Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Harvey	Student ID:	20985954
Other name(s):	Phillip Yang		
Unit name:	Machine Learning	Unit ID:	COMP3010
Lecturer / unit coordinator:	Qilin Li	Tutor:	Donna
Date of submission:	5/4/2024	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Phillip Harvey Date of signature: 5/4/2024

(By submitting this form, you indicate that you agree with all the above text.)

1. Table of Contents

INTRODUCTION **3**

1. DATA CLEANING **3**

1.1 MISSING VALUES 3
1.2 OUTLIERS 3
1.3 DUPLICATES 4
1.4 INCORRECT VALUES 4

2. DATA PROCESSING **5**

2.1 ENCODING 5
2.2 NORMALISATION 5
2.3 FEATURE ENGINEERING 6
2.3.1 Event Number and Denoising *6*
2.3.2 Basic Feature Engineering *6*
2.3.3 Research *7*

3. MODEL DEVELOPMENT **8**

3.1 MODEL SELECTION 8
3.1.1 Linear Regression *8*
3.1.2 Support Vector Regression (SVR) *8*
3.1.3 Selected Models *9*
3.2 EVALUATION AND HYPERPARAMETER TUNING 9
3.2.1 Random Forest *9*

4. REFLECTION **12**

Introduction

We have been tasked with creating a model through Machine Learning that can be used for predicting the peak blast pressure caused by BLEVEs (“tgt_pressure”), given certain factors about the scenario. This report will detail the process of creating such a model.

1. Data Cleaning

Before being able to create a model, we must ensure that the data is free of errors that may hamper model development. The train data has many errors – detailed below – which are thankfully not present in the external test set, meaning we can filter these out to train our model, before using it to predict tgt_pressure in the external test set.

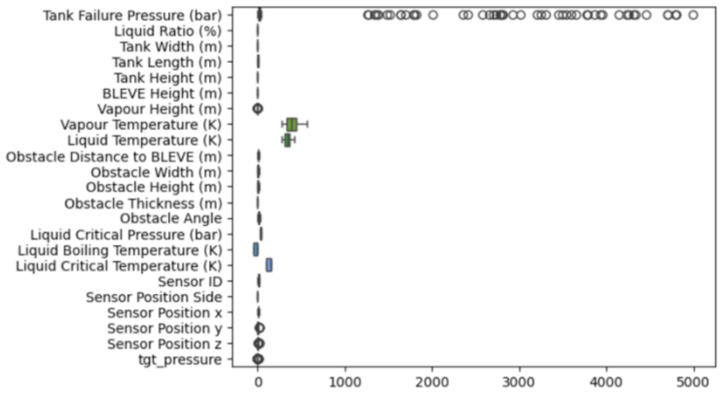
1.1 Missing Values

The train data contains multiple NaNs. To deal with this, we must either 1) remove them from the data, or 2) use an appropriate statistical method to eliminate them. 112 of the 10050 datapoints (1.11%) contain NaNs. Whilst it’s tempting to simply delete these due to apparent lack of significance, we can observe closer and see that only 10 of these contain 2 or more NaNs. These 10 rows contain exactly 20 NaN values, so we will decide to replace the single-NaN values with the median for that row and drop any row which contains more than 1 NaN.

We must be careful when applying this logic to categorical data. All categorical data in our dataset can be encoded as binary values, so whilst taking the median is still possible, we must consider the effect this has. This is further in section 2.1.

1.2 Outliers

When removing outliers, we are attempting to remove anomalous values from the data that are apparent errors. Importantly, we are not simply trying to remove the largest and smallest values past some threshold.



There is one main set of anomalies in the data, coming from the “Tank Failure Pressure (bar)” variable (see right). Digging deeper, we can see that a set of datapoints have values within the range [1000, 5000], whilst the remaining values are roughly between 4 and 40 – around 100x smaller. If we consider the fact that another popular unit of measurement for pressure, kPa, is 100x the value of bar, these “outlier” values appear to simply be measurement errors.

