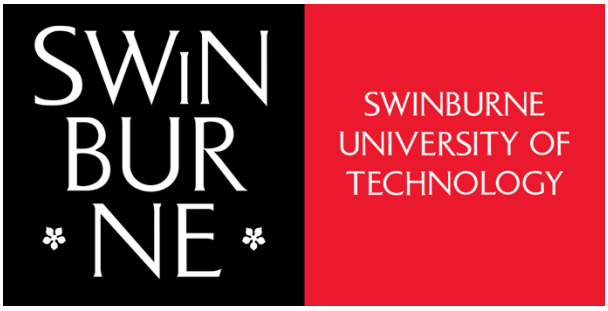


Artificial Intelligence for Engineering

Portfolio Assessment 3

Develop an AI model by your own decision

Xuan Tuan Minh Nguyen - 103819212
Studio 1-6

Step 1: Data Preparation *3*
1. Shuffling and randomly get 1000 data points for real-time testing *3*
2. Remove constant value columns 3
3. Convert columns with few integer values to categorical feature 4
4. Methods for making class distribute evenly 4
5. Exploring and adding composite features 6
Step 2: Feature selection, Model Training and Evaluation *8*
1. Feature Selection *8*
2. Models Training *9*
3. Models Evaluation *10*
4. Best model selection and models saving *14*
Step 3: ML to AI *15*
1. Data processing and predicting process *15*
2. Prediction result *16*
Step 4: Develop rules from the ML model *17*
Appendix *18*

Step 1: Data Preparation

1. Shuffling and randomly get 1000 data points for real-time testing

To effectively shuffle the data, I have used the previously applied shuffling method, which is the `sample()` function, and saved it into a file called **shuffle_vegemite.csv** for later use.

Figure 1: Code to shuffle and save the shuffled dataset

To create the dataset with 1000 data points, I have used the `model_selection.train_test_split()` method provided by `scikit-learn`, which is normally used to split training and testing datasets. However, I have used a special parameter called `stratify`, which will help generate the dataset where those 1000 samples will have the near equal distribution. Once the splitting process is finished, I have saved the train and test datasets into the equivalent csv file for further use.

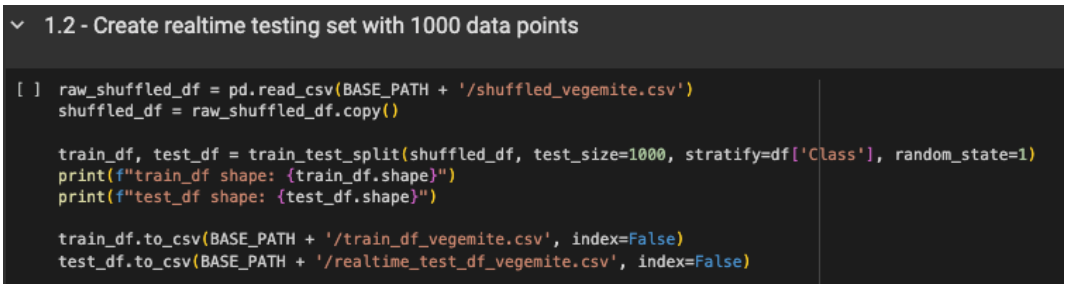


Figure 2: Code to create 1000 data points for real time testing

2. Remove constant value columns

Now with the remaining 14000+ data points, I have started to find out the constant value columns, which could be easily done with the `nunique()` function, which helps finding columns that their values only have one value. Once I have done the investigation, in which the columns of **TFE Steam temperature SP** and **TFE**

Product out temperature have only one value, I attempted to drop it using the `drop()` function provided by `pandas`.

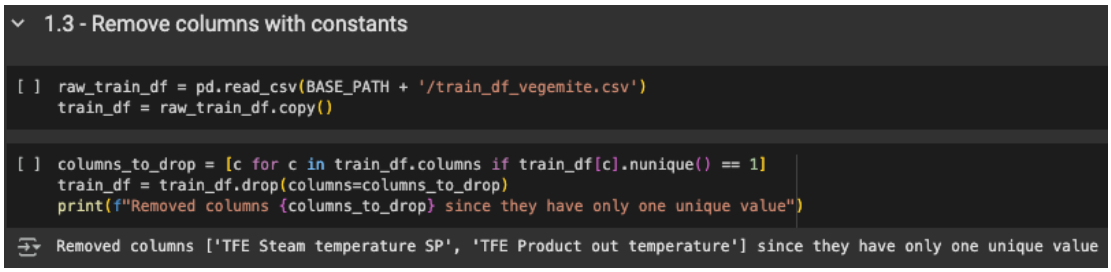


Figure 3: Finding columns with one unique value and drop it

3. Convert columns with few integer values to categorical feature

For this section, I have decided to convert columns that have the unique values less than or equal to 5 values, in which after a short exploration using `nunique()` function, the columns that need to be converted are **FFTE Feed tank level SP**, **FFTE Pump 1**, **FFTE Pump 1 - 2**, **FFTE Pump 2** and **TFE Motor speed**.

