COMP3010 Machine Learning
Assignment One Report

S1 2024
Peak Pressure caused by BLEVEs

Student Name: Zhenqi Zhang

Student ID: 20080833

Due Date: 5th of May 2024

Kaggle Name: Beiii

Document version: 1.1 (2015-11-15)

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.

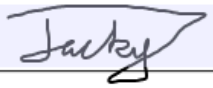
Last name:	Zhang	Student ID:	20080833
Other name(s):			
Unit name:	Machine Learning	Unit ID:	COMP3010
Lecturer / unit coordinator:	Qilin	Tutor:	Qilin
Date of submission:	05/05/2024	Which assignment?	(Leave blank if the unit has only one assignment.)

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:  Date of signature: 05/05/2024

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

Table of Contents

Introduction

This report is intended to document and explain the machine learning techniques and methods used during this project. This report will include initial data cleaning, data preprocessing, model selection, model training, model evaluation and the analysis of predictions given by different models.

The intention of this project is not only to record the predictions given by the model, but also to conduct high-dimensional data prediction, optimisation and comparison by learning and exploring different machine-learning models and techniques. During my studies, I learned three different types of models: Regression Model (SVR), Gradient Boosting Decision Tree (LightGBM) and Neural Network (MLP and RNN). I also compared different models and assembled them to make the best prediction for the given dataset.

Data Cleaning

According to the Machine Learning for Dummies book by Mueller and Massaron (2021), the first step in data processing for machine learning models is to gather and clean up the data. During this period, I first uploaded the train.csv and test.csv to the Jupyter Notebook and had them ready for further data clearing.

After gathering the data, the next step is to clean the data. In this part of the project, I performed the following cleaning techniques on my data:

Missing data -
For the missing data, I first checked for the number of features that were missing in each row. If a row turns out to be missing most of the features, then that row will be dropped entirely. The reason for dropping that row is that if a row is missing most of the features, then no matter how well we fill in the missing value, it still won't give us the best result and will thus affect the prediction of our model. The rows with only a few missing features will be explained in the preprocessing part of this report.

Removing Duplicates -
For the machine learning models, having duplicates could cause the model to be skewed. For our dataset, the duplicates can be easily identified with the 'ID', so I first dropped all the rows of duplicates for the 'ID' column.

Data Type Conversion -
In the given dataset, the 'Status' feature is the only feature that has a different data type (String) to all other features (float). After looking into the 'Status' feature, I realised that the 'Status' is simply a categorical feature, the value is either Superheated or Subcooled, but it contains a number of spelling errors. Since the 'Status' feature is basically a categorical feature with only two unique values, I decided to fix all the spelling errors first, changing [Superheated, superheated, Superheat and Saperheated] into superheated and changing [Subcooled, Subcool and Subcoled] to subcooled to make sure that the feature only has two unique features. After fixing the spelling mistakes, I converted the superheated to 0 and subcooled to 1 which is helpful for the further steps in model training to make the processing more efficient and decrease the model complexity.

Splitting the Target Value out of the Training Dataset -
Since I have done all the steps that involve removing rows, I decided to split the Target Value out of the training dataset so I don't have to worry about it later when I preprocess the data.

Data Processing

After the data gathering and cleaning, the next step is to preprocess the dataset so that it can be passed to the machine learning models to make the predictions. In the data processing part, I have done the following steps:

Handling Missing Values with K-Nearest Neighbour - ("sklearn.impute.KNNImputer," n.d.)
For the rows with only a few features that are missing, I chose to use the K-Nearest Neighbour Imputer technique to fill in the missing values. The reason that I decided to use this technique is that it has the ability to find adjacent averages in the training set and fill in the missing value based on the nearest neighbours. This is useful for datasets that are not randomly distributed and can be helpful for model training to capture the patterns. Compared to the KNN Imputer, methods such as replacing with the mean or most frequent value are still helpful, but that will make it hard for the machine learning models to capture the patterns within those features.

