Gem: Gaussian Mixture Model Embeddings for Numerical Feature Distributions

Hafiz Tayyab Rauf
Department of Computer Science,
University of Manchester
Manchester, UK.
hafiztayyab.rauf@manchester.ac.uk

Alex Bogatu
National Biomarker Centre, CRUK-MI,
University of Manchester
Manchester, UK.
alex.bogatu@manchester.ac.uk

Norman W. Paton
Department of Computer Science,
University of Manchester
Manchester, UK,
norman.paton@manchester.ac.uk

André Freitas
IDIAP Research Institute
Martigny, Switzerland.
Department of Computer Science,
National Biomarker Centre, CRUK-MI,
University of Manchester
Manchester, UK.
andre.freitas@manchester.ac.uk

ABSTRACT

Embeddings are now used to underpin a wide variety of data management tasks, including entity resolution, dataset search and semantic type detection. Such applications often involve datasets with numerical columns, but there has been more emphasis placed on the semantics of categorical data in embeddings than on the distinctive features of numerical data. In this paper, we propose a method called Gem (*G*aussian mixture model *em*beddings) that creates embeddings that build on numerical value distributions from columns. The proposed method specializes a Gaussian Mixture Model (GMM) to identify and cluster columns with similar value distributions. We introduce a signature mechanism that generates a probability matrix for each column, indicating its likelihood of belonging to specific Gaussian components, which can be used for different applications, such as to determine semantic types. Finally, we generate embeddings for three numerical data properties: distributional, statistical and contextual. Our core method focuses on numerical columns without using table metadata for context. However, the method can be combined with other types of evidence, and we integrate attribute names with the Gaussian embeddings to evaluate the method's contribution to improving overall performance. We compare Gem with several baseline methods for numeric only and numeric + context tasks, showing that Gem consistently outperforms the baselines on five benchmark datasets.

1 INTRODUCTION

Data repositories, such as data lakes and open government data, often contain substantial amounts of numerical data [20], which forms the backbone of various analytical and predictive models. Indeed, numerical data often outnumbers non-numerical and categorical data [18]. Applications such as semantic type detection of numerical data are thus important, but numerical data presents several challenges, including variability in data distributions (e.g., consider two columns both labeled *"weight"* in different datasets: one representing *"package weight"*, and another representing *"human weight"*; although both columns share the same label and numerical nature, their distributions and contexts differ significantly). Even when columns have similar distributions, their semantics might differ. For instance, a column representing *temperature* and another representing *test score* can have similar distribution shapes but different semantics. Figure 1 illustrates the challenge of comparing numerical distributions, where columns from different semantic types share similar values.

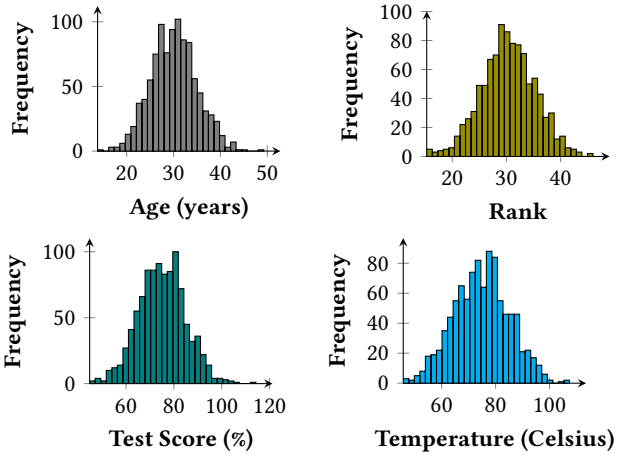


Figure 1: A histogram with a Kernel Density Estimate (KDE) overlay distributions of four numerical columns: Age, Rank, Test Score, and Temperature. Despite the similar distribution shapes — Age and Rank both showing a normal distribution around a mean of 30 and Test Score and Temperature around a mean of 75 — the semantic contexts differ significantly and refer to the different semantic types and units. For example, *"Age"* might be measured in years, *"Rank"* in a hierarchical position, *"Test Score"* as points out of 100, and *"Temperature"* in degrees, Fahrenheit and Celsius. These variations illustrate the complexity of semantic type detection of columns with different distributions. In this context, existing methods struggle to distinguish these overlapping columns. However, our proposal can effectively distinguish between these columns by focusing on their distributional properties.

The central contribution of this paper is a method for creating embeddings of numerical columns. In proposing a method that focuses on numerical data, we have two objectives:

(i) To provide an approach to the embedding of numerical columns where metadata is absent or unreliable. For example, it has been reported that 20% of web tables do not have column headers [9].
(ii) To provide more effective embeddings for numerical data that can then be combined with other contextual information. We note that state-of-the-art proposals for column annotation that perform well on non-numerical data, have been shown to perform much less well on repositories that contain significant amounts of numerical data [18]. In the absence of a single unifying embedding method that works well on both numeric and non-numeric data, the state-of-the-art involves hybrid proposals that include specialized techniques for numerical data [20]. All such hybrid techniques stand to benefit from advances in numerical data embedding.

Although several approaches handle numerical columns using bespoke deep learning techniques [8, 13, 16, 19, 29, 30, 34], they heavily rely on the context extracted from non-numerical data. Our contribution complements that of existing proposals, in providing a new approach to handling numerical features that can be combined with other features or used in isolation where contextual information is limited. Furthermore, existing methods often overlook the distributional differences between columns with similar column names but different values. For example, two columns representing temperature readings in different regions might have similar schemas but different distributions due to varying climates. Similarly, existing approaches may fail to capture fine-grained domain-specific information from numerical data distributions. For instance, financial transaction amounts and sales figures might overlap in certain ranges but differ in others, presenting specific challenges such as variability in data distributions and similar contextual information. Numerical columns often have diverse distributions, such as normal, skewed, or multimodal, which can be challenging to model accurately. Existing methods may struggle to differentiate between columns with similar value ranges but different underlying distributions. Additionally, many approaches rely heavily on contextual information from table names and neighboring columns, which might not always be available. This reliance can lead to misclassification when context is absent or incomplete.

In this paper, we aim to address challenges associated with numerical data, and propose an approach based on a Gaussian Mixture Model (GMM) [4, 23, 26] to identify data distributions existing in different columns. Gem focuses solely on numerical columns without utilizing context from table names or neighboring columns. However, we later incorporate context from column headers (attribute names) to investigate how numeric-only embeddings contribute to improvements in downstream tasks. We defined a signature mechanism to draw a probability matrix from each column, which shows the probability of a column belonging to a particular Gaussian component or distribution, which can be interpreted as a semantic type.

The contributions of this paper are:

(1) A method for producing embeddings for numerical columns that leverages GMMs to handle numerical distributions in tabular data. This approach utilizes the statistical properties of distributions and a unique signature method to form a probability matrix from Gaussian distributions, focusing exclusively on numerical data.
(2) An investigation into the contribution of embeddings produced from numerical values in combination with header information. This includes thoroughly analyzing the impact of integrating numerical data distributions and header embeddings using transformer models.
(3) A comprehensive comparative analysis of Gem against state-of-the-art bespoke methods that reveals that Gem consistently achieves superior performance, both when incorporating contextual information and when using only numerical values.

2 RELATED WORK

We review the literature on embeddings for numerical data in two categories: (i) approaches that employ GMMs and other mixture models, and (ii) numerical embedding methods for tabular data.

2.1 Mixture models for embeddings

Several approaches have adopted mixture models and other distributional techniques to encode numerical data via distributions. One notable method [8], focusing on tabular deep learning, proposes two mechanisms to encode numerical data: piecewise linear encoding (PLE) and periodic activation functions (PAF). PLE divides the numerical range into segments and fits linear functions within each segment, capturing non-linear relationships among numerical features. PAF maps numerical values to a higher-dimensional space using sinusoidal transformations, which helps capture periodic patterns. The evaluation reports that the custom embeddings can provide improved performance when included in a variety of architectures.

Another related approach embeds numeral contexts within traditional word embeddings, enhancing numeral understanding in text data applications [13], utilizing Self-Organizing Maps (SOM) and GMM to create numeral embeddings. Both SOM and GMM integrate numerical values and neighboring textual data to produce embeddings, which are calculated as weighted averages of prototype numeral embeddings determined by a similarity function and integrated into traditional word embeddings. The proposed approaches are shown to outperform baselines that do not incorporate numerical embeddings.

The MULTIHIERTT framework [35] handles numerical reasoning over hybrid datasets that integrate hierarchical tables and textual data. It is developed on MT2Net [35], combining numerical data and contextual information from table structures, such as headers and neighboring text. Unlike GMM and SOM [13], which mainly focus on numerical columns, MT2Net enhances its reasoning capabilities by utilizing the structural metadata from tables. The MULTIHIERTT framework works in two stages: a fact-retrieval module first gathers relevant numerical and textual information, followed by a reasoning module that applies both symbolic and arithmetic operations to integrate and reason over the retrieved data. Several other recent approaches adopt numerical reasoning methods to embed numerical data, each tailored to specific applications, including number decoding [31], automatic data generation [7], numerical attribute estimation [14, 17] and data-to-text generation [27].

