School of Science, Computing and Engineering Technologies

COS40007

Artificial Intelligence for Engineering

Portfolio Assessment-3: "Let's develop AI model by your own decision'"

Studio 4 – Portfolio 4

Students' Name: Duc Thinh Pham - 104169675

Lecturer: Dr. Trung Luu

Date: October 5th, 2024

CONTENTS

1 INTRODUCTION

This assessment is designed to evaluate your understanding of machine learning (ML) model development. The objective is to explore a dataset, analyze its features, and compare various ML models, with a focus on data preparation, feature selection, and model evaluation. Through hands-on experience, you'll create, evaluate, and deploy an AI model based on the dataset provided, which contains data related to vegemite production. This task also involves transitioning from ML to AI deployment by utilizing unseen data and generating decision rules to optimize the model's predictions. The assessment requires comprehensive analysis and a well-documented approach, culminating in a final report and code submission.

2 PORTFOLIO WORK

2.1 DATA PREPARATION

2.1.1 Does the dataset have any constant value column. If yes, then remove them

A **constant value column** is a column in a dataset where all the values are identical, meaning the column contains only one unique value across all the rows. These columns do not provide any useful information for model training because they lack variability and, therefore, cannot help the model distinguish between different outputs. In other words, they do not contribute to the predictive power of the model.

```
constant_columns = [col for col in df_train.columns if df_train[col].nunique() == 1]

# Remove those columns
df_train = df_train.drop(columns=constant_columns)

print("Columns removed:", constant_columns)
```

```
Columns removed: ['TFE Steam temperature SP', 'TFE Product out temperature']
```

Figure 1: Identifying and Removing Constant Value Columns

2.1.2 Does the dataset have any column with few integer values? If yes, then convert them to categorial feature.

A column with few integer values is one that contains a small number of distinct values, even though the data type of the column is numerical. Such columns often represent categories or discrete classes rather than continuous numerical data. These columns should be treated as categorical features because their values represent specific categories or labels rather than actual measurements or quantities. Treating these columns as continuous numerical data could mislead the machine learning model, leading to incorrect assumptions about the relationship between these features and the target variable.

To handle this scenario, we define a threshold (in this case, 10 unique values or fewer) to identify columns that should be converted to categorical features:

```
# Define the threshold for "few" unique values (e.g., 10 or fewer)
threshold = 10

# Identify integer columns with few unique values
cols_to_convert = [col for col in df_train.columns if col != 'Class' and df_train[col].nunique() <= threshold]

cols_to_convert
```

```
['FFTE Feed tank level SP',
 'FFTE Pump 1',
 'FFTE Pump 1 - 2',
 'FFTE Pump 2',
 'TFE Motor speed']
```

Figure 2: Identifying few integers-values column

These columns need to be converted to categorical features. We use **label encoding** to assign a unique integer label to each distinct category. The **LabelEncoder** from the **sklearn.preprocessing** module is applied to each identified column. This converts the original integer values to categorical labels. The encoder objects are stored in the **label_encoders** dictionary for future reference, allowing us to decode or reverse the transformation if needed. Finally, we save the transformed dataset:

```
label_encoders = {}

for col in cols_to_convert:
    le = LabelEncoder()
    df_train[col] = le.fit_transform(df_train[col])
    label_encoders[col] = le

df_train.to_csv('vegemite_converted.csv', index = False)
```

Figure 3: Converting these columns to categorical labels

2.1.3 Does the class have a balanced distribution? If not, then perform necessary undersampling and oversampling or adjust class weights.

The target column exhibits an imbalanced distribution, with **Class 2** being the most prevalent, followed by **Class 1** and **Class 0**. To tackle this issue of class imbalance, we opted to use the **Synthetic Minority Over-sampling Technique (SMOTE)** for oversampling, a widely recognized and effective method.

