Assignment 2 – Regression

Name: Gavin Skehan
Student ID: 21440824

Algorithm 1: Random Forest Regression (RFR)

Random Forest regression is a supervised ensemble learning algorithm that improves prediction accuracy by combining outputs of multiple decision trees. The Random Forest creates many decision trees on randomly selected subsets of the data making it less likely to overfit on the training data. The model then analyses the outputs of each decision tree to make an overall prediction for unseen data, this works by averaging predictions. Random Forests can process larger datasets and capture more complex relationships than individual decision trees.

How Random Forest Regression Works

Random Forest regression is an ensemble machine learning algorithm that combines predictions from multiple decision trees to improve accuracy scores for predicting a continuous target variable. Using averaging predictions helps reduce variance and mitigate the impact of high variance in individual trees.

Random Forest uses bagging (Bootstrap Aggregating), meaning trees are created by the random selection of data subsets of the training data (controlled by max_samples if bootstrap=True). Only a subset of features are considered at each split of the tree, reducing overfitting as the trees are trained on different various sub-sets of the training data meaning they are less likely to follow specific patterns within the training set thus improving the model's ability to generalize to new unseen data.

Each decision tree looks for the best split for each node, measuring the quality using variance reduction or squared error to minimize prediction error. The objective is to divide the data so that the target values within each terminal node (leaf) are as close to the mean of the target values in that node. This minimizes the L2 loss, which is equivalent to the mean squared error (MSE).

The model's final prediction is the average of the outputs from all the trees making the model less sensitive to high deviations within individual trees, ensuring the model generalizes well making for unseen data. [Reference 1, 4, 10]

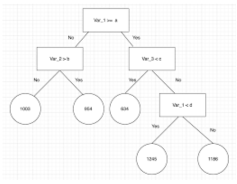


Figure 1: Random Forest Visualization [Reference []]

Why I Chose Random Forest

Random Forest is an ensemble method, which means that the algorithm can create a large amount of individual decision trees to formulate the final prediction. This adds more of an efficient way to predict the target variable compared to decision trees. Changes in the datasets won't affect the overall standing of the model due to the large number of trees in the random forest. Building upon that point, random forests can handle large amounts of data and we can tune this further by increasing the number of trees in the random forest to improve accuracy.

Combining the trees enables the model to avoid overfitting. Random Forests can also handle non-linear relationships.

Hyperparameter Tuning

Hyperparameter 1: n_estimators = number of trees in the forest. Increasing generally improves the performance of the model but also increases the computational cost of training and predicting. The optimal value will reduce variance as the prediction is gathered from the averages of all the trees with each tree trained on a different subset of training data making it less influenced by noise.

Hyperparameter 2: max_depth = max number of levels in each decision tree. Higher values can lead to overfitting while low values can lead to underfitting. The default value to none, if the value is not set, the nodes are expanded until all leaves are pure nodes or contain less than the min_samples_split. When tuned to optimal value max_depth can control bias and enable trees to capture meaningful patterns without focusing on noise, reducing the risk of high variance. [12]

Model Training and Evaluation

Load in the dataset to ensure it is being read. Next, we define the x (features) and y (target variable) as the 'tensile_strenght', dropping this from the x. Visualize the data to get a grasp of the correlations between the features and the target variable, potentially identifying any outliers.

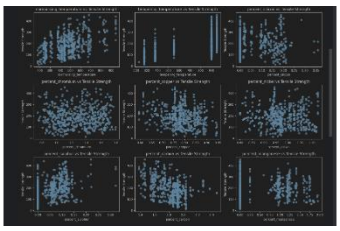
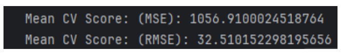
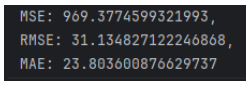


Figure 2: represents the correlation between features and target variable (tensile_strength)

Split the data into training and test sets using the 'train_test_split' with an 80-20 split of training to test data. Initialize the model with the default parameters (n_estimators=100, max_depth=none, criterion='squared error', min_samples_split=2, min_samples_leaf=1) and fit the model to the training data. The next step is to cross-validate [11] the training accuracy on the training data using Mean Squared Error (MSE) and Root Mean Squared Errors (RMSE). The RMSE of 32.51 indicates that the model's predictions on average are 32.51 units away from the actual tensile strength values. We can see there is room for improvement with hyperparameter tuning. When evaluating the test data the Mean Absolute Error (MAE) was 23.80, MSE was 969.38, and the RMSE was 31. This suggests that the model is generalizing well on unseen data but highlights the need for tuning to improve the accuracy further.