Handle Skew Data - ("Sklearn.Preprocessing.PowerTransformer," n.d.)
For skew data, which is where the features were not distributed normally around the mean, most of the data was either too far to the left or right of the scale. Without handling these skewed data, the model might be affected by bias and thus affect the performance and accuracy of the prediction. To address these skewed data, I decided to use the Yeo-Johnson technique, as it can effectively transform the feature to a much better (normal) distribution. The reason that I chose to use the Yeo-Johnson technique is that it is capable of dealing with both positive and negative values within data.

Handle Outliers - ("6.3. Preprocessing Data," n.d.)
For the outlier within the dataset, I decided to use the RobustScaler technique, as it can very effectively scale the outlier to reduce the effect it has on machine learning model training. During the process, the outliers are subtracted from the medium of the features and then divided by the IQR of the features, which then gives a scaled value for the outlier. So, the data near the 25% percentile will be scaled to around -0.5, and data near the 75% percentile will be scaled to around +0.5. An extreme outlier, for example, an extreme positive outlier, might be scaled to around +5.5 to minimise its effect. With the use of RobustScaler, the effect of the outliers would be significantly reduced, and compared to other techniques like removing the outlier, RobustScaler helps keep the patterns there for the model during the training but minimises the effect of the outlier.

Scaling the Data for Model Training -

This part of preprocessing basically just scales the dataset so that all the values are scaled into a range to ensure that there are no features that will significantly impact the model training just because they have higher values. I decided to use MinMaxScaler to keep the feature space balance within the range of -1 and 1, helping to enhance the performance of the models I train.

Splitting the Training Dataset into Train Data and Validation Data -

This part simply splits the training dataset into 75% training and 25% validating so that we can evaluate the model's performance after the training and apply more techniques, such as hyperparameter tuning to the model, which then helps to provide the best model to predict for the test dataset.

Model Selection

When deciding on a model, the first thing that needs to be determined is whether the model used can effectively understand the data, cope with the complexity of the task, and achieve effective computing efficiency, as the dataset given to us is not considered large and complex, so models that are too simple might not meet our need. After considering these key points, I decided on three types of models that I will use: a Regression Model, a Gradient-Boosting Decision Tree, and a Neural Network.

Regression Model (SVR) - (Sethi 2024)

The first model I implemented was the Support Vector Regression (SVR) of the regression model. This model can effectively handle nonlinear problems through kernel tricks. This type of model also includes models such as Lasso and Ridge, and its advantages are its simplicity and interpretability. On the other hand, its disadvantages are also very obvious. Due to their simplicity, they usually cannot capture data well and predict efficiently when dealing with overly complex data patterns. Due to the Dataset being too complex, the SVR did not really give a very good result compared to the other models.

Gradient-Boosting Decision Tree (LightGBM) - (Patel 2023)

After discovering that SVR did not meet my expectations, the second model type I tried was the Gradient Boosted Decision Tree (LightGBM). The gradient-boosting decision tree is good at dealing with non-linear relationships as it can capture the important things about a feature by creating branches. These relationships can be further improved with the gradient. In GBDT, each gradient improvement corrects the errors in the previous gradient to better adapt to the nonlinear relationship in the data.

Among the many GBDT models, I chose to use LgihtGBM. A very important reason why I chose LightGBM is that it can process large data sets very effectively, and our training data is because of very large data sets. Also, Lightgbm can continuously optimise itself and be able to handle situations where not much preprocessing is required to find the importance of different features before feature analysis, thereby improving accuracy. It is for these reasons that I chose LightGBM as one of my models, and the results it gave me were also very impressive.

Neural Network (MLP and RNN) - (R 2023)

The last model type I chose was Neural Network. Neural Network has good nonlinear modelling capabilities. It can better discover complex nonlinear relationships in data through multiple layers of neurons. Each layer of neurons is a new change, capable of capturing different relationships each time. Like GBDT, a Neural Network performs very well when processing large data sets and can continuously improve results through automatic learning. On the other hand, one of the

shortcomings of Neural Networks is that it requires a lot of data to improve slowly. Fortunately, our data set is very large.

