Supervised Machine Learning Tasks on Statlog (Vehicle Silhouettes) Data Set

Project Report

Project Members:

Yik Kien Tse

Jiayun Kang

Gagan Kumar B. A.

Atsumi Hirose

Qianxi Feng

1. Introduction

- Purpose/Use of the task: The dataset is taken from real-world images of vehicles and can be used to train machine-learning models to categorize various kinds of vehicles based on their silhouettes. This has real-world uses in industries like transit and surveillance. Additionally, the dataset's 18 unique features, each of which describes a different aspect of the car silhouette, make it possible to teach machine learning models to recognize multiple patterns and make accurate predictions.
- Learning tasks used:
  - Decision Trees
  - Instance-based learning
  - Neural networks
  - Bayesian Learning
  - Model Ensembles
- Performance measure: Accuracy and Precision shall be used to measure performance. Accuracy calculates the proportion of instances that are properly classified. It can be determined by dividing the number of correctly predicted labels by the overall number of true labels. Precision calculates the ratio of true positive results to all positive results. It can be determined by dividing the number of true positives by the total of true positives and erroneous positives. The confusion matrix gives a holistic interpretation of the performance of the learning tasks.
- Manual approach:
  - The first step is to clean and preprocess the data as required. This will include removing any empty values and encoding any categorical factors.
  - Second, choosing a subset of characteristics that are most important for classification. Statistical techniques like principal component analysis, feature significance ranking, and correlation analysis could be used to accomplish this.
  - Next, visual analysis can be done by looking over the images of the car silhouettes in the dataset and looking for any patterns or traits that can be used to categorize them. This could apply to the car's form, size, aspect ratio, or other characteristics.
  - Later, a set of criteria can be specified for categorizing the vehicles into four groups based on the visual analysis and feature selection (bus, van, saab, and Opel). If a car has a specific aspect ratio and a lot of corners, for instance, we can describe it as a bus.
  - Further, each piece of data is sorted into one of the four groups by applying the classification rules to the dataset.
  - Finally, by comparing the manual classification outcomes with the original class labels in the dataset, the performance of the above process can be evaluated in terms of accuracy and precision.

2. Learning Tasks

The learning task for this dataset is classification. Predicting the class of an object based on its features is the objective of the supervised learning activity known as classification. In mathematics, classification is modeled as a function that converts a collection of input features (x) into a set of output classes (y):

$$y = f(x);\ x \text{ has dimension of } (n * m)$$

Where n is the number of data, m is the number of features, x is a matrix of input features with dimensions (n x m), and y is a vector of output classes with dimension (n x 1). In a classifier, the function f generates the value 'y' from the input 'x'.

1. Decision trees: Detailed explanation is provided in section 5.
2. Instance-based learning:
   K-Nearest Neighbors (KNN) is a supervised learning algorithm used for the classification task here. Given a new input sample, KNN determines its class or label based on the class or label of its K-nearest neighbors in the training data.

   If X is the input dataset consisting of n samples, each with m features: X = {x_1, x_2, ..., x_n}, where x_i = (x_i1, x_i2, ..., x_im) is a feature vector for ith sample.

   if Y = {y_1, y_2, ..., y_n} is the corresponding class labels or regression targets for each sample in X.

   The KNN algorithm takes two inputs: A new sample x_q, represented as a feature vector of m dimensions and the number of nearest neighbors to consider, k.

   KNN outputs a class label for the new sample x_q: In the classification task, the output is the majority class label of the k-nearest neighbors of x_q.

   The distance between two samples is calculated using a distance metric, such as Euclidean distance or Manhattan distance. The algorithm then finds the k-nearest neighbors of the new sample x_q based on the distance metric and outputs the corresponding class or target value.
3. Bayesian:
   Bayesian classifiers are probabilistic classifiers, in that the methods return p(h|D) or the probability of a hypothesis given data D and assign the most likely class to a given sample. That is, they solve for the maximum aposteriori hypothesis, h_{map}.

$$h_{map} = \arg\max P(h|D)$$
$$= \arg\max \frac{P(D|h)P(h)}{P(D)} \quad \text{...(from the Bayes Theorem)}$$
$$= \arg\max P(D|h)P(h)$$

4. Neural network: Detailed explanation is provided in section 5.
5. Model Ensemble: Detailed explanation is provided in section 5.

3. Material

- Data description: The Statlog (Vehicle Silhouettes) dataset consists of several car features that were taken from the silhouette pictures of these vehicles. The dataset includes a total of 846 instances, each representing a distinct vehicle, and each described by 18 features. Geometric and non-geometric characteristics, including compactness, circularity, distance circularity, radius ratio, pr. axis aspect ratio, max. length aspect ratio, scatter ratio, elongated ness, pr. axis rectangularity, and others, were used to extract the features from the images.
  Every instance in the dataset is given a label for the sort of car it is: bus, van, saab, or opel. The dataset was obtained from the UCI Machine Learning Repository. It is extensively employed in machine learning and pattern recognition research and instruction.

