

Regression with an Abalone Dataset

AI6102 Final Project

Aldi Hilman Ramadhani
G2304133A
aldi001@e.ntu.edu.sg

Puteri Khatya Fahira
G2304095C
pu0001ra@e.ntu.edu.sg

Reinelle Jan Bugnot
G2304329L
bugn0001@e.ntu.edu.sg

Abstract—In this project, we participated in a Kaggle competition, *Regression with an Abalone Dataset*, which aims to predict the number of rings of an abalone from physical measurements. Our approach explored various feature engineering techniques, including PCA, Polynomial features, and manual feature engineering. We tested different models, including Lasso, LGBM, XGBoost, and CatBoost, and employed ensemble techniques such as Simple ensembling and Voting ensembling, for model combination. The best performance, with an RMSLE of 0.14583, was achieved using a voting ensemble of LGBM, XGBoost, and CatBoost models on manually feature-engineered data. A key factor contributing to this success was the use of Optuna, an automated hyperparameter tuning framework, which streamlined our tuning process and led to more refined model configurations. Our results demonstrate the effectiveness of careful feature engineering, model selection, and hyperparameter tuning in improving model performance in machine learning tasks.

I. INTRODUCTION

For this project, we participated in an ongoing Kaggle competition called *Regression with an Abalone Dataset* [1]. The objective of the competition is to develop a regression model that can accurately predict the number of rings of an abalone based on several given physical measurements. This competition lasts for a month, which commenced on April 1, 2024 and ends on May 1, 2024. While the machine learning task itself is relatively simple to execute, developing a high-performing model that could outperform the work of other competitors requires careful data handling and model design, as well as extensive parameter tuning.

II. COMPETITION DETAILS

A. Dataset

The provided dataset for this competition is a generated tabular dataset which retrieved from a deep learning model trained on the original Abalone dataset [2]. There are 90,615 labelled rows provided to train our model, and 60,411 unlabelled rows for testing. The dataset consists of 9 columns, including 1 id column, 1 categorical / nominal feature column for the abalones' sex, and 6 continuous feature columns describing several measurements for each abalone. The target itself is discrete values called 'Rings', indicating the age of the abalone, hence the regression problem.

To further enrich our training data, we also use the original Abalone dataset to build our model, which consists of 4,177 rows of data with identical features with the given training

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	F	0.550	0.430	0.150	0.7715	0.3285	0.1465	0.2400
1	F	0.630	0.490	0.145	1.1300	0.4580	0.2765	0.3200
2	I	0.160	0.110	0.025	0.0210	0.0055	0.0030	0.0050
3	M	0.595	0.475	0.150	0.9145	0.3755	0.2055	0.2500
4	I	0.555	0.425	0.130	0.7820	0.3695	0.1600	0.1975

Fig. 1. Dataset Sample

dataset. The schema for both original and generated dataset is similar, hence we don't need to conduct any preprocessing to combine both datasets. In total, we have 94,792 rows of data, using 75,834 (80%) for training, and reserving the other 18,958 (20%) rows to validate the model via 5-fold cross-validation.

B. Evaluation Criteria

The main evaluation metrics in the competition is Root Mean Squared Logarithmic Error, which is calculated as:

$$\text{RMSLE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2} \quad (1)$$

RMSLE error is claimed to be more robust and stable than RMSE [3]. RMSE is more likely to encounter magnitude exploding when the prediction contains outlier compared to RMSLE.

III. EXPLORATORY DATA ANALYSIS

The first step towards any machine learning experiment is data analysis. Here, we performed simple EDA to gain better understanding of the data before conducting the experiment. After cleaning the dataset, such as handling missing values and duplicate entries (which we didn't find any), we first analyzed the data distributions between the provided competition data vs the original abalone data to see if we can use the original data to extend our training data, then we checked for outliers within the features and selected strategies to deal with them, and then we analyzed the relationships between the different physical measurements features through correlation matrices to guide our feature engineering strategies later on.

