CS5011 Practical 1: Machine Learning

Part 1

Design

The flowchart in Figure 1 details the Machine Learning pipeline design, showing the pre-processing steps taken on the raw input data before using the chosen model configurations to perform cross validation and make predictions on the provided test data.

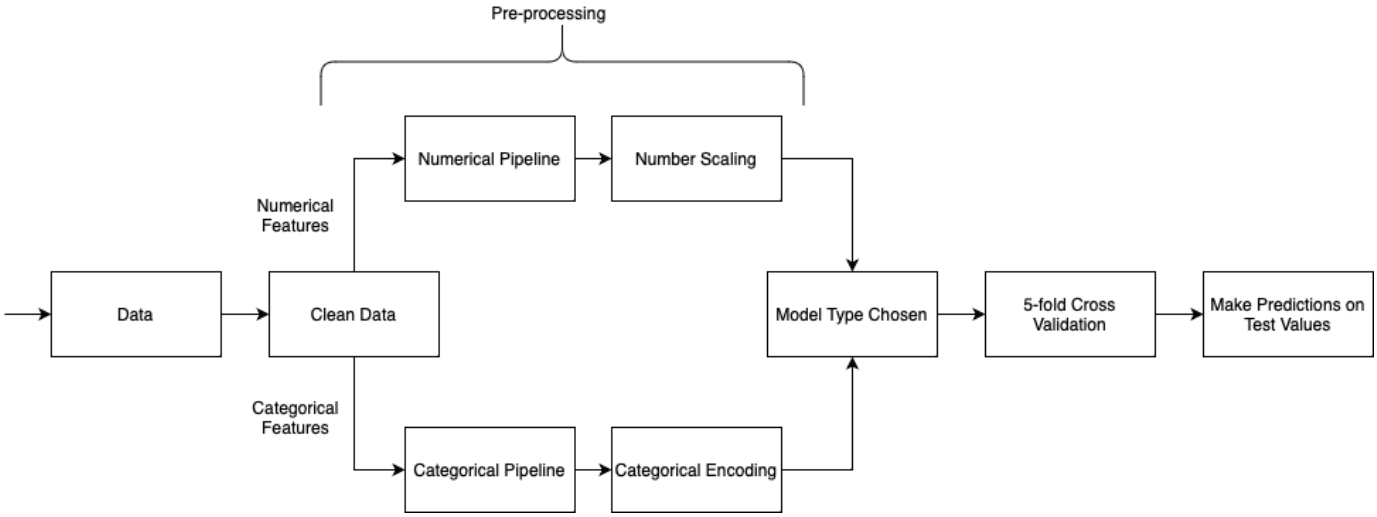

Figure 1 Machine Learning Pipeline

Data Cleaning

Initially, from analysing the raw dataset some data cleaning was required. This was due to factors such as incorrect data types as per the context of the data and unnecessary data columns being present in the data set.

The analysis of the data set highlighted columns within the data set that were not valid to include as part of the models training data such as: "*id*", "*date_recorded*", "*wpt_name*", "*recorded_by*" as these do not provide any useful information for the model to learn from. Including these might encourage the model to overfit by learning details about the data that are irrelevant, so removing these fields will improve the model's overall ability to generalise to unseen data.

Further, there were some features in the data set that were stored as an integer value but were categorical features. Not explicitly altering these data types would lead to mischaracterisation of some of the features to a numerical feature rather than a categorical feature. An example of this from the data is the "*region_code*" which is stored as an integer value, so would be treated as a numerical feature when it should be treated as a categorical feature. This is achieved by changing the type to a string.

Pre-processing: Categorical Features

To handle categorical features different categorical encoding techniques were adopted. One-Hot Encoding, Ordinal Encoding and Target Encoding. One-Hot Encoding converts each category into a binary vector representation and is best suited to categorical features that

are unordered and with relatively low unique values. Ordinal Encoding maps each category to an integer based on a given defined order and is most suitable when there is a natural order. Target Encoding replaces each category with a statically calculated value, often the mean of the target variable for the category. Useful for high-cardinality features.