- Dataset summary: Number of instances: 846
  Number of features: 18
  Number of classes: 4 (bus, van, saab, opel)
  Class distribution:
  Bus: 218 (25.8%)
  Van: 199 (23.5%)
  Saab: 217 (25.6%)
  Opel: 212 (25.1%)

No	Attribute	Type	% missing value	Target attribute
1.	Compactness	Integer	0	Yes
2.	Circularity	Integer	0	Yes
3.	Distance circularity	Integer	0	Yes
4.	Radius ratio	Integer	0	Yes
5.	Pr.axis_aspect_ratio	Integer	0	Yes
6.	Max.length_aspect_ratio	Integer	0	Yes
7.	Scatter ratio	Integer	0	Yes
8.	Elongated ness	Integer	0	Yes
9.	Pr.axis_rectangularity	Integer	0	Yes
10.	Max.length_rectangularity	Integer	0	Yes
11.	Scaled_variance	Integer	0	Yes
12.	Scaled_variance.1	Integer	0	Yes
13.	Scaled_radius_of_gyration	Integer	0	Yes
14.	Scaled_radius_of_gyration.1	Integer	0	Yes
15.	Skewness_about	Integer	0	Yes
16.	Skewness_about.1	Integer	0	Yes
17.	Skewness_about.2	Integer	0	Yes
18.	Hollows_ratio	Integer	0	Yes

Data gathering and handling: Code to access the UCI URL and combine the nine .dat files into one.

```
import urllib.request
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Access the URL with the dataset
url_link          =          "https://archive.ics.uci.edu/ml/machine-learning-
databases/statlog/vehicle/"

# Defines the list of file names to be downloaded
data_names = ["xaa.dat", "xab.dat", "xac.dat", "xad.dat", "xae.dat", "xaf.dat",
"xag.dat", "xah.dat", "xai.dat"]

# Initialize an empty list to store the data from each file
list_data = []

# Loop over each file and download its contents
for file_name in data_names:
    # Define the full URL for this file
    file_url = url_link + file_name

    # Download the file and append its contents to the list
    response = urllib.request.urlopen(file_url)
    data = response.read()
    list_data.append(data)

# Combine the data from all the files into a single string
combined_data = b"\n".join(list_data)

# Write the combined data to a file
with open("combined_data.dat", "wb") as f:
    f.write(combined_data)
```

Note: The *combined_data.dat file should be deleted from the local directory each time the notebook is restarted. If not the data will be appended to the existing file. This might lead to improper classification.*

Initial Statistical data analysis:

1. *Looking for missing values in each attribute and the data type of each attribute*

```
import numpy as np

# Function to calculate the percentage of missing values
def percent_missing(dfx):
```

```
    missing_total = dfx.isnull().sum().sort_values(ascending=False)
    missing_percent = (missing_total / len(dfx)) * 100
    return pd.concat([missing_total, missing_percent], axis=1, keys=['Total',
'Percent'])

# Function to display the data types and percentage of missing values in each
feature
data_info = pd.concat([dfx.dtypes, percent_missing(dfx)], axis=1, sort=False)
data_info.columns = ['Data Type', 'Total Missing', 'Percent Missing']
print(data_info)
```

There were no missing values in the dataset.

2. Determining the statistical features of each attribute, such as mean, standard deviation, count, minimum, maximum values, etc.

```
dfx.describe()
```

Since there are 18 attributes, the output of the above code is partially shown below due to space constraints. The complete table of statistics can be seen in the *'Data_exploration.ipynb'*

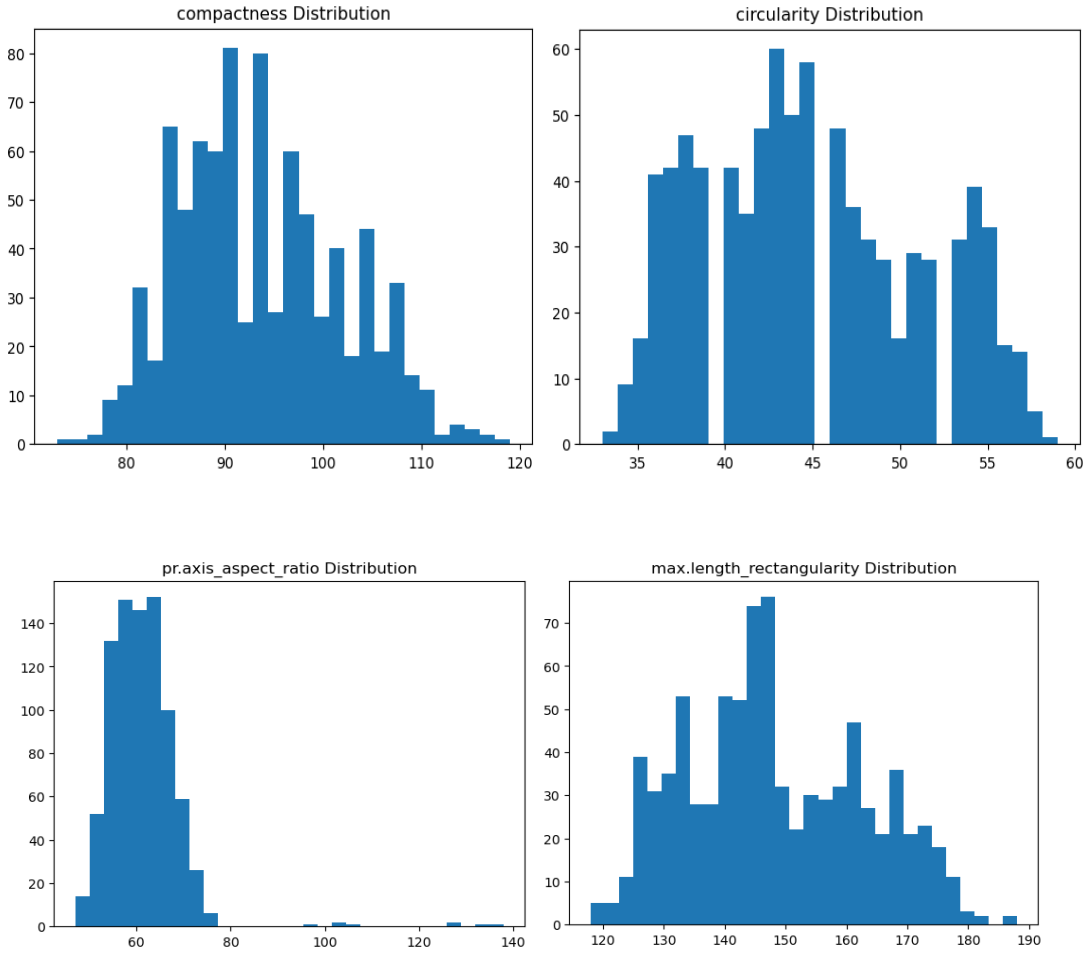
	compactness	circularity	distance circularity	radius ratio	pr.axis_aspect_ratio
count	846.000000	846.000000	846.000000	846.000000	846.000000
mean	93.678487	44.861702	82.088652	168.940898	61.693853
std	8.234474	6.169866	15.771533	33.472183	7.888251
min	73.000000	33.000000	40.000000	104.000000	47.000000
25%	87.000000	40.000000	70.000000	141.000000	57.000000
50%	93.000000	44.000000	80.000000	167.000000	61.000000
75%	100.000000	49.000000	98.000000	195.000000	65.000000
max	119.000000	59.000000	112.000000	333.000000	138.000000

