BERT-Sort: A Zero-shot MLM Semantic Encoder on Ordinal Features for AutoML

Anonymous[1]

[1]Anonymous Institution

Abstract Data pre-processing is one of the key steps in creating machine learning pipelines for tabular data. One of the common data pre-processing operations implemented in AutoML systems is to encode categorical features as numerical features. Typically, this is implemented using a simple alphabetical sort on the categorical values, using functions such as *OrdinalEncoder*, *LabelEncoder* in *Scikit-Learn* and *H2O*. However, often there exist semantic ordinal relationships among the categorical values, such as: quality level (i.e., ['very good' > 'good' > 'normal'> 'poor']), or month (i.e., ['Jan'< 'Feb' < 'Mar']). Such semantic relationships are not exploited by previous AutoML approaches. In this paper, we introduce BERT-Sort, a novel approach to semantically encode ordinal categorical values via zero-shot Masked Language Models (MLM) and apply it to AutoML for tabular data. We created a new benchmark of 42 features from 10 public data sets for sorting categorical ordinal values for the first time, where BERT-Sort significantly improves semantic encoding of ordinal values in comparison to the existing approaches with 27% improvement. We perform a comprehensive evaluation of BERT-Sort on different public MLMs, such as RoBERTa, XLM and DistilBERT. We also compare the performance of raw data sets against encoded data sets through BERT-Sort in different AutoML platforms including AutoGluon, FLAML, H2O, MLJAR and TPOT to evaluate the proposed approach in an end-to-end scenario. BERT-Sort successfully improves the performance of models in 4 different AutoML tools with 10.61% performance gain.

1 Introduction

An Automated-Machine Learning (AutoML) platform aims to automate the process of feature engineering, data engineering, hyper-parameter optimization, training, prediction, and deployment, where it minimizes human supervision in all stages (He et al., 2021). Each data set may contain a variety of data types including ordinal values where the order of values is important. Let $\mathcal{C}_1 \prec \mathcal{C}_2 \prec ... \prec \mathcal{C}_n$ denote a fixed set of arbitrary classes of $\mathcal{C}$. For instance, *UCI Audiology data set* (Porter, 2019) includes a feature of *Air* with a set of unique values of *Normal* $\prec$ *Mild* $\prec$ *Moderate* $\prec$ *severe* $\prec$ *profound*. Although this field is not a target feature, the order of the values carries semantic meaning. AutoML platforms encode each feature based on their types and content of values. Often AutoMLs encode categorical features as an integer array (Pedregosa et al., 2011) function. For instance, *H2O AutoML* (LeDell et al., 2018) and *Scikit-Learn* use *categorical_encoding* and *OrdinalEncoder* (SkL, 2021) functions, respectively, to transform categorical values to an integer array. However, all categorical encoders rely on an alphabetical sort function such as *Numpy Sort* (num, 2021) where the encoded values are based on the sorting results of categorical values alphabetically (Oliphant, 2006) LeDell et al. (2018). Such a simple method may fail to capture the semantic relationships between values. For instance, the value *'profound'* in the *Air* feature of *UCI Audiology* data set represents more serious level than the value *'severe'* but *OrdinalEncoder* returns the opposite result. Similarly, alphabetically sorting values of *num-of-cylinders* feature in *UCI Automobile Data Set* returns *['eight'* $\prec$ *'five'* $\prec$ *'four'* $\prec$ *'six'* $\prec$ *'three'* $\prec$ *'twelve'* $\prec$ *'two']* as $[\mathcal{C}_1 \prec \mathcal{C}_2 \prec ... \prec \mathcal{C}_7]$, respectively, where the ordinal values have been misplaced. As a result, such incorrectly encoded values can pose more challenges to any machine-learning algorithms to predict the target value of *price* based on increase/decrease value of *cylinders*.

 © 2022 the authors, released under CC BY 4.0

One hypothesis to be verified in this paper is that AutoML platforms incorrectly encoding ordinal categorical features might result in degraded performance. To address this issue, we propose a novel approach *BERT-Sort*, which utilizes pre-trained Masked Language Models (MLM) (Devlin et al., 2018) in a zero-sot setting to semantically sort and encode ordinal values. The following are our main contributions in this study.

(i) A zero-shot systematic sorting algorithm to sort ordinal values is introduced in Section 3;

(ii) We compose a benchmark of 10 real-world data sets with 42 ordinal features for the first time, which is explained in Section 4 (detail in Appendix A);

(iii) We conduct a comprehensive performance evaluation of benchmarks by i) comparing the results of BERT-Sort (with initialization on 4 different publicly available MLMs) and OrdinalEncoder (which is widely used in different AutoML tools), and ii) evaluating between raw data set and encoded data set through BERT-Sort in 4 different AutoML platforms of AutoGluon, FLAML, H2O, MLJAR and TPOT in Section 4.

