University of St Andrews

School of Computer Science
North Haugh, St Andrews

CS5011

Practical 1 - Machine Learning

Data analysis of 'Pump It Up' dataset

Student ID: [redacted]

February 17, 2025

Contents

1 Part 1

This sections aims to analyse the dataset shortly to construct a preprocessing pipeline, and to run and evaluate different combinations of scaling, encoding and models based on the cross-validation accuracy score. Tab.1 shows an overview of selected features with their missing and unique values.

Feature	Missing Values	Percentage	Unique Values	Top-1 Coverage (%)
scheme_name	28810	48.502	2695	2.229
installer	3655	6.153	2145	31.217
permit	3056	5.145	2	68.955
subvillage	371	0.625	19287	0.861
basin	0	0.000	9	17.253
region	0	0.000	21	8.912
recorded_by	0	0.000	1	100.000

Table 1: Dataset statistics

1.1 Dataset and Preprocessing

Tab.1 shows an overview of the cardinalities of selected features, along with the percentage occupied by the most frequent value. The latter gives a hint of the distribution of the categories in each column and the weight that the most frequent values have on the data. The column *recorded_by* only features a single value, since this column also has no missing values. To filter out such columns, *VarianceThreshold* will be used in our preprocessing pipeline. The dataset was also analysed to find similar features, and for identical columns like *quantity* and *quantity_group*, one feature was dropped. All other features were kept since it is not given that dropping features results in a better performance.

For categorical values, the choice was made to replace missing values with the impute strategy of substituting a constant value `"missing"` to preserve the information the missing values could have. Another feature to highlight would be *scheme_name*, which has 28810 missing values, which is nearly half of all entries. The column also has very high cardinality with 2695 categories, but the choice was made to keep the feature because of a strong correlation to the targets.

To extract information from the *date_recorded* column, the values were converted to the difference to the oldest date. In this way, we have numerical values that provide additional information to our data. Another column to highlight is *construction_year*, which has 20709 zeros in it. This greatly influences the mean, as shown in Tab.2. *Longitude* and *latitude* were combined to form a single column that specifies the distance to the geographical centre of all water points, as a correlation between these distances and the targets was found.

Standardisation to a mean of zero is helpful in equalising the weight among all features, as well as is needed for some models, such as linear regression and neural networks, to work more efficiently [1]. For example, if we compare the columns in Tab.2, we see very different maximal values and therefore a necessity to standardise.

Statistic	amount_tsh	gps_height	population	construction_year
mean	317.650	668.297	179.910	1300.652
std	2997.575	693.116	471.482	951.621
min	0.000	-90.000	0.000	0.000
max	350000.000	2770.000	30500.000	2013.000

Table 2: Summary statistics for selected numerical features.

Using the StandardScaler (StSc) helps preparing the dataset for models expecting normally distributed input data.

1.2 Pipeline

One-Hot Encoder (OHE): Encodes categories to a matrix, results in high dimensionality, and should be used when categorical variables do not have an ordinal relationship and the number of unique categories is relatively small. Tab.1 shows that there are a number of features with low cardinality, where OHE could be impactful. Regarding training, `max_categories=50` was chosen to ensure a reasonable size of the output matrix after encoding. Furthermore, since we deal with very high cardinality columns, `handle_unknown=ignore` was selected.

Ordinal Encoder (OE): Convert each value to an integer from 0 to #categories − 1, but needs natural order among categories. For our dataset, the feature *water_quality* would be a case where it could be ordered based on health implications. The only modification here was to set `handle_unknown=use_encoded_value`, to deal with unseen categories during the training and validation process.

Target Encoder (TE): Converts values to probability estimates of the target attribute. It should be used when categorical variables have high cardinality (customer IDs, product IDs, ZIP codes). Tab.1 shows that the data set has numerous features in which the target encoding is applicable. For this encoder, the default values are kept.

For training and validating the model, *StratifiedKFold* is used to maintain a proportional representation of classes in every fold. `shuffle=True` was chosen to avoid any ordering bias that might be present in the dataset.

For the models, `max_iter = 500` was set for LogisticRegression (LR) to avoid excessive runtimes, but still allow more than the specified default. The RandomForestClassifier (RFC) was kept in default settings, as well as the GradientBoostingClassifier (GBC) and the HistGradientBoostingClassifier (HGBC). For the MLPClassifier (MLPC), `max_iter=500` was choosen to allow more iterations than the default, but to avoid excessive runtimes and overfitting, `early_stopping=True` was chosen.