Univariate and Bivariate Visualization

1. Histograms were plotted to visualize the individual attributes.

```
# loop over each column in the DataFrame
for col in dfx.columns:
    plt.figure()
    # create a histogram of the data
    plt.hist(dfx[col], bins=30)
    # add a title to the plot
    plt.title(col + ' Distribution')
    # show the plot
    plt.show().describe()
```

The result of the above code is shown below. Only four attributes are included in the report. The histograms of all the attributes can be seen in the *'Data_exploration.ipynb'*

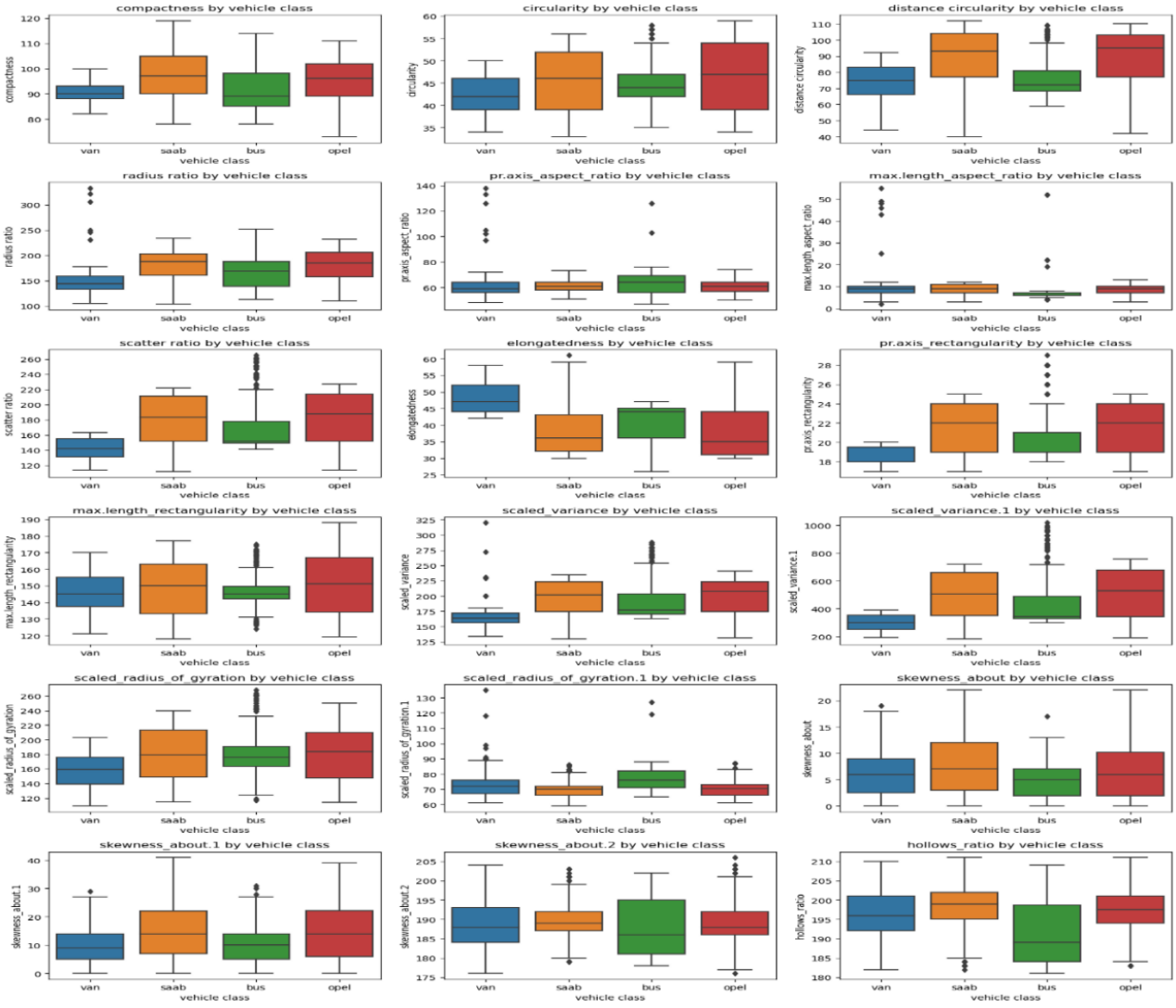


2. Box plots were plotted to visualize the relationship between classes and individual attributes.

```
# Setting figure size and layout
fig, axes = plt.subplots(nrows=6, ncols=3, figsize=(15,20))
axes = axes.flatten()

# Loop each attribute and create individual boxplot
for i, col in enumerate(dfx.columns[:-1]):
    sns.boxplot(x='class', y=col, data=dfx, ax=axes[i])
    axes[i].set_xlabel('vehicle class')
    axes[i].set_ylabel(col)
    axes[i].set_title(col + ' by vehicle class')

# Remove unutilised subplots
for i in range(len(dfx.columns), len(axes)):
    fig.delaxes(axes[i])
```

```
# Adjust subplot spacing
fig.tight_layout()

# Show the plot
plt.show()
```