2 Related Works

In the supervised approach, researchers and practitioners utilize existing limited data sets with ordinal values to develop a model where it can encode ordinal values and predict unseen ordinal values for encoding purposes. However, the adaptation of a trained model from one domain to another is extremely limited. Therefore, most related works in the supervised approach can be used in a set of selective domains/languages and it can be used in a form of training a model for a particular domain. An early study by McCullagh (1980) and later other studies by Christensen (2015) and Harrell (2015), introduced a general class of regression models for ordinal values where it utilizes the ordinal values through various modes of stochastic ordering. In another recent study by Lausser et al. (2020), the authors developed an ordinal subcascades detection and encoding process, but the authors mentioned that their works have a limitation where analyzing suitable data representations may give a better answer for ordinal encoding. Dahouda and Joe (2021) proposes a deep-learned embedding technique for categorical features encoding. The proposed technique is a distributed representation for categorical features where each category is mapped to a distinct vector, and the properties of the vector are learned while training a neural network. In our proposed approach, we utilize the semantic understanding of MLMs to overcome the supervision and the limitation of ordinal subcascade encoding and other similar approaches.

In addition to the aforementioned studies, categorical encoders have also been widely applied in AutoML tools. Auto-sklearn provides *ordinal encoding* or *one-hot encoding* as choices for categorical features (Feurer et al., 2020). MLJAR (mlj, 2022) (Płońska and Płoński, 2021) converts categorical features into numeric with *label encoder*, *one-hot encoder* or *target encoder*, which is automatically selected based on feature cardinality and AutoML training stage (mlj, 2022). As explained by LeDell et al. (2018), H2O utilizes tree-based models (Gradient Boosting Machines, Random Forests) to support group-splits on categorical variables, so categorical data can be handled natively. However, as explained by Zhou and Hooker (2021), the default split-improvement method is biased towards increasing the importance of features with more potential splits especially when we are dealing with a large number of ordinal values. H2O also uses *categorical_encoding*, which specifies the encoding scheme to use for handling categorical features. In AutoGluon, each categorical feature is mapped to monotonically increasing integers (Aut, 2022a). *AutoKeras* defines an argument *categorical_encoding*, which specifies whether to encode the categorical features to numerical features (Aut, 2022b). However, these categorical encoders rely on purely alphabetically sort functions such as *Numpy Sort*(num, 2021) where it rearranges the values by alphabetically sorting values, then it assigns an integer value based on the index of sorted values (Oliphant, 2006) LeDell et al. (2018). Although the method can encode the categorical data sets, it fails to encode ordinal values

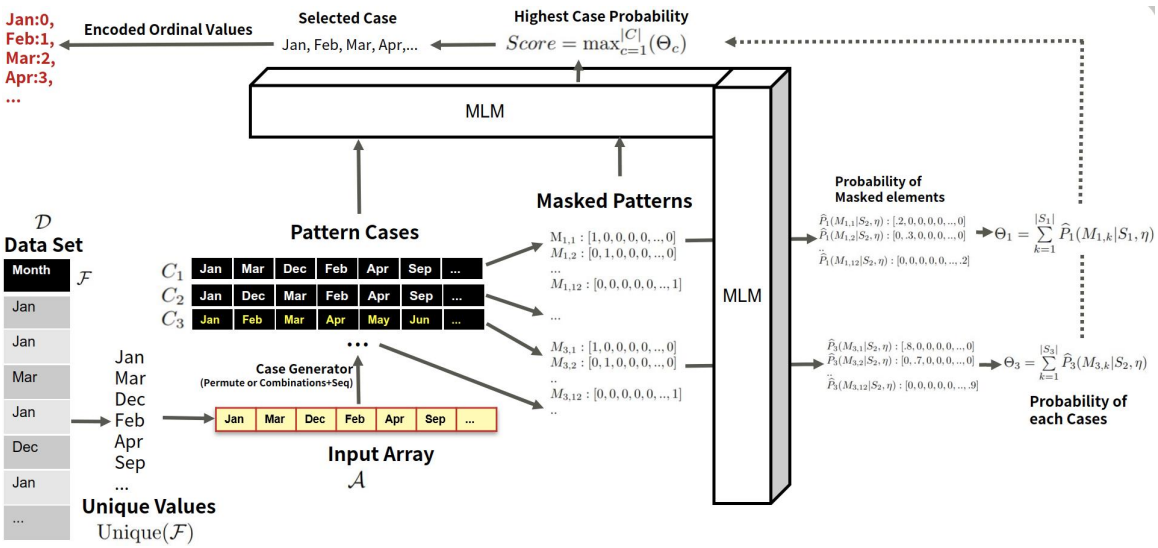

Figure 1: An overall process of BERT-Sort approach

by misplacing the values in incorrect orders. In this paper, our target is introducing an approach that takes advantage of ordinal values and their semantic definitions and defines an approach that can be universally applied toward diverse domains/languages without any supervision.

3 BERT-Sort

The basic idea of BERT-Sort is to utilize the power of a pre-trained language model to recognize the order of a given sequence of values. Language models learn from a large number of text corpora where they processed the context of sentences and paragraphs. One of the algorithms to process the context is Masked-Language Modeling (MLM), where it computes the probability distribution of a masked token among the candidates of tokens. A masked language model can be used across different down-stream tasks such as text generation, sentiment analysis etc. Each feature of the given data set may contain categorical ordinal values. The tokens of ranked ordinal values could be seen in many documents. For instance, we may find a large number of documents that includes *'February'* appearing after *'January'*. In NLP down-stream tasks, such as text generation, we may replace a subset of tokens of a given input string with a distinctive character of *[MASK]* and the objective is predicting the masked token. We utilize this unmasking process to find the orders of a set of given values based on their probability of orders according to a pre-trained MLM.