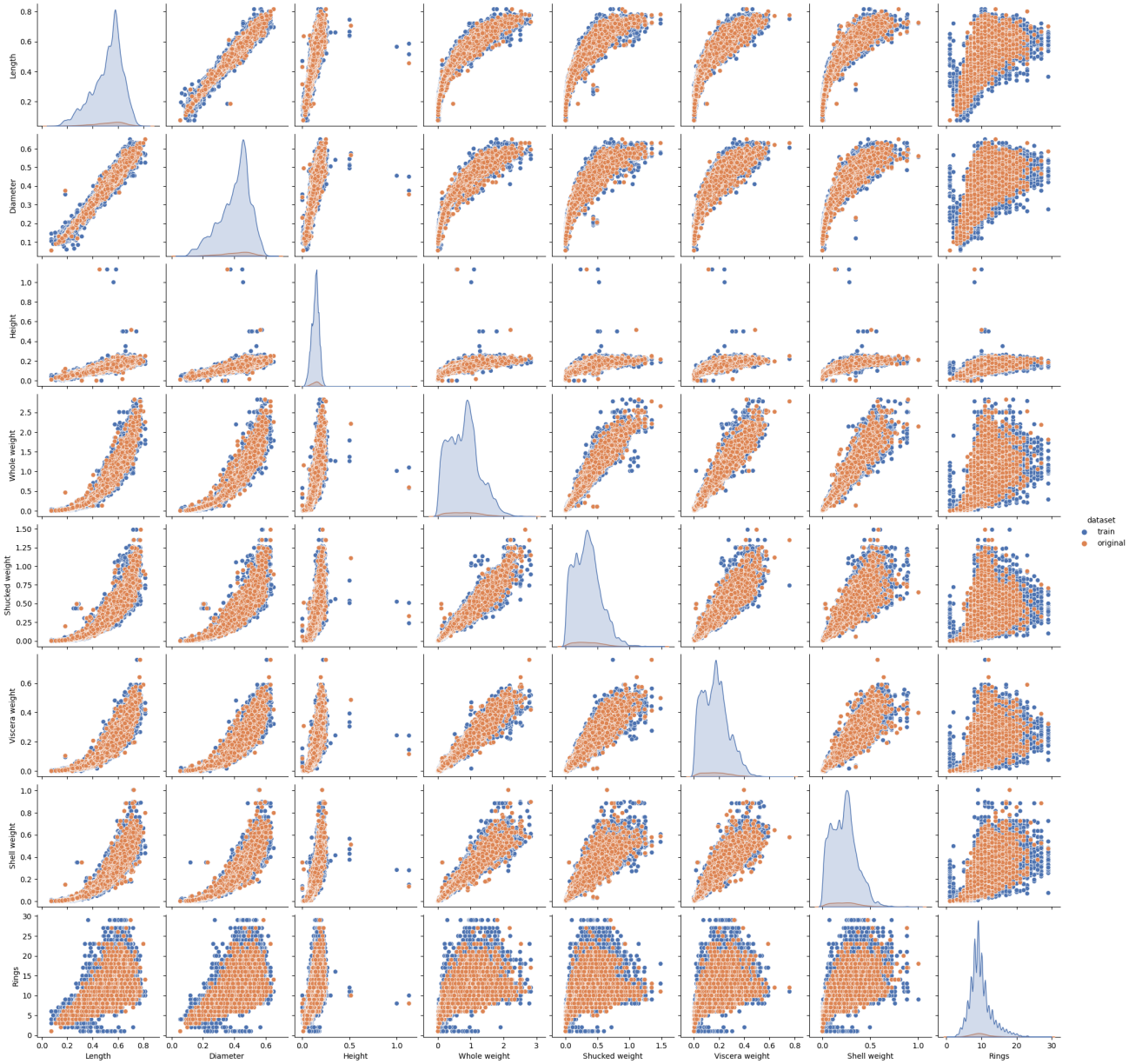


Fig. 2. Pairplots between Numerical Features of Both Dataset. Orange is Original Dataset, Blue is Competition Dataset

A. Distribution of Competition Data vs Original Data

We first compared the given training dataset with original dataset to ensure distribution match before utilizing both datasets to train our model. Figure 2 presents the pairplot across all of the features and target variable against each other. The diagonal entries in this figure shows the distribution of values between the pairs of features, while the off-diagonal entries shows the scatterplot of datapoints between pairs. We observe that the distribution of the target labels from both datasets are similar, a closer look of which is shown on Figure 3. The fluctuation on the train dataset is likely to be caused by target label Rings being discrete and the train dataset is much larger than the original dataset. Furthermore, we examined

the distribution of each features from both dataset. We find that both dataset give similar distribution for each features. We observe that the skew is relatively similar, even though the given dataset shows more fluctuation on the distribution which is expected due to the big difference of the dataset size. A few examples of Kernel Density Estimate (KDE) plot of some selected features are shown on Figure 4. Furthermore, from Figure 2, it is evident that all numerical features shows strong positive correlation to the Rings size, as shown by the linear relationship of the datapoints in the scatterplots.

We also compared the categorical data distribution between the provided training data and the original data. In this case the only categorical data present is the Sex feature. Figure 5 shows that the features are evenly distributed both for the

training dataset and the original dataset. Hence, no upsampling or downsampling methods are needed to align the training data distribution with the test data distribution.

