Project Report: Predicting Water Pump Functionality in Tanzania

220034672

March 13, 2024

Contents

1 Introduction

This project leverages a dataset from the "Pump it Up: Data Mining the Water Table" competition hosted by Driven Data and is used throughout (Driven Data 2021) . The dataset comprises features of water pumps across Tanzania and their operational status: functional, functional needs repair, or non-functional. The goal is to predict the status of water pumps based on the dataset using machine learning models, thereby also gaining insight into the models' effectiveness. This task necessitates thorough data pre-processing, including handling categorical features, missing values, and numerical scaling, to prepare the data for model training and evaluation. The implementation uses Python, with sci-kit-learn, pandas, and numpy libraries.

2 Setup and Initial Data Exploration

The objective was to acquire a comprehensive understanding of the dataset's structure, including its features and target variables. We focused on identifying and understanding the types of features and detecting any missing values. This step was crucial, as I quickly identified many problems with the dataset, which were relevant for planning the subsequent pre-processing steps.

3 Data Pre-processing

Our objective was to prepare the dataset for modelling by addressing issues identified during the exploration phase.

The actions we took were as follows:

Dropping Unnecessary Columns: The first step was to delete unnecessary columns. Many features were duplicates or deemed logically redundant the dropped columns were:

- date_recorded
- payment_type
- wpt_name
- source, source_class

- waterpoint_type_group
- management_group, scheme_name, scheme_management
- recorded_by
- sub village
- region
- lga
- ward
- extraction_type_group
- extraction_type_class
- quality_group
- quantity_group
- source_type
- num_private

4 Categorical Feature Encoding Strategies

To effectively handle the diverse nature of categorical data within our dataset, we employed two primary encoding strategies: Target Encoding for high cardinality features and One-Hot Encoding for binary features.

4.1 High Cardinality Features

For features with a high number of categories—such as 'funder', 'installer', 'basin', 'extraction_type', 'management', 'payment', 'water_quality', 'quantity', and 'waterpoint_type'—we applied Target Encoding according to Micci-Barreca (2001). This method is particularly beneficial for its efficiency in managing features with numerous unique categories. Rather than generating a vast array of dummy variables, which could introduce issues related to high dimensionality, Target Encoding transforms these categories into numerical values based on their association with the target variable. This preserves essential information in a more compact representation. To ensure consistent processing of missing values, they were imputed with 'Unknown', allowing the model to utilize all available data without excluding any due to missing entries.

4.2 Binary Features

One-Hot Encoding was utilized for binary variables such as ‘public_meeting‘ and ‘permit‘. Given these features only contain boolean values and lack a natural order, this encoding strategy creates a binary column for each category, signifying the presence or absence of a category with a predictor value. This approach is straightforward and effective for binary data.

5 Numerical Feature Scaling

Numerical feature scaling is a crucial preprocessing step to enhance machine learning model performance. We adopted the StandardScaler from the scikit-learn library for this purpose. This scaler standardizes each feature to have a mean of zero and a standard deviation of one.

Selected numerical features for scaling included:

- ‘amount_tsh‘: Total static head (the amount of water available to the waterpoint).
- ‘gps_height‘: Altitude of the well.
- ‘longitude‘ and ‘latitude‘: GPS coordinates.
- ‘region_code‘, ‘district_code‘: Geographic locations (administrative).
- ‘population‘: Population around the well.
- ‘construction_year‘: Year the waterpoint was constructed.

5.1 Implementation Details

The StandardScaler was initialized and fitted to our dataset, calculating mean and standard deviation for each numerical feature. This ensures data is scaled uniformly, preventing any single feature from disproportionately influencing the model due to its magnitude.

Post fitting, the scaler transformed the selected numerical features, centering their distribution around zero and scaling to a unit standard deviation. This standardization is crucial for minimizing bias in models sensitive to feature magnitude, such as SVMs and k-NN.

We applied this scaling consistently to both the training and test datasets. For the training set, we fitted the scaler and transformed the data accordingly. For the test set, we applied the transformation using the training set's parameters, maintaining evaluation integrity by ensuring consistent scaling.