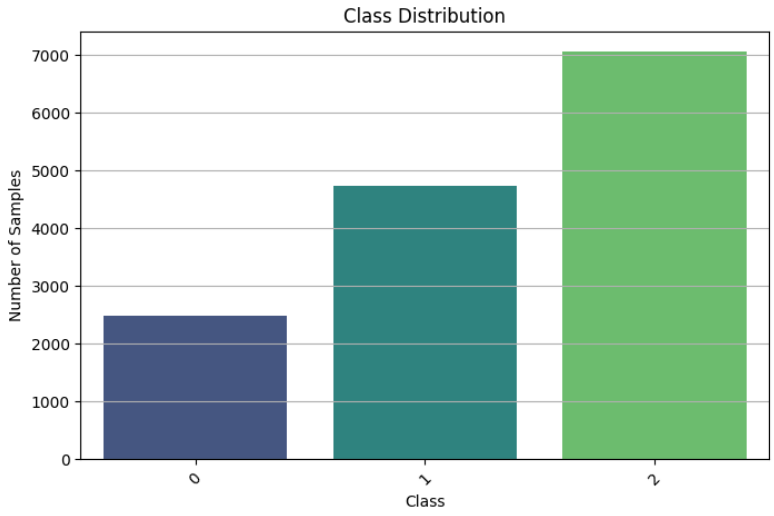

Figure 4: Target Column (class) Distribution

The following code snippet demonstrates how to balance an imbalanced dataset using **SMOTE (Synthetic Minority Over-sampling Technique)** for oversampling and **Tomek Links** for under-sampling. This approach is particularly effective in improving the performance of machine learning models by ensuring that all classes are adequately represented.

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import TomekLinks
from collections import Counter

X = df_train.drop('Class', axis=1)
y = df_train['Class']

print(f"Initial class distribution:\n{Counter(y)}")

smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)

tomek_links = TomekLinks()
X_balanced, y_balanced = tomek_links.fit_resample(X_smote, y_smote)

print(f"Class distribution after SMOTE and Tomek Links:\n{Counter(y_balanced)}")

df_train = pd.concat([pd.DataFrame(X_balanced, columns=X.columns), pd.Series(y_balanced, name='Class')], axis=1)
```

✓ 0.8s

```
Initial class distribution:
Counter({2: 7053, 1: 4716, 0: 2468})
Class distribution after SMOTE and Tomek Links:
Counter({0: 7053, 1: 6959, 2: 6932})
```

*Figure 5: **SMOTE (Synthetic Minority Over-sampling Technique)** for oversampling and **Tomek Links** for under-sampling*

2.1.4 Do you find any composite feature through exploration? If so, then add some composite feature in the dataset.

During the exploration of the dataset, we analyzed the correlation between various features and the target column, **Class**. We specifically focused on identifying features that exhibited small positive correlations with the target variable.

Composite Features:

A **composite feature** is a new feature derived from the combination of two or more existing features. These composite features can capture interactions or relationships that might not be evident when considering the individual features in isolation.

Findings:

1. **Correlation Analysis**:
    - We established a threshold of **0.1** to identify features that have small positive correlations with the target column.
    - The correlation matrix was computed, and we identified features with correlations greater than **0** but less than or equal to **0.1**.
2. **Identifying Pairs**:
    - We examined the filtered correlation matrix to identify pairs of features that also demonstrated small positive correlations with each other.
    - The identified pairs were listed for further consideration in creating composite features.

```
threshold = 0.1
corr_matrix = df_train.corr()

# Get the correlation of each feature with the target column
target_corr = corr_matrix['Class']

# # Find features with small positive correlations with the target column
small_positive_corr = target_corr[(target_corr > 0) & (target_corr <= threshold)].index.tolist()

# Filter the correlation matrix to include only the small positive correlations
filtered_corr_matrix = corr_matrix.loc[small_positive_corr, small_positive_corr]

pairs = []

for i, feature in enumerate(small_positive_corr):
    for other_feature in small_positive_corr[i+1:]:
        if 0 < filtered_corr_matrix.loc[feature, other_feature] <= threshold:
            pairs.append((feature, other_feature))