2.2 Numerical embeddings for tabular data

In applications such as semantic column type detection, it is challenging to detect the type of a column solely from numerical

data, and as a result many proposals combine numerical features with other context. Our work aims to increase the extent to which numeric column values can inform such applications by considering distributions from the column values themselves.

Most existing approaches consider contextual evidence from neighboring columns, rows, tables and metadata, such as table descriptions or names. For instance, DICE [30] produces embeddings to reflect actual distances on the number line, utilizing contextual information from surrounding words to enhance numerical reasoning. This involves creating vector representations (embeddings) for numerical values such that the cosine similarity between these embeddings corresponds to the numerical difference between the values. For example, if two numbers in a column are 5 and 10, the DICE embeddings would ensure that the vector representations for these numbers are placed in such a way that the cosine similarity between them reflects the numerical distance of 5 units.

Research has often focused on leveraging as much contextual evidence as possible to improve column semantic type detection. For example, Sato [34] uses column values from numeric and non-numeric neighboring columns, table metadata, and global table features. Sato employs a structured prediction model that integrates evidence from individual columns and evidence involving adjacent columns to capture inter-column semantic relationships. These relationships define the embeddings within the same table, as certain semantic types often co-occur across columns. For instance, a *"Date"* column is semantically related to a *"Payment Due"* column in finance data. Similarly, Sherlock [12] is a multi-input neural network-based architecture that detects semantic data types by analyzing features extracted from numerical and non-numerical contexts. Sherlock utilizes metadata from column headers, adjacent textual data within the table, and numerical values. Sherlock composes features to generate comprehensive embeddings, including character distributions, word embeddings, and paragraph vectors. Unlike Sherlock, which focuses on intra-table context, RECA [29] extends the contextual scope by incorporating data from related tables, providing a broader contextual framework. RECA utilizes a graph neural network to integrate features from related tables, capturing complex inter-table relationships.

A recent proposal, Pythagoras [20], outperforms Sato [34], Sherlock [12] and Doduo [28] by focusing on a more holistic integration of numerical and non-numerical contexts. Pythagoras employs a Graph Neural Network (GNN) and constructs a heterogeneous graph to integrate various contextual signals, including table names, numerical data, and metadata from neighboring columns. Unlike earlier methods that either emphasize related tables (RECA [29]), intra-table relationships (Sherlock [12]), or token-level interactions (Doduo [28]), Pythagoras synthesizes these contexts within a unified graph structure.

In terms of numerical embeddings, most existing approaches focus on statistical properties of numeric columns rather than distributional properties, e.g., Pythagoras [20] and Sato [34]. Conversely, *ad hoc* methods that capture distributional properties based on Gaussian Mixture Models (GMMs) or similar techniques have proven effective for other numerical tasks, such as clustering and density estimation. However, these approaches have not been widely generalized for data management tasks within tabular data. As a result, there remains a gap in fully leveraging the numerical features of tabular data. Existing methods do not focus on drawing distributions from numerical columns and clustering them based on similar distributions, missing the opportunity to utilize the inherent properties of numerical data. Our approach addresses this gap by extracting numerical data distributions.

3 GEM-BASED SIGNATURES FOR NUMERICAL COLUMNS

Building on the related work, Gem seeks to address the limitations of existing methods by maximizing the use of numerical data distributions to generate embeddings. Our proposed method uses numerical data distributions to identify and cluster columns with similar semantic types. Gem provides a unique approach to tackling numeric columns by grouping distributions (in other words, histograms) from tables that refer to the same semantic type. Additionally, it is designed to combine the numerical embeddings with other types of evidence, as explored in the experiments (see Section 4.2). This process involves extracting numerical data from tabular data, fitting a GMM to capture their distributional characteristics, and then calculating a probability matrix for each column based on these distributions. Gem takes stacks of numerical columns and uses a signature mechanism to predict the probabilities of each column belonging to corresponding Gaussian components. These probabilities are then aggregated for each column to form a likelihood distribution across the different components, effectively capturing the underlying numerical characteristics. Gem then calculates additional statistical features for each column and integrates contextual information from headers. These combined features enhance clustering, distinguishing similar distributions based on distributional, statistical, and contextual properties. In Figure 2, we illustrate the transformation of numerical columns into final embeddings. In the following, we describe Gem for producing embeddings from numerical columns.

3.1 Modelling Value Distributions Using GMMs

Assume we have a dataset comprising n columns, each representing a distinct set of numerical values. The primary representational goal is to capture the underlying distributions from which the values in each column are drawn. GMM offers an approach to this problem, leveraging the ability to deliver an expressive probabilistic model to identify the latent Gaussian distributions that collectively describe the numerical values.

A GMM is a probabilistic model representing a mixture of m Gaussian distributions. GMM represents the dataset's probability density function (pdf) as a weighted sum of multiple Gaussian distributions. The pdf of a GMM is given by [23, 26]:

$$p(x) = \sum_{j=1}^{m} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j) \tag{1}$$

where:

- x is a numeric value.
- π_j is the mixing coefficient for the j-th Gaussian component, with $\sum_{j=1}^{m} \pi_j = 1$.
- $\mathcal{N}(x|\mu_j, \Sigma_j)$ is the Gaussian distribution with mean μ_j and covariance Σ_j.

To estimate the parameters (μ_j, Σ_j and π_j) of the GMM, we employ the Expectation-Maximization (EM) algorithm [4], which iteratively optimizes these parameters to maximize the likelihood of the observed numeric columns. The EM algorithm includes two main steps: the Expectation step (E-step) and the Maximization step (M-step). Initially, the means μ_j, covariances Σ_j, and weights

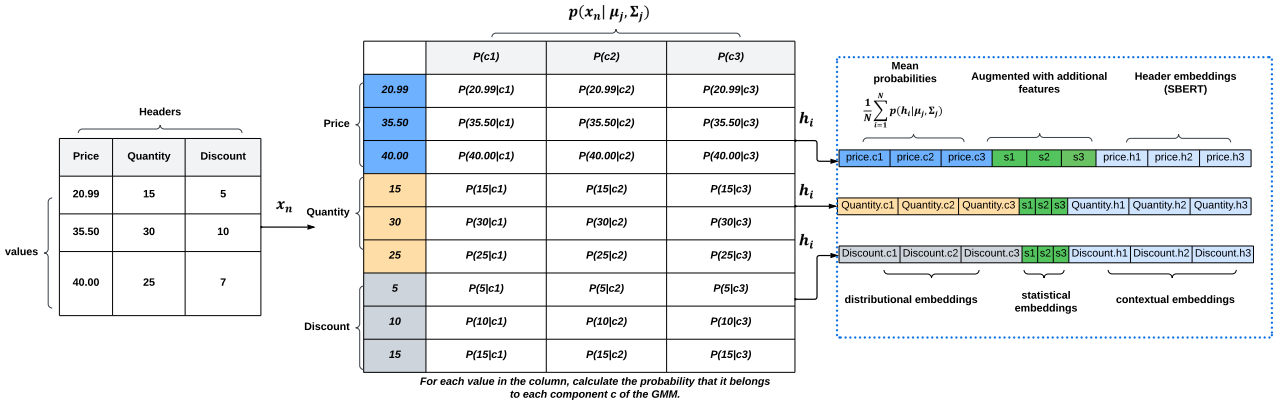


Figure 2: The process of transforming a table with three numeric columns *(Price, Quantity, Discount)* into a final embedding matrix. First, the GMM is fitted to the values in each column. For each value x_n in a column, the probability $p(x_n \mid \mu_j, \Sigma_j)$ that it belongs to each component C_j of the GMM is calculated using Equation 6 where μ_j and Σ_j are the mean and covariance matrix of component j, respectively. Next, the mean probabilities for each component are computed: $\mu_{C_j} = \frac{1}{N}\sum_{i=1}^{N} p(h_i \mid \mu_j, \Sigma_j)$ where N is the number of values in the column. These mean probabilities are augmented with additional statistical features $(s1, s2, s3...sn)$. Simultaneously, the column headers are transformed into embeddings using the SBERT model. Finally, the normalized probability matrix (value embeddings) and the normalized SBERT embeddings (header embeddings) are combined to form the final embedding matrix for the table. The final embedding vector for each column includes the distributional embeddings using GMM, the statistical embeddings using data properties, and the contextual embeddings from headers, resulting in a comprehensive representation of the column data.