High-cardinality features are present within the data set. This impacts the performance and time taken of the One-Hot Encoding technique so to deal with this problem a minimum frequency value was implemented to remove categories that are infrequent, in my implementation this value was 5% after experimenting and this was found to be a suitable choice.

To configure the Ordinal Encoding ordered categories were specified, these categories were: "*water_quality*", "*quality_group*", "*quantity*", "*quantity_group*" and the order chosen was derived trivially or when not so clear through research into common ordering conventions related to the category context.

Pre-processing: Missing Values

To handle missing values within the data set I implemented a *SimpleImputer* into the *preprocessor* of my *Pipeline* design. Imputation is the process of filling in data that is missing with substitute values so that a complete data set can be used for analysis.

For numerical values any missing values would be replaced by the mean value of the category. Utilising the mean value for this purpose means that there is no undue bias introduced into the training data. Additionally, imputation allows for no data to be discarded preserving the size of the data set even when missing values are present.

For categorical this process is slightly different, my approach involved replacing any missing categorical data with a constant value "missing". This also prevents bias being introduced by creating an extra category for missing rather than forcing the missing value into an already existing category.

Utilising the *Pipeline* functionality made implementing these imputation processes very easy and allowed for customisation in the approach taken.

Pre-processing: Numerical Feature Scaling

Numerical feature scaling standardises the scale of values within the numerical categories. This is implemented through configuring the numerical *Pipeline* to include *StandardScaler*. Utilising this mechanism often leads to improved convergence of models, additionally the scale of numerical values is consistent meaning that no single feature can dominate due to a large-scale range leading to increased numerical stability. Models such as *LogisticRegression* and *MLPClassifier* should benefit from numerical scaling as they rely on gradient-based optimisation and distance calculations.

When numerical scaling is not utilised the numerical features retain the original scales, which varies widely across different features. This could give rise to model bias by importance being misplaced on features with large magnitudes. Additionally, not scaling can lead to slow convergence or the model getting stuck in local minima. Tree-models such as *RandomForestClassifier*, *GradientBoostingClassifier* and *HistGradientBoostingClassifier* typically perform well without scaling as they are based on decision trees that are insensitive to the magnitude of the features.

Pre-processing: Datetime Features

To deal with date and time features within the data set when these are encountered they are split into three parts: day, month and year. This allows the model to use the temporal features of the date and time values more easily. This process

Result Analysis

The accuracy results in the figures below are generated use 5-fold Cross Validation. This ensures that a consistent result is achieved considering the whole data set for training and validation.

As shown from the graphs in Figure 2 it can be observed that the implementation of numerical feature scaling improves the accuracy of the *LogisticRegression* and *MLPClassifier* models whilst the other models are unchanged. As mentioned previously the models that are unimpacted are tree-based models so are insensitive to the differences in numerical scaling. Whereas the *LogisiticRegression* and *MLPClassifier* are impacted as they rely on gradient-based optimisation so the scaling should improve the performance, as is shown in the graphs for Figure 2.