len(small_positive_corr), len(pairs)
```

✓ 0.0s

```
(16, 25)
```

Figure 6: Implementation of Composite Features

2.1.5 How many features you have in your final dataset?

After the various preprocessing steps and feature engineering techniques applied to the dataset, we have a total of **70 features** in the final dataset. This includes the original features as well as any additional composite features created during the exploration process. The inclusion of these composite features aims to enhance the model's ability to learn from the data and improve predictive performance.

2.2 FEATURE SELECTION, MODEL TRAINING AND EVALUATION

2.2.1 Does the training process need all features? If not, can you apply some feature selection technique to remove some features? Justify your reason of feature selection

Feature Selection Technique Applied

In this case, we utilized the **SelectKBest** method from the sklearn library, which is based on statistical tests to select the top features that have the strongest relationship with the target variable.

1. **Standardization**: Before applying the feature selection technique, we standardized the features using **StandardScaler**. This ensures that each feature contributes equally to the distance calculations in the selection process, which is particularly important when dealing with features on different scales.
2. **Top Features Selected**: After applying **SelectKBest**, we identified the top 10 features, which are:
    - FFTE Production solids SP
    - TFE Out flow SP
    - FFTE Heat temperature 1
    - FFTE Temperature 1 - 1
    - FFTE Temperature 1 - 2
    - FFTE Temperature 2 - 1
    - FFTE Temperature 3 - 2
    - FFTE Unk Temperature
    - TFE Steam temperature
    - TFE Temperature

Justification for Feature Selection

Feature selection reduces overfitting by eliminating irrelevant features, thus simplifying the model and minimizing the risk of learning noise instead of patterns. This simplification also enhances model performance, as focusing on the most informative features allows for faster, more accurate predictions. Additionally, a simpler model is easier to interpret, which is valuable in real-world applications where understanding prediction factors is crucial. With fewer features, training times are reduced, enabling quicker iterations and experimentation. Ultimately, selecting only the most relevant features improves the model's generalization ability, enhancing its predictive capabilities on unseen data.

```python
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=['Class'])
y = df['Class']

#Standardize the features
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Use the SelectKBest class to extract the top 10 features
selector = SelectKBest(score_func=f_classif, k=10)
X_new = selector.fit_transform(X, y)

# print the selected features
selected_features = X.columns[selector.get_support()]
print(selected_features)

#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.3, random_state=42)

#Print the shape of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.0s

```
Index(['FFTE Production solids SP', 'TFE Out flow SP',
       'FFTE Heat temperature 1', 'FFTE Temperature 1 - 1',
       'FFTE Temperature 1 - 2', 'FFTE Temperature 2 - 1',
       'FFTE Temperature 3 - 2', 'FFTE Unk Temperature',
       'TFE Steam temperature', 'TFE Temperature'],
      dtype='object')

((14660, 10), (6284, 10), (14660,), (6284,))
```

Figure 7: Applying SelectKBest for Feature Selection

2.2.2 Train multiple ML models (at least 5 including DecisionTreeClassifier) with your selected features.

In this step, we train multiple machine learning models using the selected features, which include Logistic Regression, Random Forest, Decision Tree, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Each model is trained on the training dataset, then evaluated on the test dataset. For evaluation, we capture both the classification report, which includes precision, recall, and F1-score, and the confusion matrix for each model. These metrics help assess how well each model performs and highlight their strengths and weaknesses across different classes. This comprehensive evaluation allows us to compare the models and select the one with the best overall performance.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split


# Models initialization
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier()
}

# Train and evaluate each model
results = {}
for model_name, model in models.items():
    print(f"Training {model_name}...")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Save evaluation metrics
    results[model_name] = {
        'classification_report': classification_report(y_test, y_pred),
        'confusion_matrix': confusion_matrix(y_test, y_pred)
    }
```

✓ 10.0s