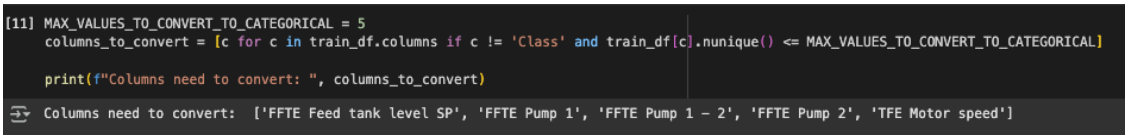


Figure 4: Exploring columns that has less than or equal to five unique values

Once discovered and temporarily saved into an array, I have used `LabelEncoder()` to convert those columns into categorical features, then saved it into a `dict` for reusing in next steps.

Figure 5: Features conversion using `LabelEncoder` and save for further usages

4. Methods for making class distribute evenly

```
[13] raw_converted_df = pd.read_csv(BASE_PATH + '/converted_vegemite.csv')
     converted_df = raw_converted_df.copy()

     distrib_of_class = converted_df['Class'].value_counts()
     print(distrib_of_class)

     sns.barplot(x=distrib_of_class.index, y=distrib_of_class.values)
```

Figure 6: Code to plot the distribution of the **Class** column

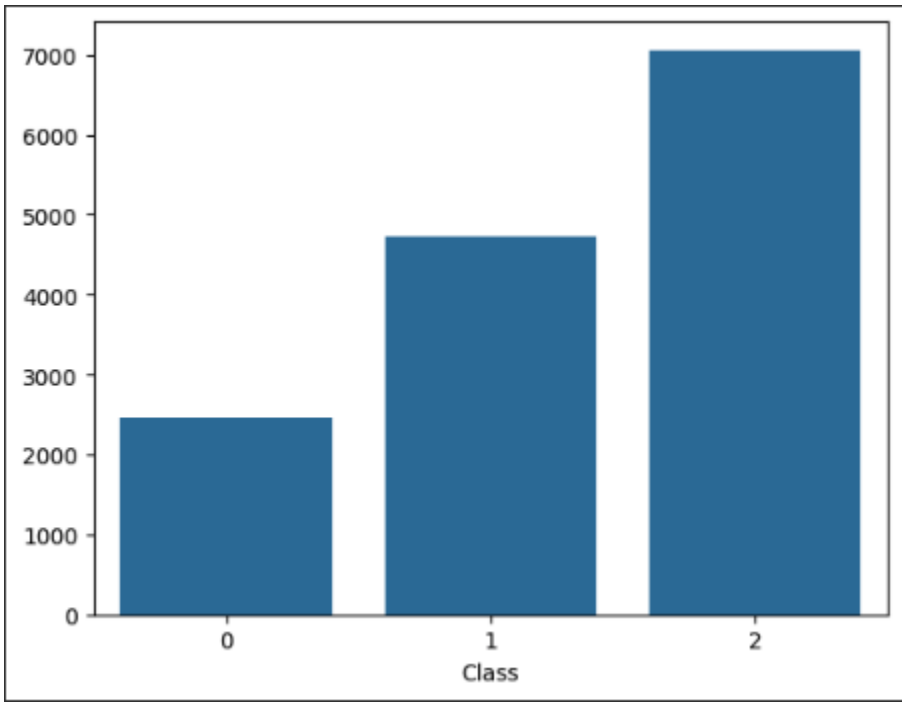


Figure 7: Value distribution of **Class** column

As described in the bar plot, the value distribution of the **Class** column is not balanced at all, in which the different range between values are around approximately 2500 - 5000 data points. To overcome this, I have used two popular techniques for **undersampling** and **oversampling**, which are **Synthetic Minority Oversampling Technique (SMOTE)** and **Tomek Links**. These techniques are built-in with `sklearn` API, and could be used via the exposed API of `over_sampling.SMOTE` and `under_sampling.TomekLinks` methods.

```
[14] X = converted_df.drop('Class', axis=1)
     Y = converted_df['Class']

     print(f"Class before re-distributed: ({Y.value_counts()})")
     smote_resampler = SMOTE(random_state=42)
     tomek_links_resampler = TomekLinks()

     X_smote, Y_smote = smote_resampler.fit_resample(X, Y)
     X_balanced, Y_balanced = tomek_links_resampler.fit_resample(X_smote, Y_smote)

     print(f"Class after re-distributed: ({Y_balanced.value_counts()})")
     resampled_df = pd.concat([pd.DataFrame(X_balanced, columns=X.columns), pd.Series(Y_balanced, name='Class')], axis=1)
     resampled_df.to_csv(BASE_PATH + '/resampled_vegemite.csv', index=False)
```

Figure 8: Undersampling and Oversampling code

```
Class before re-distributed: (Class
2    7059
1    4719
0    2459
Name: count, dtype: int64)
Class after re-distributed: (Class
0    7059
1    6957
2    6932
Name: count, dtype: int64)
```

Figure 9: Result before and after re-distribute

As we can see in **Figure 9**, after the redistribution process, the distribution is now more balanced, with the gap between values being no more than 200 data points, which is good for the training process.

5. Exploring and adding composite features

Before exploring and adding composite features, I have scaled the values of all features using `MinMaxScaler()` and saved it into a file for further usage.

```
[16] features = resampled_df.drop('Class', axis=1)
     target = resampled_df['Class']

     scaler = MinMaxScaler()
     scaled_features = scaler.fit_transform(features)

     scaled_df = pd.DataFrame(scaled_features, columns=features.columns)
     scaled_df['Class'] = target.values
     print(scaled_df.info())
     scaled_df.to_csv(BASE_PATH + '/scaled_vegemite.csv', index=False)
```

Figure 10: Scaling values using `MinMaxScaler`

Once I have done with scaling, I reused the saved datasets, which is `scaled_vegemite.csv`, to draw the correlation heatmap to visualise the relationship of variables.

```
1.6 - Correlation analysis

raw_scaled_df = pd.read_csv(BASE_PATH + '/scaled_vegemite.csv')
scaled_df = raw_scaled_df.copy()
print(len([c for c in scaled_df.columns]))

45

[71] correlation = abs(scaled_df.corr())
     print(correlation.T)
```

