CS5011: P2 - Machine Learning

190018035

March 13th 2024

Contents

1 Introduction

For this practical, I was tasked with implementing and evaluating different machine learning models on the Pump It Up: Data Mining the Water Table [1] dataset. This involved experimenting with different preprocessing steps, hyperparameters, and machine learning models to find the best model to predict the status of water pumps in Tanzania. The checklist below details my achievements in the practical for each of the specified parts.

1.1 Project Achievements

- Part 1: Attempted and Fully Working

- Part 2: Attempted and Fully Working

1.2 Usage Instructions

To run the script, navigate to the root directory and run the following command:

```
python3 part1.py <train-input-file> <train-labels-file> <test-input-file> <
    numerical-preprocessing> <categorical-preprocessing> <model-type> <test-
    prediction-output-file>
```

The values for each of the arguments are equivalent to that described in the assignment specification. However, one difference is that the `<numerical-preprocessing>` and `<categorical-preprocessing>` arguments now take in a `Manual` option, which allows features to be preprocessed based on the encoding that are most suitable for the model.

2 Part 1

In this section, I will detail the various preprocessing steps I took to prepare the data for the machine learning models, guided by a deep exploration of the dataset. I will also discuss the design decisions I made while implementing this part, including the parameters of the preprocessing methods and the hyper-parameters of the machine learning models.

2.1 Preprocessing Steps

Prior to implementing the machine learning models, I performed a number of preprocessing steps to clean and prepare the data for the models. These steps were guided by a deep exploration of the dataset, where I analyzed the distribution of the features, the relationships between the features, and other important characteristics of the dataset. In conjuction with the steps outlined below, I will also discuss my findings from the exploration that lead to the decisions I made.

2.1.1 Cleaning the Data

Removing Irrelevant Features   The first step I took was to remove any features that were irrelevant to the prediction task. These features either did not directly contribute to the status of the water pumps or contained too many missing values. The features that were removed include:

- `id` The identifier for the water pump is not relevant to the prediction task

- `wpt_name` The name of the water pump is unlikely to contribute to the prediction task

- `scheme_name` The name of the water scheme is unlikely to contribute to the prediction task

- `num_private` This feature contains mostly missing values, with over 99.24% of the values missing (defined as 0). Additionally, the feature does not have a clear definition, so it is unclear how it would contribute to the prediction task.

- `amount_tsh` Similarly, this feature contains mostly missing values, with over 70.1% of the values missing (also defined as 0).

Removing Single-Value Features   I also removed any features that only contained a single value, as they do not contribute to the prediction task. There was only one feature that met this criteria, which was `recorded_by`.

Removing Redundant Features There were a number of features that were redundant, meaning they contained the same information as another feature. I removed these features to reduce the dimensionality of the dataset. There were a few considerations I made when deciding which of the features that are redundant to remove. The first consideration was the cardinality of the feature - having too many unique values would make the feature difficult to encode, whilst having too few unique values would make the feature less informative. The second consideration was the number of missing values in the feature - if a feature had a large number of missing values, it would be less informative. The features that were removed include:

- `payment` This feature contains the same information as `payment_type` with the same value counts, so I removed it and retained `payment_type`.

- `quantity` Similarly, this feature contains the same information as `quantity_group` with the same value counts, so I removed it and retained `quantity_group`.

- `extraction_type` This feaure was removed in favor of `extraction_type_group` and `extraction_type_class`. The `extraction_type_class` feature provides a more general classification of the extraction type. It contains only 7 unique values, which makes it easier to encode, and the smallest value count for a given category for this feature is 117. The `extraction_type_group` feature, on the other hand, contains 13 unique values, where the smallest value count for a given category is 98. Finally, the `extraction_type` feature contains 18 unique values, where the smallest value count for a given category is 2. From this information, we can see that the `extraction_type_class` feature is the most general, and the `extraction_type` feature is the most specific. In order to provide more useful information to the model, I decided to remove the `extraction_type` feature and retain the other two, as this feature has the highest cardinality with the smallest value count for a given category whilst the other two features have a lower cardinality with a higher value count for a given category, thereby providing more useful information to the model.

- `scheme_management` This feature was removed in favor of `management`. These two features contain most of the same categories; however, the `scheme_management` feature contains 3878 missing/unknown values whereas management only contains 561 unknown values. This makes the `management` feature more informative and easier to encode, so I decided to remove the `scheme_management` feature.

- `water_quality` This feature was removed in favor of `quality_group`. These two features contain mostly the same information, where `water_quality` splits "salty" and "flouride" into two subcategories (introducing a "salty abandoned" and "flouride abandoned" subcategory). These subcategories contain comparatively few values, so I decided to remove the `water_quality` feature and retain the `quality_group` feature to reduce the dimensionality of the dataset.

- `source` This feature was removed in favor of `source_type` and `source_class`. The `source_class` feature is the most general, containing only 3 unique values, and the `source_type` feature sits in the middle, containing 7 unique values and containing very similar columns to `source`. The `source` feature contains 10 unique values. Since `source_class` provides the most general information, and since `source` contains the highest cardinality, I decided to remove the `source` feature and retain the other two so there is a balance between the general and specific information provided to the model.

- `waterpoint_type_group` This feature was removed in favor of `waterpoint_type`. The `waterpoint_type` splits up the "communal standpipe" category into "communal standpipe"

and "communal standpipe multiple". However, since "communal standpipe multiple" only contains contains relatively few values (6103), I decided to remove the `waterpoint_type_group` feature and retain the `waterpoint_type` feature to reduce the dimensionality of the dataset.

Replacing Construction Year with Decades The `construction_year` feature contained a large number of missing values, with 34.8% of the values missing. To make the feature more informative, I decided to categorize the construction years into decades. This would allow the model to learn the relationship between the construction year and the status of the water pumps, without imputing values that might distort the underlying distribution of the data. The decades were defined from 1960s to the 2010s, with the missing values being replaced with the "Unknown" category.

Imputing Missing Values

Fixing Formatting Issues

Limiting High Cardinality Features There were a few features in the dataset, specifically `funder`, `installer`, subvillage and `ward`, that contained a large number of unique values. Such high cardinality could pose challenges, as encoding these features could significantly increase the dimensionality of the dataset, potentially leading to issues like overfitting and increased computational demand. To address this, I set a threshold (95%) that identifies the most frequent categories that cumulatively account for 95% of the values in the feature. This threshold was determined through manual experimentation, but can be set through hyperparameter optimization. Through this strategy, I was able to reduce the number of dummy variables needed for encoding and makng the dataset size more manageable. The remaining categories were replaced with the "Other" category. Additionally, I also replaced the missing values with the "Unknown" category. This strategy resulted in the following reductions in the number of unique values for each of the features (after fixing formatting issues):

Feature	Original Unique Values	Reduced Unique Values
funder	1883	405
installer	1905	397
subvillage	19287	16336
ward	2092	1578

Table 1: Reduction in Unique Values for High Cardinality Features

Converting Datetime Features

2.1.2 Encoding the Data

2.2 Design Decisions

2.3 Evaluation

3 Part 2

For this part of the practical, I utilized Optuna [2], an automated hyper-parameter optimization (HPO) tool to find the best hyper-parameters for each of the models. In this section, I will detail

my HPO process and analyze the results of the experiments.

3.1 Hyper-Parameter Optimization

3.2 Evaluation

4 Conclusion

References

[1] URL: https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/.

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2019.