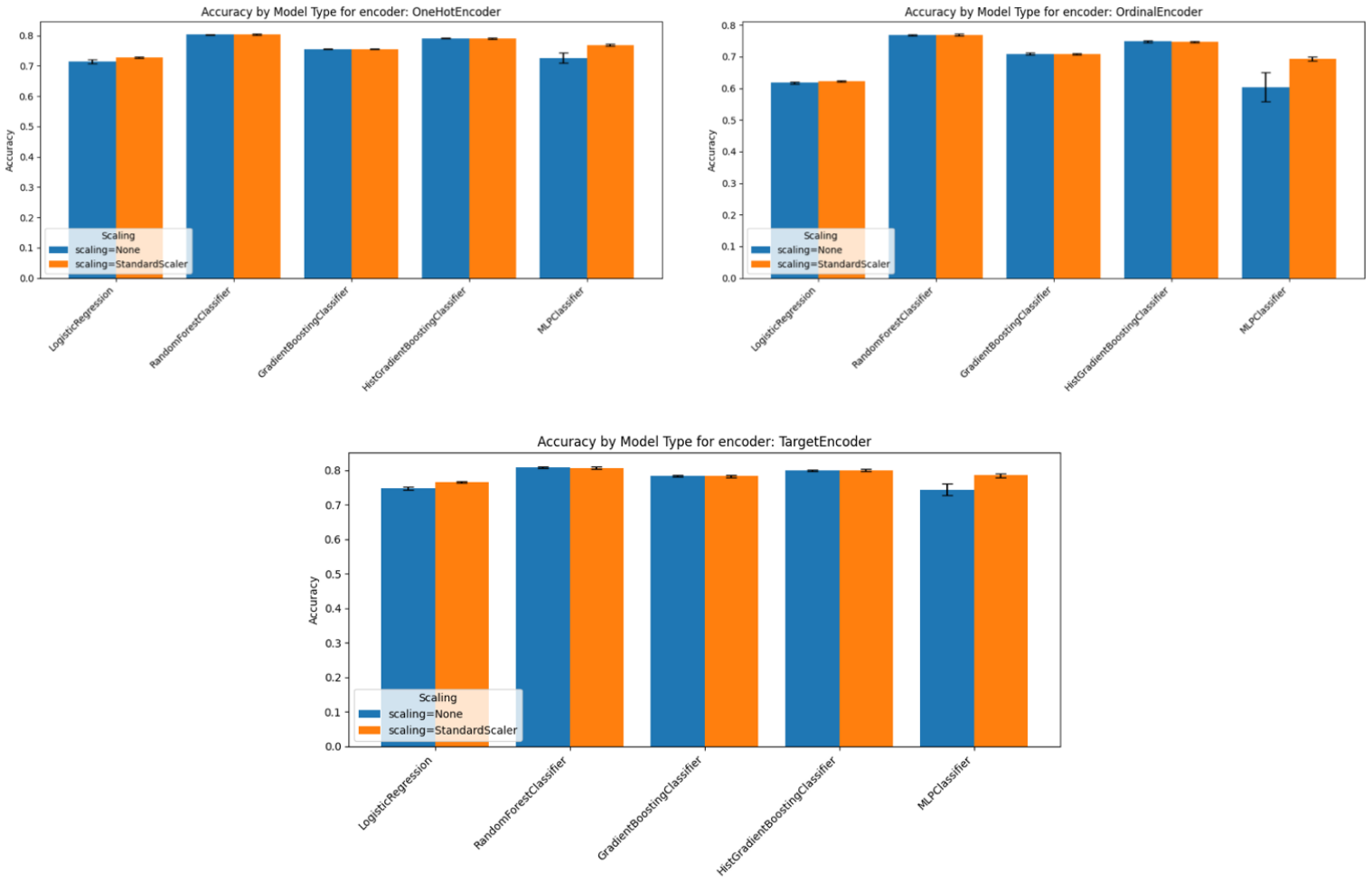

Figure 2 Accuracy per Encoder and Model Type, Standard Scaler vs None

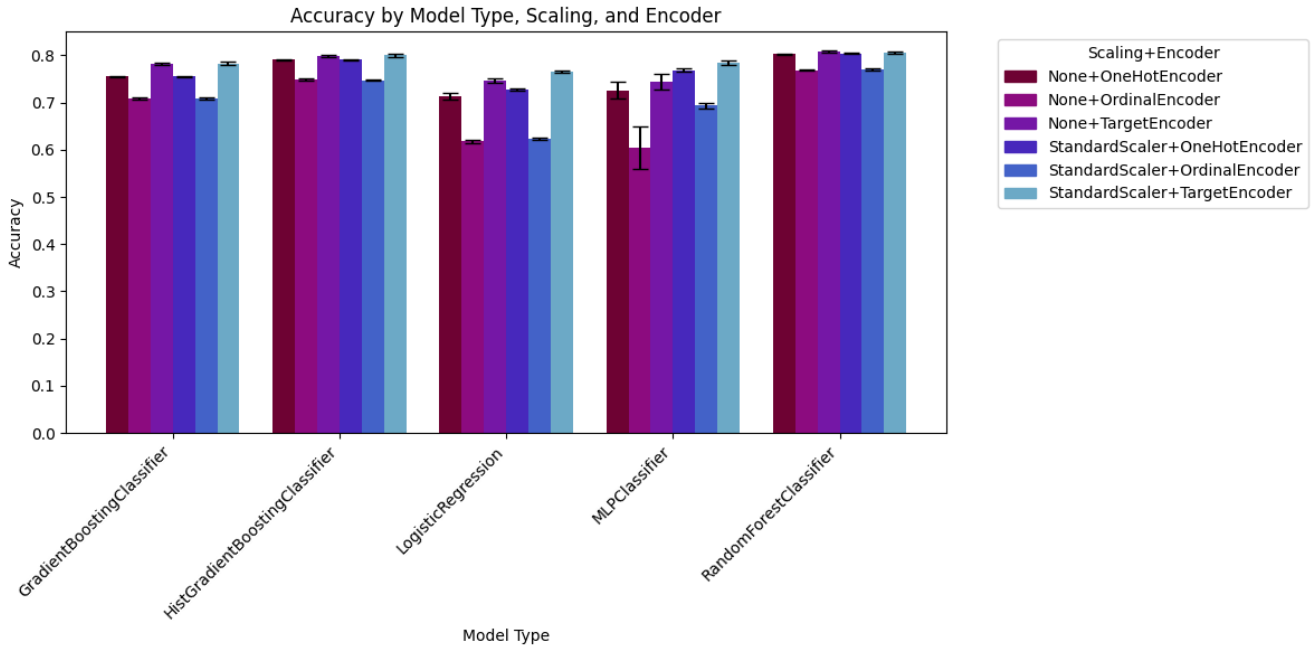

Figure 3 Accuracy by Model Type, Scaling and Encoder

From Figure 3 it can be observed the *StandardScaler* and *TargetEncoder* combination is consistently the highest performing for all the model types. A target encoding strategy is often effective when there are high-cardinality features present and given this is a present characteristic within the data this concurs with the performance values that are produced for the model configurations that utilise target encoding.

Further, it can be seen that the *HistGradientBoostingClassifier, RandomForestClassifier* and some of the *GradientBoostingClassifier* configurations performance on average the best across the different categorical encoding and numerical scaling approaches. A main reason for this is that both *HistGradientBoostingClassifier* and *RandomForestClassifier* are ensemble methods. Both combine multiple "weak learners" that reduces variance and can often achieve higher accuracy than a single model which can be seen in Figure 3. Moreover, decision trees are less sensitive to outliers and can handle missing data in a more graceful manner, especially when using histogram-based splits such as for the *HistGradientBoostingClassifier.*

When considering the performance of a machine learning model it is also important to consider the amount of time it takes to complete training. A model that is only marginally inferior in terms of performance but takes significantly less time to train may be preferred in certain contexts, so the amount of time taken to produce the trained model is an important metric to consider. From analysing the time taken for the models to complete some trends can be seen. Such as that the *OrdinalEncoder* tends to lead to the fastest training time, this is due to the nature of the *OrdinalEncoder* only considering the ordered categories in the classification. On the other hand, from the results using *OneHotEncoder* can greatly increase the amount of time taken to train this is due to the process of the OneHotEncoder and the impact high-cardinality categories have on this approach. Further, if the minimum frequency configuration setting was not implemented this downside would be magnified, and the training time would be increased further.

Part 2

Hyper-parameter Optimisation Configuration

For conducting the experiment utilising the hyper-parameter optimisation (HPO) tool the configuration space available includes all the pre-processing and model choices available previously. Additionally, for each machine learning model three hyper-parameters were chosen to be able to be optimised within a defined range as shown in Figure 4.

