BLEVE TARGET PRESSURE PREDICTION

Kaggle name: bruceomondi

Final Kaggle score: 0.28914

Introduction

Boiling Liquid Expanding Vapour Explosions (BLEVEs) induced by Liquified Petroleum Gas (LPG) or Liquified Natural Gas (LNG) can occur during processing and storage or during transportation and can cause damages to different civilian structures, e.g., tunnels and highways, buildings, bridges, or onshore/offshore facilities (Bariha et al., 2017, Li and Hao, 2021) [2][4]. Being able to predict the pressure of these blasts could help us better prepare for and mitigate such disasters from occurring. This report aims to predict the pressure from blast waves of Boiling Liquid Expanding Vapour Explosions (BLEVEs) using different machine learning models – random forest, support vector regressor, neural networks and XGBoost.

Methodology

This report will walk you through the different steps involved in preprocessing and cleaning the data, some exploratory data analysis (EDA), feature engineering, model training, predictions using the models, and saving the output to CSV files.

I first imported the necessary libraries and added any relevant ones as I progressed. I then opened the data files ('train.csv' & 'test.csv') and checked their structures including the features, data types, and number of missing values.

Missing values

The training data had missing values for all each features:

Features	Missing values
Liquid Critical Temperature (K)	30
Liquid Critical Pressure (bar)	30
Liquid Boiling Temperature (K)	29
Vapour Temperature (K)	28
Liquid Temperature (K)	27
Target Pressure (bar)	10
BLEVE Height (m)	10
Sensor Position z	9
Vapour Height (m)	9
Tank Length (m)	9
Tank Width (m)	9
Liquid Ratio (%)	9
Tank Height (m)	8
Obstacle Distance to BLEVE (m)	8
Sensor Position y	8
Obstacle Angle	8
Sensor ID	8
Sensor Position Side	8
Tank Failure Pressure (bar)	7
Obstacle Thickness (m)	7
Status	7

Sensor Position x	7
Obstacle Height (m)	6
Obstacle Width (m)	6
ID	5

Compared to 10050 total entries, these were relatively few missing entries, so I decided to impute them using SimpleImputer library and with their respective medians since the data is not normally distributed and median is also more robust to outliers as compared to the mean.

I also checked the test dataset and there were no missing values, so I left it as it was.

Data types

'Status' was the only feature with categorical data and due to some data entry mistakes, there were 9 distinct categories instead of the only 2 that were required – Subcooled & Superheated.

Training data:

Status	
Subcooled	**6285**
Superheated	**3683**
Subcool	**23**
subcooled	**20**
Subcoled	**14**
Superheat	**8**
superheated	**6**
Saperheated	**4**

I merged these categories to ensure that I had only the 2 that I needed.

Status	
Subcooled	**6367**
Superheated	**3683**

The test set had no issues as the categories were clearly marked.

Status	
Subcooled	**1890**

Superheated	**1313**

The ‘Status’ feature was useful in our modelling, so I converted it to numerical values using dummy variables, which essentially binarizes the data, i.e., if the entry status is Superheated = 1, if entry status is Subcooled = 0.

Duplicate entries

I then checked for duplicate entries in the data and dropped them as we do not the extra set of the same entries. This could have occurred due to errors in data collection. I filtered by ID and Sensor ID to make it easier to spot the duplicate entries. The training data had 50 duplicate entries which were all dropped, leaving 10000 entries.

```
There are 50 duplicate entries
# After removing the duplicate entries
(10000, 26)
```

The test set did not have any duplicate entries.

```
There are 0 duplicate entries
(3203, 25)
```

Outlier detection and analysis

I plotted the boxplots for all features and included a function to list all the outliers in the plot, to which I found the following outliers:

- Tank Failure Pressure (bar) – 50 extreme value outliers
- Vapor Height (m) – 71 regular outliers
- Sensor position (Y) – 10 regular outliers
- Sensor position (Z) – 91 regular outliers

The Tank Failure Pressure feature had some very extreme values (compared to its median – 22.96) which could potentially affect the model.