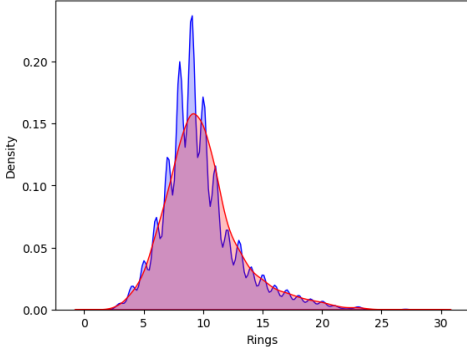


Fig. 3. Label Distribution of Competition Dataset and Original Dataset

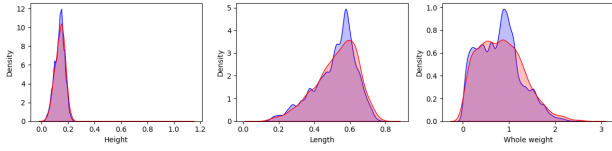


Fig. 4. Sample Feature Distribution of Competition Dataset and Original Dataset

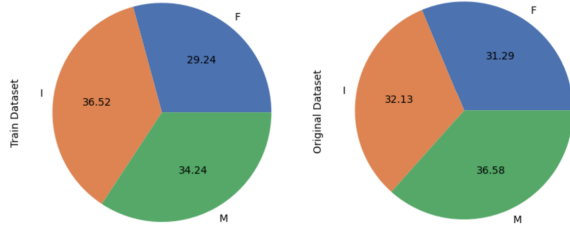


Fig. 5. Distribution of Categorical Feature (Sex) of Both Dataset

Based on these empirical observation of both dataset distribution. We decided to include the original dataset to enrich our data for building the model. Next, we perform EDA to help guide our decisions of whether to keep all entries and features of the dataset for training.

B. Outliers

We observe several outliers and in each features, calculating based on the IQR of each features. In total, we find there are significant portion of data, almost 8,457 out of 90,615 rows contains feature with outliers. We will asses whether removing or retaining these rows would benefit our model prediction against the testing data based on our empirical results discussed in the Results section. Some examples of the Boxplot of the given training dataset features are shown on Figure 6.

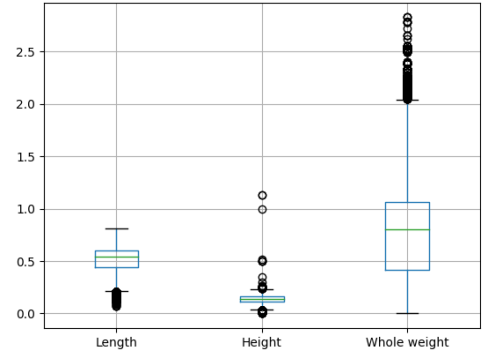


Fig. 6. Boxplot of Sample Features of Dataset

C. Feature Correlation

We evaluate the correlation of each features in the dataset, and how they correlate with the label. Observations from Figure 7 shows that weight-related features are highly correlated. Figures 8 also illustrate that every feature has positive correlation to the Rings target value, which is intuitively expected. Further investigation using Variance Inflation Factor emphasize that the features exhibit high multicollinearity. However, we can't simply reduce these correlated features as it will left us with very minimal number of features to train the model, hence, underfitting the model. Instead, we opted to performing feature engineering with the goal of reducing the cross-feature correlations. Our experiments will include empirical observation on how feature engineering of these feature will impact the training and final model performance.

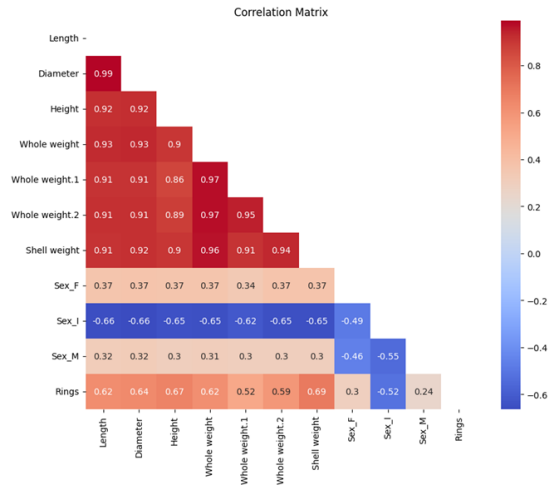


Fig. 7. Correlation Between Features

IV. DATA PRE-PROCESSING

A. Feature Engineering

1) *PCA*: Principle Component Analysis is one of the most common feature engineering techniques by finding essential patterns on the features and representing them with fewer

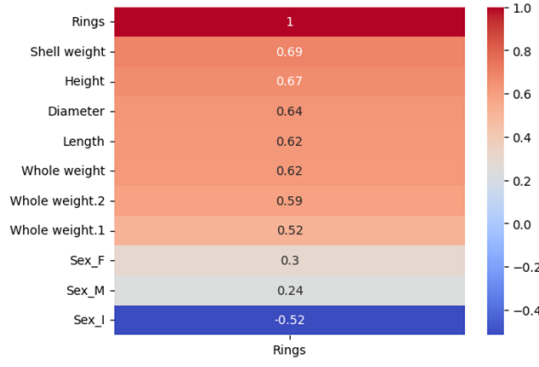


Fig. 8. Correlation Between Feature and Target Label

variables, called principal components [4]. Given the correlation in our features, we conduct multiple experiments against several number of principle components aiming to find the best number of principle components to represent the dataset effectively.