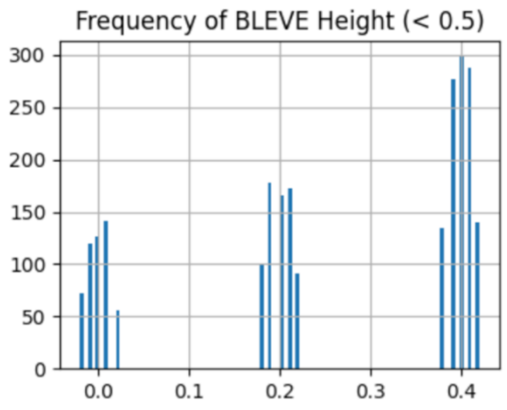
Given that these sorts of errors are present throughout the data (e.g. use of C instead of K in temperature) and the fact that the values for other variables in these "outlier" data points appear to be reasonable, it appears to be fair to assume that these values are simple measurement errors and should be divided by 100, rather than removed.

1.3 Duplicates

There are 50 sets of identical rows in the data. We will keep one of each set and remove the other – this is done through the df.drop_duplicates() method.

1.4 Incorrect Values

Next, we must consider values that work statistically, but don't make sense in real life. Note that this is done after some early data processing, as we can first remove noise and encode values, both of which makes this task more difficult.



Negative values do not make sense in many contexts, e.g. the dimensions of the tank. We should check that all negative values present in the data actually make sense. Initially, there are 4 negative values: "Sensor Position y", "Sensor Position z", "BLEVE Height (m)" and "Liquid Boiling Temperature (K)". These last 2 raise concern as 1) the height of the tank cannot be negative, and 2) 0K means absolute zero – no temperature should be below this. In regard to 1), we can see (right) that this is just a result of artificial noise – values appear to be generated according to some normal distribution about 0, 0.2, 0.4 etc., thus if we are able to denoise these first (2.2.1), we should be able to eliminate most of these negative values. After this, we are still left with roughly 25 negative values – due to the way that we denoised our data – which we will simply clip to 0.

As for point 2) negative Kelvin values should be impossible, and considering this is the boiling point, this means that it is impossible for this substance to exist in solid or liquid forms. Whilst one may be able to argue the existence of such a substance in a hypothetical scenario – where said substance is only able to exist in gaseous form – this argument makes no sense, as the explosions we're investigating are regarding BLEVEs, explosions which generate from the rapid vaporisation of liquid. Since these values are one of the few without any noise present with only 2 unique values, we can conclude that this is instead a measurement error. It is likely that the recorded temperatures (for both boiling and critical temperature) are supposed to be in ºC rather than K. This can also be validated through some research – if we consider the 2 unique sets of "liquid properties" (critical temperature, boiling point and critical pressure), we can match these up exactly with n-butane and propane, which is shown below in 2.1.

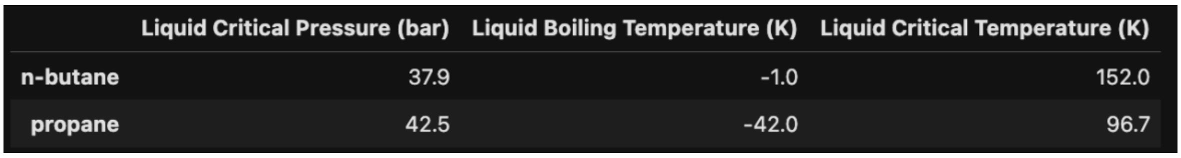
2. Data Processing

2.1 Encoding

As previously mentioned, our data contains categorical variables. The obvious example of this is “Status”, which has values “Superheated” and “Subcooled” – and some misspellings of these – however, there is also a less obvious example, the aforementioned “liquid properties”. Whilst these are technically numeric, there are only 2 unique values for each variable throughout the whole dataset, meaning that we choose if we want to use them as either numeric or categorical data (if not both).

To encode the status variable, we first need to view all the unique values in the dataset. All resemble either “Subcooled” or “Superheated”, and we can see that all values resembling “Subcooled” contain “Sub” or “sub”, whilst all values resembling “Superheated” contain “heat”, thus we will change all names to “Subcooled” or “Superheated” and give these a dummy variable instead – 1 for superheated and 2 for subcooled.

Next, if we view the raw data (without NaNs) for the liquid properties and remove duplicates, we can see that there are only 2 unique sets of values: [37.9,-1,152] and [42.5,-42,96.7]. Upon further research, we can determine that these values correspond exactly to that of n-butane and propane – given that we assume the given temperature actually in ºC instead of K – as is shown below. To add this information to our model, we will code each of the liquid properties with a “0” if it corresponds to n-butane, and “1” if it corresponds to propane, then we will create a new column, “substance”, which is equal to the product of all 3 (since all 3 values should be the same, taking the product should give us if it’s n-butane (0) or propane (1)).



