Categorical Data Deduplication

Soham Pachpande
Gehan Chopade

ABSTRACT

Data Preparation involved in Machine Learning Pipelines reduces ML user productivity and hinders progress. One of the data preprocessing steps is the identification and removal of duplicates in categorical data. Encoding Categorical duplicates as different entities poses risk to ML model accuracy. In this work, we look at statistical features of categorical data, build a dataset with duplicate and non-duplicate word pairs, and build a binary classifier to distinguish between duplicate and non-duplicate word pairs. We perform feature engineering using text comparison metrics and model selection over classic ML models such as Logistic Regression, Trees, and SVMs; and embedding-based Neural network models. Our best performing model trained on a held-out cross-validation - a Random Forest binary classifier achieves an F1 score of 0.95

KEYWORDS

datasets, text features, machine learning

1 INTRODUCTION

Data Cleaning is one of the most important and tedious sub-process of Machine Learning data preparation and involves tedious grunt work. And model performance is directly affected because of dirty data even in AutoML systems. Data Acquisition in the real world is often flawed and one of the data issues is categorical duplicates. For example, in table 1, we can see that the categories *"SWE"* and *"Software Engineer"* imply the same job title by common knowledge but are represented differently. This dilutes the strength of the category and affects downstream model accuracy.

In this project, we study the statistical features of duplicates in categorical variables and build a binary classifier to predict whether two words(categories) are duplicates of one another or not. Note that we only use the categorical words and not the metadata making this problem different from entity resolution. To the best of our knowledge, we did not find any related work which has directly worked on categorical data duplication without any contextual information or metadata.

2 OUTLINE

We divide the report into the following sections. First, we describe our dataset and data preparation process in section 3. We formally define our Task in section 4. We discuss our approaches, experimental results, and analysis in section 5. We then conclude our report and give a brief outline for future work.

Location	Job Title	Education	Salary Grade
SF	Software Engineer	Masters	A
SF, CA	SWE	MS	A

Table 1: Salary Grade Predication sample Data

3 DATA

3.1 Dataset

We use a labeled dataset for Categorical Duplicates [3] from the ML Data Prep Zoo project [2]. This dataset is prepared by annotating true entities within a Categorical column with corresponding duplicates. This dataset includes 1248 string categorical columns from 217 datasets stored as raw CSV files. Within this dataset, a total of 52 string columns (which represent 33 out of the 217 datasets) contain one or more categorical duplicates.

3.2 Data Preparation

We preprocess our dataset by creating word pairs from the dataset. To get duplicate word pairs we pick entities that occur more than once in a categorical column and we group every word with the same annotated entity as a duplicate pair.

Category Set	Category Occurrences	Total	Entity Number
Euro	25	263	1
U.S. Dollar	15	263	2
US Dollar	1	263	2
Franc	8	263	3

Table 2: Example Annotation for a single Categorical Column

For example, the table 2 shows the annotated data for a subset of a single Categorical column in our dataset. The two category sets - "U.S. Dollar" and "US Dollar" are duplicates of one another and are annotated with the same entity number. We group all such category sets which are annotated with the same entity number from every column and label them as duplicate word pairs. For non-duplicates, we randomly sample two category sets from a randomly sampled category column, ensure that the sampled category set belongs to a different entity, and label them as non-duplicates. Table 3 and Table 4 shows annotated duplicate and non duplicate word pairs respectively.

The total number of possible non-duplicate word pairs outnumber the number of duplicate words pairs by a large ratio, this leads to a heavy class imbalance. To avoid bias due to class imbalance, we

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

word1	word2	is Duplicate
Viewbank	viewbank	1
Croydon	croydon	1
adelaide	Adelaide	1
Sefton Park	SEFTON PARK	1
U.S. dollar	U. S. dollar	1
ALASKA	AK	1
New York, NY	New York, NY 10012	1
White & Cream	White/Cream	1

Table 3: Annotated Duplicate Word Pairs

word1	word2	is Duplicate
Curfew	Aggravated Assault	0
Military Department	Public Defender, State	0
French Guiana	Mauritius	0
Prep	End	0
Computer & Electronics Retail	Leisure Products	0

Table 4: Annotated Non Duplicate word pairs

ensure that the ratio of the number of duplicates to non-duplicate word pairs is approximately 1:5. We find 1782 duplicate word pairs in our dataset and we sample 9500 non-duplicate word pairs.