In the choice of Neural Networks, I ended up chosing MLP and RNN. Both MLP and RNN models can effectively handle complex nonlinear relationships and help me get better results. Both models are able to handle complex data sets very effectively and give very effective and accurate predictions.

Hyperparameter Tuning

SVM - Code Guided and Learnt from ("sklearn.svm.SVR," n.d.)

Parameters Tuned, Range and the Best Parameter:

Regularization Parameter (c) - Trade-off between Bias and Variance (Best = 10)
Kernel Coefficient (gamma) - Shape and size of decision boundary (Best = 0.1)
Epsilon - How well the model goes into the training data (Best = 0.01)

```
# Hyperparameter Tuning
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1],
    'epsilon': [0.01, 0.1, 0.2]
}
```

Strategy Employed for Searching - GridSearchCV

I used GridSearchCV as it can work through multiple combinations of the parameters and cross-validate them to find the one with the best performance. The advantage is that it finds the optimised hyperparameter. The disadvantage is that it will take a long time as it goes through all the possible combinations.

LightGBM - Code Guided and Learnt from ("Python-package Introduction — LightGBM 4.3.0.99 Documentation," n.d.)

Parameters Tuned, Range and the Best Parameter:

Numer of Leaves in each tree (num_leaves) - Complexity of the tree (Best = 128)
Regularization L1 (reg_alpha) - Adding Penalty to avoid overfitting (Best = 0.1)
Minimum number of data in a leaf(min_data_in_leaf) - Stop splitting if less than that number (Best = 20)
Learning Rate (learning_rate) - Learning rate of the model (Best = 0.05)
Number of Trees (n_estimators) - Number of Boosting Levels or Trees to build (Best = 200)

```
# Grid search for hyperparameters
param_grid = {
    'num_leaves': [31, 64, 128],
    'reg_alpha': [0.1, 0.5],
    'min_data_in_leaf': [20, 50, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200]
}
```

Strategy Employed for Searching - GridSearchCV

I used GridSearchCV as it can work through multiple combinations of the parameters and cross-validate them to find the one with the best performance. The advantage is that it finds the optimised hyperparameter. The disadvantage is that it will take a long time as it goes through all the possible combinations.

MLP - Code Guided and Learnt from ("Sklearn.Neural_Network.MLPRegressor," n.d.)

Parameters Tuned and the Best Parameter:

Numer of Layer and Node in each Layer(hidden_layer sizes) - Complexity of Model(Best = 50,50)
Activation - Strategy to learn complex patterns (Best = relu)
Solver - Strategy to optimise weight (Best = adam)
Learning Rate (learning_rate_init) - Learning rate of the model (Best = 0.05)

Parameters Range:

```
# Grid search for hyperparameters
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate_init': [0.001, 0.01]
}
```

Strategy Employed for Searching - GridSearchCV

I used GridSearchCV as it can work through multiple combinations of the parameters and cross-validate them to find the one with the best performance. The advantage is that it finds the optimised hyperparameter. The disadvantage is that it will take a long time as it goes through all the possible combinations.

RNN - Code Guided and Learnt from (Ng, n.d.)

Parameters Tuned and the Best Parameter:

Numer of Node in each Layer(hidden_layer sizes) - Complexity of Model(Best = 150)
Number of Layers (num_layers) - Number of Layers (Best = 3)
Learning Rate (learning_rate_init) - Learning rate of the model (Best = 0.001)

Parameters Range:

```
# Hyperparameters sets
hidden_sizes = [50, 100, 150]
num_layers = [1, 2, 3]
learning_rates = [0.001, 0.01, 0.1]
```

Strategy Employed for Searching - Manual Tuning

Did the stupid way of manually made a hyperparameter tuning with three for loops that check the best model from all the combination of the hyperparameters.