3.1 Problem Formulation

Figure 1 shows the overall process of the proposed approach, BERT-Sort, where it aims to find the best order of ordinal values by computing the maximum probability of appearances of values in a set of possible orders. First, BERT-Sort captures all categorical features ($\mathcal{F}$) of a given data set, $\mathcal{D}$. Then it checks if the number of unique values in any categorical feature $\mathcal{F}_i$ is less than φ, a threshold of the maximum number of unique values that can be considered for BERT-Sort processing. In Section 4, we extend this rule-based approach to automate the detection of ordinal values by applying BERT-Sort to a few cases. Once $\mathcal{F}_i$ is selected as a candidate for BERT-Sort, then it generates N different possible permutation cases (different orders of ordinal values) of $\mathcal{C}_i$ where $i = 1, .., N$. Since it is computationally expensive to generate all possible permutations for a large number of elements, later we explain a sequential approach to reduce the number of cases. For each generated case, $\mathcal{C}_i$, BERT-Sort produces k different masked patterns ($k = 1..||\mathcal{C}_i||$) where $M_{i,k}$ represents each masked pattern of $\mathcal{C}_i$. Each masked pattern of $M_{i,k}$ applied to $\mathcal{C}_i$ to generate a

sentence. For instance, for a given three elements of $[\mathcal{C}_{1,1}, \mathcal{C}_{1,2}, \mathcal{C}_{1,3}]$ and applying $M_{1,1} = [1, 0, 0]$, a sentence of "*[CLS][MASK]*, $\mathcal{C}_{1,2}$, $\mathcal{C}_{1,3}$*.[EOS]*" can be generated. Different sentence structures can be generated (i.e., adding only a blank space between each ordinal value) in this stage as we explained the detail of different structures and their performances in Appendix D. Then, BERT-Sort knows the masked element ($\mathcal{C}_{1,1}$) and find its probability in top η tokens (which is a parameter of BERT-Sort) and represented as $\mathcal{W}_\eta$.

Next, BERT-Sort makes an inference on MLM to unmask $M_{c,k}$ and retrieves the probability denoted as $\widehat{P}_c(M_{c,k}|S_c, \eta)$, if an unmasked token $M_{c,k}$ ($U(M_{c,k})$) existed in $\mathcal{W}_\eta$ for a given sequence of values S_c; otherwise it returns 0. Finally, it finds the average probability of $|S_c|$ patterns of c case from MLM denoted as Θ_c, for the final score of case c as follows:

$$\Theta_c = \begin{cases} \frac{\sum_{k=1}^{|S_c|} \widehat{P}_c(M_{c,k}|S_c,\eta)}{|S_c|}, & \text{if } U(M_{c,k}) \exists \mathcal{W}_\eta \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Θ_c denotes the score of appearing a sentence per generated order of case c, and $\Theta_M = \max_{c=1}^{|C|}(\Theta_c)$ is selecting a case with the highest score that shows the best order or sequences of input elements. Intuitively, BERT-Sort computing the most likely order from all possible permutations with pre-trained MLM. Different MLMs can be applied under the same architecture across different domains (i.e., medical domain) and languages (i.e., Chinese).

3.2 Handling a large number of ordinal values

Although BERT-Sort semantically ranks values instead of non-semantic approaches (i.e., alphabetical sort), it is computationally intractable for BERT-Sort to directly support large permutations of ordinal values. BERT-Sort applies two approaches to handling a large number of ordinal values.

First, BERT-Sort uses the divide and conquer approach for sorting elements if there are any repeated words among ordinal values. The following shows four steps for this process. i) It generates a set of groups where each group has a word in common and it denotes common word (i.e., define a *'Hot'* group leader for values of *[Lava Hot, Boiling Hot, Hot]*; ii) It sorts elements within each group. (i.e., *[Hot ≺ Boiling Hot ≺ Lava Hot]*); iii) It sorts elements of **group leaders** and **unique values** (i.e., *[Cold ≺ Hot]*); Finally, iv) it replaces each sorted group leaders with their sorted values of the group. (i.e., *[Cold ≺ Hot ≺ Boiling Hot ≺ Lava Hot]*).

Second, BERT-Sort instead of generating a large number of permutation cases when $\zeta < |\mathcal{A}|$, it is sorting only ζ number of elements, then it sorts the rest of elements one-by-one sequentially. Figure 2 shows an example of sequential sorting values for 12 ordinal value elements (months abbreviations). BERT-Sort uses a sequential approach in three steps as follows. i) It finds the best candidate for ζ blank spots (in the figure, it is initiated with $\zeta = 5$ spots); ii) Once it finds the best candidates for ζ blank spots as initial sorted elements (it finds *[Jan ≺ Feb ≺ Apr ≺ Jun ≺ Sep]* for 5 spots in the example), then it adds each remaining element to the initial sorted elements (in the second iteration, it finds the best position for *"Oct"* in initial sorted 5 elements); Finally, iii) it repeats step (ii) until all elements are added to the final sorted values.