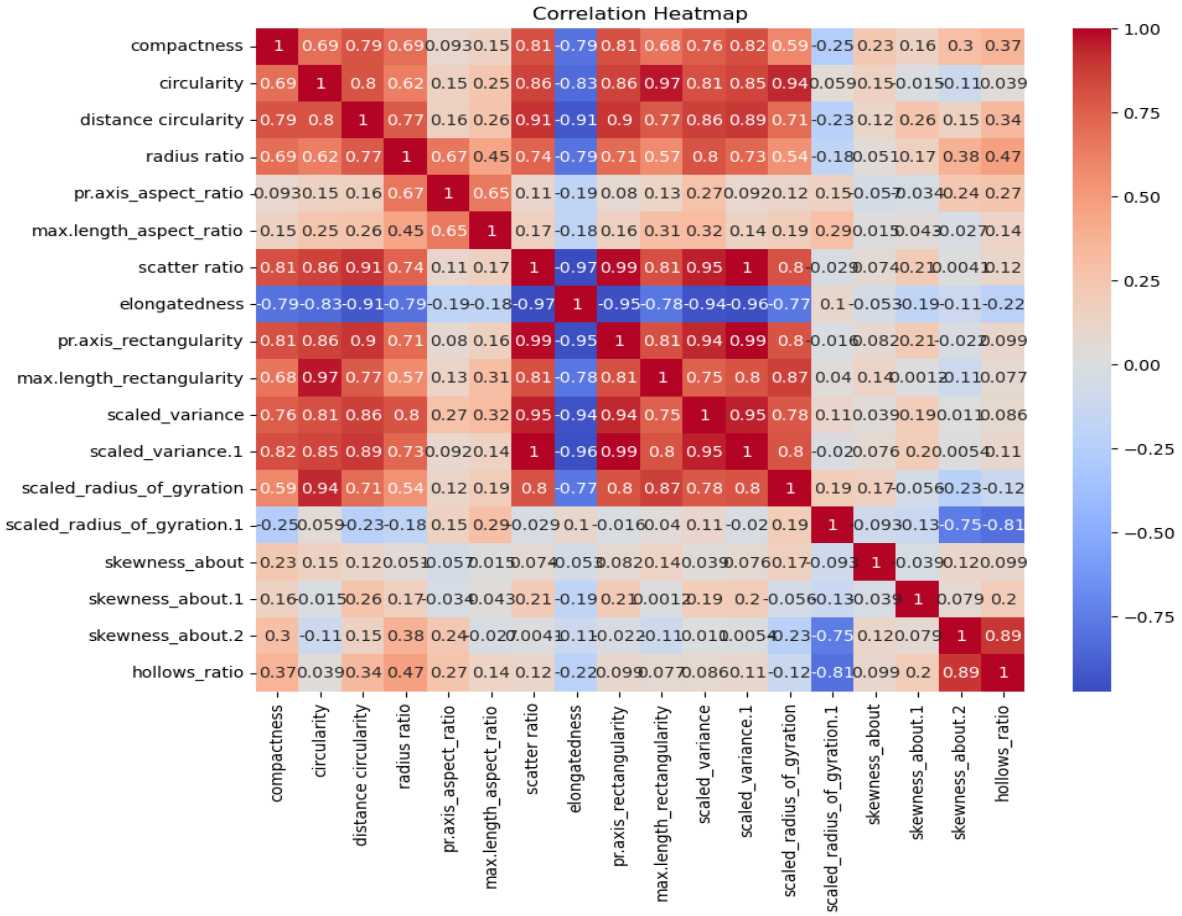
Mathematically, a boxplot summarizes the distribution of a set of data using five key values: the minimum value, the first quartile (Q1), the median, the third quartile (Q3), and the maximum value. The length of the box represents the interquartile range (IQR), which is the distance between the first and third quartiles. Any points outside the range of [Q1 - 1.5 x IQR, Q3 + 1.5 x IQR] are considered outliers and are plotted as individual points beyond the whiskers of the boxplot. The result of the above code is shown below.



