Machine Learning

Assignment 1:

**Name:** Roopan Marimuthu

**ID:** 21489965

**Kaggle Username**: NA

**Contents**

1.0: Introduction

2.0: Data Processing

2.1: Train, Test, Validation & Cross-Fold Validation

2.2: Outliers

3.0: Linear Model

4.0: Decision Tree

4.1: Random Forest

4.2: Gradient Boosting Tree

4.3: XGBoost

5.0: Neural Network

5.1: Hyperopt Tuning

**Introduction**

For this formal assignment for the UNIT machine learning (COMP3010), I will be required to formulate machine learning models for data relating to a BLEVE blast wave propagation in an obstacle environment. Specifically, I will be required to utilise at least **three fundamentally different** types of machine learning models. I must take full advantage of hyparameter tuning for each model and use compulsory evaluation metrics such as MAPE or . After developing certain models, I must consider ensemble methods, which refers to the utilization of multiple models to improve the final predication performance.

***Environment***

For this assignment, I will be using a Jupyter notebook in a PyCharm IDE.

**Data Processing**

Before building any models, I must ‘clean’ training data and optimize.

For now, there are 3 methods of Data Processing I have utilized, this includes dropping duplicate entries, dropping invalid entries and ‘one hot encoding’ categorical data.

Dropping duplicate entries was an obvious step in cleaning up the data, with little to no downsides. However, with invalid data that has missing values for certain features, there is a bit more that could be done. For its relevant features, missing values have been replaced with the median for the column, this would result in no actual data loss, and the same amount of data samples.

As for the categorical data, or in this case, ‘Status’ was a feature that could be allocated either ‘Superheated’ or ‘Subcooled’. However, faced with the first issue, there had been multiple typos in declaring this options, this made ‘hot encoding’ a bit more difficult. A simple solution was to utilise panda’s *get\_dummies()* function to retrieve categorical data and implement it as its own separate features, looking at the new added columns, we would then have to merge typos in their respective columns, with only 2 remaining columns ‘Superheated’ or ‘Subcooled’ represented with Boolean values a ‘True’ or a ‘False’. The column was concatenated with the original dataset, the ‘Status’ feature was deleted, and columns were shifted to allow for the target variable to be the last column.

**Train, Test, Validation** & **Cross-Fold Validation**

Throughout the model I will utilise, I will split data individually to see what may benefit the data the best.

Train will be utilised to train the model, validation or cross-fold validation will be used to validate and improve the model, whilst test is used **ONLY** when the model is fully trained and optimized, If I am unhappy with the scores from the test set, the model will be retrained and have its parameters tuned.

A standard scaler was also fit to **data** to optimise convergence and prevent feature overshadowing.

**Metrics**

A screenshot of a computer

Description automatically generatedTo measure the effectiveness of my models, I will utilise either, MAPE, MSE or R^2, for neural networks the primary criterion will be L1 or MSE.

Looking at the feature correlation provided by **seaborn,** we can see that some features correlate very strongly with each other, these included, *Sensor ID, Liquid Critical Temperature (K), Liquid Boiling Temperature, Superheated and Supercooled.*

Throughout the model that I will utilise, I will experiment with removing these features to see if they have an impact on the performance of predications.

A screen shot of a computer

Description automatically generated**Outliers**

From a glance at the data, there appear to be some extreme outliers in the first column, **Tank Failure Pressure (bar),**

Box Plot for *Tank Failure Pressure:*

Looking the box plot, these extreme outliers start ~around 50.

To resolve this, the column was clipped from 0, 50, where outliers would make out at 50.

**A screen shot of a computer

Description automatically generated**

*After clipping outliers:*

The box plot now retains a much more normal distribution.

**NOTE:** This process was repeated for every column, however Tank failure pressure was the only feature that contained extreme outliers.

**Linear Model**

Looking at the correlation heatmap, the features have little correlation with variables other than themselves, as for the target variable (Target pressure), there was no obvious signs of correlation, this meant that fitting a Linear relationship was already going to be trivial.