1.3 Results analysis

Tab.3 shows the train and test results of all possible combinations using 5-fold cross-validation for each model averaged over all preprocessing steps. Additionally, the variance of the individual training and validation results is shown, as well as the training time, since it provides additional information.

Model	Train Avg.	Test Avg.	Var (10^{-3})	Time
GradientBoostingClassifier	0.797	0.773	0.007	303.383
HistGradientBoostingClassifier	0.869	0.799	0.037	21.256
LogisticRegression	0.700	0.678	0.012	7.063
MLPClassifier	0.751	0.707	0.079	65.784
RandomForestClassifier	0.958	0.806	0.007	18.609

Table 3: Average scores for each model type

One thing to note are the high training scores of RFC, which looks to be overfitting to the training data. The test scores are the highest throughout all combinations as well, but have to be taken with caution since we know the model overfits. The predictions in the test set might be much worse. Furthermore, MLPC has a training and validation variance that is twice as high as HGBC and more than five times higher than the other models. GBC and HGBC show the most promising results, and the former shows the best generalisation performance, where LR and MLPC performed the worst overall.

Next, Tab.4 shows the average results of the encoders used, over all combinations. TE performed the best, beating OHE, which might be due to the high cardinality of the dataset, where OHE was capped by setting an upper limit on the categories, risking to loose information. Looking at the training time, TE provides the best trade-off in terms of efficiency, which aligns with findings by Pargent *et al.* [2]. The OE performed the worst, as the data set did not have suitable columns, where a natural order can be established and it can be assumed that the ordinal assignment of values leads to the models falsely recognising relations between features.

Encoder	**Train Avg.**	**Test Avg.**	**Var** (10^{-3})	**Time**
OneHotEncoder	0.832	0.774	0.026	147.792
OrdinalEncoder	0.770	0.699	0.038	31.828
TargetEncoder	0.844	0.784	0.021	70.037

Table 4: Average scores for each encoder type

Regarding scaling, we do see a better performance using the StSc, which is influenced by LR and MLPC performing 3% and 4% better respectively over all encoders. The tree-based models are not influenced by scaling the numeric values when looking at the scores.

Scaler	**Train Avg.**	**Test Avg.**	**Var** (10^{-3})	**Time**
None	0.810	0.748	0.032	91.153
StandardScaler	0.820	0.757	0.024	75.285

Table 5: Average scores for each scaler type

For more detailed analysis, Tab.6 shows the result of every combination in detail and will be used in the following analysis of each model.

LogisticRegression: Rows 15, 20, 25 are interesting, since LR should be used with scaled values, and the results undermine this. It is notable that OE performed about 30% worse than OHE and TE. The latter performed the best with a validation score of 0.767. This could be because of the `max_categories` parameter value set in the OHE. The low score with OE could be because the LR treats the ordinal values as meaningful differences, leading to learning incorrect relations. Furthermore, LR did not converge within the limit of 500 iterations, whereas it converged after about 300 iterations for OHE and TE.

RandomForestClassifier: Tab.6 shows that this model is not influenced by the type of encoding used in terms of the validation score, but shows less overfitting behaviour when used with the TE, which is seen in rows 11 and 26. The validation score of this combination is within the overall model average of 0.806, but the training score is closer to the test score. The improvement in generalisation performance could be partially correlated to the fact, that the TE is a good option for features with high cardinality, and does not introduce wrong relations like the OE [2].

GradientBoostingClassifier: GBC looks robust against overfitting, as validation scores are only 3% lower than training scores on average. The rows of interest are 2, 7, 12, since GBC does not need numeric scaling beforehand. Using the TE for categorical preprocessing resulted in a 3.2% increase to a validation score of 0.789. The logs show that the model always reached 200 iterations, which was set using `n_estimators` and thus, a better score might be achieved with a higher limit.

