multiclass): *Product Type Name*. Categorical var.: *Non Proprietary Name* (17k).

Employee Salaries [19] (9.2k). Salary information for employees of the Montgomery County, MD. Target (regression): *Current Annual Salary*. Categorical variable: *Employee Position Title* (385).

Federal Election [20] (3.3M). Campaign finance data for the 2011-2012 US election cycle. Target (regression): *Transaction Amount*. Categorical variable: *Memo Text* (17k).

Journal Influence [21] (3.6k). Scientific journals and the respective influence scores. Target (regression): *Average Cites per Paper*. Categorical variable: *Journal Name* (3.1k).

Kickstarter Projects [22] (281k). More than 300,000 projects from the Kickstarters platform https://www.kickstarter.com. Target (binary): *State*. Categorical variable: *Category* (158).

Medical Charges [23] (163k). Inpatient discharges for Medicare beneficiaries for more than 3,000 U.S. hospitals. Target (regression): *Average Total Payments*. Categorical var.: *Medical Procedure* (100).

Met Objects [24] (469k). Information on artworks objects of the Metropolitan Museum of Art's collection. Target (binary): *Department*. Categorical variable: *Object Name* (26k).

Midwest Survey [25] (2.8k). Survey to know if people self-identify as Midwesterners. Target (multiclass): *Census Region* (10 classes). Categorical var.: *What would you call the part of the country you live in now?* (844).

Open Payments [26] (2M). Payments given by healthcare manufacturing companies to medical doctors or hospitals (year 2013). Target (binary): *Status* (if the payment was made under a research protocol). Categorical var.: *Company name* (1.4k).

Public Procurement [27] (352k). Public procurement data for the European Economic Area, Switzerland, and the Macedonia. Target (regression): *Award Value Euro*. Categorical var.: *CAE Name* (29k).

Road Safety [28] (139k). Circumstances of personal injury of road accidents in Great Britain from 1979. Target (binary): *Sex of Driver*. Categorical variable: *Car Model* (16k).

Traffic Violations [29] (1.2M). Traffic information from electronic violations issued in the Montgomery County, MD. Target (multiclass): *Violation type* (4 classes). Categorical var.: *Description* (11k).

Vancouver Employee [30](2.6k). Remuneration and expenses for employees earning over $75,000 per year. Target (regression): *Remuneration*. Categorical variable: *Title* (640).

Wine Reviews [31] (138k). Wine reviews scrapped from WineEnthusiast with variety, location, winery, price, and description. Target (regression): *Points*. Categorical variable: *Description* (89k).

A.2 Learning pipeline

Sample size: Datasets' size range from a couple of thousand to several million samples. To reduce computation time on the learning step, the number of samples was limited to 100k for large datasets.

Data preprocessing: We removed rows with missing values in the target or in any explanatory variable other than the selected categorical variable, for which we replaced missing entries by the string 'nan'. The only additional

[15] https://www.kaggle.com/chicago/chicago-building-permits

[16] https://beachpartyserver.azurewebsites.net/VueBigData/DataFiles/Colleges.txt

[17] https://data.lacity.org/A-Safe-City/Crime-Data-from-2010-to-Present/y8tr-7khq

[18] https://www.fda.gov/Drugs/InformationOnDrugs/ucm142438.htm

[19] https://catalog.data.gov/dataset/employee-salaries-2016

[20] https://classic.fec.gov/finance/disclosure/ftpdet.shtml

[21] https://github.com/FlourishOA/Data

[22] https://www.kaggle.com/kemical/kickstarter-projects

[23] https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Inpatient.html

[24] https://github.com/metmuseum/openaccess

[25] https://github.com/fivethirtyeight/data/tree/master/region-survey

[26] https://openpaymentsdata.cms.gov

[27] https://data.europa.eu/euodp/en/data/dataset/ted-csv

[28] https://data.gov.uk/dataset/road-accidents-safety-data

[29] https://catalog.data.gov/dataset/traffic-violations-56dda

[30] https://data.vancouver.ca/datacatalogue/employeeRemunerationExpensesOver75k.htm