π_j are initialized randomly. In the E-step, the responsibilities $\gamma(z_{nj})$ are calculated, which represent the probability that a data point x_n belongs to the j-th Gaussian component:

$$\gamma(z_{nj}) = \frac{\pi_j \mathcal{N}(x_n \mid \mu_j, \Sigma_j)}{\sum_{k=1}^{m} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)} \tag{2}$$

In the M-step, the parameters are updated based on the responsibilities computed in the E-step:

$$\mu_j = \frac{\sum_{n=1}^{N} \gamma(z_{nj}) x_n}{\sum_{n=1}^{N} \gamma(z_{nj})} \tag{3}$$

$$\Sigma_j = \frac{\sum_{n=1}^{N} \gamma(z_{nj})(x_n - \mu_j)(x_n - \mu_j)^T}{\sum_{n=1}^{N} \gamma(z_{nj})} \tag{4}$$

$$\pi_j = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nj}) \tag{5}$$

This iterative process continues until convergence, typically when the change in the likelihood of the data given the parameters falls below a pre-defined threshold. In our case it is the default value of 1e-3. Once we obtain Gaussian components from the GMM for each numeric value, we use the Gem signature mechanism to compute the likelihood that each column belongs to a particular Gaussian component.

3.2 Gem Signature Mechanism

In this step, Gem treats all numerical values from the columns as a single stack (one-dimensional array) of numeric values rather than individual columns. Here, signatures refer to feature vectors extracted from each column, which capture essential characteristics for analysis. For each data point x_n, we compute the probability of it being generated by each Gaussian component j using the fitted parameters of the GMM (see Figure 2). This is done using pdf [23, 26]:

$$p(x_n \mid \mu_j, \Sigma_j) = \frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x_n - \mu_j)^T \Sigma_j^{-1} (x_n - \mu_j)\right) \tag{6}$$

Using these probabilities, we construct a probability matrix P, where each element P_{nj} represents the responsibility $\gamma(z_{nj})$ computed during the E-step. The matrix P thus encodes the likelihood of each data point belonging to each Gaussian component, summarizing the distributional features captured by the GMM.

In addition to these GMM-derived probabilities, we extract several statistical features from each numeric column to capture the statistical aspects of the column's distribution. These features were selected by systematically evaluating the Pythagoras feature set [20]. We only focus on features applicable to numerical values. Each feature's correlation with the Gem embeddings was tested, and only those with high correlation were retained. The select features includes:

- *Unique count*: Reflects the variety of distinct values in the column, indicating whether the data is largely repeated.
- *Mean*: Representing the average value in the column.
- *Coefficient of variation (CV)*: A normalized measure of spread that indicates the relative dispersion of values.
- *Entropy*: Quantifies the degree of uncertainty in the data distribution.
- *Range*: The difference between the maximum and minimum values.
- *Percentiles (10th and 90th)*: Highlight the lower and upper bounds to provide insights into the data's distribution.

Mathematically, let $\mathbf{f}_i$ represent the vector of additional features for the i-th column. Standardization transforms each feature vector $\mathbf{f}_i$ to $\tilde{\mathbf{f}}_i$, where:

$$\tilde{\mathbf{f}}_i = \frac{\mathbf{f}_i - \mu(\mathbf{f})}{\sigma(\mathbf{f})} \tag{7}$$

where $\mu(\mathbf{f})$ and $\sigma(\mathbf{f})$ are the mean and standard deviation of the feature vector, respectively. These standardized feature vectors are then integrated with the mean probabilities derived from the GMM. For the i-th column, let $\mathbf{m}_i$ represent the mean probability vector of length K (number of Gaussian components). The augmented feature vector $\mathbf{a}_i$ is formed by concatenating $\mathbf{m}_i$ and $\tilde{\mathbf{f}}_i$:

$$\mathbf{a}_i = [\mathbf{m}_i \,\|\, \tilde{\mathbf{f}}_i] \tag{8}$$

Finally, each augmented feature vector $\mathbf{a}_i$ is normalized to ensure comparability across different columns, resulting in the final row of the probability matrix P_i:

$$P_i = \frac{\mathbf{a}_i}{\|\mathbf{a}_i\|_1} \tag{9}$$

where $\|\mathbf{a}_i\|_1$ denotes the L1 norm of $\mathbf{a}_i$. This integration enhances the descriptive power of the probability matrix by combining the probabilistic information (distributional embeddings) from the GMM with statistical characteristics of the columns.

Why combine distributional and statistical features? The notion behind combining the statistical and distributional features of a column is to distinguish fine-grained patterns of similar numerical columns with different semantic types. For instance, consider two columns $\mathbf{x}_1$ (income) and $\mathbf{x}_2$ (heights). Distributional features, extracted using GMM probabilities $p(x_{i,j} \mid \mu_k, \Sigma_k)$ (Equation 6), learn the clustering patterns within the column, such as income (low, medium, high) in $\mathbf{x}_1$, forming $\mathbf{m}_1 = [m_{1,1}, m_{1,2}, \ldots]$. Statistical features $\tilde{\mathbf{f}}_1 = [f_{1_1}, f_{2_1}, \ldots]$, such as mean $f_{1_1} = \frac{1}{N_1}\sum x_{1,j}$ and variance $f_{2_1} = \frac{1}{N_1}\sum (x_{1,j} - f_{1_1})^2$, represent global trends like average income. Gem combines both feature sets into $\mathbf{a}_1 = [\mathbf{m}_1 \,\|\, \tilde{\mathbf{f}}_1]$ (Equation 8), normalizes them as $P_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_1}$ (Equation 9), and provides embeddings with fine-grained distributional clustering and broader numerical properties. The generated embedding can then distinguish columns like $\mathbf{x}_1$ (income) and $\mathbf{x}_2$ (heights), even when their value distributions show overlapping patterns.

3.3 Header embeddings

We obtain the distributional embeddings from the GMM and capture the statistical properties from the column values. Additionally, we incorporate contextual information from column headers. This step, while not always necessary, proves helpful when the distributional embeddings are highly dense and similar. The context provided by the column headers helps to disambiguate meaningful differences among columns. Our experiments (see Section 4.2) report results with and without headers, demonstrating Gem's flexibility and effectiveness. We use Sentence-BERT (SBERT) [25] to embed column headers, which captures the semantic meaning of the headers in a high-dimensional space. Let $\mathbf{s}_i$ represent the SBERT embedding for the i-th column header. To ensure compatibility with the value embeddings, the SBERT embeddings are also normalized:

$$\mathrm{S}_i = \frac{\mathbf{s}_i}{\|\mathbf{s}_i\|_1} \tag{10}$$

where $\|\mathbf{s}_i\|_1$ denotes the L1 norm of $\mathbf{s}_i$.

Finally, the normalized probability matrix $\mathbf{P}_i$ (value embeddings) and the normalized SBERT embeddings S_i (header embeddings) are concatenated to form the final combined embedding vector $\mathbf{C}_i$ for each column:

$$\mathbf{C}_i = [\mathbf{P}_i \| \mathrm{S}_i] \tag{11}$$

where $\|$ denotes the concatenation operation. This final embedding vector $\mathbf{C}_i$ includes the probabilistic and semantic information, providing a joint representation of each column in the table for downstream tasks (attribute name + represented value distribution). E.g., in clustering, each data point is assigned to the Gaussian component with the highest responsibility:

$$\text{Cluster}(x_n) = \arg\max_j C_{nj} \tag{12}$$

In addition to the contextual embeddings and probabilistic representations, we also aggregate standardized statistical features $\tilde{\mathbf{f}}_i$. These standardized features $\tilde{\mathbf{f}}_i$ are integrated with the probabilistic embeddings $\mathbf{P}_i$ and the normalized SBERT embeddings S_i, forming the final aggregated embedding vector $\mathbf{C}_i^{\text{agg}}$:

$$\mathbf{C}_i^{\text{agg}} = \left[\mathbf{P}_i \| \mathrm{S}_i \| \tilde{\mathbf{f}}_i\right] \tag{13}$$

This final aggregated embedding $\mathbf{C}_i^{\text{agg}}$ combines probabilistic, semantic, and standardized statistical information, providing a rich representation for downstream tasks.

By integrating the GMM with extracted signatures and leveraging the resulting probability matrix, we establish a comprehensive framework for managing and analyzing datasets composed of numerical columns. We provide pseudocode to generate the final embedding matrix from numerical columns in Algorithm 1.

4 EVALUATION

4.1 Datasets

We use five widely used datasets, which include Sato Tables [34], Git Tables [11], Google Dataset Search (GDS)[1], Web Data Commons (WDC)[1] and BiodivTab [1] to evaluate Gem[2]. We select numeric columns from all five datasets. The datasets have been selected for their abundance of numeric columns, rich variability in data distributions, and diverse column semantics. Dataset details are given below and in Table 1.