However, proceeding with a basic implementation of Linear Regression with no regularization, a MAPE score of ~0.45 was retrieved, since it was the first model, we could not measure the effectiveness in comparison to other models.

**Decision Trees**

**Normalisation:**  Values were not scaled for Decision Trees, as applying scalers made little to no difference on decision-tree-like models, this was due to how features were split in trees.

Decision Trees are another type of model that can better fit more complex relationships between features and labels. With a regular decision tree implementation with a max depth of 10 to prevent overfitting, MAPE score was improved by ~0.08, with a new record of ~0.37

*Ensemble Methods*

**Random Forrest** is a bagging ensemble method for a decision tree, it creates multiple decision short trees based on random sections of data and utilises a voting classifier to provide a final output, with the parameters:

n\_estimators=500, random\_state=50, bootstrap=True, max\_depth=10)

MAPE score was improved by ~0.05 through the utilisation of Random Forrest.

**Gradient Boosting Tree** is a boosting ensemble method for a decision tree, it creates iterations of decision trees that are improved based on loss for a previous iteration. It does this by combining an ensemble of trees as weak learners to improve the final predication, in the case of regression it is usually an average of all leaners.

Starting off with default parameters, *GridSearchCV* was utilized to run parallel tests of individual parameters to see what would work the best with the given dataset.

After some tuning these parameters were found to provide the best results:

n\_estimators = 500  
max\_depth = 30  
min\_samples\_split = 10  
max\_features = 'sqrt'  
learning\_rate = 0.1  
min\_samples\_leaf = 10  
subsample = 0.5

To prevent overfitting, ‘shrinkage’ was implemented, by lowering learning rate below 1.0 and subsample values below 1.0.

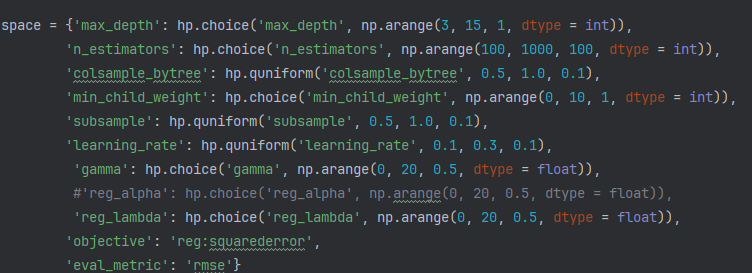
**NOTE:** Very little optimisation was used with models such as Decision Trees and GBTs as there would be better version of

**XGBOOST**

XGBoost is a popular implementation of GBDT, it is faster and grants better results due to its parallel processing capabilities and built in regularization parameters. This would be one of the two models I spend the most time optimizing due its popularity in predicating tabular data.

Since XGBoost is particularly more susceptible to outliers, clipping the first feature improved the model substantially.

To assist in optimising the tree, I used hyperopts **fmin** function, that attempts to optimise the model through Bayesian optimisation.



Initially using many parameters to feed into the algorithm, I implemented cross-validation into the returning score rather than the test set to avoid overfitting. Initially this worked relatively well, the lowest loss recorded was ~0.31. However, due to the large number of parameters that were being optimized at once, my trials began to plateau.

I also discovered that removing highly correlating features stated earlier would slightly improve model predications.

*GridSearchCV*

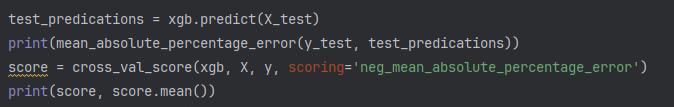
In attempt to further improve the model, I attempted to improve parameters one by one, following [this](https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/#XGBoost_Parameter_Tuning_With_Example) resource.

Starting off with:

1. Max Depth and Min Child Weight
2. Gamma (Regularization)
3. Subsample and Colsample Bytree
4. Alpha and Lambda Regularization

After optimising these params, I increased estimators to ~5000 and decreased learning rate to 0.01 and repeated the process of optimising params for further improvement of the model.