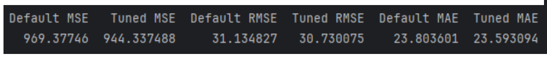

```
Mean CV Score: (MSE): 1056.9100024518764
Mean CV Score: (RMSE): 32.510152298195656
```
Training Evaluation


```
MSE: 969.3774599321993,
RMSE: 31.134827122246868,
MAE: 23.803600876629737
```
Testing Evaluation

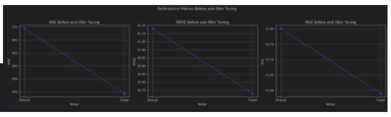
GridSearch was used to identify the best hyperparameter pair of the n-estimator and the max_depth. GridSearch aims to find the best hyperparameters by exhaustively searching the defined grid. This is done by following the template in sci-kit-learn by declaring the param_grid with the n_estimator and max_depth values. Initialize the grid_search with the following: estimator=Random_forest, param_grid=param_grid, scoring='neg_mean_squared_error', n_jobs=-1, cv=10, verbose=1 and then fit to the training data. The grid search function will run through all the possible combinations to find the best match.). The parameter ‘cv=10’ specifies that it is a 10-fold cross-validation.

Results After Fine-Tuning Hyperparameters

The best hyperparameters were chosen based on obtaining a good balance. We want to avoid overfitting (high variance) and underfitting (high bias). Low variance and low bias are the ideal scenarios for creating a curved line on the graph. We see a slight increase in accuracy due to hyperparameter tuning.



```
Default MSE   Tuned MSE  Default RMSE  Tuned RMSE  Default MAE  Tuned MAE
  969.37746  944.337488     31.134827   30.730075    23.803601  23.593094
```



Regression Performance Metrics

Mean Squared Error (MSE)

MSE measures the average difference between the expected target values in the test set and the values predicted by the model (Citation to notes). MSE penalizes larger errors more heavily. The tuned model has a lower MSE (944.34) than the default (969.38). This 2.58% decrease highlights an improvement in accuracy as the average difference is reduced.

Root Mean Squared Error (RMSE)

The RMSE is simply the root of MSE. It enables us to interpret the answers easier. A decrease in RMSE from the default: 31.13 to the tuned: 30.73 shows the model is consistently closer to the actual values for out tuned model.

Mean Absolute Error (MAE)

MAE values give us information on the average error in each prediction without considering direction. The slight decrease from 23.8 to 23.59 suggests that tuning the model reduced the average error meaning the model is more reliable post-tuning. [Reference 9]

Algorithm 2: Support Vector Regression (SVR)

Support Vector Regression (SVR) is a supervised learning algorithm for predicting continuous numerical outputs. It is built on the same principles of Support Vector Machines (SVM) but is used to handle regression tasks. The objective is to find a hyperplane to approximate target values within a margin of tolerance called the e-insensitive tube. We want to have as many points within this tube while minimizing deviations for the points that lie outside the tube. SVR can handle both linear and non-linear relationships by using a kernel to map data to a high-dimensional space. We will use the radial basis function (RBF). [Reference 7]

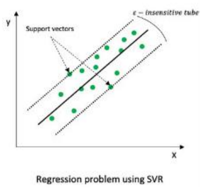


Figure 2: Regression problem in SVR showing how SVR uses the e-insensitive tube [Reference 2]

How Support Vector Regression Works

SVM is a supervised machine-learning algorithm used for regression analysis. The goal in SVR is to predict continuous output values while fitting as many data points as possible within the margin of tolerance (epsilon) or the e-insensitive tube. SVR minimizes errors outside of the margin of tolerance. The e-sensitive tube is a lost function that ignores errors within the margin and highlights points outside the margin that will be penalized based on distance from the margin. Slack variables handle errors beyond the tube by penalizing them depending on whether the predictions are above or below the tube. The C parameter for regularization is used to adjust the penalty placed on the errors outside of the tube.

Optimization Problem: SVR formula to minimize regularization and the penalties applied to errors outside the tube.

$$\begin{aligned} &\min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ \text{subject to } & y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ & w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ & \zeta_i, \zeta_i^* \geq 0, i = 1, \ldots, n \end{aligned}$$

Figure 3: Optimization Problem Formula

Prediction Formula: after training the SVR uses this formula to make predictions

$$\sum_{i \in SV} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

Figure 5: The prediction formula

- ai and ai* represent the weights of each support vector (data points that lie on or outside the boundary)
- K(xi, x) represents the kernel function which in my case is the Radical Basis Function (RBF). It enables us to handle complex non-linear relationships. Here is the formula: $\exp(-\gamma \|x - x'\|^2)$ Figure 6: RBF Kernel function [Reference 8 - Formulas]

Why I Chose Support Vector Regression

SVR can handle non-linear relationships with the use of the correct kernel. I am curious to see the impact of tuning the hyperparameters C and Gamma and how much it can improve the performance of SVR to prevent or avoid overfitting of the model on the training data while still maintaining or improving high accuracy on the test set.