This process reduces the computation time from O(n!) to the combination problem of $O(\zeta Log(n)) + \omega$ where ω is a constant value due to sequential adding values.

BERT-Sort aims to have better accuracy with higher probability score with lengthy contents (larger ζ have better accuracy). For instance, Θ_M with $U([MASK]) = Jan$ in *"[CLS][MASK], Feb, Mar, Apr.[EOS]"* is higher than *"[CLS][MASK], Feb.[EOS]"*. Therefore, we recommend to keep maximum possible of ζ elements per available computation resources. For instance, $\zeta = 5$ if $||\mathcal{A}|| \leq 12$, and $\zeta = 3$ if $12 < ||\mathcal{A}|| \leq 20$. Algorithm 1 shows the details of BERT-Sort procedures.

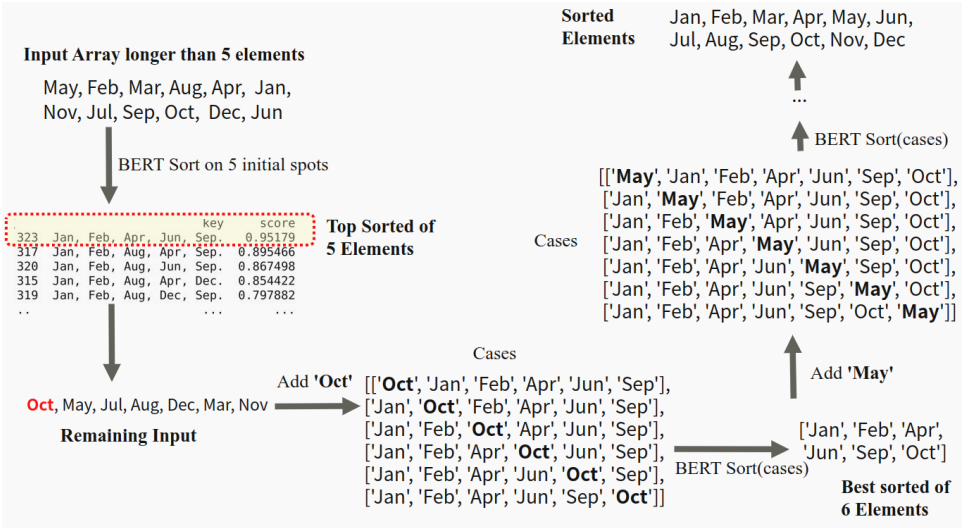

Figure 2: An example of sequential sorting elements for large number of ordinal values where $\zeta = 5$

4 Experiments

Experimental Setup. We evaluate the proposed BERT-Sort approach under two cases: i) the performance of BERT-Sort in detecting and encoding categorical ordinal features; ii) the effectiveness of encoding applied to categorical ordinal features by BERT-Sort before input to various AutoMLs.

We use 10 different publicly available real-world data sets because it includes categorical ordinal values. We annotate and generate 42 different categorical features (as explained in Appendix A), where we compare the performance of Scikit-Learn OrdinalEncoder as a baseline against BERT-Sort encoder. We initiate BERT-Sort algorithm with 4 different popular and publicly available MLMs: DistilBERT, RoBERTa, XLM-RoBERTa and BERT-base-uncased ($M_{1..4}$ respectively) with a zero-shot setting. We compare the results of different MLMs and recommend the best MLM to researchers and practitioners. The details of MLM, configurations and BERT-Sort hyper-parameters with a link to reproduction are presented in Appendix B.

Like other encoders, such as *OrdinalEncoder*, *LabelEncoder*, we use BERT-Sort to rank ordinal values semantically, then assign an integer for each element per their orders. Since the orders of the values are principal factors in either ascending or descending, we do not distinguish between two ranks. However, we expect that BERT-Sort returns ordinal values ranked in ascending order (i.,e., *'low' to 'high'* or *'Jan' to 'Dec'*) because most documents (which has been used to train MLMs) are written in ascending format. For instance, we may find many documents in Wikipedia that indicate *'Jan, Feb'* and fewer document that include *'Feb, Jan'*.

4.1 Evaluation of Detecting Ordinal Features

We define *Ordinal Value Detection* function To apply BERT-Sort effectively, where it checks whether a set of unique values of a given feature corresponds to a semantic order. We select the ordinal values with length of 3 to 20, and remove numerical values in string format as part of pre-processing stage. Then, we shuffle input unique values and generate n number of cases. If $\Theta_n > 1e-4$, BERT-Sort consider that there is an ordinal relationship between values. In our experiment, we use benchmark that includes 10 data sets with 212 features. In this experiment, $n = 3$, $\eta = 20,000$ in *BERT_Base_Sort()* and initiate MLM on *RoBERTa* model. As shown in Table 1, the ordinal value detection function predicts ordinal values with $Precision = 0.875$, $Recall = 1$ and $F1 = 0.933$. As shown in Table 1 and Table 2-row#5, BERT-Sort detects 6 false-positive features out of 212 features due to possibility of generating a probability score for a set of given values in zero-shot