	Liquid Critical Pressure (bar)	Liquid Boiling Temperature (K)	Liquid Critical Temperature (K)
n-butane	37.9	-1.0	152.0
propane	42.5	-42.0	96.7

Properties corresponding to n-butane and propane

Note that in replacing missing values with the median of a column, we inadvertently added some incorrect values to the liquid properties above (hence why we looked at the raw data without NaNs rather than our “cleaned” data). Whilst we could easily treat these separately, this can be dealt with whilst denoising, which is done in 2.2.1.

2.2 Normalisation

As each feature has its own dimensions, we will normalise each to having $\mu = 0$, $\sigma = 1$, as otherwise having so many unique dimension sizes would make it impossible to have an appropriate learning rate that isn’t too large for some variables and too small for others. However, some models that we will explore do not utilise a learning rate (e.g. random forest), and thus since our goal is to maximise MAPE, we will simply leave the data unscaled.

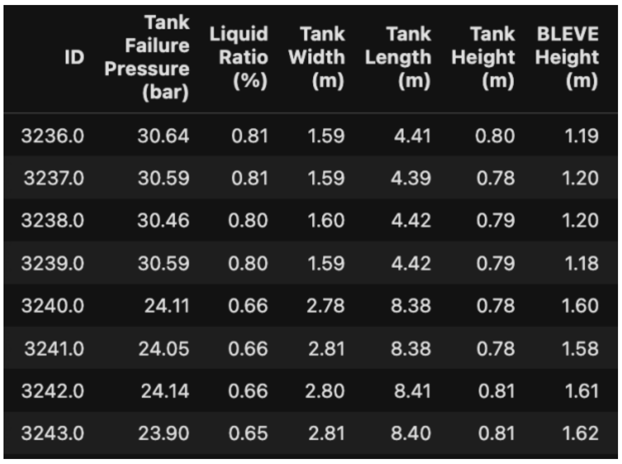
2.3 Feature Engineering

The feature engineering in this project was used both in processing the data (2.2.1) and enhancing the model (2.2.2+).

2.3.1 Event Number and Denoising

When looking at the data (sorted by "ID"), it becomes evident that data at different sensors from the same explosion is recorded sequentially (see below).

Whilst it is tempting to ignore this, we can utilise this fact to group similar data points together in attempt to de-noise the data. For a given event, constant values, such as the tank's dimensions, should remain constant, however, this is not the case, entries for a given event instead seem to be normally distributed about some value (e.g. 0.8, 1.6, 4.4). Whilst we could attempt to eyeball this number and logic to generalise this behaviour (e.g. we can see that the Tank Width column to the right has likely been generated about 1.60 and 2.80). However, we have no information about where this noise has come from and thus such an assumption of "nice numbers" is inappropriate, so we will instead use a statistical method, namely either mean or median. Whilst mean is preferred in many scenarios, we will instead use the median value as this is more robust to potential outliers whilst only having a minimal sacrifice to accuracy.



ID	Tank Failure Pressure (bar)	Liquid Ratio (%)	Tank Width (m)	Tank Length (m)	Tank Height (m)	BLEVE Height (m)
3236.0	30.64	0.81	1.59	4.41	0.80	1.19
3237.0	30.59	0.81	1.59	4.39	0.78	1.20
3238.0	30.46	0.80	1.60	4.42	0.79	1.20
3239.0	30.59	0.80	1.59	4.42	0.79	1.18
3240.0	24.11	0.66	2.78	8.38	0.78	1.60
3241.0	24.05	0.66	2.81	8.38	0.78	1.58
3242.0	24.14	0.66	2.80	8.41	0.81	1.61
3243.0	23.90	0.65	2.81	8.40	0.81	1.62

Data from two separate explosions ("events"), note the similarity between the top and bottom 4 rows

Note that this practice also addresses the inaccuracies in categorical data when we filled in our missing values earlier – we initially filled the value with the median for the column, which is either the correct or incorrect value. If the value is incorrect, we this can be seen as our 3 liquid properties will not have the same encoded value (e.g. [1,0,1] instead of [1,1,1]). However, given that there was only a maximum of 30 NaNs for these columns (with each instance having a maximum of 1 NaN), when we group different datapoints from the same explosion together and take the median, this will be corrected (as the majority of the remaining values will have [1,1,1]) and we are left with the desired output.