Multivariate Visualization

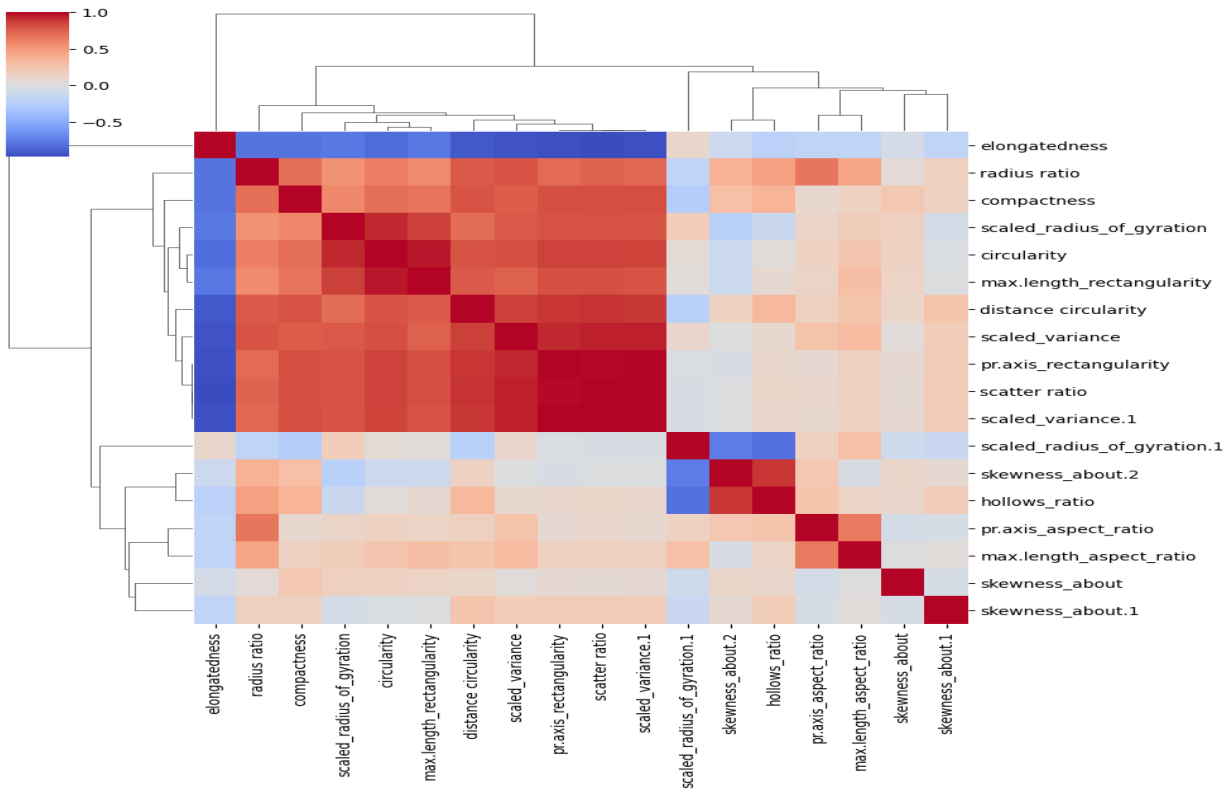
1. Determining the correlation between the attributes.

```
# Create Correlation heatmap of attributes
plt.figure(figsize=(10,8))
sns.heatmap(dfx.corr(), cmap='coolwarm', annot=True)
plt.title('Correlation Heatmap')
plt.show()
```



2. Visualizing the pattern/ relationship between attributes using clustered heatmap.

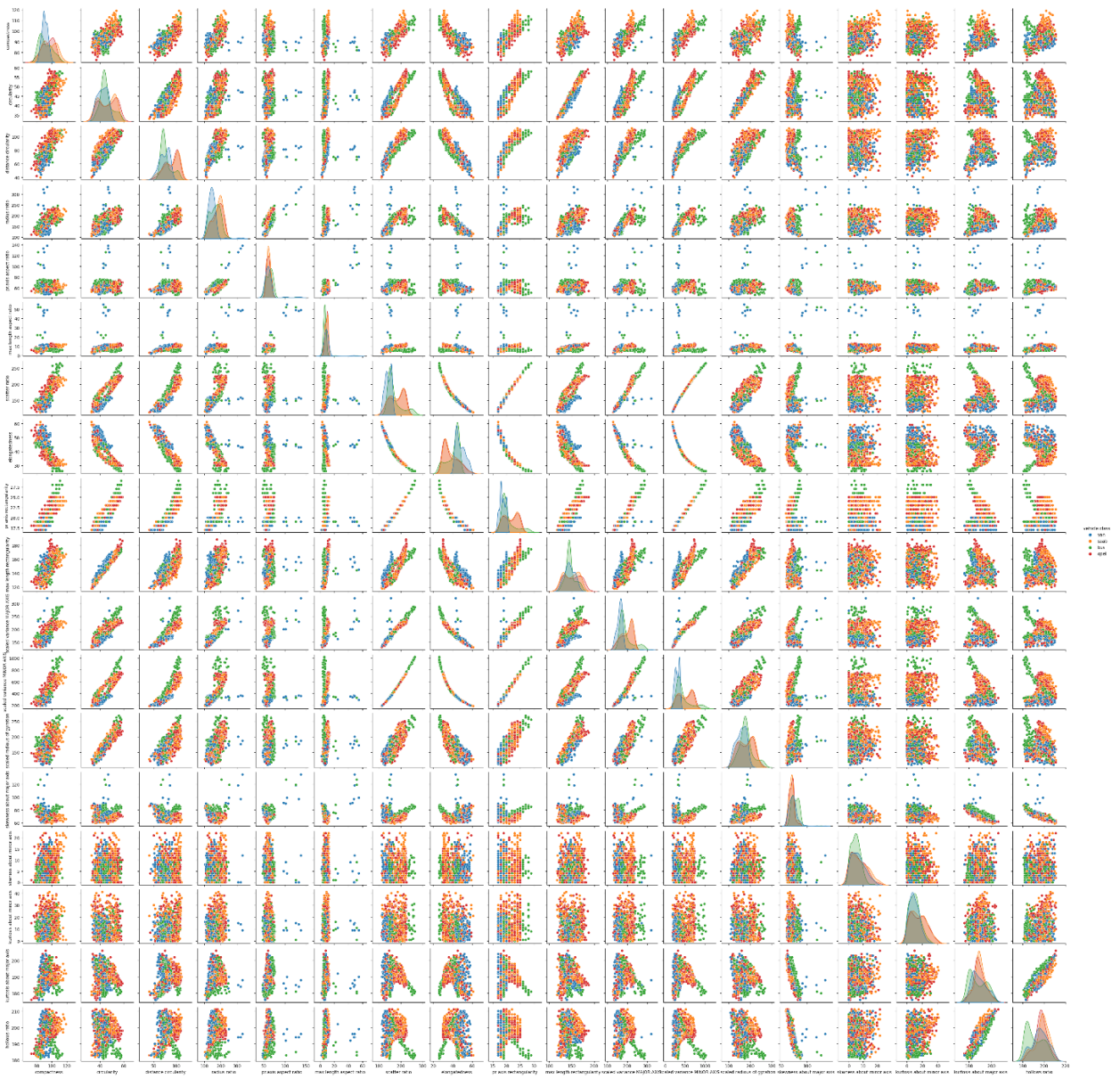
```
sns.clustermap(dfx.iloc[:, :18].corr(), cmap='coolwarm')
```