By replacing the original numerical values with their standardized equivalents, we preserved data integrity and ensured the dataset was primed for machine learning algorithms. The target variable 'status_group' was also converted into numerical values to facilitate model training: functional (2), functional needs repair (1), non-functional (0).

The preprocessing steps, including both encoding strategies and numerical feature scaling, were implemented using pandas for data manipulation and sci-kit-learn for transformations. This prepared the dataset for subsequent model training and evaluation, highlighting the importance of these preprocessing steps in the overall machine learning workflow.

6 Conclusion

The preprocessing steps outlined in this project are critical for preparing the dataset for effective machine learning model training. By addressing missing values, encoding categorical features and scaling numerical values, we have laid the groundwork for building and evaluating machine learning models to predict the functionality of water pumps in Tanzania. The project's next phase will involve selecting, training, and evaluating suitable machine-learning models using the preprocessed dataset.

7 Model Training

We trained five different models with their default settings to establish baseline performances. The models included:

7.1 Logistic Regression

In the Logistic Regression model, we opted for the `'saga'` solver and set `max_iter` to 10,000 to cater to our extensive dataset. The `'saga'` solver was chosen for its efficiency with large datasets and compatibility with multiple penalty types, allowing us to manage feature multicollinearity through regularisation. We increased the iteration limit to aim for convergence, though still not achieved as the default setting proved insufficient, a sign of our dataset's complexity.

Standardisation of specific numerical columns using StandardScaler was crucial for models like Logistic Regression, which can be sensitive to feature scales.

We configured the `multi_class` setting to `'multinomial'` to address our multi-category target variable. This approach is typically more precise than `'one-vs-rest'` for balanced datasets.

The choice of the 'saga' solver aligns with our need for computational efficiency and accuracy in modelling. As we progress, we may need to recalibrate our model parameters for optimal performance, potentially considering alternatives like 'lbfgs' or SGDClassifier.

7.2 Random Forest Classifier

We leveraged a RandomForestClassifier with 100 trees to harness the power of ensemble learning for our project. The Random Forest algorithm, renowned for its robustness against overfitting, matched our pre-processing strategy, where we selectively scaled numerical columns.

After loading our pre-processed training and test datasets, we separated the features and the target variable, `'status_group'`. The RandomForestClassifier was instantiated with 100 estimators, setting the random state to 42 for reproducibility.

During model fitting, the Random Forest algorithm built numerous decision trees and predicted the majority vote across trees for each sample. This approach captures the complex relationships between features and the target variable and provides insights into feature importance, which can guide further feature engineering efforts. Our Random Forest model, trained on the cleaned and encoded dataset, showed promising accuracy, making it a strong candidate for further tuning. While the RandomForestClassifier effectively handles feature interactions and non-linearities, we may explore additional hyperparameter tuning to refine our model's predictive power further.

The pre-processing steps, including target encoding of categorical variables and selective feature scaling, were designed to prepare the dataset for this model type. To continue developing our project, we will consider advanced techniques such as grid or random search for hyperparameter optimisation, which could lead to even better model performance.

7.3 Gradient Boosting Classifier

This ensemble technique builds models stage-wise and generalises them by allowing the optimisation of an arbitrary differentiable loss function. We instantiated the Gradient Boosting Classifier with 100 trees, a learning rate of 0.1, and a max depth of 3, aiming for a strong learner without risking overfitting. The random state was set to zero to maintain consistent results across runs. This setup provides a balanced approach to learning complex patterns without becoming too specialised in the training data. Fitting the model to our training data, the Gradient Boosting Classifier iteratively built trees that corrected the errors of the preceding ones, enhancing the model's accuracy with each step. We utilised the trained model to predict the labels of the test set, converting numerical predictions back to the original categories of water pump functionality. Our prepared submission DataFrame, holding the id and predicted status_group, encapsulates our model's generalisation capabilities. This data frame was then saved as a CSV file and ready for evaluation. The initial classification scores indicate that the gradient-boosting classifier performed well and effectively captured the non-linear relationships and interactions within the dataset.