2.3.2 Basic Feature Engineering

When predicting tgt_pressure at a sensor in a particular explosion, the 2 main factors that must be considered are 1) the strength of the initial explosion and 2) the placement of the sensor. Variables relating to the tank or liquid features can be used to help predict the first point, whilst variables related to the sensor and object placement can be used for the second.

The placement of the sensor relative to the obstacle is very important in determining the tgt_pressure at that sensor. This can be done a few ways; we could create separate models for each sensor, train separate models for sensors by the side of the obstacle they're on or try to incorporate it as a factor in our model. All 3 methods were tried, and performance varied by model – regardless of this, there's a limited capacity for feature engineering based off the location on the obstacle. However, one extremely useful predictor would be to determine the distance of the sensor from the explosion – whilst obstacle distance has been given, this does not account for the fact that some sensors may be much closer to the explosion than others based on the obstacle angle or position relative to the BLEVE. Instead, a "net_sensor_dist" is calculated to describe the net distance between any given sensor and the BLEVE. However, physics describes the relationship between intensity and distance through the inverse square law, so we will instead consider the use of $\frac{1}{sensor\ distance}^2$ (named "1/dist_sq"). We will also test other distance features (e.g. inverse, inverse cubic) to confirm our choice.

When considering the strength of the initial explosion, we must first think about what actually causes the explosion. A BLEVE, or "boiling liquid expanding vapour explosion", occurs when a pressurised vessel containing a superheated liquid is ruptured, which leads to a rapid loss in pressure as the vapour in the container escapes, causing the superheated liquid to quickly vaporise – this sudden expansion in volume (to up to thousands of times its original volume) is what causes the observed explosion. Factors affecting the magnitude of this explosion would include things like the total volumes of the liquid and vapor (both combined – i.e. tank volume – and separate), the rate at which the vapor evaporates, how much more volume the liquid takes up (both in absolute terms and relative to the tank) and more. Whilst some of the more obvious relationships can be determined easily through the data (e.g. tank_volume, vapour_volume), some of these factors are much more difficult to determine and need to be researched.

2.3.3 Research

Much of the time on this project was spent reading various papers (both by Qilin[1] and other authors) in effort to help with the feature engineering of the project, particularly regarding the strength of the initial explosion. Unfortunately this effort was mostly unsuccessful, having little in the way of results to show for it.

Most of this effort centred around finding an effective volume at explosion. This started off with utilising the liquid correction method and finding the flashing fraction f, which required the determination of both H_v and c_p, with c_p requiring linear interpolation based on data obtained from NIST for both propane and butane. Then, the net volume is calculated as $V_{net} = V_{vapor} + V_{liquid} * \frac{\rho_l}{\rho_v}$ (from Modelling and Understanding BLEVEs[2], 22.15), with ρ again requiring interpolation based off of values obtained through NIST, however, these produced absurdly large values which did not work well as a predictor for "tgt_pressure" (as seen in 3D plots trying to predict "tgt_pressure" with 1/dist_sq as the other axis, along with simulations).

One feature that ended up working well was the energy released in the expansion of vapour (reference in the same paper). This models the released energy through $E_v = 10^2 * \left(\frac{P \cdot V}{\gamma - 1}\right) \cdot \left(1 - \left(\frac{P_\alpha}{P}\right)^{\frac{\gamma-1}{\gamma}}\right)$. Using initial vapor volume as the value of vapor in this model worked ok as a prediction, however, using net vapor volume – initial vapor volume (“vapor_vol”) + vapor produced through flashing of liquid – unfortunately did not work as well. This was observed through plots and correlations between itself and tgt_pressure, as well as its performance as a predictor in models instead of E_v calculated with vapor_vol (which was deemed to be a suitable predictor due to having a stronger correlation, and a decent improvement in performance when used over correlated predictors such as net vapour volume and vapor_vol.

3. Model Development

3.1 Model Selection

The main types of models considered for this assignment were linear regression, SVR, random forest, XGBoost and Neural Networks. In attempt to prevent data leakage and improve the accuracy of local metric predictions, a hold-out style validation was used throughout. There was experimentation with cross validation and different train/validation/test splits, but ultimately the models that seemed to perform the best (based on both local and submission accuracy) were hold-out validation methods with an approx. 80/10/10 split – the final choice was an 80/12/8 split as this also allowed for a maximisation of training data but allowed for each model iteration to be evaluated on slightly more data during train. Additionally, data was split by “event_num”, keeping measurements from the same explosion together, to try and prevent overfitting (we don’t want to train and validate/test on similar data).