- **Sato Tables**, part of the VizNet dataset, includes numeric columns representing attributes such as population counts, GDP values, and personal statistics. Many numeric columns have similar distributional characteristics but different semantic types. For instance, columns labeled as "*age*," "*duration*," "*weight*," "*order*," and "*position*" exhibit similar numeric distributions, yet they have different semantic meanings. The distributional and statistical similarity between these columns is greater than 0.90, indicating their contextual (header embeddings) meanings differ significantly despite their numeric resemblance.
- **Git Tables** is a large-scale semantic type detection dataset consisting of relational tabular data from a wide range of domains. The column annotations were obtained from Schema.org and DBpedia. Git Tables represents a particularly challenging setting without additional context descriptions. For example, detecting the semantic type of a

[1] https://github.com/PierreWoL/SILM
[2] https://github.com/hafizrauf/Gem

Table 1: Dataset statistics related to the number of numeric columns and ground truth (GT) clusters. The numbers in brackets indicate the columns and semantic types in the GT, which were derived by refining coarse-grained annotations into fine-grained ones for both the GDS and WDC datasets

	GDS	WDC	Sato Tables	Git Tables	BiodivTab
# Columns	2491 (2117)	2852 (5678)	2231	459	384
#GT clusters	86 (96)	147 (325)	12	19	44

Algorithm 1 Generating the final embedding matrix from numeric columns

Require: Dataset with n numeric columns $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and headers $\{h_1, h_2, \ldots, h_n\}$
Ensure: Final embedding matrix $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\}$
Initialize lists: column_headers $\leftarrow$ [], additional_features $\leftarrow$ []
for each column i in the dataset **do**
  Extract column values $\mathbf{x}_i$ and header h_i
  Append h_i to column_headers
  Calculate additional statistical features $\mathbf{f}_i$
  $\mathbf{f}_i \leftarrow [f_{1_i}, f_{2_i}, \ldots, f_{m_i}]$
  Append $\mathbf{f}_i$ to additional_features
end for
Fit GMM with m components to all column values: GMM_model $\leftarrow$ GMM(m)
for each column i **do**
  for each data point $x_{i,j}$ in $\mathbf{x}_i$ **do**
    Compute the probability $p(x_{i,j} \mid \mu_k, \Sigma_k)$ using Equation 6
  end for
  Compute the mean probabilities for each component
  Form the augmented feature vector $\mathbf{a}_i$ using Equation 8:
  Normalize $\mathbf{a}_i$ to form P_i using Equation 9:
  Append P_i to the probability matrix
end for
Encode column headers using SBERT: $\mathbf{s}_i \leftarrow$ SBERT_model.encode(h_i)
Normalize SBERT embeddings using Equation 10:
for each column i **do**
  Concatenate P_i with $\mathbf{S}_i$ to form the final combined embedding vector $\mathbf{C}_i$ using Equation 11:
  Append $\mathbf{C}_i$ to the final embedding matrix
end for
Output the final embedding matrix

column given the values *[153, 228, 125, 273, 319, 139, ...]* to be *duration, height, length* or *volume*.

- **BiodivTab**[3] [1] forms from a wide variety of real-world biodiversity datasets and was used in the SemTab2021[4] challenge to map semantic classes from knowledge bases to table columns. BiodivTab contains several challenges, including columns with nested entities, no contextual information, and diverse numerical columns.
- **WDC** (Web Data Columns) includes numeric columns extracted from web data, such as product prices, stock quantities, and review scores. It captures a broad spectrum of e-commerce and social media numeric data. WDC attribute names are categorically coarse-grained. For example, columns like *Score_Cricket, Score_Rugby, Score_Football* are semantically annotated with *Score*. However, we transform the annotation from coarse-grained to fine-grained to better capture the different distributions of each column. For instance, while both *Score_Cricket* and *Score_Rugby* represent game scores, they have distinct contexts and distributions. Cricket scores tend to be much higher due to the nature of the game, while Rugby scores follow a different scale. Simply classifying them as *Score* would overlook these differences. Further details of the column annotation process are provided in Section 4.1.1.
- **GDS** (Google Dataset Search) is a platform developed to help researchers discover openly available datasets on the web. We used the GDS dataset, where the authors manually curated specific tables for data discovery tasks. This dataset has been refined to a fine-grained level from its original form, ensuring that each table represents distinct and specific concepts for more precise column annotation. For example, instead of having a general *"power"* column, we annotate columns with more granularity, such as *"engine_power_car"* and *"battery_power_device"*, which capture contextually relevant information about the power of car engines and electronic devices, respectively.

The following criteria guided the selection of the above datasets:

- **Numerical Columns Specificity:** Each dataset contains a significant number of columns that are composed entirely of numerical data (see Table 4).
- **GT clusters with detailed refinement**: Another criterion is the availability of GT clusters that categorize different semantic types. The initial annotations for the GDS and WDC datasets were refined from broader, coarse-grained types to more specific, fine-grained semantic categories. For instance, the GDS dataset refined clusters from 86 to 96 distinct types, while WDC refined 147 clusters into 325 semantic types.
- **Diversity across datasets:** The selected datasets provide a broad spectrum of semantic types representing different domains.

4.1.1 Data Annotation: From Coarse-Grained to Fine-Grained Labels. We use the following criteria to convert coarse-grained labels into fine-grained labels for both WDC and GDS datasets, as both datasets have coarse-grained annotations as ground truth. For example, the score of a *cricket* and the score of a *football* game can be classified under the supertype "*score*". The re-annotation was performed manually, guided by the criteria outlined below:

- Two columns should have the same annotation if they describe the same domain. Applying the equality (=) operator to values from different columns should be meaningful.

[3]https://github.com/fusion-jena/BiodivTab
[4]https://www.cs.ox.ac.uk/isg/challenges/sem-tab/2021/index.html

For example, it is not meaningful to compare a *volume* with an *area* as they have different units.
- Two values must describe the same real-world concept at a comparable scale and within the same context to be equivalent. For instance, the *height* of a person and the *height* of a building, while both referring to '*height*', are not equivalent because they belong to different real-world domains and scales.
- If subcategories exist, they must be applied at the appropriate level of specificity. For example, the *score* achieved by a *baseball player* cannot be equated to the *score* achieved by a *golf player*, even though both fall under the super category of *score*.

The manual re-annotation process was carried out systematically by following these criteria. We provide coarse-grained and fine-grained ground truth annotations in GitHub[5].

4.1.2 Evaluation Metrics. We evaluated Gem for two downstream tasks: semantic table annotation and column clustering.

Two metrics are used to evaluate column semantic type annotation, *Precision at K* and *Normalized Discounted Cumulative Gain* (nDCG). Precision at K is used to evaluate the performance of ranking algorithms, and computes the fraction of the top K results that are correct. Thus Precision at K is defined as:

$$Precision@K = \frac{TP@K}{K} \tag{14}$$

where $TP@K$ is the number of true positives in the top K. In our experiments, the true positive columns are those with the correct type annotation, and the top K are the K columns with embeddings that are most similar to a given column. In the experiments, K is set to the total number of columns with the same semantic type in the GT.

nDCG considers the ranked list's relevance and positions of retrieved columns. It can be defined as:

$$nDCG@K = \frac{DCG@K}{IDCG@K} \tag{15}$$

where, $DCG@K$ (Discounted Cumulative Gain) computes the cumulative relevance of the top K columns, and $IDCG@K$ represents the maximum possible $DCG@K$ if all true positives are ranked correctly. We consider a column is relevant if it has the same semantic label as the GT label. Unlike $Precision@K$, $nDCG@K$ takes into account the relative order of TPs in the ranked list.

We used two well-known evaluation metrics for column clustering: Accuracy (ACC) [33] and Adjusted Rand Index (ARI) [32]. ACC measures the proportion of correctly clustered columns and ranges from 0 to 1. The ARI score ranges from -1 to 1, where negative values suggest worse-than-random labeling, 0 indicates random labeling, and 1 indicates a perfect match.

4.1.3 Baselines. We selected two types of baseline method to compare with Gem. The first relies solely on numerical data. The second includes methods that consider context from headers, table names and neighboring columns, such as Pythagoras, Sato, Doduo and Sherlock. We could not directly compare these methods to Gem because Gem is purely designed to handle numerical features. To ensure a fair comparison, we modified and re-implemented Pythagoras, Sato, Doduo and Sherlock to retain their core statistical features and header information, aligning their focus on numerical data. However, we explicitly excluded all other contextual information, such as table names, neighboring non-numerical columns, and inter-column relationships, so that the comparison was based solely on the numerical embedding capabilities of the methods. We acknowledge that this re-implementation of Pythagoras, Sato, Sherlock and Doduo gives rise to a simplified and modified version of the original methods. However, it illustrates the specific impact of removing several contextual elements, allowing for a more precise comparison with Gem's context-independent approach.