7.4 Histogram-Based Gradient Boosting Classifier

When initializing the `HistGradientBoostingClassifier`, several crucial parameters were set: `max_iter` was configured to 100, `learning_rate` was set to 0.1, and `max_depth` was established at 10. Additionally, `early_stopping` was enabled to automatically determine the optimal number of iterations and further prevent overfitting.

The `validation_fraction` was set at 0.1 to utilize a subset of the training data for validation, and `n_iter_no_change` was fixed at 10 to define the number of iterations without improvement on the validation set before ceasing the training process. Training the model involved fitting it with the feature set and target variable, enabling the histogram-based optimization to swiftly process the data and construct an effective ensemble of decision trees.

We loaded our encoded test dataset for predictions and applied the same pre-processing steps as used with the training dataset, with the exclusion of the 'id' column. This ensured alignment between the training and test dataset structures. Upon predicting the test labels, we translated the numerical predictions back to their original water pump functionality categories.

The prepared submission DataFrame, which associates each 'id' with its corresponding 'status_group' prediction, was then saved to a CSV file, encapsulating our predictive insights. The `HistGradientBoostingClassifier`'s initial performance indicators demonstrate its competency in handling complex datasets efficiently. To further enhance our model's accuracy, we might consider fine-tuning additional hyperparameters, such as the number of bins used in the histograms or experimenting with different loss functions.

7.5 MLP Classifier

A Multi-layer Perceptron classifier that trains using backpropagation. Due to its reliance on gradient descent, feature scaling is crucial for performance. In our quest to model complex relationships within our dataset, we employed an `MLPClassifier`, which is part of the neural network family in sci-kit-learn. Given its sensitivity to the scale of input features, we meticulously pre-processed our data, standardizing it to have a mean of zero and a variance of one. This step is crucial for neural networks to ensure that each input feature contributes proportionately to the learning process. Once we loaded and scaled our training data, we set up the `MLPClassifier` with a single hidden layer of 100 neurons—striking a balance between learning capacity and computational efficiency. The 'relu' activation function was selected for its efficiency and effectiveness in deep learning models. We opted for the 'Adam' solver, a popular choice for large datasets due to its adaptive learning rate capabilities. Other hyperparameters were carefully chosen: a low regularization term (`alpha`) to prevent overfitting while allowing the network enough flexibility to capture underlying patterns and a `max_iter` of 200 to give the network ample iterations to converge. The random state was set to 42 to maintain reproducibility. After training the `MLPClassifier` on our scaled data, we processed the test set with the same scaler to maintain consistency. We made predictions and then transformed the numerical outputs back into their original categorical status group labels, ensuring our results were interpretable and actionable. The final step involved creating a submission DataFrame, where we paired each test set 'id' with its predicted status group. We then saved this information into a CSV file for subsequent evaluation and reporting purposes. The `MLPClassifier`, with its deep learning capabilities, provided us with a powerful tool to capture non-linear complexities. For further model enhancements, we might experiment with different architectures, adding more hidden layers or adjusting the learning rate to optimize performance. Each model was fit to the training data, and predictions were made on the test set. The results were then evaluated to determine the effectiveness of each model in predicting water pump functionality.

7.6 Insights and Model-Specific Pre-processing

Logistic Regression required a significant number of iterations and a robust solver (`saga`) due to the multinomial nature of our target variable and the large dataset size. This model had the lowest accuracy, likely due to its linear nature, which may need help with the complexity of the relationships in the data.

Random Forest showed strong performance without any feature scaling, benefiting from its ensemble nature and robustness to the varying scales of features.

Gradient Boosting and Histogram-Based Gradient Boosting Classifiers both performed well, with the latter providing a more efficient training process and slightly better accuracy due to its advanced handling of continuous features and early stopping to prevent overfitting.

MLP Classifier required feature scaling to perform adequately, which is typical for neural network models. It showed competitive performance, indicating that the dataset might contain non-linear patterns that this model can capture.