2) *Polynomial Features*: Our experiments also include enriching our dataset by implementing polynomial features. We conduct multiple experiments using different hyperparameter (degree) values of d to generate the polynomial and interaction features, expanding each of numerical features by $d * \text{times}$. One example of the new correlation matrix for the polynomial features using $d = 3$ is shown of Figure 9. We observe more portion of features especially the weight-related features are shown in lighter colors in the heatmap, meaning they are less correlated than the original data.

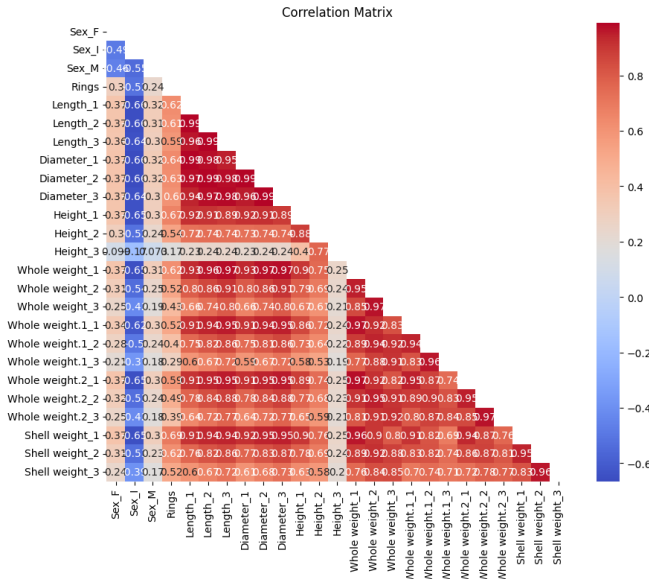


Fig. 9. Correlation Between Polynomial Features and Target Label

3) *Manual Feature Engineering*: We further implement manual feature engineering based on our prior knowledge and the existing features of the dataset. We introduce several new features, namely residual weight, square area, and density.

The residual weight is based on the difference between the whole weight and the total of other three weight features. This feature is assumed as the portion of the weight that is loss during the calculation of other weight features during the data acquisition process. The squared area is calculated from the length multiplied by diameter features. Next, we calculate density of each rows by dividing the whole weight with the volume calculated using the squared area and height features. However, we observe several rows has their height feature equal to zero, which we assumed as missing data and causing error in the new feature calculation. Hence, we introduce another feature, called area density using whole weight divided by the squared area. Finally, we implement one-hot encoding for our only categorical feature, 'Sex'. The final list of features for manual feature engineering are as follows:

Features after Manual Feature Engineering

- Sex
- Length
- Diameter
- Height
- Whole Weight
- Shucked Weight
- Viscera Weight
- Shell Weight
- **Residual Weight**
- **Square Area**
- **Area Density**

B. Scaling

We used the `StandardScaler` function from scikit-learn across all of our numerical features to mitigate the scale differences between the features that will affect the learning process. This works by removing the mean and scaling to unit variance.

$$z = \frac{x - \mu}{\sigma} \quad (2)$$

This is particularly important for machine learning algorithms that are sensitive to the scale of the input features. Fortunately, most of the models that we explored were tree-based models which are fairly insensitive to scaling, so the impact of this step might not be much apparent for experiment setups involving purely tree-based approaches.

V. HYPERPARAMETER TUNING

Hyperparameter tuning is the process of finding the optimal set of hyperparameters for a machine learning model, aiming to improve the model's predictive accuracy and generalization ability. We can perform this manually by iteratively adjusting the hyperparameters based on the model's performance on a validation set. This approach, however, can be time-consuming and may not always lead to the best results due to the high dimensionality of the hyperparameter space and the non-convex nature of the loss landscape.

Alternatively, we can use automated hyperparameter tuning methods, which systematically explore the hyperparameter space to find the optimal configuration. Common techniques available are Grid Search, where we exhaustively tries numerous combinations of user-defined hyperparameter values; and Random Search where we randomly sample the hyperparameter values to speed up the computation in contrast to Grid Search. Several frameworks have also been proposed to optimize this hyperparameter tuning process. For instance, Bayesian optimization, genetic algorithms, and gradient-based optimization are more advanced methods that aim to intelligently explore the hyperparameter space by learning from the results of previous evaluations. These methods can often find better hyperparameters more quickly than grid search or random search, but they can be more complex to implement and may require additional computational resources.