- ***Piece-wise Linear Encoding (PLE)*** [8] transforms numeric data into a series of linear segments, each representing a portion of the data range. This method simplifies complex non-linear relationships into manageable linear parts by dividing the numeric range into intervals and applying linear transformations within each segment.
- ***Periodic Activation Functions (PAF)*** [8] introduce oscillatory behavior into neural network layers, making them adept at capturing repeating patterns in numeric data. This model with periodic function efficiently learns and represents cyclical patterns and can detect semantic types that exhibit periodic behavior.
- ***Squashing_GMM*** [13]: This method begins by squashing numeric values into log space following a prototype induction using GMM to identify the clusters, each representing a prototype. Similarity functions then measure how closely numeric columns match these Gaussian components.
- ***Squashing_SOM*** [13]: This method is similar to Squashing_GMM, except for the prototype induction part, where SOM projects the log-transformed data onto a grid of lower dimensionality while preserving its topological structure, inducing prototypes representing data clusters.
- **Kolmogorov-Smirnov (KS) statistic** [22]: The KS statistic is particularly relevant in this context because it measures the maximum difference between the cumulative distribution functions (CDFs) of the empirical data and several theoretical distributions, such as normal [6], uniform [5], exponential [2], beta [15], gamma [10], lognormal [21], and logistic [15]. We evaluate how well the numerical data in columns aligns with these reference distributions; we generate features that capture the underlying semantic type of the columns because different semantic types exhibit unique distributional patterns, and the KS statistic helps identify these patterns accurately.
- ***Sherlock*** [12]: We compare Sherlock with Gem because it extracts statistical features from numerical columns, such as mean, variance, skew and kurtosis, which align with Gem's focus on numerical data. To ensure a fair comparison, we augment these statistical features with SBERT-generated embeddings from column headers, similar to Gem's use of header information. Sherlock's model processes these combined features using dense layers with dropout and a softmax layer.
- ***Sato*** [34]: We also compare Sato, which is an enhancement in Sherlock. To maintain fairness, we exclude Sato's global and local context features, which rely on neighboring nonnumerical columns, since Gem does not utilize the nonnumerical global context. In our implementation of Sato, we focus on single-column data, extracting the same statistical features as Sherlock and combining them with SBERT embeddings from the headers. These combined features are processed in Sato's neural network model.

[5]https://github.com/hafizrauf/Gem

Overall, we extract statistical features in both implementations (Sherlock and Sato) and combine them with SBERT embeddings before processing them through their respective training architectures to obtain embeddings.

- ***Pythagoras*** [20]: Pythagoras uses a graph representation of tables to capture both numerical and contextual information, such as table names and neighboring columns. The model combines pre-trained language models for initial encoding with specialized subnetworks for numerical features. In line with Gem's focus on numerical data and headers only, we re-implemented Pythagoras in a context-reduced version, where only header data was considered, excluding table names and neighboring columns. Additionally, we retained the same statistical features selected for Gem.
- ***Doduo*** [28] is a pre-trained Transformer-based model used for multi-task learning to predict column types and relations. Doduo considers non-numerical context, such as textual data from cell values, and uses a multi-column approach with attention mechanisms to capture fine-grained token-level interactions among cells within the same table. However, in our implementation, we used Doduo's context-reduced version (single-column version) to generate embeddings for numeric columns and headers, relating solely to their intrinsic data.

To differentiate the original versions of Pythagoras, Sherlock, and Sato from our modified simple implementations, we called them Pythagoras_SC, Sherlock_SC, Sato_SC, and Doduo_SC, where SC indicates *Single-Column*. This shows that our implementations work with individual numerical columns without relying on multi-column and table-wide context. In the original approaches, these methods use additional information, such as neighboring columns and metadata. However, in our adapted versions, we remove this extra context to focus exclusively on the features of single-column numerical data.

4.1.4 Parameter Setting. The number of Gaussian components does not significantly impact Gem's overall performance (see ablation study in Section 4.4). Through comprehensive experimentation, we found that each column generally exhibits between 5 to 10 distinct distributions, and further increasing the number of components beyond this range does not contribute to performance improvement. Specifically, using more than 10 Gaussian components per column leads to model complexity without corresponding gains in accuracy. However, we determine each dataset's optimal number of components using the Bayesian Information Criterion (BIC). The BIC results showed consistent performance across 5 to 100 components, with minimal fluctuations. To maintain consistency, we used 50 Gaussian components for all our analyses. In baselines, specifically Squashing_GMM [13], we use the same number of components as used in Gem; in Squashing_SOM [13], PLE [8] and PAF [8], we use 50 prototypes, bins, and frequencies, respectively. Additionally, we initialize the EM algorithm 10 times to increase the likelihood of finding the global optimum, ensuring robust convergence and avoiding local minima.

4.2 Results and Discussion

4.2.1 Numeric-Only Results. Table 2 shows the experimental results of Gem compared to the baselines, considering numeric-only data across all five datasets. As the results for *precision* and *nDCG* are similar, in the text when we discuss numbers, we always refer to *precision*. We observe the following:

(1) **Gem consistently outperforms the baseline methods when considering numeric columns, achieving the highest average precision in all datasets**. Notable improvements relative to the best baseline are in Sato Tables (0.06), Git Tables (0.03), and GDS (0.03), demonstrating its ability to handle diverse numeric data distributions.
(2) **Baseline methods, including PLE, PAF and the KS statistic, struggled to differentiate between columns with superficially similar value ranges across all datasets.** For example, the columns labeled *'Rating' [3.6, 3.8, 3.9, 3.9, 3.6, ...]* and *'Weight' [1.0, 1.0, 1.4286, 1.25, 1.0957, 2.5, ...]* were incorrectly identified as highly similar (as evidenced by high cosine similarity in the embeddings produced by PLE and PAF, and low KS statistic values), despite having distinct value distributions and underlying semantics. In contrast, Gem distinguished between these columns, correctly classifying them as true negatives. This demonstrates Gem's superior ability to capture and identify semantic differences based on the underlying value distributions of the columns.
(3) **Gem better accounts for distributional variations in detecting column semantic types.** For instance, Gem correctly identifies (true positives) the top 10 neighbors of the column *'Mileage'* with values *[5, 117000, 92000, 500...]* as *'Mileage'* on GDS. However, with Squashing_GMM and KS statistic, the top 10 neighbors are columns about *'Rank'* and *'Year'* due to the overlap in value ranges, even though these columns represent different domains. The additional statistical features combined with distributional properties in Gem effectively identify the fine-grained components among numeric columns.
(4) **Gem accurately distinguishes between *width* and *length* columns in contrast with Squashing_SOM and Squashing_GMM using the Git Tables dataset.** Gem achieves a precision of 0.61 compared to 0.41 with Squashing_SOM and 0.39 with Squashing_GMM. For example, for the column *'width:[5, 256, 5, 256, 5.12]'* and *'length: [256, 5, 256, 5, 256, 109.71, 51.2]'*, Gem identifies subtle distributional variances in both columns, such as ranges and proportions of values, unlike Squashing_SOM and Squashing_GMM, which consider the cluster properties. Squashing_SOM also reduces overlapping data to a grid representation, which overlooks the fine-grained patterns in distributions, while Squashing_GMM depends on overall variance, which is similar in both columns.
(5) **For Sato Tables, Gem tends to misclassify columns with similar value distributions, resulting in overlapping errors.** For instance, *'weight'* columns with values *[32.2, 34.3]* were consistently misclassified as *'age'* due to their repetitive values (e.g., *[32, 30, 30, 31, 31, 31, 30, 31, 31, 31]*). Values of both columns have similar means, variances, and percentile ranges, even with different lengths (due to repeated cells). Gem utilizes GMM in its signature mechanism to capture the distributions, and these comparable statistical properties result in an overlapping probability matrix. We observe that Gem struggles when columns share close numeric ranges (30-34 in the above example) and experiences repetitive patterns. These patterns

Table 2: Average precision and nDCG score on GitTables, Sato Tables, GDS, WDC and BiodivTab datasets on numeric-only data. To ensure a consistent comparison, we used the coarse-grained versions of all datasets. Gem (D+S) represents the distributional and statistical components of Gem.

Methods	Avg precision					Avg nDCG				
	Git Tables	Sato Tables	WDC	GDS	BiodivTab	Git	Sato	WDC	GDS	BiodivTab
Squashing_GMM[13]	0.25	0.28	0.19	0.29	0.72	0.30	0.30	0.20	0.34	0.80
Squashing_SOM [13]	0.19	0.31	0.14	0.28	0.53	0.22	0.33	0.14	0.31	0.58
PLE [8]	0.19	0.11	0.18	0.11	0.72	0.23	0.12	0.18	0.13	0.80
PAF [8]	0.24	0.23	0.19	0.34	0.72	0.29	0.14	0.20	0.39	0.79
KS statistic [22]	0.21	0.21	0.02	0.21	0.48	0.24	0.22	0.03	0.23	0.52
Gem (D+S)	**0.28**	**0.37**	**0.21**	**0.37**	**0.74**	**0.32**	**0.41**	**0.21**	**0.42**	**0.81**

generate a similar entropy, and their probability representations in the matrix become identical, which causes the mis-classification. Conversely, despite having a low overlap in value similarities, PLE and PAF methods still resulted in mis-classifications. For example, *'year'* columns with values ranging from *[1980, 1981, 1982, ..., 2012]* were misclassified as *'duration'* with non-similar values like *[214.0, 306.0, 248.0, ...]* or *'age'* with values *[24, 38, 36, ...]*. This indicates that PLE and PAF struggle to differentiate columns with distinct value ranges due to insufficient semantic differentiation.