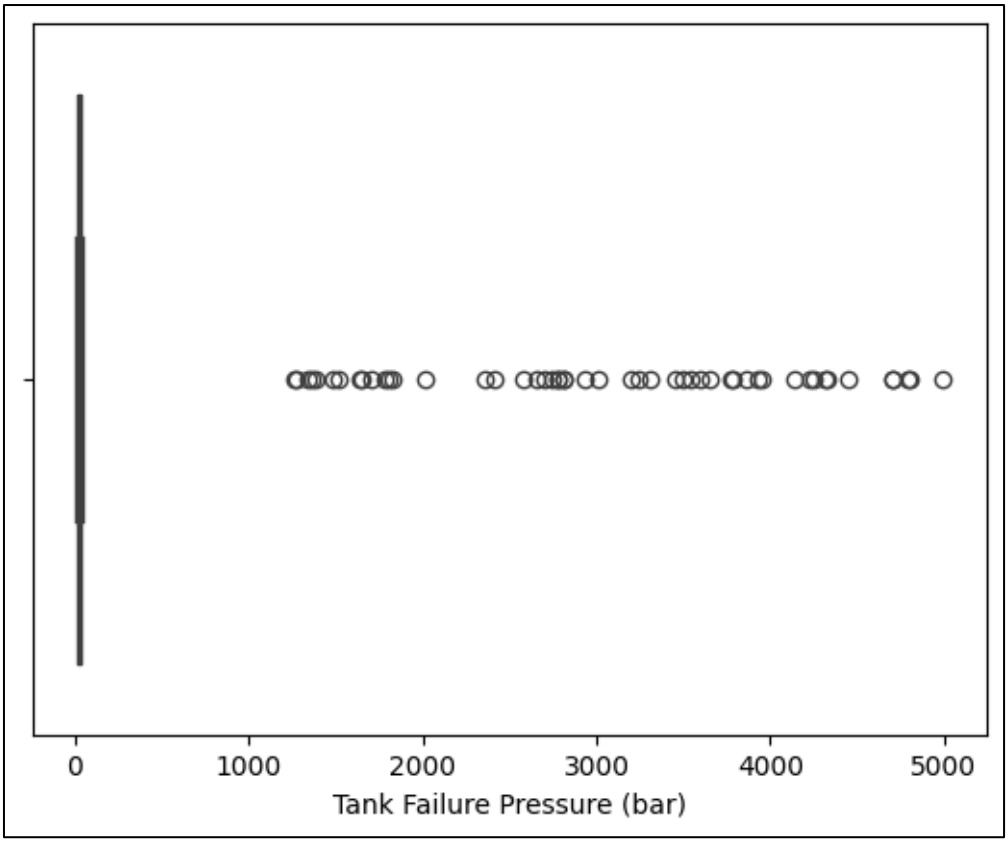

Fig: Tank Failure Pressure boxplot. You can see how extreme the outliers were

So, using the formula I had defined for outlier detection, I dropped all the 50 extreme value outliers in the feature in the training data. Just in case there were any outliers in the test, I did the same thing with the same set of outliers to make sure the test set was rid of outliers.

EDA

Correlation

To get a better understanding of how the features in the training data correlated to each other, I plotted the correlation matrix. At first it was difficult to read the plot due to the number of features, so I plotted using a threshold for features with more than 0.70 correlation [3], which would put them at risk of multicollinearity. I found that ID, Sensor ID, Liquid Boiling Temperature (K), Obstacle Distance to BLEVE (m), and Liquid Critical Pressure (bar) had very high correlations and made the decision to drop them from both the training and test sets, to mitigate the impact of multicollinearity in the model.

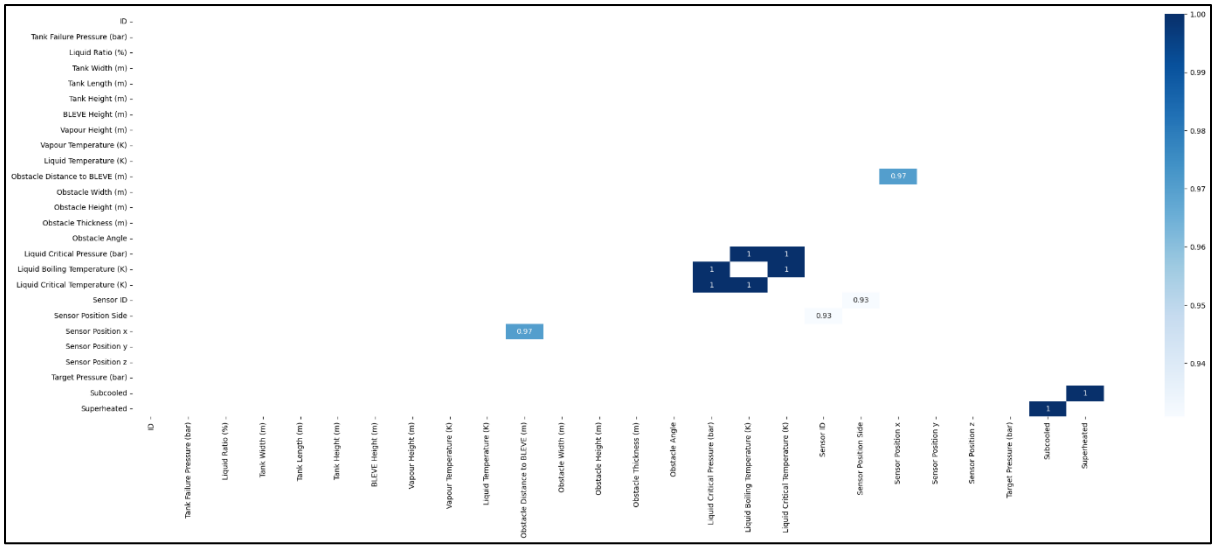

Fig: Correlation matrix with threshold of 0.7 correlation or higher.