Figure 11: Import the scaled dataset and get the correlation

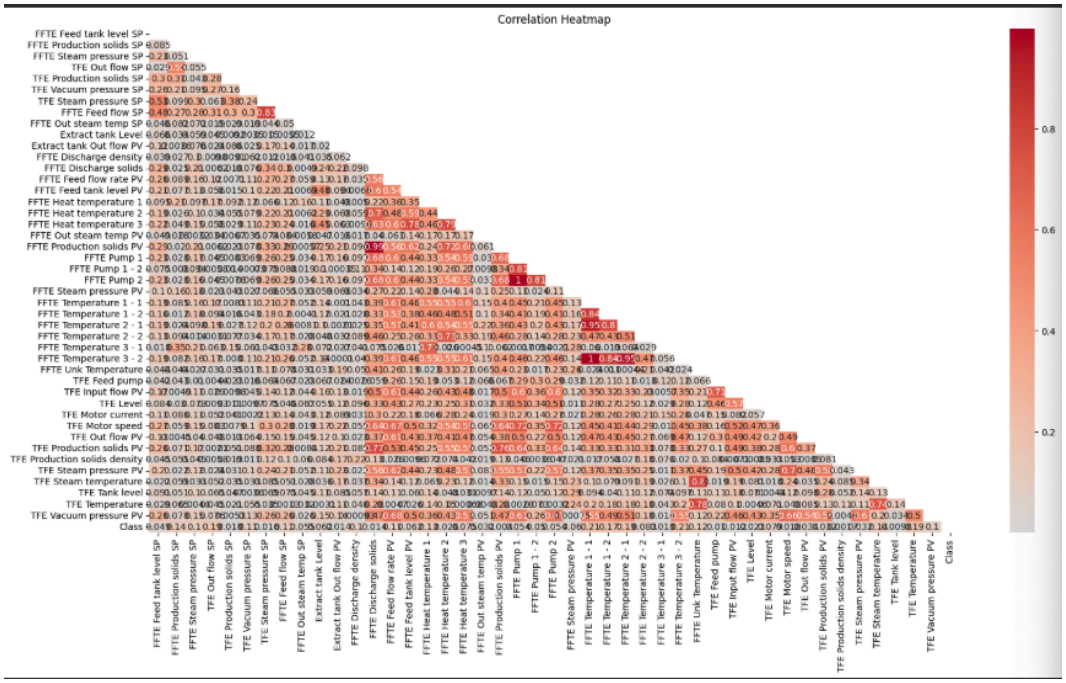


Figure 12: Heatmap of the dataset

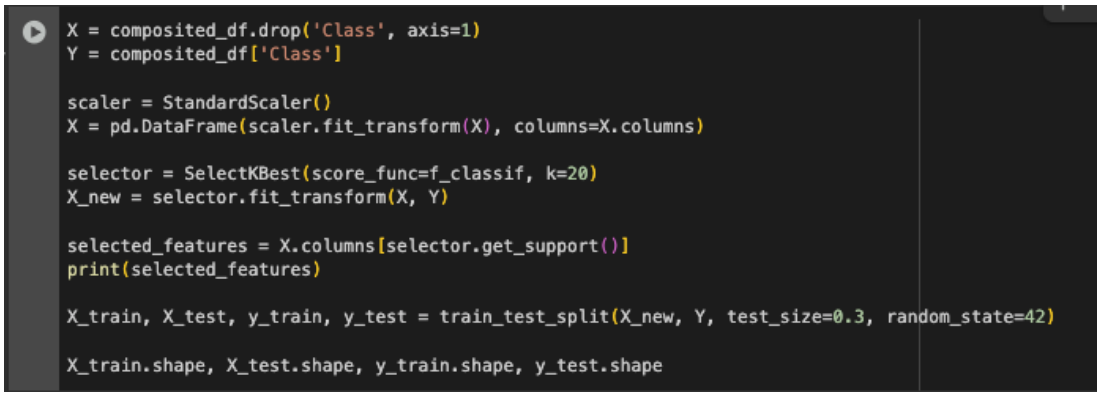
As described in the heatmap, the correlation between the **Class** column with the other features are relatively low, with just approximately around 0.02 to 0.2. Thus, I have decided to create new composite features based on the columns that have **low positive correlation** with the **Class** column, resulting in a total of 59 features in the final dataset.

Figure 13: Codebase to find the composite features and create new composite features

Step 2: Feature selection, Model Training and Evaluation

1. Feature Selection

In this section, I have performed feature selection by choosing 20 best features using the `SelectKBest` method.



```
X = composited_df.drop('Class', axis=1)
Y = composited_df['Class']

scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

selector = SelectKBest(score_func=f_classif, k=20)
X_new = selector.fit_transform(X, Y)

selected_features = X.columns[selector.get_support()]
print(selected_features)

X_train, X_test, y_train, y_test = train_test_split(X_new, Y, test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Figure 14: Select 20 best features using `SelectKBest` method

```
Index(['FFTE Feed tank level SP', 'FFTE Production solids SP',
       'TFE Out flow SP', 'TFE Production solids SP', 'TFE Vacuum pressure SP',
       'FFTE Feed flow SP', 'FFTE Out steam temp SP', 'FFTE Feed flow rate PV',
       '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',