[31] https://www.kaggle.com/zynicide/wine-reviews/home

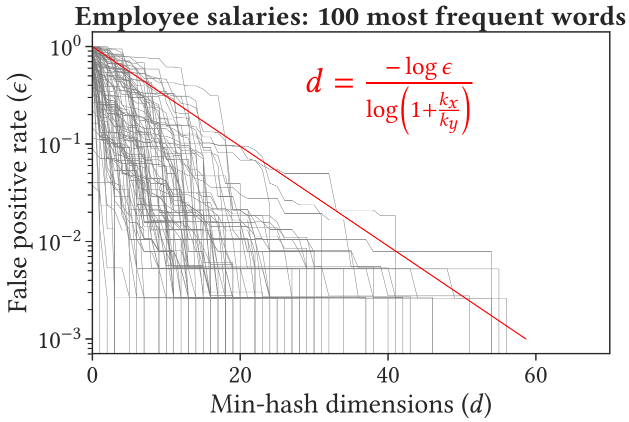


Fig. 9: **Number of dimensions required to identify inclusions.** Grey lines are the proportion of false positives obtained for the 100 most frequent words in the employee salaries dataset (H_0 corresponds to identifying categories that do not contain the given word). The red line represents the theoretical minimum dimensionality required to obtain a desired false positive rate (with $k_x/k_y = 0.125$, the inverse of the maximum number of words per category), as shown in Theorem 3.1.

TABLE 8: Median scores by dataset for XGBoost (d=30).

Datasets	One-hot + SVD	Similarity encoder	TfIdf + SVD	FastText + SVD	Gamma Poisson	Min-hash encoder
building permits	0.244	0.505	0.550	0.544	**0.570**	0.566
colleges	0.499	0.532	**0.537**	0.530	0.524	0.527
crime data	0.443	0.445	0.445	**0.446**	0.445	0.446
drug directory	0.971	0.979	0.980	**0.982**	0.980	0.981
employee salaries	0.880	0.905	0.892	0.901	**0.906**	0.900
federal election	0.135	0.141	0.144	**0.151**	0.146	0.146
journal influence	0.019	0.138	0.164	**0.221**	0.118	0.133
kickstarter projects	0.879	0.879	0.880	**0.880**	0.879	0.880
medical charge	0.904	0.905	**0.905**	0.904	0.904	0.904
met objects	0.771	0.790	0.789	**0.796**	0.791	0.794
midwest survey	0.575	0.635	0.646	0.636	0.651	**0.653**
public procurement	0.678	0.677	0.678	0.678	0.674	**0.678**
road safety	0.553	0.562	0.562	0.560	0.563	**0.566**
traffic violations	0.782	0.789	0.789	0.790	0.792	**0.793**
vancouver employee	0.395	0.550	0.530	0.509	0.556	**0.568**
wine reviews	0.439	0.671	0.724	0.657	**0.724**	0.679

preprocessing for the categorical variable was to transform all entries to lower case.

Cross-validation: For every dataset, we made 20 random splits of the data, with one third of samples for testing at each time. In the case of binary classification, we performed stratified randomization.

Performance metrics: Depending on the type of prediction task, we used different scores to evaluate the performance of the supervised learning problem: for regression, we used the R^2 score; for binary classification, the average precision; and for multi-class classification, the accuracy score.

Parametrization of classifiers: We used the scikit-learn [34] for most of the data processing. For all the experiments, we used the scikit-learn compatible implementations of XGBoost [31], with a grid search on the `learning_rate` (0.05, 0.1, 0.3) and `max_depth` (3, 6, 9) parameters. All datasets and encoders use the same parametrization.

Dimensionality reduction: We used the scikit-learn implementations of `TruncatedSVD` and `GaussianRandomProjection`, with the default parametrization in both cases.

Appendix B
Algorithmic considerations

B.1 Min-hash encoder

B.2 Gamma-Poisson factorization

Algorithm 1 requires some input parameters and initializations that can affect convergence. One important parameter is ρ, the discount factor for the fitting in the past. Figure 10 shows that choosing ρ=.95 gives the best compromise between stability of the convergence and data fitting in term of the Generalized KL divergence.