Hyperparameter Tuning

Hyperparameter 1: C: the regularization C parameter controls the trade-off between complexity and training error. When C is large, the model minimizes errors in the training data which can lead to overfitting to the training set. When C is small, this allows for more flexibility by allowing the model to pick up on fewer errors. Removing overfitting and improving generalization as we aren't fitting too close the patterns and data on the training model, the performance can improve on unseen data.

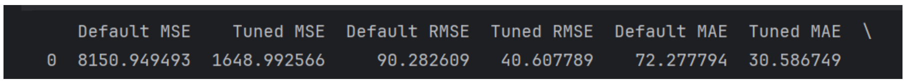
Hyperparameter 2: Gamma decides how closely the model fits the training data. Low gamma means your model not follow patterns directly from the training data which can improve generalization. High gamma means the model is trying to highlight every little detail and pattern in the training set. This could lead to overfitting, which is not good for generalization.

Model Training and Evaluation

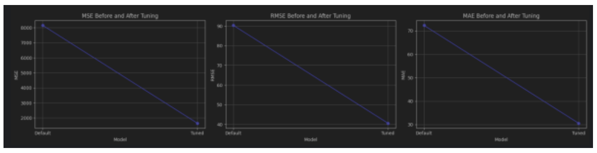
Load the dataset to ensure it is being called correctly. Define the features and the target variable (tensile_strength) by dropping the target variable from X. Next we visualize the features against the target variable to see the correlations and identify any outliers. As the data is not yet split, we do this manually by calling the ‘train_test_split’ using an 80-20 split. In SVR it is essential to standardize the data as it is sensitive to the magnitude of the features. By splitting before preprocessing we avoid any data leaks and can get a better evaluation. To scale use the ‘StandardScaler()’ function. Initialize the SVR model with the default hyperparameters (kernel=’rbf’, C=1.0, gamma=’scale’, etc). Train the model on the training dataset and evaluate it with cross-validation. The default RMSE gives a score of 79.05 meaning is it approximately 79.05 units on average away from the target variable. Evaluate the test to have a benchmark score to improve on. MSE: 8150.95, RMSE: 90.28, MAE: 72.30. To determine the most optimal hyperparameters to tune we use GridSearch. GridSearch iteratively searches all the combinations of the hyperparameters initialized in the param_grid finding the best scores determined by the negative mean squared error. This model fits 10 folds for each of the 16 candidates, totaling 160 fits. We are checking two hyperparameters, C: [0.1, 1, 10, 100] and Gamma: [0.1, 1, 10, 100]. Best Hyperparameters: {'C': 100, 'gamma': 0.1}. [Reference 2, 3, 5, 7, 8]

Results After Tuning Hyperparameters

Get the results after find-tuning by running the scaled model with the best SVR parameters on the test data and record the results. Tuned MSE: 1649, RMSE: 40.61, MAE: 30.59. We get a significant increase in accuracy after fine-tuning which highlights the importance of utilizing the most optimal parameters.



```
   Default MSE   Tuned MSE  Default RMSE  Tuned RMSE  Default MAE  Tuned MAE  \
0  8150.949493  1648.992566     90.282609   40.607789    72.277794  30.586749
```



Regression Performance Metrics

Mean Squared Error (MSE)

MSE measures the average difference between the expected target values in the test set and the values predicted by the model. MSE penalizes larger errors more heavily. The tuned model has a significantly lower MSE (1649) than the default (8151). This decrease highlights a major improvement in accuracy as the average difference is reduced.

Root Mean Squared Error (RMSE)

The RMSE is simply the root of MSE. It enables us to interpret the answers more easily to see how far on average in units the score is away from the target variable, it is simply the root of the MSE score. A decrease in RMSE from the default: 90.3 to the tuned: 40.61 shows the model is consistently closer to the actual values for out tuned model.

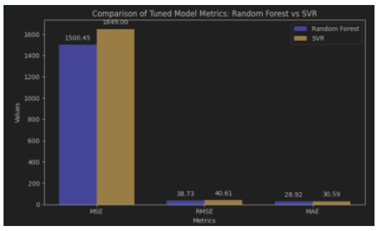
Mean Absolute Error (MAE)

MAE values give us information on the average error in each prediction without considering direction. The decrease from 72.28 to 30.59 means that tuning the model reduced the average error meaning the model is drastically more reliable post-tuning. [Reference 9]

Comparative Analysis of Models

Both models ended up performing well and relatively close in terms of performance. Where we see a drastic difference was the effect of the hyperparameter training on the SVR model. There was a huge decrease in RMSE of 55.03% and a 57.68% decrease in MAE. This indicates that hyperparameter tuning for SVR is essential. Although Random Forest didn't have dramatic accuracy improvements with hyperparameter tuning, it still improved after tuning. It performed better than SVR with an RMSE of 38.73 and an MAE of 28.92. These are slightly better than the SVR.