       'TFE Production solids SP_FFTE Discharge solids',
       'TFE Production solids SP_FFTE Heat temperature 2',
       'TFE Production solids SP_FFTE Production solids PV',
       'TFE Production solids SP_FFTE Temperature 2 - 2'],
      dtype='object')
((14663, 20), (6285, 20), (14663,), (6285,))
```

Figure 15: 20 Selected Features

2. Models Training

I have used five different models, which are **Logistic Regression, Decision Tree Classifier, Random Forest, K-Nearest Neighbors and Support Vector Machine**

2.2 - Model training

```
[151] def train_model_and_generate_report(models: dict, X_train, X_test, Y_train, Y_test):
        _results = {}
        for name, model in models.items():
          print(f"Training model: {name}")
          model.fit(X_train, Y_train)
          print(f"Generating report for: {name}")
          Y_predict = model.predict(X_test)
          _results[name] = {
              'classification': classification_report(Y_test, Y_predict),
              'confusion': confusion_matrix(Y_test, Y_predict)
          }
        return _results
```

```
[152] models = {
          'Logistic Regression': LogisticRegression(),
          'Decision Tree': DecisionTreeClassifier(),
          'Random Forest': RandomForestClassifier(),
          'K-Nearest Neighbors': KNeighborsClassifier(),
          'Support Vector Machine': SVC()
      }

      results = train_model_and_generate_report(models, X_train, X_test, y_train, y_test)
```

Figure 16: Code for training and generating report for 5 models