3. Visualizing the scatter plot of attributes.

```
# Plotting scatter plot

sns.set(style="ticks")

sns.pairplot(data=dfx,     vars=["compactness",     "circularity",     "distance
circularity",          "radius          ratio",          "pr.axis_aspect_ratio",
"max.length_aspect_ratio","scatter           ratio",           "elongatedness",
"pr.axis_rectangularity",    "max.length_rectangularity",    "scaled_variance",
"scaled_variance.1",                                "scaled_radius_of_gyration",
"scaled_radius_of_gyration.1","skewness_about","skewness_about.1","skewness_a
bout.2","hollows_ratio"], hue='class', diag_kind="kde")

plt.show()
```

The following scatter plot shows that some of the attributes are highly correlated as shown by the straight line, and have skewness. The skewness is due to the outliers.



Performing additional statistical analysis

1. Determining imbalance ratio.

```
# Count the number of instances in each class
class_counts = dfx['class'].value_counts()

# Print the class distribution
print('Class distribution:')
print(class_counts)

# Calculate the class imbalance ratio
imbalance_ratio = class_counts.min() / class_counts.max()
```

```
# Print the class imbalance ratio
print(f'Class imbalance ratio: {imbalance_ratio}')
```

Result of the above code

```
Class distribution:
bus     218
saab    217
opel    212
van     199
Name: class, dtype: int64
Class imbalance ratio: 0.9128440366972477
```

The dataset is fairly balanced.

Decision on Outliers and Correlation of attributes

From all the boxplot visualization it was found that there are some outliers in the data. But the data is highly correlated, deleting or replacing the outliers may distort the distribution of the data. Hence the decision was made to not replace the outliers from the original data. Although there are some attributes that are more highly correlated with others, it is decided to not exclude the less correlated attributes at this stage. Hence no common transformations are made.

Data Sampling

The dataset was divided after performing the preprocessing. The algorithm would be trained on 80% of the data and tested on the leftover 20%. The divided datasets were then saved to their corresponding CSV files so that the learning algorithms could use them. *These files are used only for the KNN and Bayes algorithms, the other algorithms use the combined.dat file to split the dataset into train and test depending on the characteristics of the algorithm.* A validation set is not necessary because validation was accomplished using k-fold validation on the testing set.

```
from sklearn.model_selection import train_test_split

# Split the data into features and target
X = dfx.drop('class', axis=1)
Y = dfx['class']

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
stratify=Y, random_state=42)
```

```
# Save the training and testing sets to CSV files
X_train.to_csv('train_features.csv', index=False)
X_test.to_csv('test_features.csv', index=False)
Y_train.to_csv('train_target.csv', index=False)
Y_test.to_csv('test_target.csv', index=False)
```

Bayesian specific transformations: We analyzed the distribution of each attribute and its skewness. Below shows the skewness of each of the attributes.

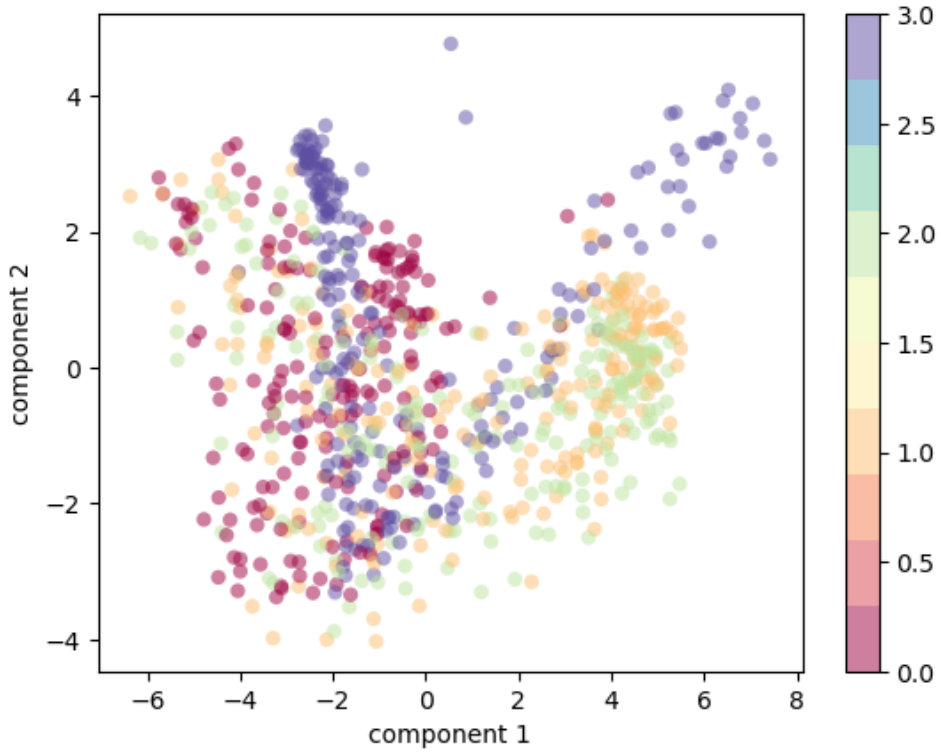
```
compactness                  0.379439
circularity                  0.262021
distance circularity         0.104986
radius ratio                 0.389976
pr.axis aspect ratio         3.821908
scatter ratio                0.606293
elongatedness                0.046447
pr axis rectangularity       0.771073
max length rectangularity    0.255280
scaled variance MAJOR AXIS   0.653124
scaled variance MINOR AXIS   0.835780
scaled radious of gyration   0.279064
skewness about major axis    2.071636
skewness about minor axis    0.772643
kurtosis about major axis    0.246930
hollows ratio               -0.228617
```