	Num.	Cat.	Model	Train Avg.	Test Avg.	Var (10^{-3})	Time
0	None	OHE	LR	0.696	0.696	0.009	13.438
1	None	OHE	RFC	0.999	0.805	0.006	20.669
2	None	OHE	GBC	0.777	0.769	0.006	604.653
3	None	OHE	HGBC	0.861	0.804	0.037	53.196
4	None	OHE	MLPC	0.757	0.750	0.061	163.992
5	None	OE	LR	0.567	0.564	0.003	3.927
6	None	OE	RFC	1.000	0.808	0.003	11.003
7	None	OE	GBC	0.774	0.760	0.008	92.684
8	None	OE	HGBC	0.885	0.790	0.053	12.510
9	None	OE	MLPC	0.634	0.584	0.163	31.483
10	None	TE	LR	0.806	0.741	0.018	6.067
11	None	TE	RFC	0.875	0.805	0.011	20.467
12	None	TE	GBC	0.840	0.789	0.007	253.131
13	None	TE	HGBC	0.862	0.802	0.021	14.278
14	None	TE	MLPC	0.815	0.756	0.081	65.796
15	StSc	OHE	LR	0.759	0.754	0.003	10.399
16	StSc	OHE	RFC	0.999	0.804	0.006	27.556
17	StSc	OHE	GBC	0.777	0.769	0.006	521.662
18	StSc	OHE	HGBC	0.861	0.804	0.037	20.117
19	StSc	OHE	MLPC	0.835	0.783	0.090	42.240
20	StSc	OE	LR	0.547	0.546	0.029	4.914
21	StSc	OE	RFC	1.000	0.808	0.006	11.890
22	StSc	OE	GBC	0.774	0.760	0.008	99.965
23	StSc	OE	HGBC	0.885	0.790	0.053	16.235
24	StSc	OE	MLPC	0.629	0.586	0.060	33.670
25	StSc	TE	LR	0.828	0.767	0.010	3.632
26	StSc	TE	RFC	0.875	0.806	0.012	20.070
27	StSc	TE	GBC	0.840	0.789	0.011	248.201
28	StSc	TE	HGBC	0.860	0.802	0.018	11.204
29	StSc	TE	MLPC	0.838	0.786	0.018	57.521

Table 6: Table with all possible combinations. The Var column shows the average of training and validation variances normalized to 10^{-3}.

HistGradientBoostingClassifier: This model showed a 8% decrease in validation accuracy compared to training accuracy, and like the other tree-based models, rows without scaling can be inspected. In contrast to GBC, very similar results could be observed using OHE and TE, with a score of 0.804 with the former. The training time for this model is approximately 14 times faster than GBC, which is consistent with the fact that the HGBC is a more efficient implementation for use on large datasets [3].

MLPClassifier: As mentioned before, the MLPC has a higher variance than the other classifiers used. This is influenced by the fact that the MLPC was also tested without scaling the values, which resulted in an increase in variance and training time of 100%. This also affected the training and validation score, where the MLPC achieved an accuracy of 0.78 with the OHE and the TE, which is a 10% increase compared to the average in Tab.4. What also stood out is that the OE did not work well for the neural model, which is a similar effect seen when analysing the LR, also producing a score that is around 30% lower than the other encoders. The reason could be that the MLPC treats the ordinal assignment as distances in the feature space, causing the model to learn arbitrary relations between categories and learning wrong patterns.

2 Part 2

To conduct a hyper-parameter optimisation, a pipeline was created that includes switching between all combinations listed in Tab.6. The choice was made to include most of the available hyper-parameters for each encoder and model to achieve a larger search space, because Optuna handles large search spaces efficiently due to dynamic adjustment and the default Tree-structured Parzen Estimator (TPE)-Sampler is designed for handling high dimensional search spaces [4]. The study history in Fig.1 shows that Optuna's efficient pruning strategy and use of the TPE likely contributed to a fast convergence to finding good combinations. Fig.1 also highlights the best result achieved using a GBC with the pipeline shown in the plot. The HGBC performed slightly worse, but these two models were the only models that consistently achieved scores above 0.8. All trial scores between 20 and 40 were achieved using the HGBC, then Optuna switched to try the GBC, as well as the other models. Trials 75 to 83 also show a very high average, achieved with the GBC again. Comparing this result aligns with the fact that in Part 1, the idea to increase the limit of estimators for GBC to improve the score seemed reasonable, and the optimisation showed that the highest scores were achieved with around 800 estimators. The good performance of gradient boosting algorithms aligns with findings in the literature, where gradient boosting methods often outperform other types of machine learning models in terms of accuracy [5].