```
Training model: Logistic Regression
Generating report for: Logistic Regression
Training model: Decision Tree
Generating report for: Decision Tree
Training model: Random Forest
Generating report for: Random Forest
Training model: K-Nearest Neighbors
Generating report for: K-Nearest Neighbors
Training model: Support Vector Machine
Generating report for: Support Vector Machine
```

Figure 17: Training and report generation process

3. Models Evaluation

Below is the overall results of the five models, which include **precision score, recall score, F1 score** and **confusion matrix**

- **Logistic Regression**

```
Overall Report for model: Logistic Regression
Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.66      0.55      2112
           1       0.51      0.32      0.39      2125
           2       0.54      0.54      0.54      2048

    accuracy                           0.50      6285
   macro avg       0.51      0.50      0.49      6285
weighted avg       0.51      0.50      0.49      6285

Confusion Matrix:
[[1391  320  401]
 [ 923  671  531]
 [ 616  325 1107]]
```

Figure 18: Classification and confusion matrix of Logistic Regression

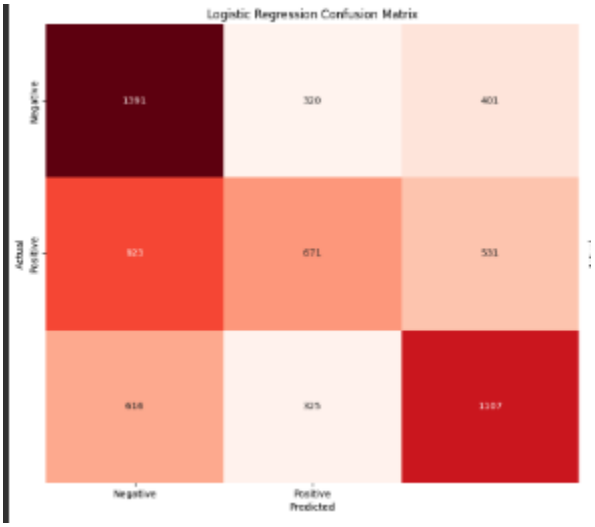


Figure 19: Visualisation of Logistic Regression's confusion matrix

- **Decision Tree Classifier**

```
Overall Report for model: Decision Tree
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      2112
           1       0.95      0.95      0.95      2125
           2       0.96      0.96      0.96      2048

    accuracy                           0.96      6285
   macro avg       0.96      0.96      0.96      6285
weighted avg       0.96      0.96      0.96      6285

Confusion Matrix:
[[2035   49   28]
 [  58 2016   51]
 [  21   64 1963]]
```

Figure 20: Classification and confusion matrix of Decision Tree Classifier

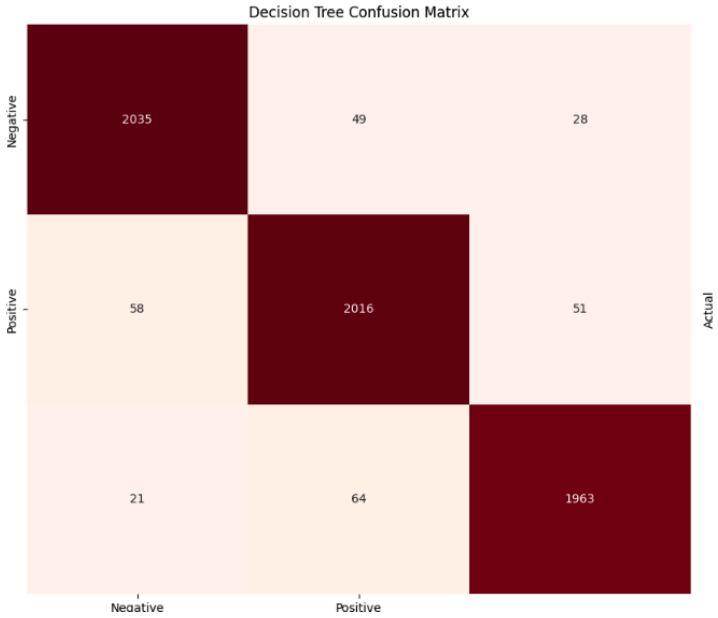


Figure 21: Visualisation of Decision Tree Classifier's confusion matrix

- **Random Forest**