As shown above, we found some of the attributes moderately skewed (e.g., ‘scatter ratio”, “pr axis rectangularity”, “scaled variance Major Axis”, “scaled variance MINOR AXIS”, “skewness about minor axis”) whilst others were highly skewed (e.g., pr. axis aspect ratio”, “skewness about major axis”). For the classifier that assumes the normal distribution (i.e., Gaussian Naive Bayes), these were log-transformed before training the classifier.

Due to the number of features, we conducted the principal component analysis to reduce the dimensionality and to visualize the dataset. We first standardised the dataset (ie centered at 0 with unit variance) using the scaler function of sklean preprocessing package, as below before conducting principal component analysis.

```
scaler = StandardScaler(with_mean=True, with_std=True)
dfx_scaled = scaler.fit_transform(dfx)
pca = PCA()
dfx_pca = pca.fit_transform(dfx_scaled)
```

Plotting the first two components (which explains roughly 70% of variance), we observed that data were overlapping especially between ‘Opel (in orange) and Saab (in yellow/green). That is, the two classes seem to share similar characteristics. Note: the labels are enumerated as `{'van': 0, 'opel': 1, 'saab': 2, 'bus': 3}`.



We also examined the correlations between attributes. Some of the attributes such as scatter ratio (column 7), elongated ness (column 8) , and pr axis rectangularity (column 9) were highly correlated with some variables. For the Naïve Bayes classifier, where independence of features is assumed, these features were removed from the dataset before training.

4. Technology

The project's primary development medium was Jupyter Notebooks. Dependent Python programs could be executed using the web-based interactive software, and the results of each code block were shown right beneath it. The software also enables the inclusion of rich text components, which were used to add details about the operation of critical code blocks.

Conda was used to handle all necessary dependencies in addition to the default libraries included in Python 3.8:

1. To extract data from the DAT documents, preprocess the data, and export the pertinent train/test splits into new CSVs, Urllib, OS, Pandas, and Numpy were used.

2. Data visualization was carried out using MatPlotLib during the preliminary data analysis step. The confusion matrices for each learning method were also shown using it. Histogram was also plotted using this.

3. Seaborn: Box plot, correlation cluster map, and scatter plot visualization were done using this program during experimental data analysis. It was also applied to the correlation matrix's creation.

4. JobLib & Pickle: used to load and export the tuned models into the model ensemble from each notebook. This was done to prevent having to train the models repeatedly.

5. Each learning model was created using Scikit-Learn, which was also used to validate the models, adjust their hyper-parameters (via GridSearch), show accuracy and precision scores, and calculate the confusion matrix. Finally, Git and Github were used for source code management and ensuring that each team member had access to the latest version of the notebooks.

5. Learning Algorithms

a. Decision Trees:

The decision tree is a non-parametric supervised learning approach designed for making classification and regression. From the IBM platform, this method is based on a hierarchical 'tree structure', which includes the root node, internal nodes, leaf nodes, and branches or edges (figure 1). The root node on the top is the start of decision trees, and the following edges can be considered as the condition where the data in the root node will be separated. The internal nodes are the subgroups after the previous classification, which are going to be split into new nodes based on the best features or attributes in these groups. As the classification goes on, the purity of the target variable in each subgroup is expected to become higher. Finally, the leaf nodes will show the result of decision trees, which predict the class of instances. Based on the adequate data volume, decision trees aim to learn the decision rules from the train datasets to build the model and predict the target variable.

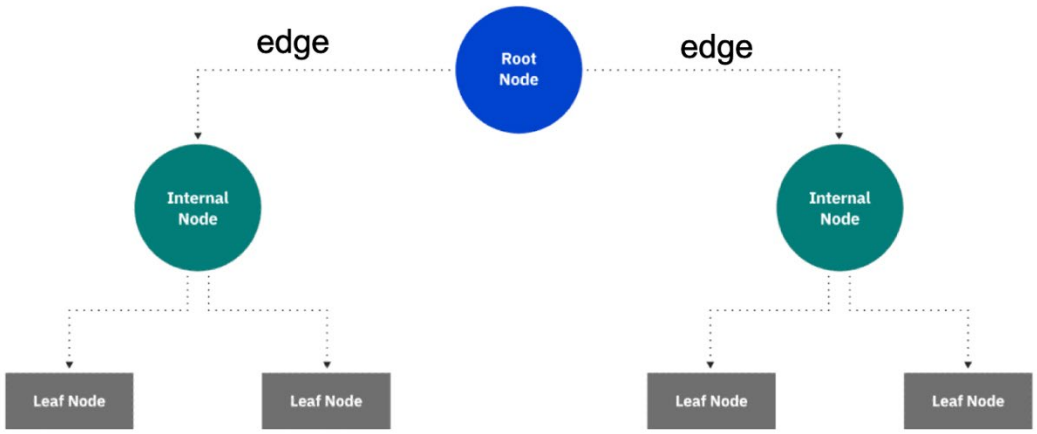


Figure 1. The basic structure of a decision tree (Source: https://www.ibm.com/uk-en/topics/decision-trees#:~:text=A%20decision%20tree%20is%20a,internal%20nodes%20and%20leaf%20nodes).