Algorithm 1 BERT-Sort Procedures

procedure BERT_Sort($\mathcal{D}, \zeta$)
▷ Sorts each feature, $\mathcal{F}$, of input data set, $\mathcal{D}$ with a given ζ parameter.
  for $\mathcal{F}$ in $\mathcal{D}$ **do** ▷ process each feature of given data set
    for $\mathcal{U}_f$ in Unique($\mathcal{F}$) **do** ▷ capture pre-processed unique values
      $\mathcal{R}(\mathcal{U}_f) \leftarrow$G$(repeated_words)$ **if** $\exists g$ **else** $\mathcal{U}_f$ ▷ grouping repeated words among unique values;
      ▷ otherwise generate a single group
      for $\mathcal{G}$ in $\mathcal{R}(\mathcal{U}_f)$ **do**
        $\mathcal{H}_\mathcal{G} \leftarrow BERT_Compose_Sort(G, \zeta)$ ▷ sort values within each group
  $\widehat{\mathcal{H}_\mathcal{G}} \leftarrow heading_words(\mathcal{H}_\mathcal{G})$ ▷ replace sorted values of each group with their header group
return $BERT_Compose_Sort(\widehat{\mathcal{H}_\mathcal{G}}, \zeta)$ ▷ sort heading groups

procedure BERT_Compose_Sort($\mathcal{A}, \zeta$)
▷ Sort input array of $\mathcal{A}$ through MLM
  if $|\mathcal{A}| > \zeta$ **then**
    $\mathcal{C} \leftarrow Combine(\mathcal{A}, \zeta)$ ▷ initial combination case generator with length of ζ
    $\mathcal{I} \leftarrow BERT_Base_Sort(\mathcal{C})$ ▷ sort initiated sequences
    for $\mathcal{E}$ in $\mathcal{A} - \mathcal{I}$ **do** ▷ sequential adding the rest of elements
      $\mathcal{C} \leftarrow \mathcal{C} \cup Seq(\mathcal{C}, \mathcal{E})$ ▷ generate new sequential cases
      $\mathcal{C} \leftarrow BERT_Base_Sort(\mathcal{C})$ ▷ sort sequential cases
    $\mathcal{O} \leftarrow \mathcal{C}$
  else
    $\mathcal{C} \leftarrow Permute(\mathcal{A}, \zeta)$ ▷ generate all new cases with all permutations of $\mathcal{A}$
    $\mathcal{O} \leftarrow BERT_Base_Sort(\mathcal{C})$ ▷ sort all possible permutation of $\mathcal{A}$
return $\mathcal{O}$

procedure BERT_Base_Sort($\mathcal{C}$)
▷ Sort input cases of $\mathcal{C}$ through MLM
Init MLM ▷ initialize a MLM model (i.e., RoBERTa)
  for $\mathcal{C}_i$ in $\mathcal{C}$ **do** ▷ process each case
    for j in range($|\mathcal{C}_i|$) **do**
      $\mathcal{M}_{i,j} \leftarrow Mask(\mathcal{C}_{i,j})$ ▷ generate mask pattern
      $\widehat{\mathcal{P}_i} \leftarrow \widehat{\mathcal{P}_{i-1}} \bigcup \widehat{\mathcal{P}_i}(\mathcal{M}_{i,j}|S_i, \eta)$ ▷ collect probability of each unmasked pattern
    $\Theta_i = \sum(\widehat{\mathcal{P}_i})$ ▷ calculate score of each case
    if $\Theta_i > \Theta_m$ **then**:
      $\mathcal{C}_m = \mathcal{C}_i$
      $\Theta_m = \Theta_i$ ▷ keep the best case with highest score
  return $\mathcal{C}_m$

		Predicted		
		Positive	Negative	Total
Actual	True	42	164	206
	False	6	0	6
	Total	48	164	212

Table 1: Evaluation of ordinal value detection on 212 features of 10 data sets

environment. Even though BERT-Sort may encode those false-positive detected features, these encoded values may not cause negative impacts to down-stream machine learning tasks.

#	Data set	Unique Values	Ground Truth	Detection Status
1	uci-audiology-original	['Normal', 'Moderate', 'Mild']	Yes	True-Positive
2	cat-in-the-dat-ii	['Warm', 'Boiling Hot', 'Freezing', 'Lava Hot', 'Hot', 'Cold']	Yes	True-Positive
3	uci-automobile	['Wagon', 'Sedan', 'Hatchback', 'Hardtop', 'Convertible']	No	True-Negative
4	uci_automobile	['Blue Collar', 'Management', 'Entrepreneur', ..., 'Admin']	No	True-Negative
5	cat-in-the-dat-ii	['Circle', 'Polygon', 'Square', 'Star', 'Trapezoid', 'Triangle']	No	**False-Positive**