3.1.1 Linear Regression

The first type of model considered was linear regression, in attempt to establish a baseline for the other models to go off of. This was done using in-built Python packages for regression, then implementing a function that would utilise forward-selection based on AIC, R^2 and MAPE along with user-inputs and decisions to create a model for all data. This initially performed terribly, with MAPE scores above 1. This eventually was reduced through separating data by the side of the obstacle they were located on, before being reduced further by creating models specific to each sensor (using the same predictors for similar sensors), however unsurprisingly, evaluation metrics were still horrible. Linear models were discontinued shortly after – whilst these could have been optimised further, modelling the explosion is a complicated, non-linear process, meaning that a linear model will not do a good job, even after potentially exploring extreme parameter optimisation and multilinear regression.

3.1.2 Support Vector Regression (SVR)

SVR was considered, but not implemented for this assignment. This was due to the apparent downsides – such as computational expense and having to choose a Kernel function, but the main reason is the fact that it performs worse when data is not easily separable. Given the fact that we had to de-noise the data ourselves and the numerous factors present in the data, it is unlikely that the data will be easily and nicely separable, and thus alternates that appeared to be more suitable were pursued instead.

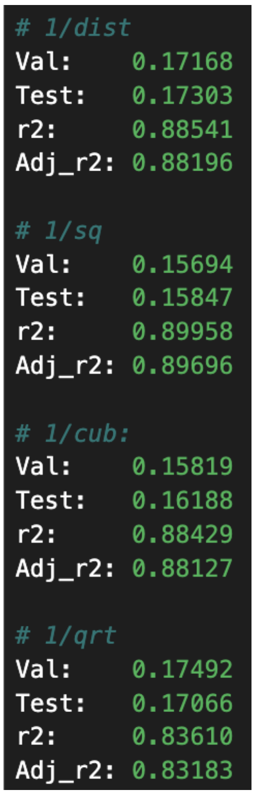
3.1.3 Selected Models

note: Qilin approved of the use of both RF and XGB)

The models selected for this exercise are Random Forest, XGBoost and Neural Network. This was due to their upside in dealing with tabular data along with their ability to model non-linear relationships predictions given numerous variables. Random Forest was utilised as more of a "baseline" metric of what our model should be capable of (given that it is an example of a bagging ensemble method), whilst the choice of neural networks was more optimistic, as these can capture incredibly complicated relationships, but generally require a comparatively large dataset to produce optimal results. On the other hand, XGBoost was seen as a good compromise between these 2 – being a boosting model, it can improve its accuracy quite well through iterations, whilst not requiring as much data as a neural network to work well.

3.2 Evaluation and Hyperparameter Tuning

Many of the decisions made through the hyperparameter tuning process were done so through informed guesses, before making a choice based off the results from given metrics. For example, the choice of what predictors to use was made through thinking about what combination of predictions would be relevant, before running simulations with multiple sets (using either more, less or different predictors than those that were initially hypothesised as being best) to determine which set produced the best results.



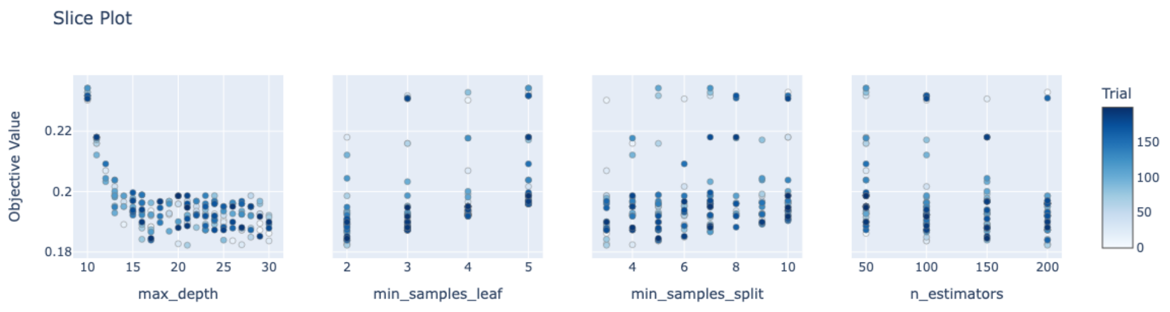
```
# 1/dist
Val:    0.17168
Test:   0.17303
r2:     0.88541
Adj_r2: 0.88196

# 1/sq
Val:    0.15694
Test:   0.15847
r2:     0.89958
Adj_r2: 0.89696

# 1/cub:
Val:    0.15819
Test:   0.16188
r2:     0.88429
Adj_r2: 0.88127

# 1/qrt
Val:    0.17492
Test:   0.17066
r2:     0.83610
Adj_r2: 0.83183
```