Model	**Hyper-parameter**	**Range / Choices**
Logistic Regression	C	0.001 – 1000
	Penalty	"l1" or "l2"
	Tol	0.00001 – 0.01
Random Forest Classifier	Estimators	50 – 300
	Max depth	2 – 20
	Min samples split	2 – 20
Gradient Boosting Classifier	Estimators	50 – 300
	Learning rate	0.01 – 1.0
	Max depth	2 – 10
Hist Gradient Boosting Classifier	Max iterations	50 – 300
	Learning rate	0.01 – 1.0
	Max depth	2 – 20
MLP Classifier	Hidden layer size	(50,) or (100,), (50,50)
	Alpha	0.00001 – 0.1
	Solver	"adam" or "sgd"

Figure 4 Model Type and Hyper-parameter Ranges

These choices were made as they cover a wide range of values allowing the HPO tool to experiment with varying ranges of settings to produce the configuration to produce the best accuracy result possible. Further, including the pre-processing and model choices from Part 1 further emphasise the range of possible configurations.

Results Analysis

In my experiment on utilising the HPO tool I conducted 50 trials containing the entire configuration space of model type, encoder, and number scaling as well as the hyper-parameter choices. From these trials the best accuracy result was produced when using the configuration shown in Figure 5.

Model Type	Gradient Boosting Classifier
Encoder	Target Encoder
Number Scaling	None
Estimators	270
Learning rate	0.1000123462705364
Max depth	10
Accuracy	~ 0.81

Figure 5 Best Performing Configuration

Without utilising the HPO tool the same configuration of model type, encoder and number scaling achieved an accuracy value of ~0.78. A ~0.03 increase can be observed when utilising the HPO tool. This is related to the optimisation of the hyper-parameters for the Gradient Boosting Classifier: Estimators, Learning rate and Max depth.

A learning rate of about 0.1 is a common and effective choice in boosting frameworks. It suggests that each tree contributes a moderate amount of correction, while the relatively high number of estimators at 270 indicates that a deep ensemble is necessary to capture the complex patterns in the data.

A max depth of 10 means that each individual tree is allowed to become quite complex. This depth allows the model to learn intricate feature interactions, which is critical when the underlying relationships in the data are non-linear.

The fact that *TargetEncoder* was chosen implies that the categorical variables have a strong and direct association with the target variable. Target encoding, which replaces categorical levels with the corresponding target statistics, often outperforms more generic approaches such as *OneHotEncoder* and *OrdinalEncoder* when the relationship between the category and the target is strong.

Conclusion

In conclusion I have made some key findings regarding the performance of the different machine learning models and pre-processing approaches. Regarding pre-processing steps using number scaling does not impact the tree-based models and improves the accuracy of the gradient-based optimisation models.

Further, it could be observed that the *OridinalEncoder* approach consistently produces the worst accuracy across all models. This mechanism is sensitive to the ordering of the categories meaning that any incorrect ordering of a feature can have a drastic impact on the results. Moreover, a limited number of ordered categories could lead to not enough detail being able to be learnt by the model or more importance being placed on a feature that is contextually valid. On the other hand, it could be observed that the *TargetEncoder* on average performed the best of all the encoders across the models which was due to the high-cardinality features present in the data set.

From utilising the hyper-parameter optimisation tool, it can be observed that the configuration shown in Figure 5 produced the highest accuracy value in my experiment, this utilised the *GradientBoostingClassifier* and *TargetEncoder* which concurs with the findings found from the analysis of Part 1 as these were determined to be amongst the highest performing configurations.

The best performing configuration optimised the hyper-parameters to maximise the *max depth* in the allowed range which allows complex feature interactions. Further, the *learning rate* was optimised to 0.1 which has been found to be a very commonly used *learning rate* to produce optimal results for machine learning models. Moreover, the higher bound of the *estimators* that was used to produce the optimal result in my experiment highlighted that a deep ensemble helped to capture the complex patters in the data set.