```
Training Logistic Regression...
Training Random Forest...
```

Figure 8: Training multiple ML models

2.2.3 Evaluate each model with classification report and confusion matrix

2.2.3.1 Classification report

Model	Accuracy	Precision	Recall	F1-Score
LogisticRegression	0.48	0.48	0.48	0.47
RandomForest	0.98	0.98	0.98	0.98
DecisionTree	0.95	0.95	0.95	0.95
SVM	0.68	0.68	0.68	0.68
KNN	0.89	0.89	0.89	0.89

Table 1: Classification Report Summary for All Models

Observations:

- The **Random Forest** model outperformed all other models in all metrics, achieving an accuracy of 0.98.
- The **Decision Tree** and **K-Nearest Neighbors (KNN)** models also showed strong performance, with accuracy values of 0.95 and 0.89, respectively.
- The **Logistic Regression** model had the lowest performance, with an accuracy of 0.48, indicating it struggled to effectively classify the instances in this dataset.

- The **Support Vector Machine (SVM)** model had moderate performance, achieving an accuracy of 0.68.

2.2.3.2 Confusing Matrix

Model Evaluations:

1. **Logistic Regression**: This model performs poorly overall, with significant misclassifications across all three classes. It struggles particularly with Class 1, resulting in low precision and recall for this category. The linear decision boundaries of Logistic Regression likely contribute to its inability to capture the complexities in the data, making it the least effective model among those evaluated.
2. **Random Forest**: This classifier excels in performance across all classes, with very few misclassifications. It demonstrates high precision and recall for Classes 0, 1, and 2, making it the best-performing model based on the confusion matrices. The Random Forest's ensemble approach effectively reduces overfitting and captures the underlying patterns in the data well.
3. **Decision Tree**: This model performs reasonably well but exhibits significant issues in predicting Class 2, with a notable number of misclassifications occurring between classes. While it achieves decent overall accuracy, the tendency to overfit and the sensitivity to variations in data lead to challenges in distinguishing Class 2 from the others.