(6) **For BiodivTab, PLE struggled to cluster *"year"* columns effectively and did not manage to group columns with varying ranges into** $TP@K$**.** In contrast, Gem showed a stronger ability to handle this. For instance, Gem successfully grouped multiple *"year"* columns from different tables, covering ranges between *1997* and *2017*, to give a precision of 0.84, while PLE achieved only 0.31 precision.

(7) **Compared to PLE and PAF, Gem successfully differentiates between columns with overlapping numerical ranges by learning distributional embeddings and capturing statistical features from Sato Tables.** For instance, two columns—one representing *'weight'* with values [32.2, 34.3] and another representing *'age'* with values [30, 31, 34]. PLE and PAF struggle with these because of the overlap in numerical values. However, Gem effectively determines the difference by identifying that *'weight'* values follow a continuous distribution, while *'age'* exhibits a clustered distribution at specific points.

(8) **Gem consistently maintains high similarity (semantic similarly of embedding vectors) scores even when columns with the same semantic types have varying cardinalities, outperforming PAF and KS statistics**. For instance, Gem analyzes a column *'year'* with 33 distinct values against another year column with 48 distinct values. Despite the difference in cardinality, Gem put them in a single cluster compared to PAF and KS statistics, which classify them into different clusters.

4.2.2 Numeric + Headers Results. In this section, we examine if the numerical embeddings obtained using Gem can contribute to further improvements when considering more evidence from columns to detect the semantic types. To achieve this, we obtained header embeddings for two datasets, GDS and WDC, and composed them with value embeddings using different composition approaches.

Table 3: Average precision and nDCG scores considering headers + values on fine-grained versions of GDS and WDC. In Gem, D represents Distributional data, S represents statistical data and C represents Contextual data.

Methods	Avg precision		Avg nDCG	
	WDC	GDS	WDC	GDS
Pythagoras_SC [20]	0.02	0.01	0.01	0.01
Sherlock_SC [12]	0.002	0.27	0.003	0.31
Sato_SC [34]	0.003	0.25	0.003	0.31
Doduo_SC [28]	0.12	0.35	0.14	0.38
Gem D+S+C (aggregation)	0.41	0.81	0.45	0.85
Gem D+S+C (AE)	0.40	0.81	0.45	0.85
Gem D+S+C (concatenation)	**0.43**	**0.82**	**0.47**	**0.86**

In Gem, we experimented with three composition methods to merge embeddings: concatenation, aggregation, and learning embeddings through autoencoders (AE). In the concatenation approach, the probabilistic features from the GMM, statistical features from the columns, and contextual embeddings from the headers are combined into a single vector by joining them side by side. In contrast, the aggregation approach summarizes these different embeddings into a single representation. The third approach, learning embeddings through autoencoders, compresses the combined information into a lower-dimensional latent space. We record the average precision and nDCG score for all methods in Table 3. Similar to Table 2, when discussing numbers, we always refer to *precision* in the text.

For this experiment, we use the fine-grained versions of both datasets. We observe the following:

(1) **Concatenation proved to be the most effective composition method for both datasets compared to aggregation and learning embeddings through AE when header contextual embeddings are combined with value embeddings**. The concatenation method preserves the integrity of each embedding type, ensuring that distributional features, statistical characteristics, and semantic context from the headers are all maintained in the final embedding. This allows the model to leverage both numerical properties and semantic signals effectively. At the same time, aggregation with three embeddings into a single representation risks losing some information as it compresses diverse characteristics into a less detailed form. In the case of AE, it is effective for capturing high-level patterns and

lacks in capturing specific details, particularly in scenarios where distributional properties are less distinct. This results in a loss of granularity crucial for tasks requiring precise differentiation between similar value distributions.

(2) **Gem's distributional embeddings help to improve the classification of overlapping values**. For example, in the WDC dataset, columns such as *'Rating_Movie' [10, 10, ...10]*, *'Rating_Book [5, 3, 5 ...5]*,' and *'Rating_Hotel [4.0, 5.0, 0.0, 3.0, 5.0, 0.0, 4.0, ..., 5.0, 5.0, 3.0]*' are clustered together using SBERT due to their high syntactic similarity. However, while all three columns represent ratings on a 1-10 scale, Gem's distributional embeddings capture the different rating patterns within each column. For example, *'Rating_Hotel'* includes a wider spread than the other ratings, with lower and zero scores. This demonstrates how integrating distributional data with contextual embeddings enhances the accuracy of embeddings.

(3) **Pythagoras_SC, which relies on headers as context for numerical embeddings, has demonstrated significant limitations when applied to the GDS dataset, where the headers are highly diverse.** It struggles to distinguish between columns with similar values, even when the headers differ. For instance, the column *"Acceleration"* was incorrectly identified as being highly similar to columns like *"Age"* and *"Dry weight"*. Pythagoras_SC's dependence on header context proved insufficient, whereas Gem performed better in these scenarios. Likewise, on the WDC dataset, where the header information is more complex and heterogeneous, Pythagoras_SC produced poorer results, as its GCN model failed to combine contextual and statistical features effectively.

(4) **Sherlock_SC, which relies on statistical features and embeddings derived solely from headers, shows a substantial difference in performance across the two datasets.** On the WDC dataset, where header information is more complex and varied, the precision drops significantly to 0.002, suggesting that Sherlock_SC struggles to generalize across more diverse column types. However, on the GDS dataset, where the column headers are more standardized, Sherlock_SC performs better with a precision of 0.27; however, in both cases, it is outperformed by Gem. For example, one mis-classification can be seen when Sherlock_SC embedding vectors of two columns have a high similarity score of 0.99: one containing years of publication for books *([2019, 1990, 2019, 2018])* and another with telephone data related to hotels *([13.943, 13.837])*, confusing these distinct categories of *"Book"* and *"Hotel"*.

(5) **Similar to Sherlock_SC and Pythagoras_SC, Sato_SC, which also uses header-based embeddings, demonstrates difficulties on the WDC dataset, achieving a precision of only 0.003.** In contrast, on the GDS dataset, Sato_SC performs worse than Sherlock_SC and Gem but better than Pythagoras_SC, with a precision of 0.25. For example, the embedding vectors of two columns using Gem embeddings show a high similarity score of 0.98: one column contains house prices in various cities *([320000, 450000, 210000])*, while the other represents population sizes in different regions *([50000, 120000, 30000])*. Despite their distinct semantic categories of *"Economic"* and *"Demographic"*, Gem embeddings were unable to distinguish between the two.

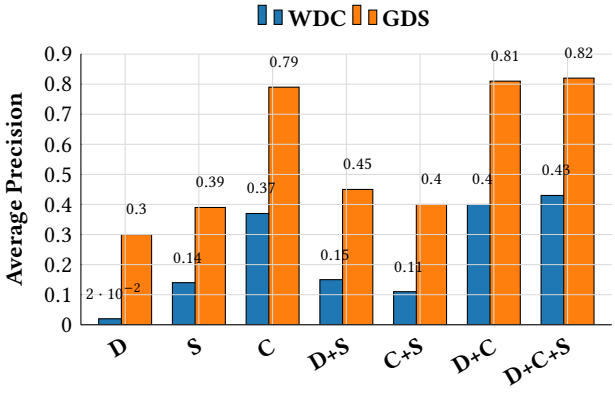


Figure 3: Average Precision for WDC and GDS across different feature settings. *'D'* is distributional features, *'S'* is statistical features and *'C'* is contextual (headers) features. The results illustrate the performance of these feature combinations for both the WDC and GDS datasets.

(6) **When handling numerical columns, Doduo_SC outperformed other adapted single-column baselines (Pythagoras_SC, Sherlock_SC, and Sato_SC)**. With a precision of 0.12 and recall of 0.35, Doduo_SC showed a stronger capability to extract meaningful representations from individual numerical columns without relying on multi-column and table-wide context. However, the precision for Doduo_SC is closer to Gem (0.14) with no context, which shows that Doduo_SC still needs header context to embed numeric columns compared to Gem with no context from headers.