Among these widely-used recent frameworks for hyperparameter tuning is *Optuna*, which is an agnostic framework that optimize the values based on searching and pruning strategies. At it's core, it operates using Bayesian optimization with an algorithm called TPE (Tree-structured Parzen Estimator). We implemented *Optuna* to facilitate the automatic hyperparameter tuning for our models during the experiments. This framework allows a more efficient search for hyperparameter values by monitoring the intermediate evaluation result of the model and pruning the unpromising trials to speed up the exploration. Optuna will conduct multiple explorations based on the user defined range of potential hyperparameter values and evaluate the results against the defined evaluation metrics. Thus, we could conduct more hyperparameter exploration in more timely manner compared to other framework or manual tuning. Shown in the image below is an example set-up of the Optuna objective() function.

```
# Define Objective and Hyperparameter Suggestions
def objective_lgbm(trial, X, Y, y_raw):

    params = {
        'num_leaves': trial.suggest_int('num_leaves', 100, 200),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 1.0, log=True),
        'n_estimators': trial.suggest_int('n_estimators', 800, 1200),
        'subsample_for_bin': trial.suggest_int('subsample_for_bin', 1e5, 2e5),
        'min_child_samples': trial.suggest_int('min_child_samples', 10, 50),
        'reg_alpha': trial.suggest_float('reg_alpha', 1e-9, 1.0, log=True),
        'reg_lambda': trial.suggest_float('reg_lambda', 1e-9, 1.0, log=True),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.25, 1.0),
        'subsample': trial.suggest_float('subsample', 0.25, 1.0),
        'max_depth': trial.suggest_int('max_depth', 1, 25)
    }

    model = LGBMRegressor(**params,
                           n_jobs=-1,
                           metric='rmse',
                           verbosity=-1,
                           bagging_freq=1,
                           boosting_type='gbdt',
                           objective='regression',
                           random_state=1)

    rmse = rmse_cross_validation(X, Y, y_raw, model)
    return np.mean(rmse)
```

Fig. 10. Example Optuna objective function for an LGBM Regressor

The Optuna objective function contains the *trial* method that systematically selects parameter values based on a given range specified using the `.suggest_float()`,

`.suggest_int()`, and `.suggest_categorical()` functions. A narrow suggestion range will yield relatively more consistent performance, but limits the optimization capacity of the model. Hence, identifying the proper range to test out values is crucial for the successful implementation of Optuna. The key assumption we make here is that the global optimum is located within the section of the high-dimensional parameter space defined by our suggestion range selections.

```
if TUNING_HYPERPARAM:
    # Instantiate Optuna Study
    study = optuna.create_study(study_name='LGBM', direction='minimize')
    study.optimize(lambda trial: objective_lgbm(trial, X_scaled, Y, y_raw), n_trials=100)

    print(study.best_params)

    best_params_lgbm = study.best_params
    save_params(best_params_lgbm, 'best_params_lgbm')
```

Fig. 11. Performing parameter search using Optuna

After specifying the selection range, we can instantiate an Optuna study object and call the `.optimize()` function to run $n_trials=100$ trials, where a unique set of parameters within the specified range is tested for each trial. Each candidate parameter set were evaluated using 5-fold cross validation using the RMSLE metric. In the snippet of the objective function in Fig. 10, we only needed to calculate for the RMSE because we already transformed the target variable into the logarithmic space using $y = \log(1 + y)$ as part of our preprocessing steps; which essentially yields RMSLE. The *best parameters* were taken from the trial that yielded the lowest 5-fold RMSLE. Figure 12 shows the objective values generated for each of the 100 trials for tuning an LGBM model. The red line marks the best parameters recorded up until trial t . We can see from the figure how well Optuna is able to focus the parameter search within just a narrow region that previously yielded high performance.

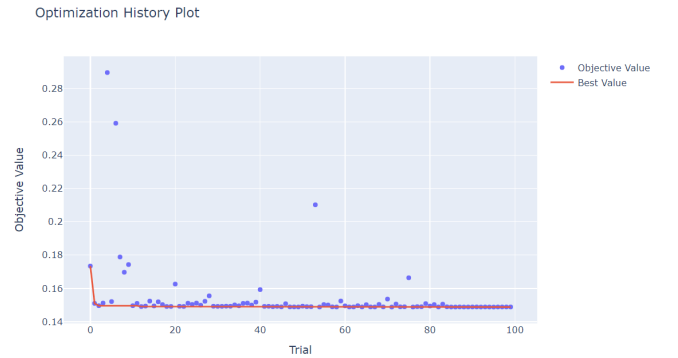


Fig. 12. Optuna Optimization History plot showing the progress of hyperparameter tuning across 100 trials for the LGBM model.

VI. MODELS

Our exploratory data analysis shows that our data is relatively simple and small in structure. Thus, we're focusing our experiment to employ more simple models as we're not confident that our data is enough to build a complex model (e.g. Deep Learning).

A. Regression Technique

1) *Lasso*: Lasso (Least Absolute Shrinkage and Selection Operator) is an adaptation of linear regression method that penalizes the absolute size of regression coefficients. It encourages simple, sparse models by setting some coefficients to exactly zero [5].