4. **Support Vector Machine (SVM)**: This classifier shows good performance in identifying Class 0 but encounters difficulties in distinguishing between Classes 1 and 2. The non-linear decision boundaries of the SVM may contribute to its performance issues, particularly in the overlap between these classes, which could lead to confusion in classification.
5. **K-Nearest Neighbors (KNN)**: This model performs well, achieving a good balance of precision and recall across all classes, particularly in Class 0. However, it struggles slightly with Class 1, resulting in more misclassifications. Despite this, KNN shows strong overall effectiveness and is a reliable choice among the evaluated models.

Summary:

The evaluations highlight the varying strengths and weaknesses of each model. The **Random Forest** model stands out as the best performer, while **Logistic Regression** exhibits the most significant challenges. The remaining models each have their unique advantages and areas for improvement, suggesting that model selection should be based on the specific requirements of the classification task.

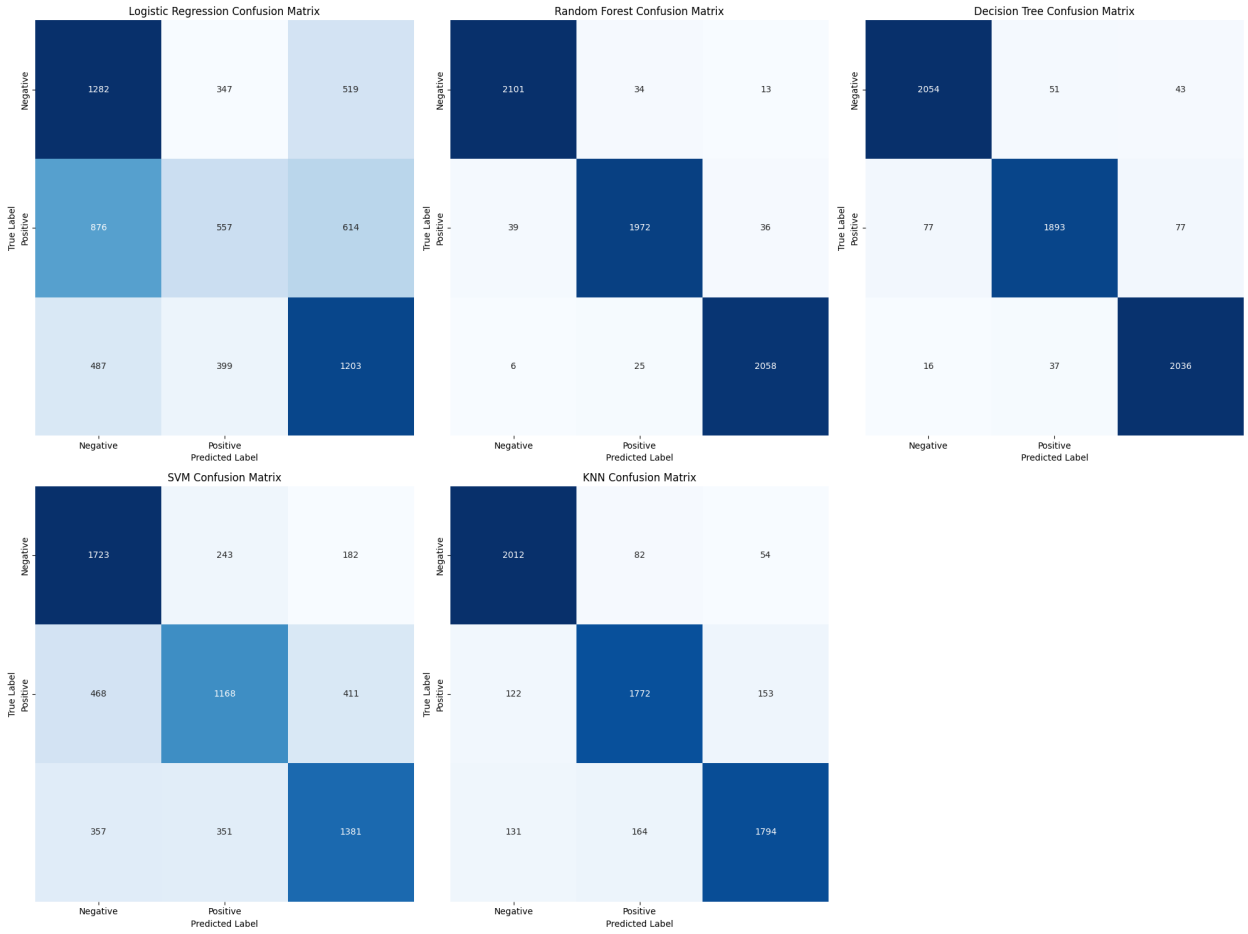

Figure 9: Confusing Matrix Plots across All Models

2.2.4 Now select your best performing model to use that as AI. Justify the reason of your selection

The **Random Forest** model is selected as the best-performing AI model due to its exceptional accuracy of **0.98** across all classes and its ability to maintain balanced performance with minimal misclassifications. Unlike other models, such as Logistic Regression and Support Vector Machine, which struggled with specific classes, the Random Forest demonstrated high precision and recall for each class, ensuring reliable and robust predictions. Its ensemble approach effectively reduces the risk of overfitting while capturing complex interactions among features, making it well-suited for this classification task. Additionally, the model's ability to provide insights into feature importance enhances interpretability, allowing for informed decision-making and further improvements in feature engineering.

2.2.5 Now save your selected model

I import and use pickle to save Random Forest model as the best one.

Save model

