Course Code: CT4101	Assignment: 3 (Regression Report)
Name: *Seshadri Sundarrajan*	Student ID: 19230107

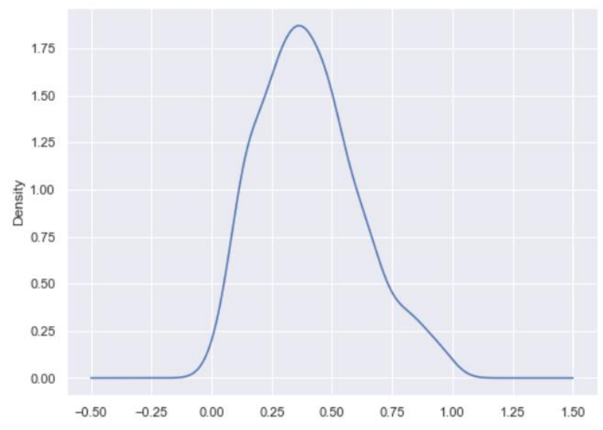
LIBRARY USED: SCIKIT-LEARN

Reasons for opting scikit-learn

- It is one of the commonly used libraries by ML practitioners because of its end-to-end documentation and community support.
- It provides extension for both supervised and unsupervised algorithms along with model selection, dimensionality reduction and pre-processing (most of the core functionalities needed for ML are captured).
- Pipelining capability provided is more useful while designing and altering end to end algorithm workflow.
- Moreover, library is in Python programming language, which was familiar to me and is highly regarded language by Data Scientists owing to its readability, robustness and accessibility.
- It is built to support data structures from Numpy and Pandas (Libraries for Data mining and Analysis) and model built can be fed into Matplotlib library for visualisation. This makes whole process simple and flexible by catering most of our needs.

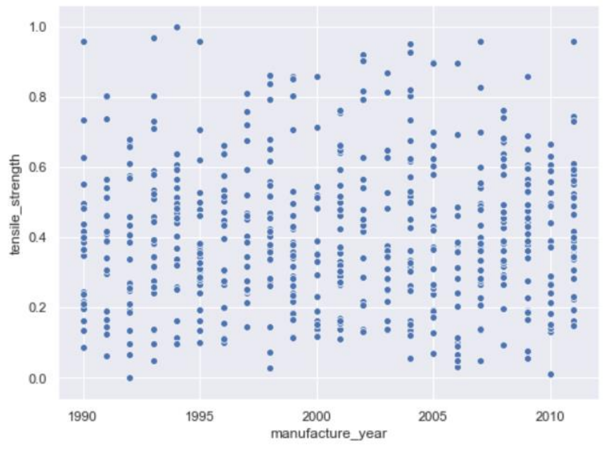
DATA PRE-PROCESSING

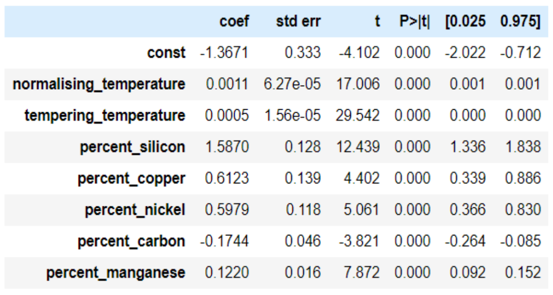
- Provided dataset doesn't have null values, column header and has each instance are along the row. So, after loading file into Pandas data frame added column header.
- Dependent variable 'tensile strength' is in the range 12 – 450. In order to interpret the regression metrics and analyse model performance scaled the variable to be in the range 0 – 1 (using Min Max Scaler). Verified if tensile strength is skewed but it looked normally distributed.



Feature Engineering

1. **'Sample'** column indicates the serial number to distinguish instances, which has no relation to our dependant variable. Hence, removed the particular column.
2. **'Manufacture year'** is also removed as it has no impact on prediction, this can be seen from the scatter plot with points evenly spread around without definite pattern.
3. Followed **Backward Elimination** technique to reduce features that are redundant and has least significance towards prediction. Used Ordinary Least Square method to get p-value of each coefficients and removed the variable with highest p-value (i.e p value >0.05). Then fitted model with new set of variables and repeated the process until there is no feature to be removed.
4. Final set of significant features are shown in the OLS summary table, the same has been used for building the models.





