# Assignment 2 – Classification using Scikit-learn

**Student Name**: Robert Walsh **Student ID**: 18103802 **Programme**: 4BCT

## Algorithm 1: Random Forest Regressor – Overview & Detailed Description

Random Forest Regressor is one of the most popular algorithms in machine learning because it's strong, stable, and works well on many different types of datasets. The idea behind it is that instead of relying on one decision tree, it builds many of them. Each tree is trained on a random sample of the training data (which is called bootstrapping, where rows are picked with replacement) so every tree ends up seeing a slightly different version of the dataset.

When a tree is being built, it doesn't look at all the features at every split. Instead, it only considers a random subset of features. The randomness (random rows & random features ) helps ensure that trees don't all learn the exact same patterns leading to overfitting. Because the trees are less correlated, the final model avoids overfitting and generalises better to new and unseen data.

During training the tree splits the data based on features and thresholds that reduce the error within each node. For regression tasks, the splitting usually relies on mean squared error (MSE) or something similar, because the goal is to reduce the variance of the target values in each split. The trees keep splitting until a stopping rule is met. One important hyperparameter that controls this is max_depth, which limits how deep trees are allowed to grow.

When it comes to making predictions, the process is as follows - a new input sample is sent through every tree in the forest. Each tree produces its own continuous prediction. The Random Forest then takes the average of all these predictions. This averaging effect is powerful because it reduces the impact of noisy or unreliable trees. If one tree makes a bad prediction, the other trees usually cancel it out, which makes the overall model more accurate and stable.

### Why I choose this algorithm.

I chose Random Forest Regressor mainly because I wanted it to contrast with my second algorithm (KNN). Random Forest is very different in how it learns & it can capture complicated, non-linear relationships between the features without needing any scaling or heavy preprocessing. It handles mixed feature ranges naturally because tree splits are not affected by the scale of the values. Random Forest is strong at learning global patterns in the data. It looks at the bigger structure of the dataset and can find important interactions/correlationa between features. However, it doesn't always perform as well when the relationship between inputs and outputs is very local or based on small neighbourhood patterns. That is the type of situation where KNN tends to perform better. By choosing two very different algorithms, one tree based and one distance based, I felt I could get a better overall understanding of the steel dataset. Each algorithm has strengths where the other one is weaker, so comparing them should give more meaningful insights than choosing two very similar models.

### Hyperparameter Details for Tuning.

*Hyperparamer 1*:

**n_estimators**

As it is one of the most important hyperparameters in a Random forest model. It controls the number of individual trees that make up the forest. Increasing this number generally improves

performance because more trees mean more averaging, which usually reduces variance and creates more stable/accurate predictions. However, too many trees can increase training time without providing much extra benefit. Because it directly influences the strength and stability of the model, it is a beneficial hyperparameter to tune and compare.

*Hyperparamer 2:*

**max_depth**

Because it limits how many splits a tree can make from top to bottom. If trees are allowed to grow too deep they can memorise the training data and overfit badly. If the depth is too shallow each tree becomes too simple and underfits. This hyperparameter directly controls the model's bias/variance balance. Deeper trees lower bias but increase variance, while shallower trees do the opposite. Because of this, max_depth has a major effect on model behaviour and is an ideal hyperparameter to tune in order to observe overfitting/underfitting patterns.

## Algorithm 2: KNN Regressor – Overview & Detailed Description

K-Nearest Neighbours, KNN is a simple but widely used algorithm in machine learning and can be used for both classification and regression problems/tasks. In regression, the aim is to predict a continuous target value for a new input by looking at the target values of its K closest neighbours in the training data. The prediction is just the average of those neighbour targets. The basic idea behind this is that datapoints that are close to each other in feature space should usually have similar outcomes. To measure how close points are, KNN uses distance metrics. The two most common ones are Euclidean distance and Manhattan distance, but there are other options too if the dataset demands it [4]. The choice of distance metric can change how the algorithm behaves, especially when the input features have very different scales.

One of the main hyperparameters in KNN is K, which decides how many neighbours we look at before making a prediction. This hyperparameter is essential for KNN to work & without a value for K, the algorithm has no definition of what a neighbourhood even is. If the user does not specify K, many libraries including scikit-learn will default to K= 5. When new data is introduced the algorithm calculates the distance from that new point to every point in the training set and identifies the K smallest distances and then averages their target values. The algorithm compares the feature values of the new data point to the training data.

Since KNN is completely dependent on distance calculations, preprocessing is a crucial step, especially when feature values are on different scales. In the steel dataset, features like normalising temperature can be in the hundreds, while percentages of chemical elements range from 0 to 1. Without scaling, the large-range features (like temperature) would dominate the distance metric and the algorithm would likely ignore the smaller-range features such as carbon or sulphur percentages. To prevent this, we apply StandardScaler() or another scaling method so that all features contribute fairly/equally to the distance calculations. Doing this makes the comparisons meaningful and avoids model bias towards high magnitude features.