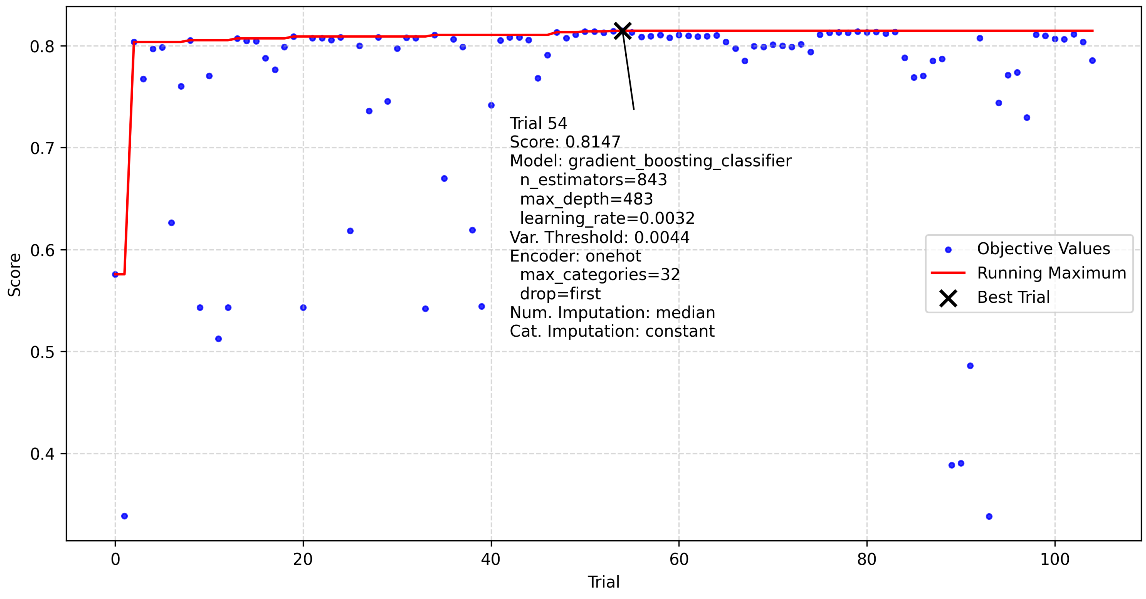

Figure 1: History Graph of Optuna Study Scores

Fig.2 shows the most influential hyper-parameters during the study. It comes as no surprise, that the type of model, as well as encoding, is important, but it is interesting to note that the variance threshold had such a big influence on the score. The reason behind this could be that features with minimal variance are less likely to contribute information and can add noise to the model. Setting the variance threshold thus can lead to a reduced dimensionality, leading to the model learning more robust patterns and better generalisation. Additionally, the fact that the data set has redundant features, that we might have not removed in preprocessing allows for improvement through feature selection.

Another interesting result is that the numerical imputing strategy turned out to be more impactful than the encoder type. The typical reason behind this is that numerical features often have a continuous and direct relationship with the target variables and effect model learning more directly. What is interesting is, that the *SimpleImputer* is set to only regard `np.nan` as missing values instead of zeros, and Tab.1 suggests that all the missing values found using `DataFrame.isnull()` are in categorical columns.