	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.3671	0.333	-4.102	0.000	-2.022	-0.712
normalising_temperature	0.0011	6.27e-05	17.006	0.000	0.001	0.001
tempering_temperature	0.0005	1.56e-05	29.542	0.000	0.000	0.000
percent_silicon	1.5870	0.128	12.439	0.000	1.336	1.838
percent_copper	0.6123	0.139	4.402	0.000	0.339	0.886
percent_nickel	0.5979	0.118	5.061	0.000	0.366	0.830
percent_carbon	-0.1744	0.046	-3.821	0.000	-0.264	-0.085
percent_manganese	0.1220	0.016	7.872	0.000	0.092	0.152

MODEL 1: GRADIENT BOOSTING

It's is an ensemble technique belonging to boosting family. It internally uses trees to produce its weak learners. Since it's a regression tree, we use Weighted Mean Squared Error as our Impurity measure in determining the best split.

The main ideology behind Gradient boosting is to combine the predictions of each weak learner and form a strong base learner that generalises all tree outputs to produce final outputs. Trees are sequentially built such that error made by previous tree is minimised by next tree. The algorithms flow is as follows -

1. The process starts by picking a value (can be assumed like a single leaf) as initial prediction (usually average of our target values) and residuals are calculated by comparing this value to actual value.

2. Depending upon the residual calculated we start building the initial tree to **predict the residual calculated in the previous step**. The reason for predicting residual is we scale each tree output and add them with the initial guess we made. Thus, we edge closer in minimising the gap between actual and predicted values.

3. For each data point, the residual predicted from the first tree is calculated and scaled by our learning rate and added to initial guess (this is our predicted output). Here learning rate ranges between 0 to 1, lesser the value smaller the steps to the final prediction. Similar to gradient descent learning rate.

4. Once the first tree is complete the next tree gets built using bootstrap sample from train data, with new set of residuals taking previous predictions into account.

Predicted Value = Initial Guess + Learning_rate * Tree_1 Residual predicted
+ Learning_rate * Tree_2 Residual predicted
+ ....until covergence

Loss Function = $\frac{1}{2}$ **(Observed - Predicted)**^{2}

5. The above process gets repeated until convergence, and at each tree we try to minimise the residuals and finally arrive at optimal prediction. To get predictions for test data,we get the tree outputs and scale them.

Convergence: Gradient boosting builds tree until the number of trees are met or the newly constructed tree fails to improve the model fit. Individual tree can be pruned early as we are taking smaller steps to our goal.

Parameters used:

Name	**Value**	**Description**
loss	ls	Loss function = least squares regression
learning_rate	0.1	Contribution of each tree to the final output = 0.1
n_estimators	75	Total number of trees (Weak learner) to be used
max_depth	5	Maximum depth of each tree = 5
max_features	4	Maximum features selected for assessing best split
Max_leaf_nodes	30	Maximum number of leaf nodes present in a tree = 30

- Performed pruning using tree depth and max features with values less than total features to restrict growth.
- Noticed that if amount of contribution from each tree (learning rate) is reduced then model performs poorly, potential reasons may be number of trees we have, so more the number of trees then lesser is the learning rate.

Cross Validation Score:

Mean Value of Adjusted R2 after 10-fold Cross validated is 0.933 (i.e **93%** of variance in tensile strength is captured)

Test Metrics [Using *RMSE* as it penalises outliers heavily and *Adjusted R2* as it penalises less significant feature more than R2]:

Root mean Square Error = **0.066** (i.e standard deviation of our prediction is 0.066)

- **Note:** We have scaled our tensile strength to lie between 0 and 1. There is always an error associated with our prediction which deviates from actual value by 0.066.

Adjusted R2 score = **90%** (Which is almost similar to our cross validated Adjusted R2 score and as both scores are comparatively high, which indicates our model is neither overfit nor underfit)

```
RMSE   : 0.066

MAE    : 0.044

R2     : 0.911

Ajd R2 : 0.909
```

MODEL 2: MULTI LAYER PERCEPTRON

It is a supervised neural network algorithm, that uses neuron as fundamental element. These neurons are interconnected to one another (like neurons in our brain).