(*The diagram is as legible here, so please refer to my Jupyter notebook if required*)

Feature engineering

As soon as I saw the variables in the dataset, I identified at least 3 features that I could engineer from the given variables: obstacle volume, tank volume, and the ratio of vapor to liquid temperature. To make this possible, I first initialised the baseline and the engineered data, then calculated their respective MAE scores to see how the engineered data fits on a Random Forest regressor, then I assigned the engineered data to both the training and test datasets [1]. After trying these steps for a couple runs, I came up with the following features:

- **Vapour Liquid Temp Ratio** = Liquid Temp (K) / Vapour Temp (K)
- **Obstacle Volume (KL)** = Obstacle Height (m) * Obstacle Thickness (m) * Obstacle Length (m)
- **Tank Volume (KL)** = Tank Length (m) * Tank Width (m) * Tank Height (m)

(*where K=Kelvin, m=meters, KL=Kilolitres*)

After performing feature engineering, I dropped the original features to avoid redundancy in the datasets.

Model creation & training

For tasks where the goal is to predict the outcome variable, it is common practice to use regression models such as the ones I chose for this task – Support Vector regressor, Random Forest regressor, a Sequential Neural Network, and XGBoost Regressor. All of these models (1) handle different data types and data complexities well, (2) can work with large and high dimensional data, (3) are not as restrictive with assumptions like other regression models, i.e., linear models. Individually:

SVR :

- good at handling complex data patterns with the use of kernels
- better interpretability due to support vectors

Random Forest:

- efficiently trains on large datasets
- can better show which features contribute most to predictions

Sequential NN:

- can handle many features with high dimensionality well
- good at detecting intricate patterns in large datasets

XG Boost:

- has in-built regularization which is good for preventing overfitting
- in-built measures to handle missing values by imputation during training
- fast and can work well with large datasets, making it more scalable

I first defined the 'model_eval' function containing the metrics that I would use to assess model performance: MAPE and R^2.

Then for each model, I first initialised the data to be used for training, initialised the standard scaler which I fitted on the training data, then used the scaler to transform both the training and test data.

Scaling helps to make sure all features have the same scale, which helps improve model performance on the data. It ensures that features contribute equally to distance calculations, leading to better model performance for distance-based calculation models that rely heavily on distances between data points for classification or regression, e.g., Support Vector regressor.

I then define hyperparameters for all the models, search for the best parameters for the model using RandomizedSearchCV [5], compare the model's performance with base parameters against the best parameters, and finally pick the best performing model use it to make predictions on the test data.

Support Vector regressor

I defined the hyperparameters and their grid as follows:

```
# checking and tuning hyperparameters for support vector regressor
kernel = ['linear', 'rbf', 'poly'] # define the kernel
C = [1.5, 10]
gamma = [1e-7, 1e-4]
epsilon = [0.1,0.2,0.5,0.3]


SVR_grid ={'kernel': kernel,
           'C': C,
           'gamma': gamma,
           'epsilon': epsilon}
```

RandomizedSearchCV gave me the following list of best parameters:

```
{'kernel': 'linear', 'gamma': 1e-07, 'epsilon': 0.5, 'C': 1.5}
```

And model performance was as follows:

- **MAPE score:** 0.43881967714138526
- **R^2 value:** 0.6981871662235553

I was able to make predictions and saved them under a 'SVRoutput.csv' file.

Random Forest regressor

I defined the hyperparameters and their grid as follows:

```
# next thing is defining and tuning hyperparameters
max_features = ['auto','sqrt']  # how many features at each split
estimators = [int(x) for x in np.linspace(start = 200, stop =3000, num
= 10)] #no of trees in the forest

maxidepth = [int(x) for x in np.linspace(10,200, num=10)] # max depth
of each tree
maxidepth.append(None)

min_samples_split = [2,5,8,10]  #min no of samples required to split
node
min_samples_leaf = [1,2,3,4]  #min no.of samples needed to be a leaf in
the tree
bootstrap = [True, False]

#all the hyperparameters
RFgrid = {
            'max_features': max_features,
            'n_estimators': estimators,
            'max_depth': maxidepth,
            'min_samples_split': min_samples_split,
            'min_samples_leaf': min_samples_leaf,
            'bootstrap':bootstrap

          }
```

RandomizedSearchCV gave me the following list of best parameters:

```
{'n_estimators': 200,
 'min_samples_split': 8,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 52,
 'bootstrap': True}
```

And model performance with the best parameters was as follows:

- **MAPE score:** 0.1739233181200429
- **R^2 value:** 0.923272045004485

Performed much better than SVR (much lower MAPE score, much higher R^2 score)

I made predictions on the test set using this model and saved them to ‘RFoutput.csv’.