To validate these models, I would get the MAPE score from the test set, as well as the cross\_val\_score from the entire training set, this would give me an idea of how the model would perform with Kaggle predictions.



Where score.mean() would give me an idea of what my Kaggle score would look like.

**Tuning of Feature Engineering:**

The iteration of the model also **ran better** with the highly correlated features and rather than removing NA entries, they were replaced with the mean for the column.

‘ID’ was still removed as it caused overfitting.



These were params I settled on, for the time being due to increasing processing times as n\_estimators were increased and as learning\_rate was lowered.

**Neural Network**

To start off, I prepared the appropriate tensors for the training, validation, testing and Kaggle set.

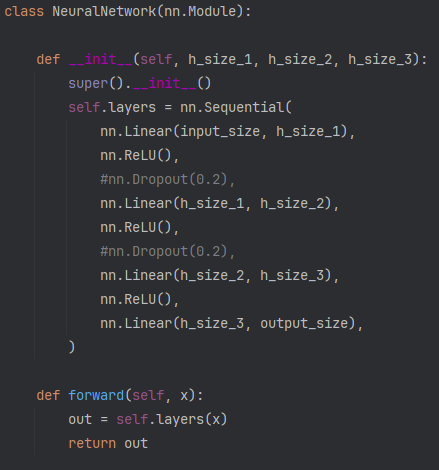
Later, when we utilised optimisation, training would be quite slow and cumbersome, so CUDA devices were attached to the tensors Aswell as the model to utilise my GPU for training and validating.

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

At first, I implemented a simple 2-layer neural network that was validated and improved based on the *test set* however, this caused major overfitting and poor performance on the Kaggle set.

To resolve this issue a 3-way split was created with the data, the train, test and validation set, where validation would assist in optimizing parameters.

To prevent overfitting, I also experimented with **2 dropout layers set to 20%,** I would routinely check if it had an impact on the validation set.



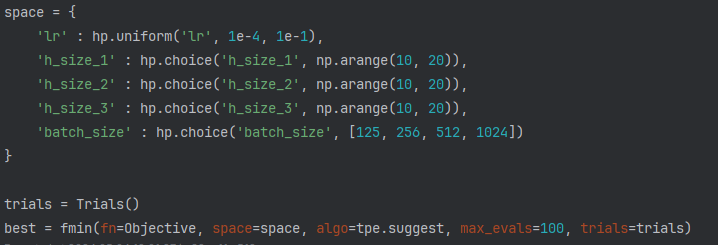
**NN Class Declaration**

This is what my NeuralNetwork would look like.

ReLu activation was chosen since it performed better than sigmoid and softplus.

**Optimisation**

To assist in optimizing the model, hyperopt was used:



For the time being, the only parameters I would optimise were the neuron counts in the 3 layers, the learning rate and the batch\_size for the Tensor Dataset.

Running hyperopt for a few hours, these were the params that provided the best result:



Which when run on my holdout set (test set), provided a MAPE of ~0.27, on par with my XGBoost model.

**Note:** Hyperopt optimised the NN with dropout enabled as to reduce overfitting on the final dataset.

**Conclusion**

Throughout the multiple Models tested, the two that provided the best results was XGBoost and Neural Networks.

Linear Regression performed poorly due to the complexity of the data, and due to the lack of time spent optimizing parameters, lack of implementation of L1 or L2 regulazation, this was due certain time constraints where I felt more time should be spent on models that may be able to fit the complexity of the dataset.

Similarly base decision trees were not a model I planned to improve much as there existed better ensemble methods such as **Random Forest** or **XGBOOST**.

**Neural Network** were cumbersome to train, taking up to ~40-50 minutes for a 100 eval tune if I decided to alter the NN class, for this reason it was not optimised to the data as well as I wanted it to.

**Difficulties**

Initially I have trouble with data fitting too well to the test set, as it was used to validate the data rather then used as a holdout. This was resolved with three ways split of data, or cross-fold validation on the training-set.