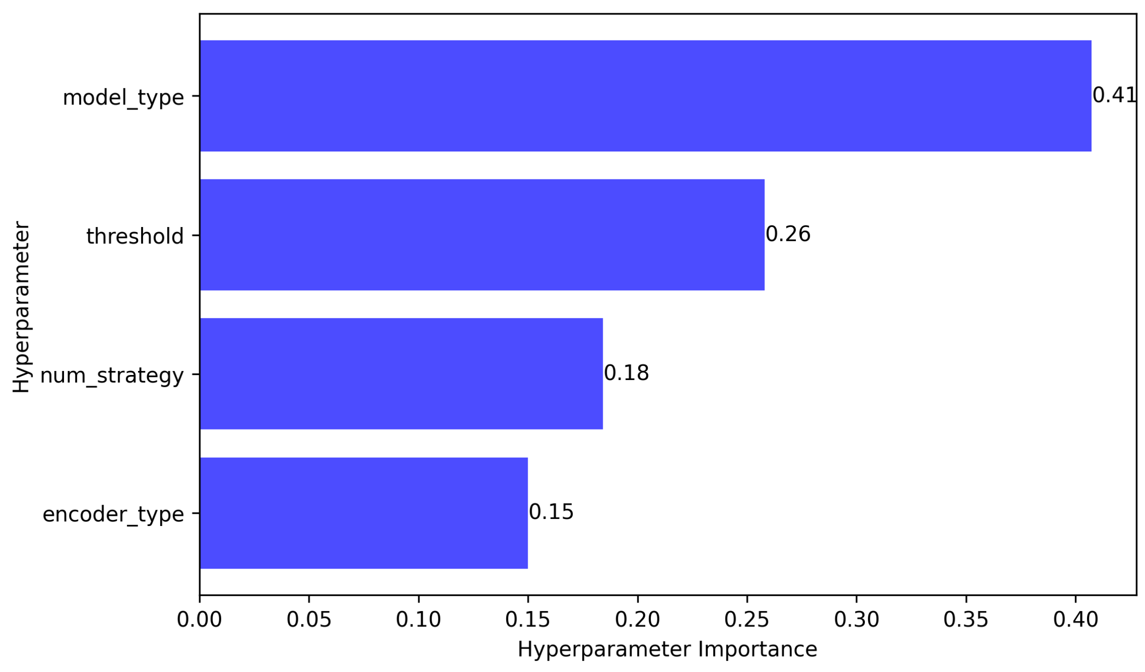

Figure 2: Hyper-parameter Importance

The above said facilitates the need to mention some limitations. The fact that a very large search space was chosen for the Optuna study might have needed a lot more trials to accurately deliver information about the effect of singular hyper-parameters. As mentioned above, the high influence of the numeric imputation is questionable and more trials would be needed to verify this result. This also applies to the variance threshold. Furthermore, is is possible that some of the chosen hyper-parameters were incorrectly bounded, hindering the optimisation process from finding the true maximum. Finally, thorough manual feature selection and adaption could also significantly influence the study results in ways that cannot be achieved with hyper-parameter optimisation.

3 Summary

In conclusion, the following key insights were gained through this study: The two experiments showed that scaling is important for the numeric models LogisticRegression and MLPClassifier, but does not influence the performance of the tree-based models. For dealing with categorical data, Target Encoder proved to be the most effective, particularly for features with high cardinality, closely followed by the One-Hot Encoder, which had a 100% increase in average training time.

Among the five classifiers tested, GradientBoostingClassifier showed the best overall performance, taking into account that RandomForestClassifier performed better but suffered from overfitting. The HistGradientBoostingClassifier had a comparable performance to the GBC, but was approximately 14 times faster in training, which makes it the preferable choice if training time is important.

Hyperparameter optimisation using Optuna showed the importance of model type and categorical encoding methods, as well as the influence of the numerical imputation strategy on the score. The variance threshold also played a crucial role, highlighting the importance of feature selection in improving model performance.

Overall, the study concluded that tree-based models like GradientBoostingClassifier and HistGradientBoostingClassifier, in combination with either One-Hot Encoder or Target Encoder are suited for this data set, depending on the computational and time constraints that influence the choice.

References

[1] Jason Brownlee, *Machine Learning Algorithms from Scratch* (Machine Learning Mastery), v1.7. 2019, 237 pp. [Online]. Available: https://deepintelligence.ir/wp-content/uploads/2024/12/Machine-Learning-Algorithms-From-Scratch-With-Python.pdf (visited on 02/06/2025).

[2] Florian Pargent, Florian Pfisterer, Janek Thomas, *et al.*, "Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features," *Computational Statistics*, vol. 37, no. 5, pp. 2671–2692, Nov. 2022, ISSN: 0943-4062, 1613-9658. DOI: 10.1007/s00180-022-01207-6. arXiv: 2104.00629[stat]. [Online]. Available: http://arxiv.org/abs/2104.00629 (visited on 02/17/2025).

[3] "Ensembles: Histogram gradient boosting," scikit-learn. (), [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html#histogram-based-gradient-boosting (visited on 02/17/2025).

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, New York, NY, USA: Association for Computing Machinery, Jul. 25, 2019, pp. 2623–2631, ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330701. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330701 (visited on 02/17/2025).

[5] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021, ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-020-09896-5. [Online]. Available: https://link.springer.com/10.1007/s10462-020-09896-5 (visited on 02/17/2025).

Acronyms

GBC GradientBoostingClassifier. 3–7

HGBC HistGradientBoostingClassifier. 3, 5–7

LR LogisticRegression. 3–5, 7

MLPC MLPClassifier. 3–5, 7

OE Ordinal Encoder. 3–5

OHE One-Hot Encoder. 3–5, 7

RFC RandomForestClassifier. 3–5, 7

StSc StandardScaler. 3–5

TE Target Encoder. 3–5, 7

TPE Tree-structured Parzen Estimator. 6