4.3 Ablation Study

We conducted an ablation study to understand the contribution of each feature type in Gem to numerical embeddings' performance. We tested combinations of Gem's distributional, statistical and contextual feature types, and calculated the average precision for each semantic type across the WDC and GDS datasets. The feature combinations we evaluated were Distributional (D), Statistical (S), Contextual (C), Distributional + Statistical (D+S), Contextual + Statistical (C+S), Distributional + Contextual (D+C), and Distributional + Contextual + Statistical (D+C+S).

For each combination, we generated an embedding matrix between column pairs and calculated precision by determining how often the top-k most similar columns matched the ground truth labels. This experiment provides insights into the impact of each feature type in accurately detecting column semantics using numerical embeddings. The results, highlighting the performance of each feature combination, are presented in Figure 3 for both WDC and GDS datasets. The following can be observed:

(1) **Among the individual feature types in Gem, Contextual performs better than Statistical that performs better than Distributional.** The Contextual features act on column headers (and not values) whereas both Statistical and Distributional features act on column values (and not headers). The results for all three feature types are better in GDS than WDC. The Distributional features, by way of the GMM, are designed to model underlying latent distributions, and work well in cases where the data is naturally segmented into distinct distributions. However,

such distributions are not well-defined in GDS and WDC for numerical columns. In such cases, GMM fails to give a full characterisation of a column.

(2) **Distributional features combine effectively with both Statistical and Contextual features.** This is reflected in the fact that (D+S) performs better than both D and S independently and that (D+C) performs better than both D and C independently. This is in contrast with Statistical features, which combine less well with Contextual features; (C+S) performs worse than C on its own for both datasets. Combining Distributional features, by way of the GMM, with Statistical features compensates for their weaknesses in isolation. GMM captures fine-grained distributional details, while statistical features provide broader, high-level insights. Together, they form a more comprehensive representation.

(3) **All three features together perform better than pairs of features**. Indeed, (D+C+S) performs much better than (C+S) and (D+S), but only slightly better than (D+C).

4.4 Impact of Gaussian components

In this section, we assess the impact of the number of GMM components on Gem's performance. We vary the number of GMM components from 100 to 500 for all datasets, and the results are presented in Figure 4. This ablation study aims to investigate how Gem performs on high-dimensional numerical features. However, the numerical features do not represent the raw dimensional numerical features; instead, they correspond to the number of Gaussian components extracted from each column. Since our use case is based on single-column distributions, the dimensions of the raw data do not affect the evaluation performance. Our observations indicate that the number of Gaussian components does not significantly impact Gem's overall performance. Precision results for GitTables remain consistently around 0.26 to 0.28, with minimal fluctuations as the number of components increases. Similarly, Sato Tables show a stable range of 0.36 to 0.37, while GDS consistently remains around 0.43 to 0.44. For WDS, precision scores slightly vary between 0.19 and 0.21, indicating no significant improvement with more Gaussian components. Similarly, for BiodivTab, the minimum precision bound appears as 0.72 and goes to 0.74; however, it achieved the maximum of 0.74 at 50 Gaussian components, quite early. For WDC and GDS, we perform the ablation study in a dedicated environment with parallel processing due to the high number of Gaussian components extracted from each single column. This stability across different numbers of components suggests that Gem's performance is robust to the choice of Gaussian mixture complexity.

4.5 Scalability analysis

We perform a scalability analysis (see Figure 5) of Gem with benchmark methods to evaluate how each method's runtime scales as the number of columns in the dataset increases. We measured each method's run time to generate embeddings. To ensure consistency, we measured the runtime for each dataset size five times and calculated the average.

Figures 5a and 5b compare the runtime of Gem and baseline methods in the numeric-only setting, whereas Figures 5c and 5d provide a runtime comparison considering the context for Gem and bespoke solutions. From Figures 5a and 5b, we observe that PLE and KS statistics demonstrate consistently low runtimes as the number of columns increases. However, PLE shows an

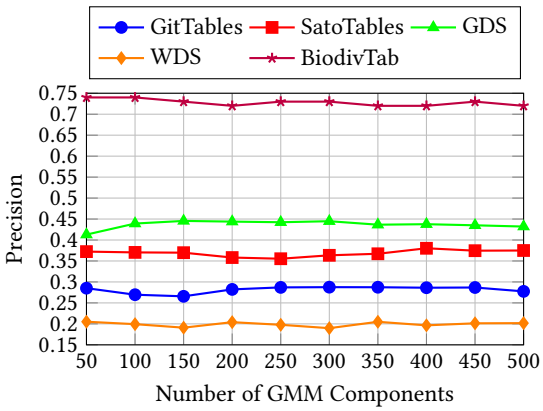


Figure 4: Performance comparison across different numbers of GMM components for all datasets.

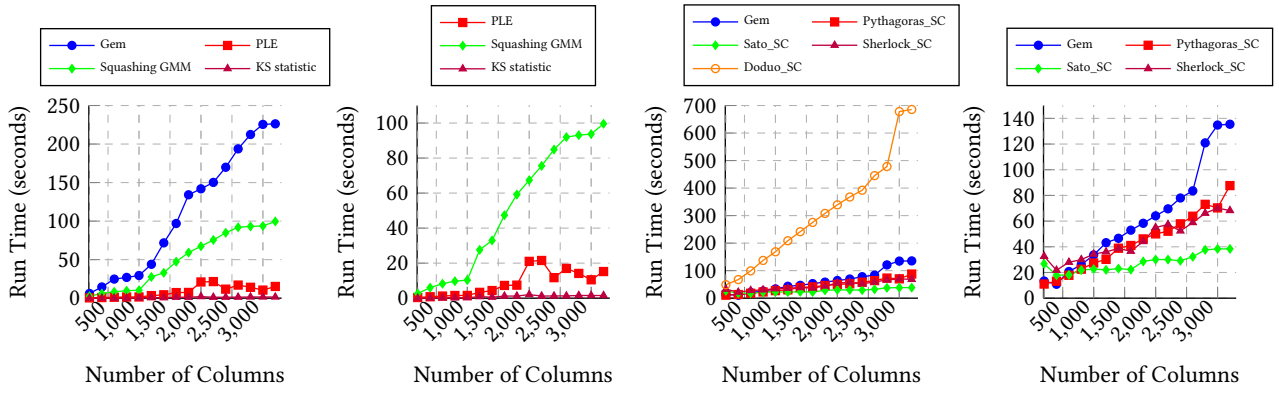
irregular trend after 2000 columns due to skewed distributions in a subset of columns; the quantile bins in PLE divide the data unevenly under skewed distributions, leading to imbalanced encoding. While Gem is mostly linear, it shows a slight irregular trend with a substantial spike at 2000 columns due to computational differences in Gaussian distributions. Squashing GMM demonstrates less predictable behavior, with significant jumps at specific column counts (e.g., 1200 and 2000), which is due to higher sensitivity to individual dataset sizes. Overall, Gem and Squashing GMM scale reasonably, in linear time.

From Figures 5c and 5d, Sato_SC, Pythagoras_SC and Sherlock_SC show linear runtime growth as the number of columns increases. However, Pythagoras_SC shows a slight peak at 3200 columns before stabilizing. Gem and Doduo_SC follow linear trends, although both experience a slightly steeper rise at a higher number of columns. Overall, Sato_SC, Sherlock_SC and Pythagoras_SC scale nearly linearly, while Doduo_SC and Gem appear linear at a higher number of columns.

4.6 Clustering Results

We evaluated Gem for an additional downstream clustering task by clustering columns with similar semantics using Deep Clustering (DC) algorithms. We applied SDCN [3], a well-known DC algorithm, and TableDC [24], which was specifically designed to support clustering in data management tasks. This analysis evaluated how well Gem integrates with the clustering methods. We also compare Gem embeddings with the ones generated through Squashing_SOM to see how different embeddings affect the clustering performance. In the clustering environment, the distributional embeddings produced by Gem and Squashing_SOM are an input for the autoencoder in the DC algorithm. The results are shown in Table 4. We observe the following:

(1) Gem consistently outperforms Squashing_SOM for both TableDC and SDCN when considering numerical embeddings in the GDS dataset. For example, TableDC with Gem obtained a higher 0.10 ARI and 0.16 ACC than TableDC with Squashing_SOM on GDS, while the improvement for SDCN with Gem embeddings is 0.08 on ARI and 0.13 ACC. Squashing_SOM's preserved topological structures, however, struggled to integrate the rich semantic context from SBERT. On the other hand, Gem, which focuses on modeling numerical distributions using GMM, better integrates the contextual information than Squashing_SOM.



(a) Numeric only (b) Zoom view of numeric only. (c) Context + numeric (d) Zoom view of context + numeric

Figure 5: Run time comparison of different methods with and without context from headers: (a) Overall view (no context); (b) Zoomed-in view (no context); (c) Overall view (context + numeric) (d) Zoomed-in (context + numeric)

Table 4: Clustering results on the GDS and WDC datasets. We compare the performance of Gem and Squashing_SOM using ARI and ACC metrics. The best result for each dataset, based on ARI and ACC, is highlighted in bold. Notation: HO = Headers only; VO = Values only; H+V = Headers plus Values.