With respect to the initialization of the topic matrix $\mathbf{\Lambda}^{(0)}$, the best option is to choose the centroids of a k-means clustering (Figure 11) in a hashed version of the n-gram count matrix $\mathbf{F}$ in a reduced dimensionality (in order to speed-up convergence of the k-means algorithm) and then project back to the n-gram space with a nearest neighbors algorithm.

Appendix C
Additional results

C.1 Online Resources

Experiments are available in Python code at https://github.com/pcerda/tkde2019. Implementations and examples on learning with string categories can be found at http://dirty-cat.github.io. The available encoders are compatible with the scikit-learn's API.

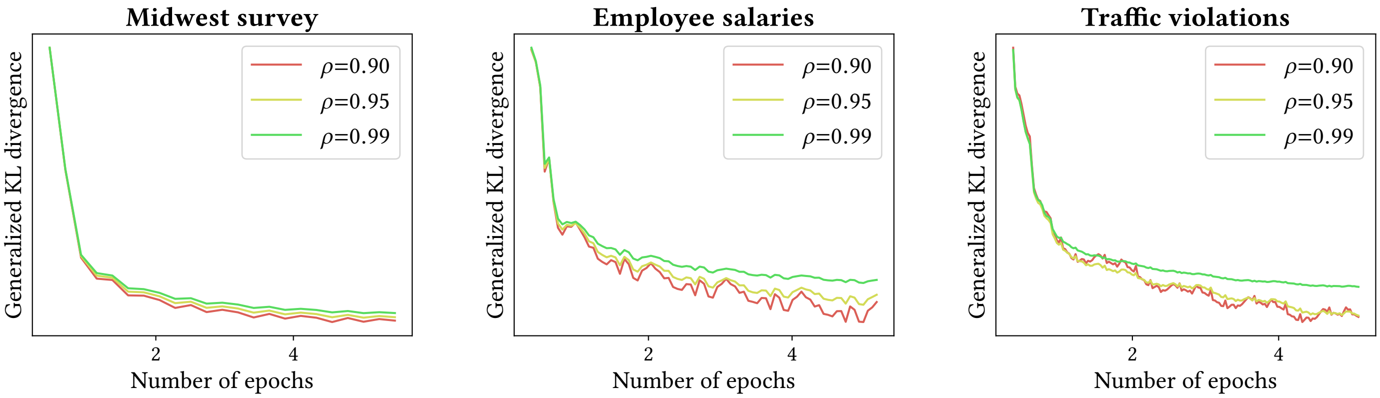


Fig. 10: Convergence for different ρ values (discount factor) for the Gamma-Poisson model.