2) *LGBM*: LightGBM is a high-performance gradient boosting framework based on decision trees. It splits trees leaf-wise, leading to better accuracy and faster training. It's efficient, memory-friendly, and compatible with large datasets [6].

3) *XGB*: XGBoost (Extreme Gradient Boosting) It's a gradient-boosted decision tree model that sequentially combines many weak models (trees) to create a strong predictive model. It optimizes loss functions by minimizing errors and adding new trees iteratively [7].

4) *CatBoost*: CatBoost (Categorical Boosting) is a gradient-boosting algorithm designed for categorical data. It uses ordered boosting, random permutations, and gradient-based optimization [8].

B. Model Ensembling

We implement model ensembling where we combine the predictions of our previously mentioned models to create a more robust and accurate overall prediction.

1) *Simple*: We uniformly distributes the total weight among the ensemble members, by assigning equal weight to each regressor model.

2) *Weighted*: Weighted averaging ensembling assigns custom weights to each model's prediction. We're aiming to emphasize certain models based on their expected performance on specific subsets of the data.

VII. RESULTS

For consistency in our results, the following steps were performed constantly across all of our experiments (where applicable):

- 5-fold cross-validation
- Standard Scaling (using z-score)
- Optuna Hyperparameter tuning with 100 trials

Table I presents the consolidated results generated from various experimental settings, showing how the model performance varies significantly with different feature engineering (F.E.) strategies and model configurations. From here, we can see that the best test (or leaderboard) RMSLE of 0.14583 was achieved through the weighted/voting ensemble of 3 models; namely, LGBM, XGBoost, and CatBoost, on the manually feature-engineered data.

A key factor that lead to this outcome is the extremely refined hyperparameter tuning process that we implemented using Optuna. Normally, traditional parameter tuning would require a significant amount of manual effort and domain knowledge to identify the optimal set of hyperparameters. This process is often time-consuming and may not guarantee the best performance due to the complex interactions between different hyperparameters. For instance, the LGBMRegressor()

documentation lists 18 tunable hyperparameters. Even with advanced domain knowledge on decision tree-based models, optimally identifying which of these 18 parameters to tune and by what degree is an extremely challenging, let alone inefficient, task. However, by using an automated hyperparameter optimization framework like Optuna, we were able to systematically and efficiently explore the hyperparameter space to generate very fine parameter selections, as shown in Table II, that would not have been possible to identify heuristically. With this, we were able to allocate much of our effort towards exploring numerous different strategies and playing around with the data through feature engineering, and testing a variety of models and their combinations.

A. Model Selection

There are four models that we used for the experiment. This models could be separated into two category. One relatively simple model that is Lasso, and three more sophisticated models consisting of LGBM, XGB, and CatBoost. Another combination or ensemble model is also used, with ensemble method of simple or voting are used.

In general the performance shows that the simple model perform worse than more complex model, with the ensemble model perform the best. Lasso the relatively simplest model received the best validation loss of 0.16574. This shows a need for a more complex model to increased performance.

Meanwhile, the more complex model like LGBM, XGB, and CatBoost have a higher performance. The best single model performance for the validation loss is 0.14757 with LGBM regressor. However, the best leaderboard loss is 0.14583 with the use of voting ensemble. This showcase the combination of each part of the models are needed to be able to get a high score in the leaderboard.

B. Impact of Feature Engineering

1) *Outlier and Low Correlated Feature Removal*: In one experiment, the outlier data will be removed from the whole test dataset to check if the accuracy performance increased. The other experiment is that the removal of low correlated feature column. This low correlated feature is calculated with the respect of the target column.

The Table I suggest that the removal of low correlated feature doesn't contribute much to the performance. This conclusion suggest that the model is sufficient enough to learn which data to use. Hence, this removal won't be used in future experiments. Another thing worth noting is that the outlier removal generate a better loss value. This is a little deceiving, since the removal of the outlier only remove data in the "visible" dataset. This automatically remove the outlier when doing cross-folding, but the test dataset could potentially have outlier data that the model can't learn from. Hence, the outlier removal also won't be implemented in the next experiment.

2) *PCA component*: The next experiment of feature engineering is the use of PCA. Each of the numerical features in the dataset got transformed into its PCA component. This new PCA component now will be added as the new column