```
Overall Report for model: Random Forest
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2112
           1       0.99      0.99      0.99      2125
           2       0.99      0.99      0.99      2048

    accuracy                           0.99      6285
   macro avg       0.99      0.99      0.99      6285
weighted avg       0.99      0.99      0.99      6285

Confusion Matrix:
[[2096    9    7]
 [  19 2096   10]
 [   2   12 2034]]
```

Figure 22: Classification and confusion matrix of Random Forest

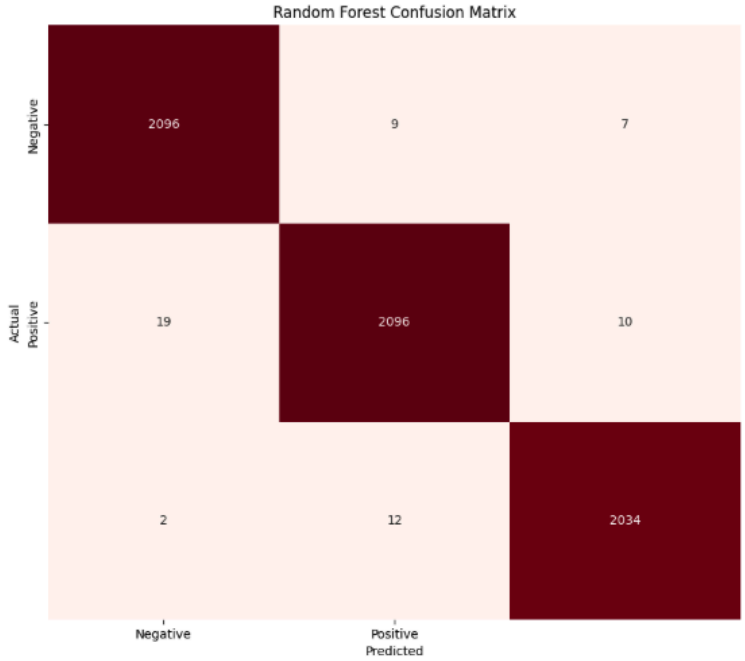


Figure 23: Visualisation of Random Forest's confusion matrix

- **K-Nearest Neighbors**

```
Overall Report for model: K-Nearest Neighbors
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2112
           1       0.91      0.91      0.91      2125
           2       0.95      0.91      0.93      2048

    accuracy                           0.93      6285
   macro avg       0.93      0.93      0.93      6285
weighted avg       0.93      0.93      0.93      6285

Confusion Matrix:
[[2027   61   24]
 [ 104 1943   78]
 [  65  128 1855]]
```

Figure 24: Classification and confusion matrix of K-Nearest Neighbors

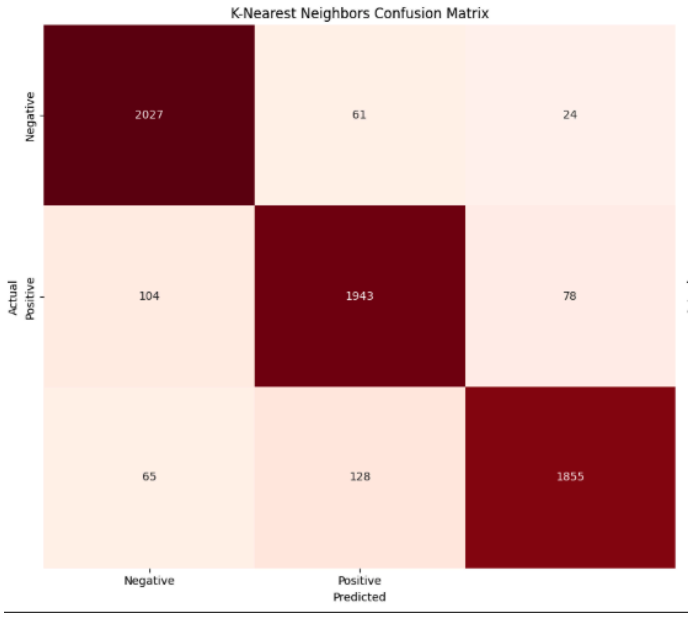


Figure 21: Visualisation of K-Nearest Neighbors's confusion matrix

- **Support Vector Machine**

```
Overall Report for model: Support Vector Machine
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.90      0.79      2112
           1       0.78      0.66      0.71      2125
           2       0.81      0.72      0.76      2048

    accuracy                           0.76      6285
   macro avg       0.77      0.76      0.76      6285
weighted avg       0.77      0.76      0.76      6285