Table 2: Five examples of the ordinal value detection

4.2 Evaluation of Encoding Ordinal Values

In addition to classical evaluation of classification problem, such as *Accuracy* (Mosley, 2013)(Urbanowicz and Moore, 2015), we proposed a new metric based on ordinal value error rate where it calculates the ordinal distance error between the ground truths and predictions as follows:

$$Ord_{Acc} = \sum_{i=1}^{\|A\|} \frac{|\mathcal{L}_i - \widehat{\mathcal{L}}_i|}{\|A\|} \tag{2}$$

where $\|\mathcal{A}\|$ denotes the length of input array (unique values of a given feature, $\mathcal{F}$), $\mathcal{L}_i - \widehat{\mathcal{L}}_i$ is the distance of i-th element in predicted order ($\widehat{\mathcal{L}}_i$) from the actual position ($\mathcal{L}_i$), and Ord_{Acc} represents the *Ordinal Accuracy*. For instance, as shown in Table 3, row#1, both values of *Jan* and *Feb* have distance (error rate) of one to their actual positions and thus $Ord_{Acc} = 0.87$. On the other hand, the accuracy for this example is 0.5 because two of four elements are incorrect. In contrast, another example shown in row #2 also has the same accuracy equal to 0.5. However, the second example has worse *Ordinal Accuracy* $Ord_{Acc} = 0.75$ than the first one since row #2 has a longer distance (higher error rate) for both *Jan* and *Mar* to their ground truth positions. Although we evaluate the benchmark on both evaluation metrics, we recommend Ord_{Acc} for ordinal evaluation.

#	Ranked Values	Acc	Ord_{Acc}
1	*[**Feb** ≺ **Jan** ≺ Mar ≺ Apr]*	0.5	0.87
2	*[**Mar** ≺ Feb ≺ **Jan** ≺ Apr]*	0.5	0.75

Table 3: A comparison between a classical accuracy metric (Acc) and an ordinal accuracy metric (Ord_{Acc}) for ground truth ordinal values of *[Jan ≺ Feb ≺ Mar ≺ Apr]*

Table 4 shows the evaluation results on 10 different data sets for 42 distinctive features of annotated ground truth of ordinal values. Ord_{Acc} corresponds to ordinal accuracy and Acc corresponds to classical accuracy metric. As part of the pre-processing stage, we consider a set of special categories where it can be removed from unique values due to general information or non-ordinal values, such as: "?", Null string, Null value, "unknown", "unmeasured", etc. We remove special categories from both BERT-Sort and *OrdinalEncoder* to have an unbiased comparison. As shown in this table, BERT-Sort improves the performance in terms of both Ord_{Acc} and Acc on all four different initiated MLMs. BERT-Sort with M_2 initialization (RoBERTa) achieves the best performance with significant improvements of 27% and 55% against the baseline based on Ord_{Acc} and Acc metrics, respectively.

4.3 AutoML Evaluation.

We use 5 AutoML tools including AutoGluon, FLAML, H2O, MLJAR and TPOT to evaluate the effectiveness of encoded categorical ordinal features by BERT-Sort to the ultimate machine learning performance. We evaluate two versions of 10 data sets, the original data set as baseline and the data set after applying BERT-Sort encoding, in 5 AutoML tools. We split each data set into 75% which is sent to each AutoML platform, and 25% is used to test the trained model. We limit the execution time of AutoML tools to 1 hour. Note that the data sets in our evaluation include all different features including numerical, text, categorical, etc.

All AutoML tools failed on two data sets (regressions) and success on 8 classification problems (explained in Appendix C). Although nominal values and other features (such as text, and numerical) may contribute to model prediction, BERT-Sort improves the performance of 4 AutoML tools as shown in Table 5. Our evaluation of all data sets on 5 different AutoML tools shows that encoding ordinal values through BERT-Sort improves the overall performance of platforms. In this experiment, the results shows that BERT-Sort encoded data set achieved 10.61% *performance gain* in compare to

Table 4: A Comparisons of semantic ordinal value evaluation of BERT-Sort with initiation on 4 different MLMs of DistilBERT, RoBERTa, XLM, BERT-base-uncased ($M_{1..4}$, respectively), and OrdinalEncoder with two metrics of Ordinal Accuracy (Ord_{Acc}) and classical Accuracy (Acc) metrics on 10 different data sets and 42 distinctive features; (champions marked in **bold**; 🏆 indicates BERT-Sort feature champion in all models based on both metrics; 🏆 indicates OrdinalEncoder feature champion against BERT-Sort based on both metrics)