TABLE I
CONSOLIDATED EXPERIMENT RESULTS

FE.	Models	Ensembling	Validation RMSLE	Leaderboard RMSLE
None Low Correlated Removal Outlier Removal	Lasso	None	0.16650	-
	Lasso	None	0.16651	0.16454
	Lasso	None	0.16571	-
pca=2	Lasso	None	0.16576	-
pca=3	Lasso	None	0.16574	-
pca=5	Lasso	None	0.16576	0.16454
poly_feature=3 poly_feature=3 poly_feature=3 poly_feature=3 poly_feature=5 poly_feature=5 poly_feature=5 poly_feature=5 poly_feature=7 poly_feature=9	CatBoost	None	0.14902	-
	LGBM	None	0.14898	-
	XGB	None	0.14932	-
	CatBoost, LGBM, XGB	Voting	0.14878	0.1463
	CatBoost	None	0.14906	-
	LGBM	None	0.14904	-
	XGB	None	0.14936	-
	CatBoost, LGBM, XGB	Voting	0.14885	0.14632
	CatBoost, LGBM, XGB	Voting	0.14887	0.14634
	CatBoost, LGBM, XGB	Voting	0.14888	0.14634
None None None None Manual Manual Manual + Outlier Adjust Manual Manual Manual	CatBoost	None	0.14928	-
	LGBM	None	0.14881	0.14680
	XGB	None	0.14912	-
	CatBoost, LGBM, XGB	Simple	0.14788	0.14666
	CatBoost	None	0.14818	-
	LGBM	None	0.14809	0.14677
	LGBM	None	0.14757	0.14793
	XGB	None	0.14833	-
	CatBoost, LGBM, XGB	Simple	0.14865	-
	CatBoost, LGBM, XGB	Voting	0.14761	0.14583

TABLE II
BEST PARAMETERS FOR LIGHTGBM, XGBOOST, AND CATBOOST MODELS (TEST RMSLE = 0.14583)

Parameter	LightGBM	XGBoost	CatBoost
num_leaves	138	-	-
booster	gbdt	gbtree	-
learning_rate	0.01851440025520457	0.011733966748427322	0.09992185242598203
n_estimators	913	1137	-
subsample_for_bin	185680	-	-
min_child_samples	34	-	-
reg_alpha	5.916235901972299e-09	0.013043045359306716	-
reg_lambda	6.943912907338958e-08	1.7487237399420372	-
colsample_bytree	0.4339090795122026	0.5748511749872887	-
subsample	0.799314727120346	0.486382907668344	0.83862137638162
max_depth	15	10	15
grow_policy	-	lossguide	lossguide
gamma	-	0.03816426816838989	-
min_child_weight	-	7	-
tree_method	-	hist	-
max_bin	-	-	464
min_data_in_leaf	-	-	78
bootstrap_type	-	-	Bernoulli
l2_leaf_reg	-	-	8.365422739510098
random_strength	-	-	3.296124856352495

of the transformed dataset. The transformed dataset is then experimented on the same model, that is Lasso regressor, to check if the use of PCA help the performance of the regressor.

The Table I above presents the outcomes of experiments using Principal Component Analysis (PCA). The number of PCA components was varied across the experiments: two, three, five, and no components. The main thing to note from the table is that by using addition PCA column, the performance of the model increased from 0.16650 to 0.16574 or 0.16576. This indicates that the PCA column helps the model to create a better regression function.

3) *Polynomial Feature*: The next experiment about feature engineering is using the polynomial feature. In this experiments, each of the numerical feature got separated into its polynomial combinations of a certain degree. For example, if the number of polynomial degree is two and the number of numerical features is 10, then the total number of transformed numerical features is 20.

The Table I details the performance of various models to a dataset with polynomial feature transformation. The experiments are categorized by the degree of the polynomial features—3rd, 5th, 7th, and 9th degrees. Three different types

of gradient boosting models are used: CatBoost, LightGBM (LGBM), and XGBoost. Additionally a voting ensemble model is listed, which is a combination of those listed models.

One important finding in this section is that the loss for the ensemble models is always better than the single model. This shows the benefit of using the ensemble rather than predicting with a single model. Also worth to note that the increase of polynomial degree doesn't translate with lower loss value. This is proven by the lowest loss is achieved with the use of 3rd degree polynomial. The single model and the ensemble model also observed this behavior. The best single model have validation loss of 0.14898 with LGBM regressor. Meanwhile, the best ensemble model have validation loss of 0.14878 and the leaderboard loss of 0.14630. This is the second best entries we have in term of leaderboard loss.

C. Individual Models vs Ensembling

We can see from the consolidated results in Figure I that applying ensembling methods generally outperform the individual models they're comprised with. This is because ensembling methods leverage the power of multiple learning algorithms, allowing them to produce better predictive performance compared to a single model. The key principle behind ensemble methods is that a group of weak learners (LGBM: 0.14809, XGB: 0.14833, CatBoost: 0.14818) can come together to form a strong learner (Ensemble: 0.14761). Each model in the ensemble makes its own prediction and the final prediction is determined by combining these predictions via simple or weighted averaging. In our setup, the weights for the voting regressor is treated as a hyperparameter in itself, and therefore was identified using Optuna and is presented in Table III below.

TABLE III
LEARNED ENSEMBLE WEIGHTS FOR VOTING REGRESSOR

Model	Ensemble Weight
LGBM	4.811792271697856
XGB	0.11433122823276876
CatBoost	4.307780525851765

The superior performance of the ensemble model over single models demonstrates the importance of model diversity in regression tasks. Different algorithms may be good at capturing different patterns in the data, so an ensemble of these models can potentially capture more patterns and therefore make better predictions.