4 TASK

We define our task as follows: Given a word pair, we aim to build a binary classifier to identify whether the given words are duplicates or non-duplicates. We resolve Categorical duplicates at a column level with no extra metadata.

5 APPROACHES AND RESULTS

We present three approaches. For each approach, we give a background observation and the algorithm. Finally we analyse the strength and weakness of each approach. The experimental results are described in table 5.

5.1 Heuristic based Approach

5.1.1 Model: We observe that a large portion of duplicate words differ in special characters (ex: *"Black-Blue"* vs *"Black/Blue"*), Capitalization (ex: *"United States"* vs *"united states"*) and presence of extra information (ex: *"San Diego"* vs *"San Diego, CA"*). We come up with a heuristic baseline to first remove white spaces and special characters (via a *strip* function) and convert words to a uniform lower case representation. We then compute the edit distance between processed string forms of the word pair and compare with a manually set threshold to classify word pairs as duplicate or non-duplicate. The algorithm 1 describes our heuristic-based method. To compute the edit distance, we penalize insertion and deletion by a factor of 1 and substitution of characters by a factor of 2.

Algorithm 1 Heuristic Algorithm to identify duplicates

```
Require: Threshold(Th), Minimum Length(min)
  $w1 \leftarrow lowercase(strip(word1))$
  $w2 \leftarrow lowercase(strip(word2))$
  $editDistance \leftarrow editDistance(w1, w2)$
  if $editDistance \geq Th$ then return False
  else return True
  end if
```

5.1.2 Analysis: The results for this model can be seen in table 5. We find a good precision but a very poor recall for this approach. Thus the heuristic model is good at identifying non-duplicates but struggles with accurately identifying duplicates. This is not surprising as non-duplicates often drastically differ in semantics as well and syntax whereas duplicate word pairs have very similar syntactic features. We, therefore, shift our attention to recall and build approaches to give a good recall and overall F1 score.

5.2 Embedding based Neural Networks

5.2.1 Model: Semantic word embedding models of languages represent each word with a real-valued vector such that word embeddings for any two words with similar semantic meaning or embeddings are more similar than embeddings for dissimilar words. We observe that categorical duplicates have similar meanings and propose a model to use the GloVe Embedding model [1] to represent our word pairs and train a shallow Neural Network to perform the required binary classification task. Figure 1 describes our neural network architecture. We generate vector embeddings of length 100 using GloVe embedding pre-trained model [1] and concatenate the embeddings for our word pair to get a vector of length 200. These concatenated vector embeddings are passed onto a fully connected neural network which is trained using Stochastic Gradient Descent with cross-entropy as the loss metric.

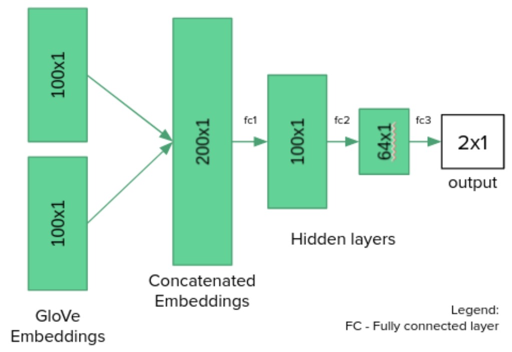

Figure 1: Embedding Based Neural Network Classifier

5.2.2 Analysis: One major caveat of this approach is that vector embedding models contain vector representations of words that belong to the corpus. This is not true in the case of our categorical duplicate dataset. Often words have spelling mistakes or do not have any semantic meaning. During our experiments, we found that only 37.2% of words in our dataset had word embeddings. This

Features	Model	Precision	Recall	F1-Score
Original Words	Heuristic	.87	.73	.78
GloVe Embeddings	Embeddings (on a subset of dataset)	.95	.93	.94
N-Grams	Logistic Regression	.76	.76	.76
	SVM	.76	.73	.73
	Random Forest	.95	.95	.95
Distance Metrics	Logistic Regression	.90	.81	.85
	SVM	.90	.86	.88
	Random Forest	.95	.94	.95

Table 5: (Macro-averaged) Results for All Approaches

renders embedding models to be impractical. Therefore, despite the good model performance, we shift our attention back towards syntactic textual features.