Distance metric selection example

Part of this decision regarding which predictors may work best came down to correlation between predictors – where predictors had a high degree of correlation between each other, only 1 was chosen to be included in the model to avoid multicollinearity and overfitting. For example, it was evident early on that including a metric for the distance between the BLEVE and the sensor would be an important feature of the model, however the choice of whether to use the pure distance or $\frac{1}{dist}, \frac{1}{dist}^{2}, \frac{1}{dist}^{3}$ etc. was not as clear. Online research did not produce a definitive result on which metric would be best but based off prior knowledge of the inverse square law, it was predicted that $\frac{1}{dist}^{2}$ would be the best measure to use. This was tested through trial and error (see right) – initially hyperparameter optimisation was conducted on each through Optuna to get a consensus for the best hyperparameters to use for the model (XGBoost), before fitting 4 models with these given hyperparameters, 1 for each distance metric. We can see that the difference is relatively marginal, but it appears that both MAPE and R^2 were best with a predictor of $\frac{1}{dist^2}$, as was predicted.

3.2.1 Random Forest

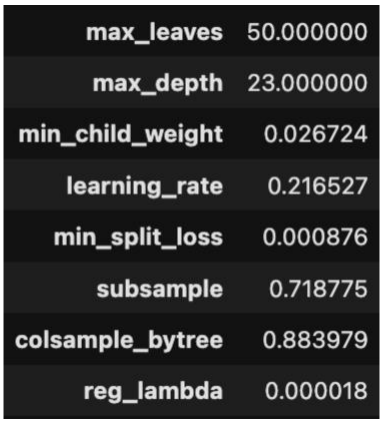
Hyperparameter tuning for individual models was initially tone through GridSearchCV. However, due to the long computational time, as well as the limited information gained, this was switched to a Bayesian optimisation method through the Optuna package.



Example hyperparam tuning with Optuna

With Optuna, we can feed in a range of values, and let Optuna perform Bayesian optimisation on our parameters. Through some of Optuna's visualisation tools, we can observe the trials through time, the relationship between our hyperparams, the performance and more. An example of this is shown above – when tuning our random forest model, we use Optuna to help select the values of 1 of 4 variables. The decision we need to make is based on 1) what parameters should we tune and 2) what range of values do we want to use. The approach used throughout this project was to use all suitable parameters for a given function, and observe if they were significant and what hyperparameter values they ended up suggesting. This was iterated through many times, so we could obtain the best hyperparams for the given function. E.g. above, we can see that max_depth values below roughly 14 lead to a higher score, so we will adjust our parameters to take values between 14 and 30 for max_depth, 1-4 for min_samples_leaf, 2-5 for min_samples_split and [150,200,250] for n_estimators.

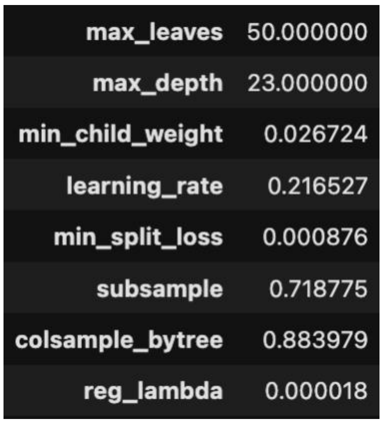
Additionally, we would swap between the transformed (normalised and quantised) dataset with the untransformed set and see which one produced the best results. In the case of random forests, these came when the data was untransformed – this shouldn't come as much of a surprise given that one of the main advantages of transforming the data is creating the same scales across different parameters to make the learning rate applicable to the whole dataset, however, random forest doesn't utilise a learning rate and hence this is unnecessary.