Confusion Matrix:
[[1897  118   97]
 [ 471 1393  261]
 [ 301  267 1480]]
```

Figure 20: Classification and confusion matrix of Support Vector Machine

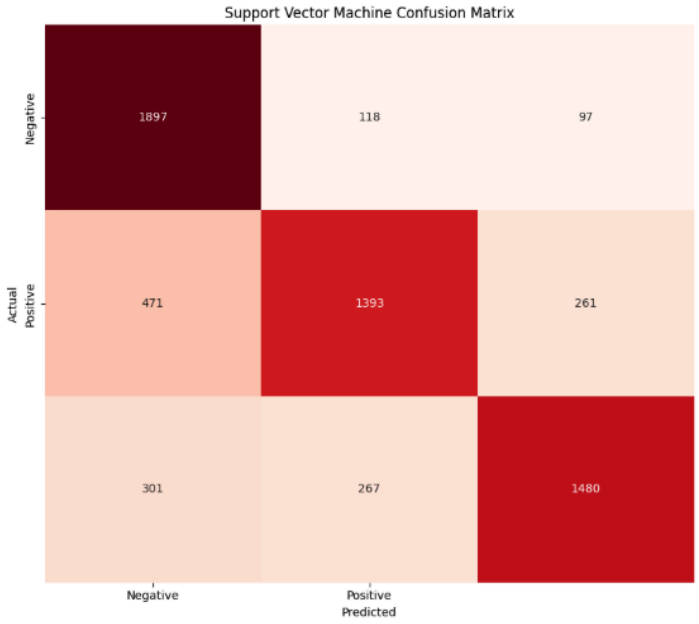


Figure 21: Visualisation of Support Vector Machine's confusion matrix

Model	Accuracy	F1 Score	Precision	Recall
Logistic Regression	50%	49%	51%	50%
Decision Tree	96%	96%	96%	96%
Random Forest	99%	99%	99%	99%
K-Nearest	93%	93%	93%	93%
SVM	76%	76%	77%	76%

Table 1: Comparison table for different models with different evaluation metrics

4. Best model selection and models saving

After evaluating the models with different methods and with a confusion matrix, I decided to go with the Random Forest model. Although the **accuracy, F1 Score, Precision and Recall metrics** seem to have a slight chance of the model being overfitted, when considering the evaluation matrix with the other models, Random Forest performs way better than the rest of the models. The best model and the rest of the models are being saved to six different files using the below piece of code.

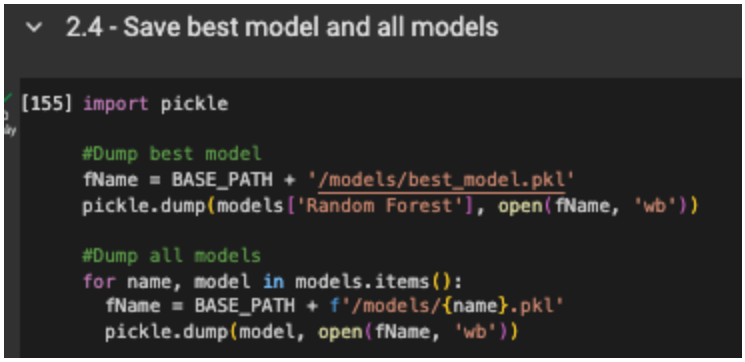


Figure 22: Best model and models saving process

Step 3: ML to AI

1. *Data processing and predicting process*

Figure 23: Import real time dataset

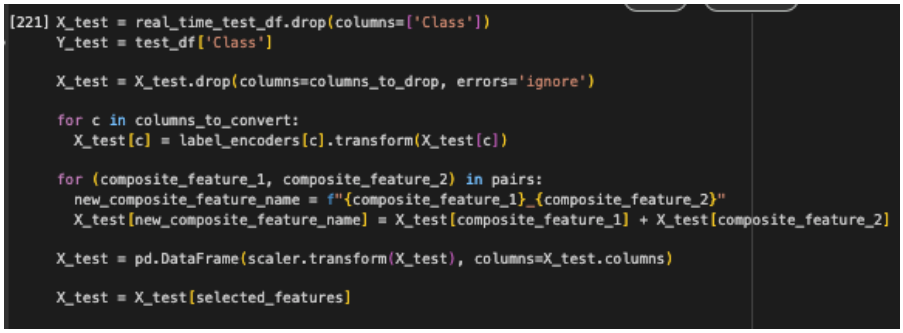


Figure 24: Data processing process