Table 1: Model Evaluation Based on Classification Accuracy

Model	**Accuracy**
Logistic Regression	0.5519
Random Forest Classifier	0.8048
Gradient Boosting Classifier	0.7620
Histogram-Based Gradient Boosting Classifier	0.7906
MLP Classifier	0.7727

7.7 Conclusion

Our evaluation showed that ensemble methods, particularly the Random Forest and Histogram-Based Gradient Boosting Classifiers, provided the best performance out of the box for our dataset. These models present promising starting points for further optimisation and exploration to enhance predictive accuracy.

8 Optimisation Process

For each model, we split our dataset into training and validation sets to provide an unbiased evaluation of the model fit during the hyperparameter tuning process. We then used Optuna to define a search space for each hyperparameter of interest:

- **Logistic Regression**: We varied C for regularisation strength, max_iter for the number of iterations, solver for the optimisation algorithm, and multi_class for the strategy in handling multi-class classification.

- **Random Forest Classifier**: Our search included n_estimators for the number of trees, max_depth for the maximum depth of the trees, min_samples_split, and min_samples_leaf to control the tree structure and prevent overfitting.

- **Gradient Boosting Classifier**: Parameters like n_estimators, learning_rate, max_depth, min_samples_split, and min_samples_leaf were optimised to balance model complexity and training speed.

- **MLP Classifier**: We experimented with hidden_layer_sizes to define the network architecture, activation functions, solver for optimisation, α for regularisation, learning_rate_init, and max_iter for the number of epochs.

Optuna's efficient search algorithm guided us towards the best combination of parameters by evaluating the model's accuracy on the validation set.

9 Results

After an extensive series of trials, the optimised models outperformed the baseline models with the following accuracy scores:

Table 2: Model Performance: Baseline vs Optimised

Model	**Baseline Accuracy**	**Optimised Accuracy**
Logistic Regression	0.5519	0.7358
Random Forest	0.8048	0.8081
Gradient Boosting	0.7620	0.8116
Histogram-Based Gradient Boosting	0.7906	0.7912
MLP Classifier	0.7727	0.7749

10 Analysis

The optimisation process led to tangible improvements in model accuracy across the board. Notably, the Gradient Boosting Classifier and the Random Forest Classifier saw significant enhancements in performance. While the improvements for the MLP Classifier and the Histogram-Based Gradient Boosting Classifier were more modest, they still demonstrated the value of meticulous hyperparameter tuning.

10.1 Model-Specific Hyperparameter Tuning

10.1.1 Logistic Regression

For the Logistic Regression model, the best trial was achieved with the following parameters:

- $C = 4.09002$: The inverse of regularization strength; smaller values specify stronger regularization.
- max_iter = 381: The maximum number of iterations taken for the solvers to converge.
- solver = 'newton-cg': The algorithm used for optimization. 'newton-cg' refers to Newton's method for optimization.
- multi_class = 'ovr': The strategy used for multi-class classification; 'ovr' stands for one-vs-rest.

These parameters yielded a validation accuracy of 0.7416, a significant improvement from the baseline.

10.1.2 Random Forest Classifier

The optimised Random Forest Classifier used the following parameters:

- n_estimators = 247: The number of trees in the forest.
- max_depth = 29: The maximum depth of the trees.
- min_samples_split = 4: The minimum number of samples required to split an internal node.
- min_samples_leaf = 1: The minimum number of samples required to be at a leaf node.

This configuration led to a slight increase in performance.

10.1.3 Gradient Boosting Classifier

In Gradient Boosting, the following hyperparameters were found to be optimal:

- n_estimators = 252: The number of boosting stages to be run, which directly impacts the model's complexity.
- learning_rate = 0.1028: Shrinks the contribution of each tree, preventing overfitting.
- max_depth = 10: Limits the number of nodes in the trees.

- min_samples_split = 6: The minimum number of samples required to consider a split.
- min_samples_leaf = 1: The minimum samples per leaf node, ensuring sufficient data points per model decision.

These settings led to the largest accuracy increase among the models tuned.