D. Kaggle Competition Outcome

As of the time of writing this report, the Kaggle competition is still ongoing. Figure 13 in the Appendix shows the snapshot of our leaderboard position out of approximately 1,300 participants, taken around the 3rd week of April after we finished most of our experiments. Figure 14 in the Appendix shows a screenshot of the Kaggle Submissions page, listing the top few leaderboard scores that we achieved across different experimental setups (which can also be found in Table I).

VIII. CONCLUSION

In this project, we joined the Kaggle competition, *Regression with an Abalone Dataset*, with the objective of predicting the number of rings of abalones, using various physical measurements. Several feature engineering techniques were tested, including Principal Component Analysis (PCA), polynomial feature generation, and manual feature engineering. We also implemented several regression models such as Lasso, LightGBM (LGBM), XGBoost, and CatBoost. Notably, we found that an ensemble of models, particularly a voting ensemble, consistently outperformed individual models. This ensemble approach led to the best leaderboard Root Mean Squared Logarithmic Error (RMSLE) performance of 0.14583.

A critical component of our approach was the utilization of Optuna, an automated hyperparameter tuning framework based on Bayesian Optimization. This significantly optimized the model tuning process, underscoring the fact that efficient parameter tuning can substantially improve performance. Interestingly, our experiments also revealed that outlier removal does not necessarily lead to improved model performance. This counter-intuitive finding highlights the complexity of machine learning model development and the importance of rigorous empirical evaluation.

Overall, our results showcase the effectiveness of combining different models and fine-tuning their configurations to achieve optimal predictive accuracy. While the challenge itself is easy to execute due to the simplistic nature of the dataset and task, achieving high leaderboard score is a tough challenge that tests our capabilities in maximizing the performance of a machine learning model. These findings, therefore, showcase our broader understanding of machine learning model development and performance optimization.

REFERENCES

- [1] Kaggle, "Regression with an Abalone Dataset," Kaggle, <https://www.kaggle.com/competitions/playground-series-s4e4> (accessed April 1, 2024).
- [2] W. Nash, "Abalone," UC Irvin Machine Learning Repository, 1995. <https://archive.ics.uci.edu/dataset/1/abalone> [Online].
- [3] S. Saxena, "What's the Difference Between RMSE and RMSLE?," in *Analytics Vidhya*. 2019.
- [4] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *Philosophical Magazine*, vol. 2, no. 11, pp. 559–572, 1901.
- [5] R. Tibshirani, "Regression shrinkage and selection via the lasso," in *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
- [6] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
- [7] T. Chen, C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.
- [8] A. V. Dorogush, V. Ershov, A. Gulin, "CatBoost: unbiased boosting with categorical features," arXiv preprint arXiv:1706.09516, 2017.

APPENDIX


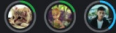


116	Dimitar Ruskov		0.14582	10	5d
117	Not from AI6102 class in NTU		0.14583	21	2h
 Your Best Entry! Your submission scored 0.14645, which is not an improvement of your previous score. Keep trying!					
118	sun9sun9		0.14583	14	4d

Fig. 13. Leaderboard position taken 3rd Week of April 2024 (out of approx. 1,300 participants)











Submission and Description	Public Score ⓘ	Select
 solution (3).csv Complete · Rein Bugnot · 15d ago · test - voting regressor	0.14583	<input checked="" type="checkbox"/>
 Abalone Regression - Optuna - Version 12 Complete · Aldi Hilman Ramadhani · 3h ago	0.14630	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 10 Complete · Aldi Hilman Ramadhani · 7d ago · Notebook Abalone Regression - Optuna Version 10	0.14634	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 17 Complete · Aldi Hilman Ramadhani · 3h ago · Notebook Abalone Regression - Optuna Version 17	0.14634	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 16 Complete · Aldi Hilman Ramadhani · 3h ago · Notebook Abalone Regression - Optuna Version 16	0.14634	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 8 Complete · Aldi Hilman Ramadhani · 8d ago · Notebook Abalone Regression - Optuna Version 8	0.14641	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 6 Complete · Aldi Hilman Ramadhani · 8d ago · Notebook Abalone Regression - Optuna Version 6	0.14642	<input type="checkbox"/>
 submission.csv Complete · Aldi Hilman Ramadhani · 7d ago	0.14645	<input type="checkbox"/>
 Abalone Regression - Optuna - Version 13 Complete · Aldi Hilman Ramadhani · 3h ago	0.14653	<input type="checkbox"/>
 solution_04-08_04_59_21.csv Complete · Rein Bugnot · 13d ago · v4: 100 trial hyperparameter tuning - ensemble voting regressor on lgbm/xgb/catboost	0.14661	<input type="checkbox"/>

Fig. 14. Kaggle Submission Scores taken 4th week of April 2024 (not exhaustive)