### Why I choose this algorithm.

I chose KNN Regressor as my second algorithm because it is very different from RandomForestRegressor, both in how it learns and in the types of patterns it tends to capture. KNN

is a distance based method that works directly with numerical features and it generally performs well on small or medium-sized datasets which fits the steel data. It does not require any encoding because all of the features in this dataset are already numeric.

One of the main reasons for choosing KNN is that it has strengths where Random Forest tends to struggle. Random Forest is an ensemble of decision trees and is good at picking up complex global patterns and interactions within features but it does not always perform well when the relationship between inputs and outputs is very local or smooth. KNN works in almost the opposite way where it makes predictions based on local neighbourhoods in the data so it can capture local structure more effectively & especially after scaling.

Using two algorithms that are fundamentally different gives a more meaningful comparison than using two similar ones. Random Forest relies on many deep trees that produce piecewise splits, while KNN relies on distances and produces smoother more continuous predictions. This contrast makes it easier to analyse differences in bias and variance & to compare how each model reacts to scaling and to understand which type of structure the steel dataset supports better. Overall, the two algorithms should balance each other's weaknesses and provide clearer insights when evaluated side by side.

### Hyperparameter Details for Tuning.

*Hyperparamer 1:*

***n_neighbors***

K controls how many neighbours are considered when making a prediction. It is the main hyperparameter in KNN and can significantly affect model performance. If K is too small, the model becomes very sensitive to noise and can overfit. If K is too large, the model becomes too smooth and may underfit because it averages across too many points. Tuning K helps find a balance between these two extremes so it makes sense to include it in the hyperparameter search.

*Hyperparamer 2:*

***p***

The p hyperparameter determines which distance metric the model uses. When p = 2, KNN uses Euclidean distance, and when p = 1, it uses Manhattan distance. The choice of distance metric has a direct impact on how the algorithm identifies the closest neighbours, especially when the features vary in how they scale or interact. Since KNN is entirely dependent on distances, tuning p can lead to meaningful differences in performance and is a suitable hyperparameter to tune for this dataset.

## Algorithm 1 – Random Forest Regressor - Model Training and Evaluation

### Training and Evaluation Details

### Discussion of results

# Algorithm 2 - <Name of Algorithm> - Model Training and Evaluation

### Training and Evaluation Details

### Discussion of results

## Conclusions

### Key Findings

- Sem viverra aliquet eget sit amet tellus cras adipiscing enim. Nulla facilisi cras

### Comparative Analysis of Algorithm Performances

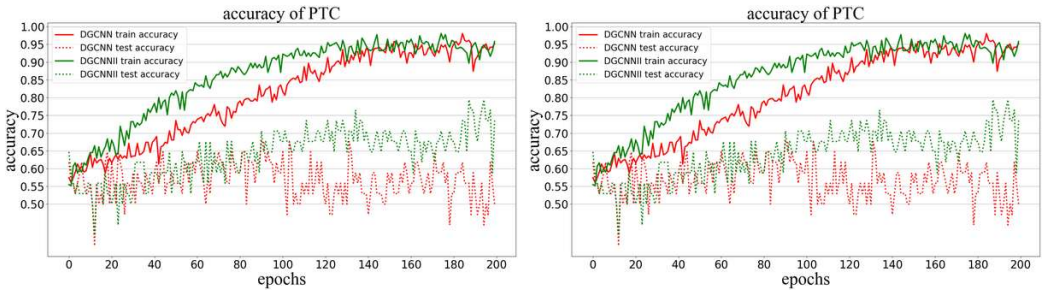


*Figure 1: Graphed comparison of results from the two algorithms*

### Recommended Hyperparameter Valued based on Results

*Algorithm 1 Hyperparamer 1:*

*Algorithm 1 Hyperparamer 2:*

*Algorithm 2 Hyperparamer 1:*

*Algorithm 2 Hyperparamer 2:*

### Concluding Remarks

## References

1. https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/
2. https://www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/
3. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
4. https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbors-knn-regression-with-scikit-learn/
5.

## Report Update Log

| Date | Update Description | Time Spent |
|---|---|---|
| **08/11/25** | Prepped template, changing headings and removing filler content | 0.5 hours |

| | | |
|---|---|---|
| **11/11/25** | Populated thee template with Random Forest Regressor overview, description and ran the initial code without hyperparameters to review results prior to implementing hyperparameters. | 2 Hours |
| **12/11/25** | | |