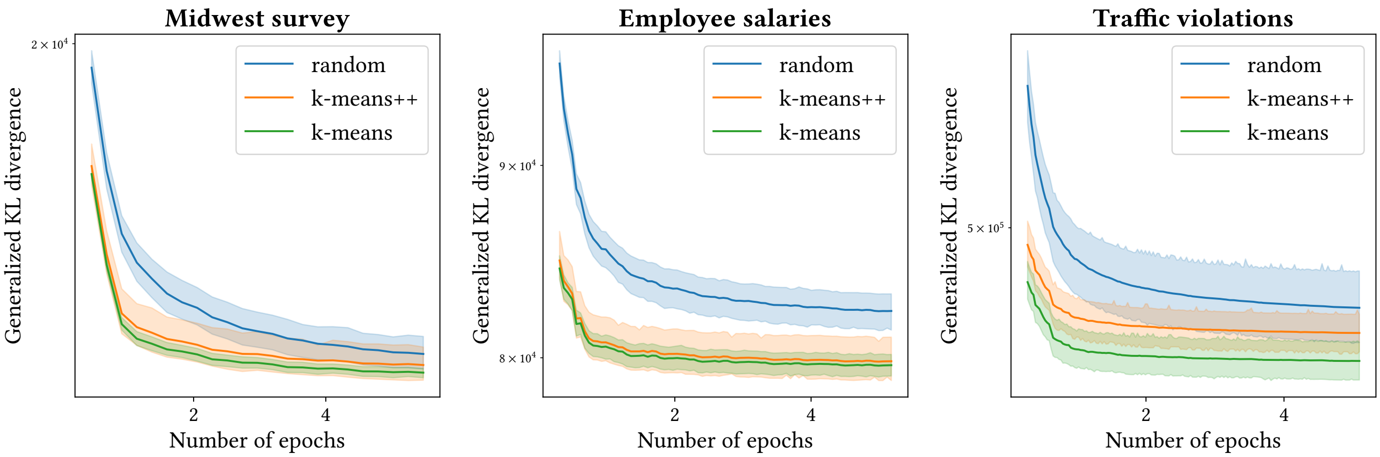


Fig. 11: Convergence for different initializations for the Gamma-Poisson model.

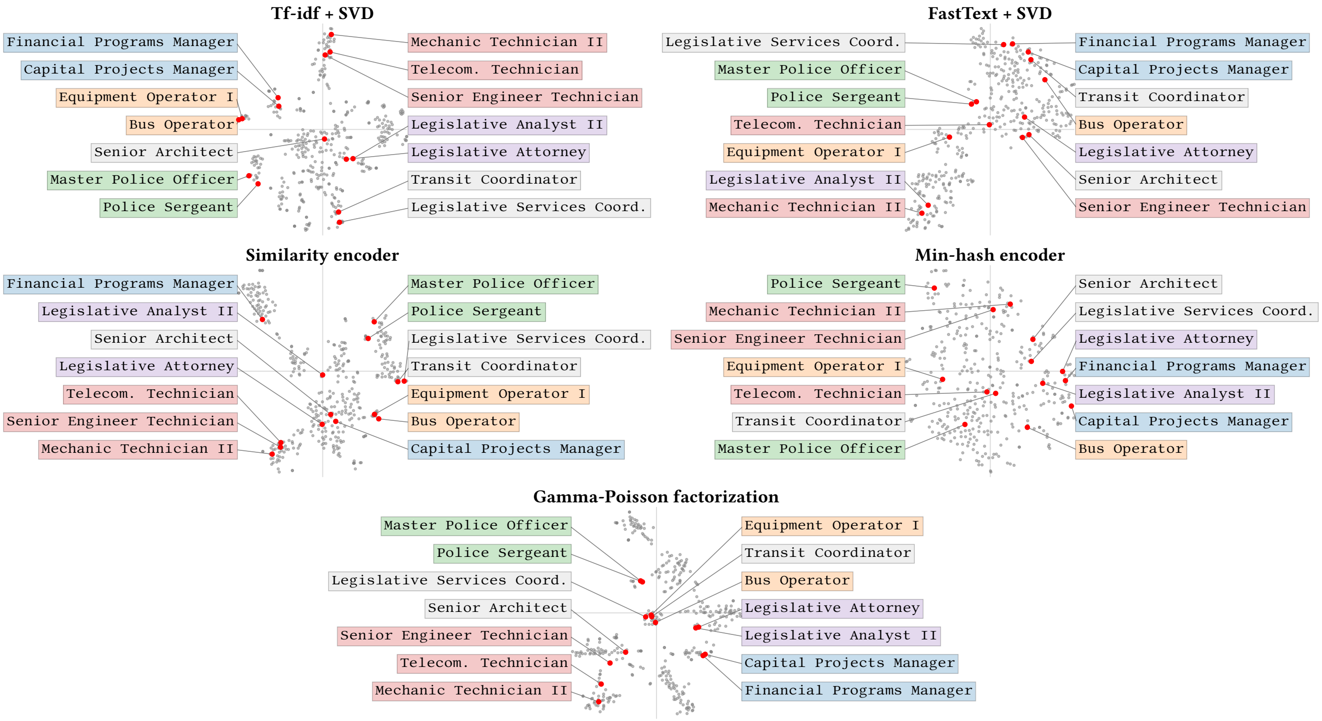


Fig. 12: Projection with t-SNE for different encoding methods for the categorical variable *Employee Position Title* in the *Employee Salaries* dataset. The original dimensionality for each encoding is d=10. Gray dots represent the rest of observed categories in the dataset.