```
import pickle

filename = 'model/bestmodel.pkl'
pickle.dump(models['Random Forest'], open(filename, 'wb'))

for model_name, model in models.items():
    filename = f'model/{model_name}_model.pkl'
    pickle.dump(model, open(filename, 'wb'))
```

✓ 0.0s

Figure 10: Saving best model

2.3 ML TO AI

2.3.1 Now measure the performance of other model using these 1000 data points. Have you observed same result of model selection that you identified through evaluation?

Model	Accuracy
BestModel	0.495
LogisticRegression	0.495
DecisionTree	0.495
SVM	0.495
KNN	0.316

Table 2: Classification Report Summary for All Models using Test dataset

The evaluation of the models reveals that most classifiers—namely Decision Tree, Random Forest, Support Vector Machine (SVM), and Logistic Regression—achieved identical accuracy of **0.4950**, indicating a shared ineffectiveness in distinguishing between the classes, especially for Classes 0 and 1. In contrast, the K-Nearest Neighbors (KNN) model exhibited a notably lower accuracy of **0.3160**, suggesting significant challenges in classification, likely influenced by class imbalance and the dataset's feature space. This overall low accuracy across models points to potential issues with class representation, emphasizing the need for further investigation. Although the Random Forest was previously selected as

the best model, its performance does not significantly surpass simpler alternatives, indicating that preprocessing, feature selection, or hyperparameter tuning may require reevaluation. Consequently, additional steps, such as implementing SMOTE to address class imbalance and exploring advanced modeling techniques, are necessary to improve predictive performance and ensure more effective classification outcomes.

2.4 DEVELOP RULES FROM ML MODEL

The outcome of my Decision Tree:

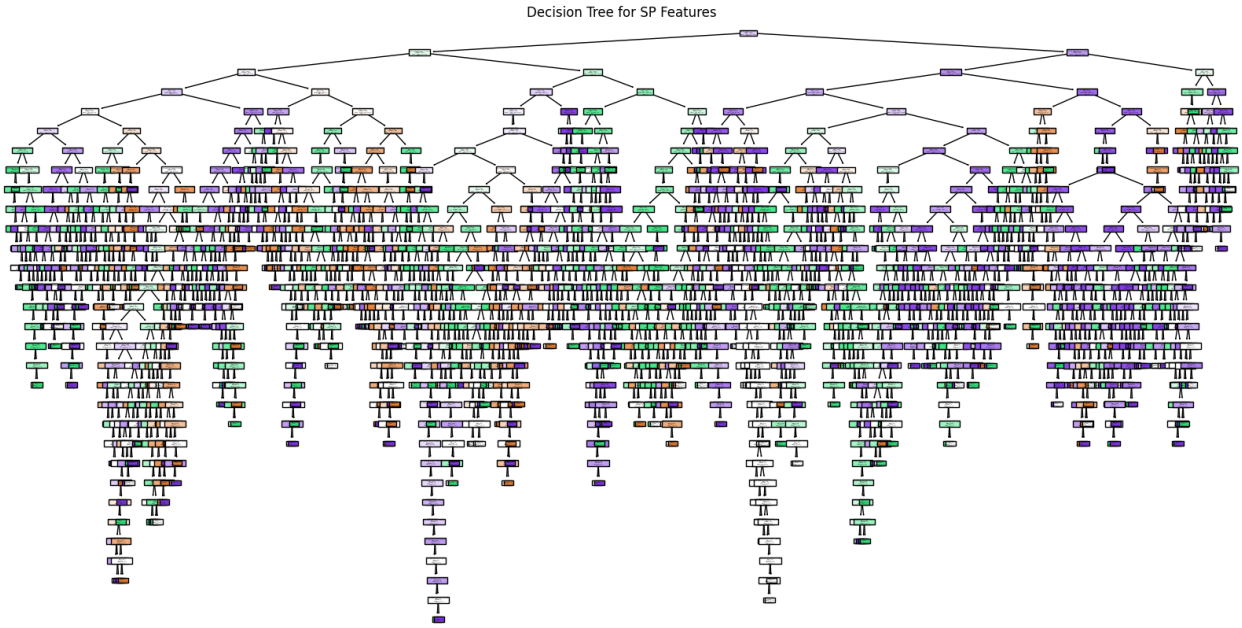

Figure 11: Decision Tree for SP Features

Define Rules for Each Class

Based on the generated rules, I can define specific conditions for each class:

- **Class 0**:
    - If **FFTE Production solids SP** ≤ 40.5 and **TFE Steam temperature SP** ≤ 60.0, then Class = 0.
- **Class 1**:
    - If **FFTE Production solids SP** ≤ 40.5 and **TFE Steam temperature SP** > 60.0, then Class = 1.
    - If **FFTE Production solids SP** > 40.5 and **FFTE Temperature 1 SP** > 50.0, then Class = 1.
- **Class 2**:
    - If **FFTE Production solids SP** > 40.5 and **FFTE Temperature 1 SP** ≤ 50.0, then Class = 2.

3 APPENDIX

My GitHub repository: thinhpham1807/COS40007---Artificial-Intelligence-for-Engineering (github.com)

Data files:
https://drive.google.com/drive/folders/1YKb1ne0l4HChTw7oVUK59oTyg78tcgmy?usp=sharing