max_leaves	50.000000
max_depth	23.000000
min_child_weight	0.026724
learning_rate	0.216527
min_split_loss	0.000876
subsample	0.718775
colsample_bytree	0.883979
reg_lambda	0.000018

Random Forest Hyperparams

3.2.2 XGBoost



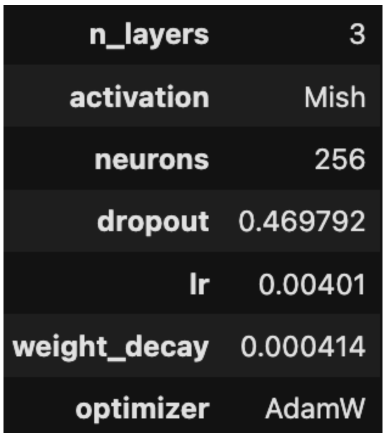
max_leaves	50.000000
max_depth	23.000000
min_child_weight	0.026724
learning_rate	0.216527
min_split_loss	0.000876
subsample	0.718775
colsample_bytree	0.883979
reg_lambda	0.000018

XGBoost Hyperparams

Our best model ended up being XGBoost. The hyperparameters tuned were against inspired by those shown in Qilin's paper – we started off with these values, before narrowing it down to the list seen on the right. We also experimented heavily with the right mix of predictors to use – as discussed above, we tried using variables that we thought would be relevant, then more variables (even if they were correlated) and less variables. Rather surprisingly, set with more variables performed (a total of 20) appeared to perform the best out-of-sample, and thus it was used for this model – it also appeared to be the best parameters for the other 2 models, hence was also used for those.

The final predictions were obtained through this model – transforming the external test data, training a model based on the full in-sample data, making predictions on this external data and then transforming these back to the original scale using our quantile transformation.

3.2.3 Neural Network



n_layers	3
activation	Mish
neurons	256
dropout	0.469792
lr	0.00401
weight_decay	0.000414
optimizer	AdamW

Neural Network Hyperparams

There were many more hyperparameters to train in the neural network model than in previous models, namely the number of layers, number of neurons to be used in each layer, type of activation function to use, dropout rate, learning rate, weight decay and the type of optimiser to be used. Initially, the ranges from Table 5 in Qilin's paper were entered into input for Bayesian optimisation through Optuna, before being narrowed down over time to give the eventual best set of parameters. Rather surprisingly, the unscaled data also seemed to perform best in this set.

4. Reflection

Overall, this project helped me understand the utility and capability of machine learning models for real-world use. It was nice to see how well models could perform simply given input data without much direction otherwise, with minimal knowledge of any complicated physics and chemistry equations.

I am a little unsatisfied with how my project ended up – more than half my time went into researching feature engineering to try and get better predictors, but it appears that I have made a mistake in my implementation of this, or have missed its significance, as I was unable to utilise a factor that accurately describes the vapor produced from the actual BLEVE process (instead only using the vapour already in the tank). This obsession over finding good features meant that much less thought was put into model production and evaluation than I would've liked, meaning that 1) my current models are likely not the best that they could be (even given the parameters I have access to), and 2) I didn't have sufficient time to explore transformer models, which appear to be the best tool to use, according to Qilin's paper.

Additionally, there appeared to be clear data leakage from my in-sample test set. This became apparent when my predicted MAPEs would be extremely low, even on the test set (normally below 0.2), but when I would use this to predict on the external set, these scores would be 5-10% higher. Despite my best efforts, I could not determine where this leakage was coming from – I went through the pre-processing items again, and tried to ensure that the test set was kept completely separate from everything else (and not used when applying a data-wide calculations, such as standardisation), but to no avail.

The project was overall challenging admittedly enjoyable – it was nice to be able to get experience with actual using these algorithms for a data-science purpose, which has a lot of real-world applications.

5. References

[1] Qilin Li, Yang Wang, Yanda Shao, Ling Li, and Hong Hao. A comparative study on the most effective machine learning model for blast loading prediction: From gbdt to transformer. Engineering Structures, 276:115310, 2023. https://www.sciencedirect.com/science/article/pii/S0141029622013864

[2] J. Casal, J. Arnaldos, H. Montiel, E. Planas-Cuchi, and J. A. Vilchez "Chapter 22 Modeling and Understanding Bleves." *Yumpu.Com*, www.yumpu.com/en/document/read/4893393/chapter-22-modeling-and-understanding-bleves