5.3 Machine Learning Models on Textual Features: N-Grams

An N-gram is a sequence of n words generated from a text that can be modeled as features for language models. In this work, we decided to leverage character-level n-grams, taking motivation from word-level n-grams used in language modeling. The general idea here is to find probabilities associated with the occurrence of characters that occur in two words next to each other. To demonstrate using an example, if we were to compare two words such as San Diego and San Diego City, we will the occurrence of characters in both these sequences can be used to develop features. If we onlt consider bi-grams generated from these two words, we will get [sa,an,n , d,di,ie,eg,go] and [sa,an,n , d,di,ie,eg,go,o , c,ci,it,ty]. We propose that we look at bi-grams and tri-grams of the pair of words and use them as features for our machine learning models. We use these features to build Logistic Regression, Support Vector Machine, and Decision Tree models.

5.3.1 Analysis: We observe that the dataset that is generated after computing bi-grams and tri-grams for two words is significantly large and all of the features generated do not contribute to the inference that is generated by the model making most of these features redundant. It is also difficult to understand why certain ngrams are more important then others which reduces model explainability.

5.4 Machine Learning Models on Textual Features: Distance metrics based similarity score

The distance metric is used vastly in machine learning applications to recognize the similarity between two data points. We have already used edit distance in the previous section in our heuristic approach. But the heuristic approach has a shortcoming where the results of that model significantly depend on the threshold that we manually tune. There is a possibility that the threshold is a function of the textual features of words which is difficult to tune manually. To get past this shortcoming, we decided to automate the thresholding process by providing distance metrics between two words as features and feeding them to a machine learning model, and make the machine learning model predict if the two inputs belong to the same category or not. We limit our scope for this work to three distance metrics - Levenshtein distance and Cosine and Jaccard distances generated from uni-grams, bi-grams, and tri-grams. We again train these set of features with Logistic Regression, SVM, and Random forest classifier.

5.4.1 Analysis: We observe that our performance significantly improves as compared to the previous approach with only 7 features present (Levenshtein distance, cosine distance, and Jaccard distance for uni-gram, bi-gram, and tri-gram). We were further curious about which features contributed the most to this improved performance. So we ran a series of feature selection and model selection experiments. We perform a nested cross-validation experiment on our best performing model, a random forest classifier, to determine feature importance and model hyperparameters. The outer cross-validation is k-fold cross-validation with 10 folds and the inner cross is a 3 fold grid search cross-validation. We validate on a list of 10,50, and 100 trees with a list of 2,4, and 6 features. In this experiment, we observed that we get the best configuration with 4 features (cosine and Jaccard distances generated from bigrams and trigrams of the input words) and 100 trees. On further investigation, we concluded that Levenshtein distance negatively impacts the performance of the model. We used the results from our nested cross-validation experiment and validated them on our dataset. The results are displayed in table 5.

6 FUTURE SCOPE

Although we achieve f1 score of 0.95 with a random forest classifier in our distance metric approach, there is still room for improvement. We look forward to implementing this approach on an exhaustive set of datasets that have more types of categorical duplicates, especially synonyms and entity duplicates. We also plan on testing our approach with downstream benchmark models after we have removed categorical duplicates from the datasets using the approach mentioned above. This will give us a fair idea about how much does the performance varies in absence of categorical duplicates. Lastly, right now we are only considering distance metrics as features and classifying categorical duplicates. We would like to use contextual information from existing datasets and form a pair of context vectors for every pair of words which can then be used as a feature vector for our machine learning models.

7 CONCLUSION

Currently, the process of handling categorical data in a dataset is a part of a data cleaning process that is largely dependent on the exploratory data analysis task that is performed on the dataset. In the scope of this work, we have taken steps towards automating this process. We went through a holistic process of classifying categorical duplicates in this work. We start off with a basic heuristic approach and build towards an embedding-based approach that leverages deep neural networks. We then look at more explainable models diverting our attention to using textual features like n-grams and distance metrics in machine learning models where we achieve state-of-the-art results with distance metrics on a random forest classifier with a f1 score of 0.95.

REFERENCES

[1] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[2] Vraj Shah and Arun Kumar. 2019. The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD 2019, Amsterdam, The Netherlands, June 30, 2019*, Sebastian Schelter, Neoklis Polyzotis, Stephan Seufert, and Manasi Vartak (Eds.). ACM, 11:1–11:4. https://doi.org/10.1145/3329486.3329499

[3] Vraj et al. Shah. [n. d.]. An Empirical Study on the (Non-)Importance of Cleaning Categorical Duplicates before ML. https://adalabucsd.github.io/papers/TR_2021_CategDedup.pdf