	Gem								Squashing_SOM							
	GDS				WDC				GDS				WDC			
	TableDC		SDCN		TableDC		SDCN		TableDC		SDCN		TableDC		SDCN	
	ARI	ACC	ARI	ACC	ARI	ACC	ARI	ACC	ARI	ACC	ARI	ACC	ARI	ACC	ARI	ACC
HO	**0.69**	**0.76**	0.65	0.68	**0.31**	0.41	0.30	0.41	-	-	-	-	-	-	-	-
VO	**0.39**	**0.48**	**0.39**	0.46	**0.03**	**0.12**	**0.03**	**0.12**	0.29	0.32	**0.31**	**0.33**	**0.009**	**0.21**	**0.009**	0.20
H+V	**0.78**	**0.81**	0.74	0.77	**0.33**	**0.43**	0.27	0.38	**0.63**	**0.70**	0.58	0.61	**0.009**	0.20	**0.009**	**0.21**

(2) TableDC outperformed SDCN across both datasets under two experimental configurations: headers-only and headers+values. The is observed in the GDS dataset, where TableDC achieves a 0.08 increase in ACC using the SBERT in the headers-only setting. This highlights the effectiveness of TableDC in combination with column headers.

(3) Gem embeddings alone do not integrate well with TableDC and SDCN. However, contextual integration with column values in TableDC shows better performance than SDCN. For example, TableDC and SDCN perform poorly with a 0.39 ARI using values only on GDS. However, TableDC improves by 0.39 ARI when headers are included compared to SDCN, which improves by 0.35 ARI.

(4) Like column embeddings, column clustering has poorer results in the WDC dataset than in GDS for both SDCN and TableDC. This arises from the inherent complexity and overlap in the WDC headers, which leads to ambiguities in both downstream tasks. The WDC dataset has more varied and noisy data distributions, making it harder for SDCN and TableDC to cluster similar columns effectively. For example, columns *"journal_Rank"* and *"Book_Rank"* have similar ranking values, leading to large clusters in both SDCN and TableDC, which is a mis-classification.

5 CONCLUSION

Numerical data is prominent in tabular datasets, and thus embeddings for database columns can usefully treat numerical data as a first-class citizen. To enable this, we propose Gem, which focuses on numerical data through a signature mechanism that generates a probability matrix for each column, indicating the likelihood of belonging to specific Gaussian components. Experiments have (i) shown that Gem outperforms previous numerical embedding proposals (i.e., [8, 13]) for semantic type detection of column using numerical embedding over a variety of datasets; and (ii) shown that Gem embeddings can be combined effectively with other evidence on the semantics of a column, such as column headers, both for column clustering and semantic type annotation.

ACKNOWLEDGMENTS

This work was partially funded by the CRUK National Biomarker Centre, and supported by the Manchester Experimental Cancer Medicine Centre and the NIHR Manchester Biomedical Research Centre. Hafiz Tayyab Rauf is supported by a studentship from The University of Manchester.

REFERENCES

[1] Nora Abdelmageed, Sirko Schindler, and Birgitta König-Ries. 2021. BiodivTab: A Tabular Benchmark based on Biodiversity Research Data. In *SemTab@ISWC, submitted.*

[2] K Balakrishnan. 2019. *Exponential distribution: theory, methods and applications.* Routledge.

[3] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. 2020. Structural Deep Clustering Network. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 1400–1410. https://doi.org/10.1145/3366423.3380214

[4] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)* 39, 1 (1977), 1–22.

[5] William Feller. 1991. *An introduction to probability theory and its applications, Volume 2*. Vol. 81. John Wiley & Sons.

[6] Hans Fischer. 2011. *A history of the central limit theorem: from classical to modern probability theory*. Vol. 4. Springer.

[7] Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting Numerical Reasoning Skills into Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 946–958. https://doi.org/10.18653/V1/2020.ACL-MAIN.89

[8] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. 2022. On Embeddings for Numerical Features in Tabular Deep Learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/9e9f0ffc3d836836ca96cbf8fe14b105-Abstract-Conference.html

[9] Hazar Harmouch, Thorsten Papenbrock, and Felix Naumann. 2021. Relational Header Discovery using Similarity Search in a Table Corpus. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. 444–455. https://doi.org/10.1109/ICDE51399.2021.00045

[10] Robert V Hogg, Joseph W McKean, Allen T Craig, et al. 2013. *Introduction to mathematical statistics*. Pearson Education India.

[11] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17. https://doi.org/10.1145/3588710

[12] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 1500–1508. https://doi.org/10.1145/3292500.3330993

[13] Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. 2020. Learning Numeral Embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.), Vol. EMNLP 2020. Association for Computational Linguistics, 2586–2599. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.235

[14] Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. 2021. NumGPT: Improving Numeracy Ability of Generative Pre-trained Models. *CoRR* abs/2109.03137 (2021). arXiv:2109.03137 https://arxiv.org/abs/2109.03137

[15] Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. 1995. *Continuous univariate distributions, volume 2*. Vol. 289. John wiley & sons.

[16] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. 2021. Well-tuned Simple Nets Excel on Tabular Datasets. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 23928–23941. https://proceedings.neurips.cc/paper/2021/hash/c902b497eb972281fb5b4e206db38ee6-Abstract.html

[17] Gayeong Kim, Sookyung Kim, Ko Keun Kim, Suchan Park, Heesoo Jung, and Hogun Park. 2023. Exploiting Relation-aware Attribute Representation Learning in Knowledge Graph Embedding for Numerical Reasoning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye (Eds.). ACM, 1086–1096. https://doi.org/10.1145/3580305.3599338

[18] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2023. SportsTables: A New Corpus for Semantic Type Detection (Extended Version). *Datenbank-Spektrum* 23, 3 (2023), 189–197. https://doi.org/10.1007/S13222-023-00457-Y

[19] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2023. Steered Training Data Generation for Learned Semantic Type Detection. *Proc. ACM Manag. Data* 1, 2 (2023), 201:1–201:25. https://doi.org/10.1145/3589786

[20] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2024. Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani (Eds.). OpenProceedings.org, 725–733. https://doi.org/10.48786/EDBT.2024.62

[21] Eckhard Limpert, Werner A Stahel, and Markus Abbt. 2001. Log-normal distributions across the sciences: keys and clues: on the charms of statistics, and how mechanical models resembling gambling machines offer a link to a handy way to characterize log-normal distributions, which can provide deeper insight into variability and probability—normal or log-normal: that is the question. *BioScience* 51, 5 (2001), 341–352.

[22] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.

[23] Karl Pearson. 1894. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A* 185 (1894), 71–110.

[24] Hafiz Tayyab Rauf, André Freitas, and Norman W. Paton. 2024. TableDC: Deep Clustering for Tabular Data. *CoRR* abs/2405.17723 (2024). https://doi.org/10.48550/ARXIV.2405.17723 arXiv:2405.17723

[25] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 3980–3990. https://doi.org/10.18653/V1/D19-1410

[26] Douglas A Reynolds et al. 2009. Gaussian mixture models. *Encyclopedia of biometrics* 741, 659-663 (2009).

[27] Lya Hulliyyatus Suadaa, Hidetaka Kamigaito, Kotaro Funakoshi, Manabu Okumura, and Hiroya Takamura. 2021. Towards Table-to-Text Generation with Numerical Reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 1451–1465. https://doi.org/10.18653/V1/2021.ACL-LONG.115

[28] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çagatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1493–1503. https://doi.org/10.1145/3514221.3517906

[29] Yushi Sun, Hao Xin, and Lei Chen. 2023. RECA: Related Tables Enhanced Column Semantic Type Annotation Framework. *Proc. VLDB Endow.* 16, 6 (2023), 1319–1331. https://doi.org/10.14778/3583140.3583149

[30] Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for Numeracy-Preserving Word Embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 4742–4753. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.384

[31] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP Models Know Numbers? Probing Numeracy in Embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 5306–5314. https://doi.org/10.18653/V1/D19-1534

[32] Jianlong Wu, Keyu Long, Fei Wang, Chen Qian, Cheng Li, Zhouchen Lin, and Hongbin Zha. 2019. Deep Comprehensive Correlation Mining for Image Clustering. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 8149–8158. https://doi.org/10.1109/ICCV.2019.00824

[33] Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. 2010. Image Clustering Using Local Discriminant Models and Global Integration. *IEEE Trans. Image Process.* 19, 10 (2010), 2761–2773. https://doi.org/10.1109/TIP.2010.2049235

[34] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848. http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[35] Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. 2022. MultiHiertt: Numerical Reasoning over Multi Hierarchical Tabular and Textual Data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 6588–6600. https://doi.org/10.18653/V1/2022.ACL-LONG.454