10.1.4 MLP Classifier

The MLP Classifier was optimised with an architecture featuring two layers of 50 neurons each, along with these hyperparameters:

- hidden_layer_sizes = $(50, 50)$: The layer sizes influence the model's capacity to learn from complex datasets.
- activation = 'relu': The rectified linear unit function, used for activation between layers.
- solver = 'adam': An algorithm for first-order gradient-based optimization of stochastic objective functions.
- $\alpha = 0.0182371967$: Regularization parameter to prevent overfitting.
- learning_rate_init = 0.0004069832: The initial learning rate for the 'adam' solver.
- max_iter = 246: Maximum number of epochs.

This configuration led to a marginal increase in model accuracy.

10.2 Learning Curve Analysis for Random Forest Classifier

An essential aspect of our model evaluation is understanding how the Random Forest Classifier's performance scales with the amount of training data. This is illustrated by the learning curve included in this section. A learning curve plots the model's training and cross-validation scores over varying numbers of training examples, providing insight into how well the model generalises.

10.2.1 Observations from the Learning Curve

- **Cross-Validation Score**: The cross-validation score, although lower than the training score, improves as the number of training examples increases. The gradual convergence of the training and cross-validation scores suggests that adding more data is likely to continue to enhance the model's generalisation performance.

- **Need for More Data**: The fact that the cross-validation score is still rising at the end of the curve hints that collecting more data could potentially lead to better model performance, even if the gradient is getting smaller.

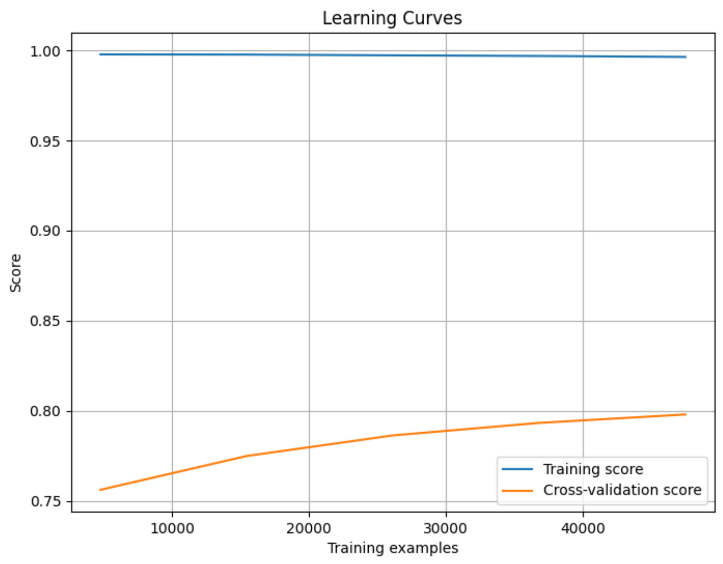


Figure 1: Learning curve of the Random Forest Classifier.

11 Feature Importance Analysis

A crucial aspect of our predictive model is understanding which features most significantly influence predictions. The following bar chart illustrates the relative importance of the various features as determined by the Random Forest Classifier.