```
[215] best_model = pickle.load(open('models/best_model.pkl', 'rb'))

      correct_predict = 0

      for idx, row in X_test.iterrows():
        X_2d = row.values.reshape(1, -1)
        Y_real = Y_test.iloc[idx]
        Y_predict = best_model.predict(X_2d)[0]
        if Y_predict == Y_real:
          correct_predict += 1

      acc = correct_predict / len(X_test)

      print(f"Correct predict: {correct_predict}/{len(Y_test)}")
      print(f"Accuracy: {acc:.2f}")

      print(classification_report(Y_test, best_model.predict(X_test)))
```

Figure 25: Use the best model for predicting and generating predict results

```
[216] saved_models = {
        'Decision Tree': pickle.load(open('models/Decision Tree.pkl', 'rb')),
        'K-Nearest Neighbors': pickle.load(open('models/K-Nearest Neighbors.pkl', 'rb')),
        'Logistic Regression': pickle.load(open('models/Logistic Regression.pkl', 'rb')),
        'Random Forest': pickle.load(open('models/Random Forest.pkl', 'rb')),
        'Support Vector Machine': pickle.load(open('models/Support Vector Machine.pkl', 'rb'))
    }

    accuracies = {}
    results = {}

    for name, model in saved_models.items():
      Y_pred = model.predict(X_test)
      accuracies[name] = accuracy_score(Y_test, Y_pred)
      results[name] = classification_report(Y_test, Y_pred)

    for name, acc in accuracies.items():
      print(f"Accuracy of {name}: {acc:.2f}")
```

Figure 26: Use the rest models for predicting and generating predict results

2. *Prediction result*

Model	Accuracy
Logistic Regression	19%
Decision Tree	34%
Random Forest (Best Model)	33%
K-Nearest	49%
SVM	49%

Table 2: Prediction results of 5 models with 1000 unseen data points

For the 1000 unseen data points, the results are significantly different. The best model, which is the Random Forest, only performs 33% of accuracy score, while Logistic Regression and Decision Tree are having the scores of 19% and 33% respectively. The models that have the most accurate prediction are K-Nearest and Support Vector Machine, which are 49%.

Step 4: Develop rules from the ML model

```
[217] raw_shuffled_df = pd.read_csv('shuffled_vegemite.csv')
      shuffled_df = raw_shuffled_df.copy()

      X_shuffled = shuffled_df.drop(columns=["Class"])
      Y_shuffled = shuffled_df["Class"]

      X_shuffled = X_shuffled.filter(like='SP')
      X_shuffled.columns
```

Figure 27: Filter out features that include keyword **"SP"**

```
from sklearn.tree import export_text, plot_tree

model = DecisionTreeClassifier(random_state=42)
model.fit(X_shuffled, Y_shuffled)

tree_rules = export_text(model, feature_names=X_shuffled.columns.tolist())
print(tree_rules)
```

Figure 28: Generate the rules using `DecisionTreeClassifier`

The Decision Tree structure is then generated and could be listed as the following table:

Class	Rule	Condition
0	1	TFE Vacuum pressure SP > -76.93
0	2	TFE Production solids SP < 37.68
0	3	FFTE Steam pressure SP > 109.00
0	4	TFE Production solids SP > 84.00 and PTTU Steam pressure SP > 109.50
1	1	TFE Production solids SP > 84.00 AND PTTU Steam pressure SP > 109.50

1	2	TFE Production solids SP > 84.00 AND TFE Vacuum pressure SP > -56.38 AND FFTE Steam pressure SP < 110.96
1	3	TFE Production solids SP > 84.00 AND TFE Vacuum pressure SP > -56.38 AND PTTU Feed flow SP > 10030.00
1	4	TFE Out flow SP <= 2068.23 AND PTTU Out steam temp SP < 50.11 AND FFTE Steam pressure SP <= 114.35
2	1	TFE Vacuum pressure SP > -57.26
2	2	TFE Production solids SP < 48.52 AND PTTU Feed flow SP > 9460.69
2	3	PTTU Steam pressure SP <= 110.96
2	4	TFE Vacuum pressure SP > -42.68

Table 3: Decision Tree Rules for each class

Appendix

- Source code + datasets: https://github.com/cobeo2004/COS40007/tree/main/Assignment3
- Decistion Tree rules: https://github.com/cobeo2004/COS40007/blob/main/Assignment3/tree_rules.txt
- Saved models: https://github.com/cobeo2004/COS40007/tree/main/Assignment3/models