Evaluation	Ord_{Acc}					Acc				
Approach	BERT-Sort				OrdinalEncoder	BERT-Sort				OrdinalEncoder
Model / Feature	M_1	M_2	M_3	M_4		M_1	M_2	M_3	M_4	
$\mathcal{F}_1$	**0.92**	**1.00**	0.76	**0.84**	0.76	**0.60**	**1.00**	**0.40**	**0.40**	0.00
$\mathcal{F}_2$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_3$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_4$	**1.00**	**1.00**	0.78	**0.78**	0.56	**1.00**	**1.00**	**0.33**	**0.33**	0.00
$\mathcal{F}_5$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_6$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_7$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_8$	**0.84**	**1.00**	**1.00**	**0.84**	0.76	**0.40**	**1.00**	**1.00**	0.60	0.20
$\mathcal{F}_9$	**0.76**	**0.76**	**0.76**	**0.84**	0.68	0.00	0.00	0.20	**0.40**	**0.20**
$\mathcal{F}_{10}$	**0.94**	**0.78**	**0.78**	**0.89**	0.72	**0.67**	0.00	0.17	**0.67**	**0.33**
$\mathcal{F}_{11}$	**0.78**	**0.78**	**0.78**	**1.00**	0.56	0.33	0.33	0.33	**1.00**	0.33
$\mathcal{F}_{12}$	1.00	1.00	0.78	0.78	1.00	1.00	1.00	0.33	0.33	1.00
$\mathcal{F}_{13}$	**0.74**	**0.93**	0.72	**0.82**	0.72	**1.00**	**0.50**	0.00	**0.25**	0.00
$\mathcal{F}_{14}$	0.75	**1.00**	**1.00**	**0.88**	0.75	0.00	**1.00**	**1.00**	**0.50**	0.25
$\mathcal{F}_{15}$ 🏆	**1.00**	**1.00**	**1.00**	**0.88**	0.75	**1.00**	**1.00**	**1.00**	**0.50**	0.25
$\mathcal{F}_{16}$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\mathcal{F}_{17}$ 🏆	1.00	0.78	1.00	1.00	**1.00**	1.00	0.33	1.00	1.00	**1.00**
$\mathcal{F}_{18}$ 🏆	**1.00**	**1.00**	**1.00**	**0.78**	0.56	**1.00**	**1.00**	**1.00**	0.33	0.33
$\mathcal{F}_{19}$ 🏆	**0.78**	**1.00**	**1.00**	**0.78**	0.56	**0.33**	**1.00**	**1.00**	**0.33**	0.00
$\mathcal{F}_{20}$	0.75	**1.00**	**1.00**	0.88	0.75	0.00	**1.00**	**1.00**	**0.50**	0.25
$\mathcal{F}_{21}$	0.62	**0.88**	**1.00**	0.75	0.75	0.00	**0.50**	**1.00**	**0.50**	0.25
$\mathcal{F}_{22}$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.56	**1.00**	**1.00**	**1.00**	**1.00**	0.33
$\mathcal{F}_{23}$ 🏆	**0.78**	**1.00**	**1.00**	**0.78**	0.56	**0.33**	**1.00**	**1.00**	**0.33**	0.00
$\mathcal{F}_{24}$	**1.00**	**1.00**	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	**1.00**	0.00
$\mathcal{F}_{25}$	**0.84**	**1.00**	**1.00**	**0.67**	0.63	0.14	**1.00**	**1.00**	0.14	0.14
$\mathcal{F}_{26}$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\mathcal{F}_{27}$	0.78	**1.00**	**1.00**	**1.00**	0.78	0.33	**1.00**	**1.00**	**1.00**	0.33
$\mathcal{F}_{28}$	**0.78**	**0.78**	**0.78**	**1.00**	0.56	0.33	0.33	0.33	**1.00**	0.33
$\mathcal{F}_{29}$	**0.76**	**1.00**	**0.76**	**0.84**	0.68	0.20	**1.00**	0.20	**0.40**	0.20
$\mathcal{F}_{30}$	**0.75**	**1.00**	**0.75**	**0.75**	0.50	**0.25**	**1.00**	**0.50**	0.00	0.00
$\mathcal{F}_{31}$	0.78	**1.00**	**1.00**	0.78	0.78	0.33	**1.00**	**1.00**	0.33	0.33
$\mathcal{F}_{32}$	0.78	0.78	0.78	0.78	0.78	0.33	0.33	0.33	0.33	0.33
$\mathcal{F}_{33}$ 🏆	0.76	0.68	0.84	0.84	**0.92**	0.00	0.00	0.60	0.60	**0.60**
$\mathcal{F}_{34}$ 🏆	**1.00**	**1.00**	**1.00**	**0.78**	0.56	**1.00**	**1.00**	**1.00**	**0.33**	0.00
$\mathcal{F}_{35}$ 🏆	**0.78**	**1.00**	**1.00**	**0.78**	0.56	**0.33**	**1.00**	**1.00**	**0.33**	0.00
$\mathcal{F}_{36}$	**0.78**	**1.00**	**1.00**	**0.78**	0.56	**0.33**	**1.00**	**1.00**	**0.33**	0.00
$\mathcal{F}_{37}$	**1.00**	**1.00**	0.78	**1.00**	0.78	**1.00**	**1.00**	0.33	**1.00**	0.33
$\mathcal{F}_{38}$	**1.00**	**1.00**	0.78	0.78	0.78	**1.00**	**1.00**	0.33	0.33	0.33
$\mathcal{F}_{39}$	0.75	0.75	0.88	**0.88**	**0.88**	0.25	0.00	0.50	0.50	**0.50**
$\mathcal{F}_{40}$	**0.78**	**1.00**	**1.00**	**0.78**	0.56	0.33	**1.00**	**1.00**	0.33	0.33
$\mathcal{F}_{41}$	**0.78**	**1.00**	**1.00**	**0.78**	0.56	0.33	**1.00**	**1.00**	0.33	0.33
$\mathcal{F}_{42}$ 🏆	**1.00**	**1.00**	**1.00**	**1.00**	0.56	**1.00**	**1.00**	**1.00**	**1.00**	0.33
#champions	30	**35**	31	32	3	23	**31**	27	28	5
	w.r.t. OrdinalEncoder				w.r.t. M_2	w.r.t. OrdinalEncoder				w.r.t. M_2
Improvement	0.20	**0.27**	0.25	0.20	baseline	0.31	**0.55**	0.49	0.34	baseline