Neural Networks

I defined the hyperparameters and optimizer, loss function and earlyStopping criteria as follows:

```
# defining hyperparameters
# create a Sequential neural network
nw_model = Sequential([
      Dense(64, activation='relu',
input_shape=(nwtrain_X.shape[1],)),  #first layer with 64 neurons
      Dense(128, activation='relu'), # second hidden layer with 128
neurons
      Dropout(0.2), # 20% drop
      Dense(64,activation='relu'),
      Dense(1)  # output layer
])

# define the optimizer for training
optimizer = RMSprop(learning_rate=0.001)

# compile the model with loss function
nw_model.compile(loss='mean_absolute_percentage_error',optimizer=optimi
zer,metrics=['mae'])
nw_model.summary()

earlyStopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)  #early stopping callback
# training the model using train_set
nw_model.fit(nwtrain_X, nwtrain_Y, epochs=50, batch_size=32,
validation_split = 0.2, callbacks=[earlyStopping])
```

I tried to search for best parameters for the model, but I kept getting an error with the estimator for RandomSearch. I modified my code several times but I still could not get it to work, so I used the base model to make the prediction.

The performance, as a result, was as follows:

- **MAPE score:** 0.22914516721904624
- **R^2 value:** 0.6792089957681263

The Neural Network performed worse than the Random Forest model, having a higher MAPE score, and only accounting for 67.9% of the variability in Target pressure values as shown by the R^2 value (0.679).

XGBoost

My final model was XGBoost. I defined the parameter grid as follows:

```
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV

# Define hyperparameters for tuning
xgb_grid = {
    'n_estimators': [100, 500, 1000],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7, 10],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0]
}
```

RandomizedSearchCV gave me the following list of best parameters:

```
{'subsample': 0.7,
 'n_estimators': 1000,
 'max_depth': 7,
 'learning_rate': 0.01,
 'colsample_bytree': 0.8}
```

The model with this set of best parameters performed as follows:

- **MAPE score:** 0.12487389766939787
- **R^2 value:** 0.9861578680383243

This was the best performing model with the lowest MAPE score and highest R^2 value of the 4, which means it explains 98% of the variability in the Target pressure using the explanatory variables in the data.

Conclusion

I eventually selected the XGBoost regressor as my final model and used that to make predictions for the target pressure for the BLEVE. Support Vector regressor performed decently but it was nowhere near as accurate as the Random Forest model, and the Neural Network setup with base parameters came in third. In the end, I think XGBoost is best suited to predict the target pressure of the BLEVE and could possibly perform even better with some more hyperparameter tuning. It is important to also make sure to avoid overfitting through hyperparameter tuning. In future, I could potentially look into expanding the set of hyperparameters for the different models, especially the XGBoost to see if it can perform even better. I could also consider ensemble methods, which possess the same robust features as the models I chose and could potentially perform better due to their ability to compile the strengths of different models and ability to tackle overfitting, handle missing data on its own, address non-linearity, include categorical features, among other benefits.

References

[1] A Banerjee, "*What is Feature Engineering in Machine Learning and why do we need it?*" Python in Plain English. Accessed: (May 2024). [Online]. Available: https://python.plainenglish.io/what-is-feature-engineering-in-machine-learning-and-why-do-we-need-it-360ec43353a0

[2] J. Li, H. Hao. "*Numerical simulation of medium to large scale BLEVE and the prediction of BLEVE's blast wave in obstructed environment*" Process Saf. Environ.Prot., 145 (2021), pp. 94-109. https://doi.org/10.1016/j.psep.2019.05.019

[3] Mivhashbrown, "*Heatmap for datasets with large number of numerical columns."* Kaggle. Accessed: (April.,2024).[Online]. Available: https://www.kaggle.com/discussions/general/210058

[4] N. Bariha, V.C. Srivastava, I.M. Mishra. "*Theoretical and experimental studies on hazard analysis of LPG/LNG release: a review*" Rev. Chem. Eng., 33 (2017) pp. 387-432. https://doi.org/10.1515/revce-2016-0006

[5] W Keohrsen, "*Hyperparameter Tuning the Random Forest in Python."* Towards Data Science. Accessed: (May.,2024).[Online]. Available: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74