Visual Comparison: Performance Increase from Tuning



Conclusion

In conclusion, Random Forest Regression performed slightly better than SVR after hyperparameter tuning. The change from default to optimal parameters was significantly more dramatic on the SVR than on the Random Forest. The large improvements highlighted that the SVR model's default parameters were not well-suited to the dataset. After tuning we saw a major improvement, whereas the Random Forest was well-suited but still gained slight accuracy improvements with tuning.

Key Findings

SVR is much more sensitive to hyperparameter tuning, it is essential to preprocess the data and tune for this model. Random Forest performs better than SVR slightly after tuning, it is less dependent on tuning but still benefits from minor adjustments. In SVR the tuning of C to a higher value of 100 allowed the model to capture more complex patterns in the training data, reducing the bias and maintaining good generalization. Gamma was tuned to 0.1, enabling the model to focus on meaningful patterns without becoming overly sensitive to noise.

Model Performance: Suggestions for Further Improvements

Test more parameters in the GridSearch especially for SVR as we saw such a dramatic increase, I would like to see the scores if we also added the epsilon, max_iter, kernels, and degree to the parameters grid. This could make the model outperform the current random forest model. For Random Forest we could explore the max_features and min_samples_leaf. We could also incorporate a validation set into the process to ensure the models test on unseen data ensuring the model generalizes well.

References

1. AnalytixLabs, 2023. “Random Forest Regression – How it Helps in Predictive Analytics?”, “Medium”, 26 Dec. Available at: https://medium.com/@byanalytixlabs/random-forest-regression-how-it-helps-in-predictive-analytics-01c31897c1d4
2. Rasifaghihi, Niousha, 2023. “From Theory to Practice: Implementing Support Vector Regression for Predictions in Python”, “Medium”, 21 Apr. Available at: https://medium.com/@niousha.rf/support-vector-regressor-theory-and-coding-exercise-in-python-ca6a7dfda927
3. “Maulana Achsan, Beny, 2019. “Support Vector Machine: Regression, “Medium”, 10 Dec. Available at: https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345
4. Figure 1: Beheshti Nima, “Random Forest Regression”, “Medium”, 2 Mar 2022. Available at: https://towardsdatascience.com/random-forest-regression-5f605132d19d
5. Figure 2: Rasifaghihi, Niousha, 2023. “From Theory to Practice: Implementing Support Vector Regression for Predictions in Python”, “Medium”, 21 Apr. Available at: https://medium.com/@niousha.rf/support-vector-regressor-theory-and-coding-exercise-in-python-ca6a7dfda927
6. GridSearch Code: “GRIDSEARCHCV.” *Scikit*, scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed 12 Nov. 2024.
7. “Support Vector Regression Tutorial for Machine Learning.” *Analytics Vidhya*, 9 Oct. 2024, www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/#h-what-is-a-support-vector-machine-svm.
8. “1.4. Support Vector Machines.” *Scikit*, scikit-learn.org/stable/modules/svm.html#kernel-functions. Accessed 9 Nov. 2024.
9. Glavin, Franks. Topic Six Lecture Notes, Regression. Available at: https://universityofgalway.instructure.com/courses/31934/files/2141299?wrap=1
10. “1.11. Ensembles: Gradient Boosting, Random Forests, Bagging, Voting, Stacking.” *Scikit*, scikit-learn.org/stable/modules/ensemble.html. Accessed 9 Nov. 2024.
11. “Cross_val_score.” Scikit, scikit-learn.org/dev/modules/generated/sklearn.model_selection.cross_val_score.html. Accessed 20 Nov. 2024.
12. “Randomforestregressor.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor. Accessed 9 Nov. 2024.

Appendix

Cross Validation

Our data was divided into testing and training sets. The training set was further divided into k folds, or subsets. Training the data on k-1 of the folds and evaluating on the kth fold (validation data) allows us to iteratively fit the model k times. Take, for instance, fitting a mode with k = 5. We train on the first four folds and assess on the fifth in the initial iteration. We train on the first, second, third, and fourth times, and we assess on the fourth. We carry out this process three more times, assessing on a different fold each time. To determine a final validation metric for the model, we take the average of the performance on each of the folds.

Metrics:

$$\text{sum of squared errors} = \frac{1}{2}\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2$$

Mean squared error (MSE)
$$\text{mean squared error} = \frac{\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2}{n}$$

Root mean squared error (RMSE)
$$\text{root mean squared error} = \sqrt{\frac{\sum_{i=1}^{n}(t_i - \mathbb{M}(\mathbf{d}_i))^2}{n}}$$

Mean absolute squared error (MAE)
$$\text{mean absolute error} = \frac{\sum_{i=1}^{n} abs(t_i - \mathbb{M}(\mathbf{d}_i))}{n}$$

Formulas: [Reference 9]