Main ideology behind Multi-layer Perceptron is to create interconnected neural network that operates between Input neurons and Output neurons, by communicating to and forth until the goal is clinched. The algorithm flow is as follows

1. Each feature is assigned an Input node(neuron) and for regression there is a single output node.
2. Between the input and output layer of nodes we can customise our own set of **hidden** (unobservable) layers.
3. These is a weight associated with each input to a neuron and there is a single input from **bias** that also has weight and its functions similar to constant in regression equation (helps to change the intercept accordingly).
4. There is always a connection between an input node and all the other nodes in the hidden layer. They represent a bipartite graph.
5. Weighted input is passed through an **activation function** that produces single output value for a node. Depending upon our use-case we can make output at a node between 0 -1 (or) -1 to +1 (or) same as weighted input via activation function that maps input value to be inside these range.
6. Feed Forward and backward propagation are the two main process used to communicate to and forth between the output and input layers via hidden layers.
    a. **Feed Forward** mechanism carries signals from input node and that gets processed in the hidden layer and finally terminates at output node. We define a loss function that tries to minimise the error at output node by initiating Back propagation.
    b. **Back Propagation** mechanism tries to rectify the error from the feed forward stage by altering the weights associated with each input based on their contribution to error, it propagates backwards from output layer to Input layer.
7. Once our entire training data has passed through the network and model parameters are updated then we have successfully completed an **Epoch**. We need to do several Epochs so that our overall error is minimal.

Convergence: After every iteration we minimise our loss function and we compare our current error value with that of previous step and difference should be lesser than tolerance. Conversely, number of Epochs also governs convergence

Parameters used:

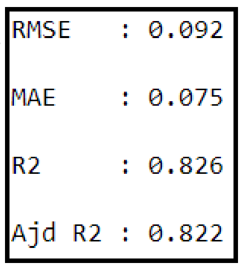
Name	Value	Description
hidden_layer_sizes	2	Total number of hidden layers present in the model.
activation	logistic	Activation function to be applied on weighted inputs at each neuron
solver	lbfgs	Optimization algorithm = Limited Memory -BFGS (Finds the local minima of our cost function with less computer memory)
alpha	0.01	Regularisation parameter to penalise less significant inputs (L2 norm)
tol	0.001	Convergence will be attained if loss or score is less than this tolerance value.

- Tried increasing hidden layers but model started to overfit and testing accuracy was less as it failed to generalise. Similarly, reducing penalising parameter 'alpha' created underfit.
- Stochastic gradient descent was not applicable as solver for our dataset, as it is suited for large datasets and failed to converge quickly, so went with L-BFGS which is suited for smaller dataset with fast convergence.

Cross validation Score:

Mean Value of Adjusted R2 after 10-fold Cross validated is 0.854 (i.e **85%** of variance in tensile strength is captured)

Test Metrics [Using RMSE as it penalises outliers heavily and adjusted R2 as it penalises less significant feature more than R2]:



```
RMSE    : 0.092

MAE     : 0.075

R2      : 0.826

Ajd R2 : 0.822
```

Root mean Square Error = **0.092** (i.e standard deviation of our prediction is 0.095)

- **Note:** We have scaled our tensile strength to lie between 0 and 1. There is always an error associated with our prediction which deviates from actual value by 0.092.

Adjusted R2 score = **82%** (Which is almost close to our cross validated Adjusted R2 score and as both scores are comparatively high, which indicates our model is neither overfit nor underfit)

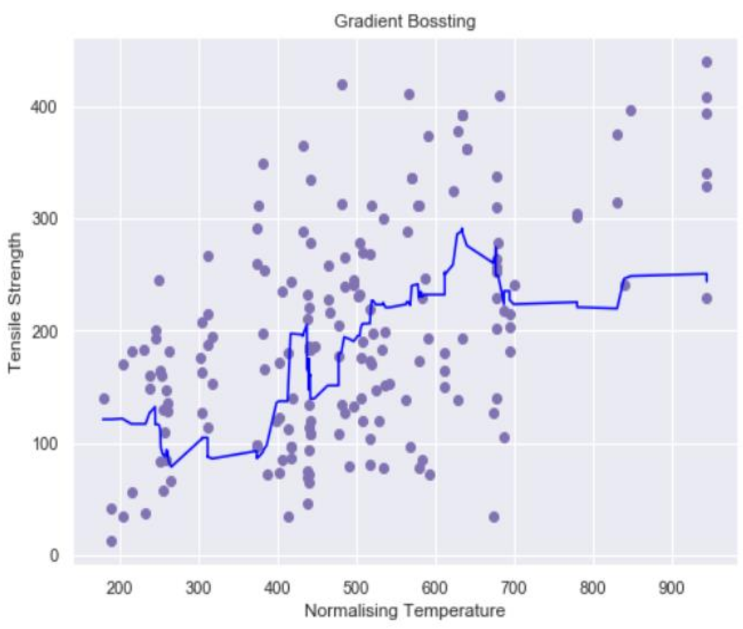
TEST TRAIN SPLIT TO PREVENT UNDERFIT AND OVERFIT:

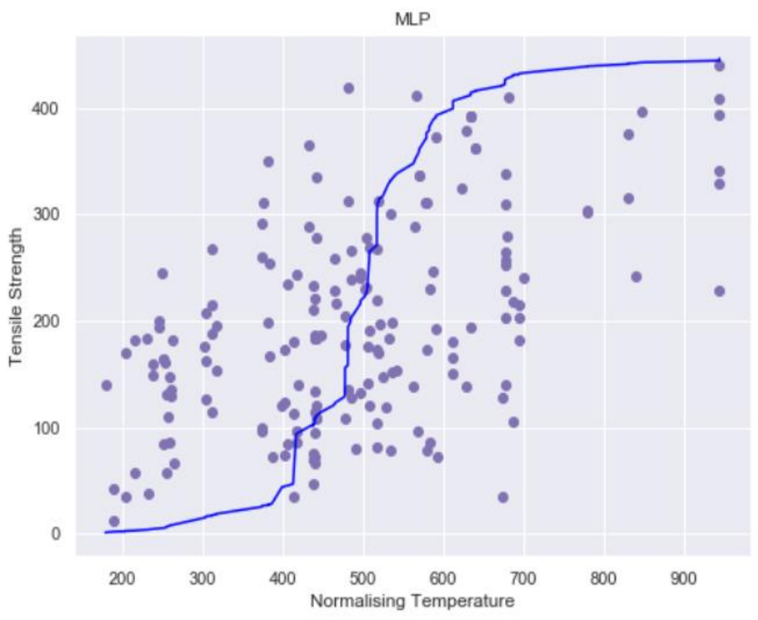
- Train and Test data are split in ratio 2:1, chose this split ratio by examining the model performance after cross validation (done on train set) and comparing with test set performance.

  - Whenever model performances were poor after cross validation performed parameter tuning and optimised values as per the problem and to increase the performance, thus overcome ***underfit***.
  - Chose ensemble and deep learning algorithms so that ***bias variance trade-off*** is in balance, though part of trade-off is taken care by algorithm and still some parameters were tuned to ensure it.
  - Initially, did some feature engineering to reduce dimensions and which in turn ***reduced noise***.
  - For GB, used tree pruning and in MLP used regularisation parameters to prevent ***overfit***.

MODEL COMPARISON

- It is clear, that **Gradient boosting** has better prediction power compared to multi-layered perceptron for the mentioned parameter setting. While predicting tensile strength (in range 0 -1) gradient boosting has *0.06 MSE* that is lesser than multi-layered perceptron with 0.09.
- Moreover, in MLP optimiser smoothens the prediction line which results in recording **lesser tensile strength variance** (i.e 82% is the explained variance).
- Likewise, in Gradient boosting we have used trees as weak-learner and have got a more complex prediction line that is bit erratic but captured nearly **90% of total variance**.
- Both the models are individually powerful as they are either ensemble or neural network, so it’s difficult to judge them on the grounds of their methodology. Yet for the given problem statement we can assure Gradient boosting provides better results.
- Regression line is visualised for both models, used only ‘Normalising Temperature’ to generate the plot as we are dealing with high dimensional data. As stated earlier, GB regression line is irregular and captures most of variations while MLP regression line is smooth and approximates a lot.





REFERENCES

1. https://machinelearningmastery.com/neural-networks-crash-course/
2. https://statquest.org/2019/04/02/gradient-boost-part-2-regression-details/
3. https://statquest.org/2019/03/25/gradient-boost-part-1-regression-main-ideas/
4. https://www.geeksforgeeks.org/ml-multiple-linear-regression-backward-elimination-technique/
5. https://scikit-learn.org/stable/modules/cross_validation.html