	#Success		F1 Score	
	Raw	BERT-Sort	Raw	BERT-Sort
AutoGluon	3	3	0.259	**0.266**
FLAML	3	3	0.284	**0.289**
H2O	8	8	**0.721**	0.688
MLJAR	7	8	0.557	**0.647**
TPOT	0	1	0.000	**0.124**
Total/Avg.	21	**23**	0.364	**0.403**

Table 5: A comparison evaluation between different AutoML tools on 10 benchmark data sets where *Raw* refers to the original data set and *BERT-Sort* refers to encoded original data set through BERT-Sort encoder; *#Success* denotes the number of successful trained models

the original data set. In Table 5, *Raw* shows the results of trained and tested on original data set; *BERT-Sort* shows the results of trained/tested on updated data set where the ordinal values have been encoded through BERT-Sort and the rest of features remain unchanged.

5 Discussions

As shown in Table 4 encoding ordinal values by BERT-Sort with initialization of any MLM indicates BERT-Sort is able to achieve the state-of-the-art semantic encoding performance on categorical features. On the other hand, these results show that widely used categorical encoding functions such as *OrdinalEncoder* will *lead to diminish the semantic order of ordinal values*.

Interestingly, BERT-Sort can go beyond expectation by sorting categorical data where it is too complex for data scientists to rank elements manually. For instance, by initiating BERT-Sort on *BioClinical BERT*(Alsentzer et al., 2019) it can sort the severity of cancer as *[Melanoma* $>$ *Leukemia* $>$ *Cancer]* without any supervision (a supervised approach used by Lausser et al. (2020) to generate this order). More details about the broader impact of BERT-Sort usages across different domains, languages (i.e., Chinese, Japanese and Spanish) and semantic image sorting are explained in Appendix E. We evaluate BERT-Sort on 10 different benchmarks by using 5 different AutoML tools. These data sets contains many other non-ordinal features including nominal, text, numerical etc. Therefore, the evaluation results may reflect the generalization of the proposed approach but it might not clearly show the significant importance of ordinal value encoding. As a future work, we plan to extend this study by utilizing web automation to apply BERT-Sort to many open public data sets where BERT-Sort can detect and generate a public universal categorical encoder.

6 Conclusion

In this paper, we introduced BERT-Sort, a novel automated approach to detect and encode categorical ordinal features during the data pre-processing step of machine learning pipelines. BERT-Sort significantly improves the performance of the state-of-the-art categorical encoders (i.e., Scikit-Learn OrdinalEncoder) by 22%. We introduced and published a benchmark that includes 10 different public data sets with 42 different ordinal features. We conducted an extensive evaluation on the benchmark where BERT-Sort is initialized on four popular MLMs of DistilBERT, RoBERTa, XLM and BERT-base. We also evaluated the effectiveness of the encoded features by BERT-Sort on various AutoML tools and compared the performances of BERT-Sort encoded data sets with original data sets in 10 benchmarks. Our experiments show that BERT-Sort successfully improved the overall performance of AutoMLs with 10.61%.

References

(2021). Numpy Sort. https://github.com/scikit-learn/scikit-learn/sklearn/preprocessing/_encoders.py#L74. Accessed: 2022-02-01.

(2021). SkLearn OrdinalEncoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html. Accessed: 2022-02-01.

(2022a). Autogluon. https://auto.gluon.ai/dev/tutorials/tabular_prediction/tabular-feature-engineering.html. Accessed: 2022-03-02.

(2022b). AutoKeras. https://autokeras.com/block/#categoricaltonumerical-class. Accessed: 2022-03-02.

(2022). Mljar feature preprocessing. https://mljar.com/automated-machine-learning/features-preprocessing/. Accessed: 2022-03-02.

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Jin, D., Naumann, T., and McDermott, M. (2019). Publicly available clinical bert embeddings. *arXiv preprint arXiv:1904.03323*.

Christensen, R. H. B. (2015). ordinal—regression models for ordinal data. *R package version*, 28:2015.

Dahouda, M. K. and Joe, I. (2021). A deep-learned embedding technique for categorical features encoding. *IEEE Access*, 9:114381–114391.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2020). Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074*.

Harrell, F. E. (2015). Ordinal logistic regression. In *Regression modeling strategies*, pages 311–325. Springer.

He, X., Zhao, K., and Chu, X. (2021). Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622.

Lausser, L., Schäfer, L. M., Kühlwein, S. D., Kestler, A. M., and Kestler, H. A. (2020). Detecting ordinal subcascades. *Neural Processing Letters*, 52(3):2583–2605.

LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., Kraljevic, T., Nykodym, T., Aboyoun, P., Kurka, M., et al. (2018). Package 'h2o'. *dim*, 2:17.

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):109–127.

Mosley, L. (2013). A balanced approach to the multi-class imbalance problem.

Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Płońska, A. and Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3.

Porter, B. (2019). UCI audiology (standardized) data set. https://archive.ics.uci.edu/ml/datasets/Audiology+%28Standardized%29. Accessed: 2022-02-01.

Urbanowicz, R. J. and Moore, J. H. (2015). Exstracs 2.0: description and evaluation of a scalable learning classifier system. *Evolutionary intelligence*, 8(2):89–116.

Zhou, Z. and Hooker, G. (2021). Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–21.