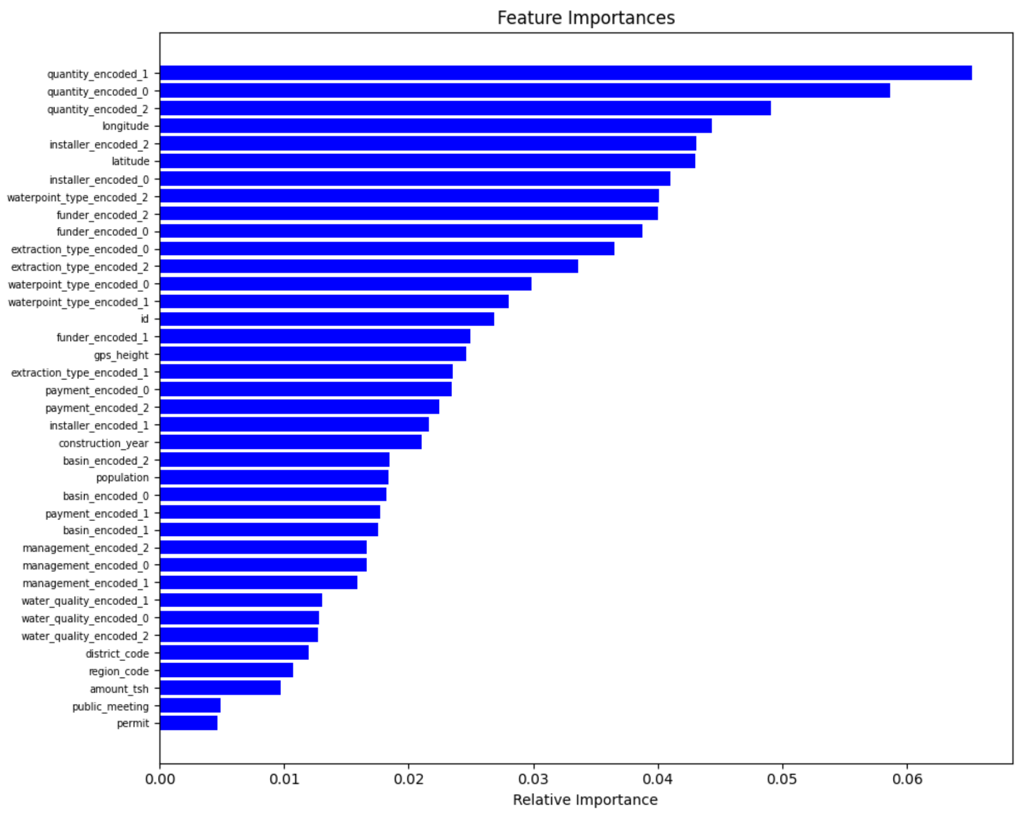


Figure 2: Bar chart of feature importances in the Random Forest model.

The chart ranks the features by their relative importance scores, providing a clear visual representation of their contribution to the model's predictive power. Notably, the features `quantity_encoded_1`, `quantity_encoded_2`, and `longitude` appear to be the most influential, with `quantity_encoded_1` showing the highest relative importance.
Less influential features, such as `public_meeting` and `permit`, contribute less to model predictions, which may suggest that they have a minimal impact on the outcome or that the information they provide is captured by other features.
Understanding these importances helps in refining the model further, allowing us to focus on the most informative features and potentially

reduce the model's complexity by eliminating less critical ones. Such an approach can improve model interpretability and reduce overfitting.
Key Takeaways from Feature Importance:

- High-impact features provide valuable signals to the model and are essential for making accurate predictions.
- Lower-ranked features might be candidates for removal during model simplification and optimization efforts.
- The domain knowledge must be applied to interpret the importance scores correctly, ensuring that any feature reduction does not omit critical information.

12 Concluding Thoughts

Through this optimisation process, we've established that hyperparameter tuning is a vital step in the machine learning pipeline. While we only ran 100 training versions of the optimisation models due to time constraints, there was still improvements across all models. It can significantly impact model performance, especially in complex datasets where the default parameters may not be sufficient.
Each model underwent retraining on the full training dataset using these optimised parameters. These refined models served as the foundation for making predictions on the test dataset, which we subsequently submitted for evaluation.
The insights gained and the performance improvements achieved highlight the importance of a tailored approach to model development. The optimised models now stand as benchmarks for further analysis and experimentation. As we continue our work, we will build upon these foundations, seeking continual improvements and refinements to enhance our predictive accuracy.