Prediction

For the final predictions, I used the ensemble method to average the result of the three models: (MLP + LightGBM + RNN) / 3 (Here, I excluded SVR since it did not perform as the complexity might have been too complex). These are the prediction scores of each model under MAPE:

1. SVR - 0.44 (In code: MAPE - 20%, R2 = 0.83)
2. MLP - 0.24 (In code: MAPE - 18.20%, R2 = 0.95)
3. LightGBM - 0.30 (In code: MAPE - 15.7%, R2 = 0.9)
4. RNN - 0.22 (In code: MAPE - 20%, R2 = 0.83)
5. Ensemble of MLP, LightGBM and RNN - 0.21 (Best Prediction on Kaggle is in the Predicted_Pressures_Ensemble_Best.csv, been 0.20157. Team Name = Beii)

Self-Reflection

For the self-reflection, I feel like I did not really get to understand the way that RNN is coded properly, I only got into the brief of RNN and thus did not get to provide the best result. One of the difficulties I encounter is actually very funny. For weeks, I have been thinking that the sample_prediction.csv file was the answer to the test.csv and because of that, I kept on getting 300-400% MAPE and got over-stressed until I tried putting it into Kaggle and got 0.7. Another difficulty would be the RNN. As mentioned above, I did not really get to look too much into the RNN. Lessons that I learnt mainly involve preprocessing. After doing this assignment, I noticed how important preprocessing is to the model training and how a mistake in preprocessing might cause a big difference in the prediction accuracy. If I had another chance to approach this assignment, I think I would spend more time on improving my preprocessing and RNN as I feel like I did not get to do everything for those two parts and thus got a bad accuracy result.

References

GeeksforGeeks. 2024. “Regression Using LightGBM.” GeeksforGeeks. April 29, 2024. https://www.geeksforgeeks.org/regression-using-lightgbm/.

Li, Jingde, Qilin Li, Hong Hao, and Ling Li. 2021. “Prediction of BLEVE Blast Loading Using CFD and Artificial Neural Network.” Process Safety and Environmental Protection/Transactions of the Institution of Chemical Engineers. Part B, Process Safety and Environmental Protection/Chemical Engineering Research and Design/Chemical Engineering Research & Design 149 (May): 711–23. https://doi.org/10.1016/j.psep.2021.03.018.

Li, Qilin, Yang Wang, Yanda Shao, Ling Li, and Hong Hao. 2023. “A Comparative Study on the Most Effective Machine Learning Model for Blast Loading Prediction: From GBDT to Transformer.” Engineering Structures/Engineering Structures (Online) 276 (February): 115310. https://doi.org/10.1016/j.engstruct.2022.115310.

Mueller, John Paul, and Luca Massaron. 2021. *Machine Learning for Dummies*. John Wiley & Sons.

Ng, Ritchie. n.d. “Recurrent Neural Networks (RNN) - Deep Learning Wizard.” https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_recurrent_neural network/#step-3-create-model-class.

Patel, Kadambari. 2023. “How Does GBDT Work in Regression? - Kadambari Patel - Medium.” Medium, July 4, 2023. https://medium.com/@kadambaripatel79/how-does-gbdt-work-in-regression-2e40f58b6033#:~:text=Gradient%20Boosted%20Decision%20Trees%20(GBDT)%20is%20a%20powerful%20technique%20for,the%20model's%20accuracy%20and%20robustness.

R, Srivignesh. 2023. “A Walk-through of Regression Analysis Using Artificial Neural Networks in Tensorflow.” Analytics Vidhya. July 18, 2023. https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/.

Sethi, Alakh. 2024. “Support Vector Regression Tutorial for Machine Learning.” Analytics Vidhya. February 9, 2024. https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/#:~:text=SVM%20regression%20or%20Support%20Vector,line%20to%20the%20data%20points.

“6.3. Preprocessing Data.” n.d. Scikit-Learn. https://scikit-learn.org/stable/modules/preprocessing.html.

“Python-package Introduction — LightGBM 4.3.0.99 Documentation.” n.d. https://lightgbm.readthedocs.io/en/latest/Python-Intro.html.

“Sklearn.Neural_Network.MLPRegressor.” n.d. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

"sklearn.impute.KNNImputer." n.d. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer.

"Sklearn.Preprocessing.PowerTransformer." n.d. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html#sklearn.preprocessing.PowerTransformer.

"sklearn.svm.SVR." n.d. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.