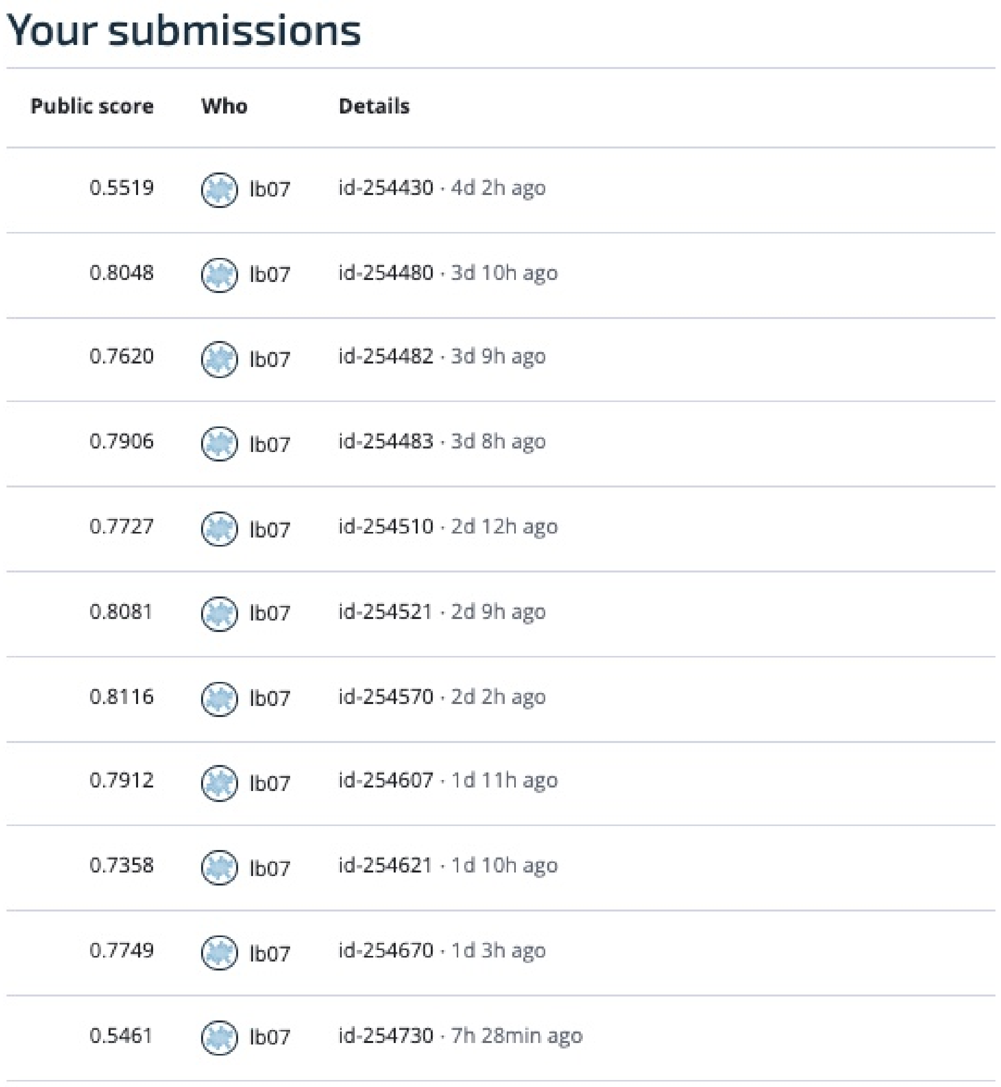
13 References

1. DrivenData. (2021). Pump it Up: Data Mining the Water Table - Competition. Available at: `https://www.drivendata.org/competitions/7/`

`pump-it-up-data-mining-the-water-table/page/25/` [Accessed 13 Mar. 2024].

2. Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, 3(1), pp.27-32.

14 Appendix



Your submissions

Public score	Who	Details
0.5519	lb07	id-254430 · 4d 2h ago
0.8048	lb07	id-254480 · 3d 10h ago
0.7620	lb07	id-254482 · 3d 9h ago
0.7906	lb07	id-254483 · 3d 8h ago
0.7727	lb07	id-254510 · 2d 12h ago
0.8081	lb07	id-254521 · 2d 9h ago
0.8116	lb07	id-254570 · 2d 2h ago
0.7912	lb07	id-254607 · 1d 11h ago
0.7358	lb07	id-254621 · 1d 10h ago
0.7749	lb07	id-254670 · 1d 3h ago
0.5461	lb07	id-254730 · 7h 28min ago

Figure 3: Submissions to the Competition