Three common algorithms in decision trees (DT):

A. In the Iterative Dichotomiser 3 (ID3) algorithm, the 'Entropy' (S) measures the disorder of the target variable in a dataset (eq.1). For example, if there is only one class of instances in the subgroup (node), the entropy for that node would be minimum (equal to 0). In addition, the 'Information Gain' (IG) represents how much reduction of the target variable's entropy can be obtained after using one attribute to split the dataset (eq.2). The core of ID3 algorithm is to create the DT nodes using the best feature which has the maximum IG. However, this algorithm can only handle discrete values (categorical datasets).

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i)) \quad \text{Eq. 1}$$

where: $p(c_i)$ is the probability of class c_i;
n is the total number of classes of the target variable.

$$Information\ Gain = Entropy_{parent} - \sum Entropy_{child} \quad \text{Eq. 2}$$

where: $Entropy_{Parent}$ is the entropy in the parent node in decision tree;
$\sum Entropy_{child}$ is the total entropy in the child nodes in decision tree after 'splitting' the parent node.

B. C4.5 is quite similar to the ID3, but it uses Gain Ratio instead of IG to determine which is the best attribute in a particular stage of classification (eq.3). Since ID3 has a bias on the features with more distinct values, the involvement of Gain Ratio will solve that issue. Compared with ID3, C4.5 is generally more reliable. Besides that, C4.5 can also be used for handling the continuous datasets by building the threshold values.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} \quad \text{Eq. 3}$$

C. Classification and Regression Tree (CART) uses Gini Impurity (GI) with a range from 0 to 1 for splitting the dataset (eq. 4). GI reflects the probability of randomly taking two distinct values from a dataset. When it is equal to 0, there is only one class of instance in the dataset. This algorithm also creates the threshold values to split the continuous datasets. On the contrary with C4.5, if one attribute has been used for the previous classification in parent node, that attribute can still be used again in the child nodes in CART.

$$Gini\ Impurity = 1 - \sum_{i=1}^{n} P\_i^2 \quad \text{Eq. 4}$$

where: n is the total number of classes of the target variable,
'P_i' is the probability of class 'i'.

In this assignment, the CART algorithm will be selected to build the decision tree, and several reasons are shown below:

- Be able to handle the continuous dataset (our dataset contains 18 attributes with continuous values.

- Easy to implement in Jupyter Notebook. The only requirement is to install the 'scikit-learn'. In the Sklearn library, there are great numbers of useful tools to build the model with CART, and visualise the tree structure like 'DecisionTreeClassifier', 'plot_tree', etc.

- Allow the attributes or features to be used for multiple times during the entire classifying process.

- There are no special preprocessing steps needed in this algorithm. It is quite kind to the users because even the nonlinear relation between the attributes will not influence the model performance.

Implementation:

i. Import the data and create the training and test datasets:

After the group members removed some outliers from the original dataset, I just used the 'pandas' package to import the new data file called 'cleaned_data.dat' directly and split the date frame into 'x' (18 attributes) and 'y' (target variable – vehicle type). Since there are totally 846 observations, I planned to use the 35% of them as the training dataset to build the model, and the rest of them would be the testing dataset. Here, 'train_test_split' function is used to randomly split the data (x_train and y_train are the attributes and the vehicle type in the training dataset, respectively).

```
# spliting the training and testing datasets:
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.35)
x_train  # 296 sets
x_test   # 550 sets
y_train  # 296 sets
y_test   # 550 sets
```

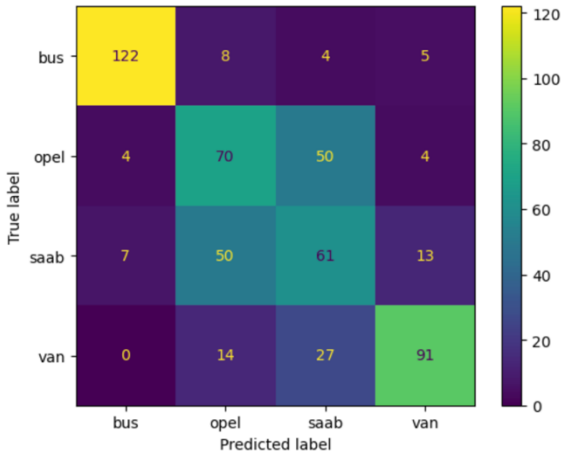
ii. The model with default hyperparameters:

Initially, the 'DecisionTreeClassifier()' with its default hyperparameters is applied to build the decision tree called 'clf_default', based on the training dataset. The prediction of the target variable (vehicle type) is also obtained while inputting the 'x_test' in the model.

```
# Create Decision Tree classifer object
clf_default = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf_default = clf_default.fit(x_train,y_train)
y_pred_default=clf_default.predict(x_test)

# Prediction Accuracy:
print("Accuracy:", clf_default.score(x_test, y_test))
```

Then, compare the model prediction with the actual target variable values in the 'y_test' and calculate the accuracy. From the result, although the overall precision is around 65%, the accuracies of predicting the Opel and Saab vehicles are quite poor which are 49% and 43%, respectively. The confusion matrix also indicates most of the errors occur while predicting the Opel and Saab (the values on the diagonal line represents the number of cases classified correctly).

```
# Create Decision Tree classifer object
clf_default = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf_default = clf_default.fit(x_train,y_train)
y_pred_default=clf_default.predict(x_test)

# Prediction Accuracy:
print("Accuracy:", clf_default.score(x_test, y_test))

# Visulise the model prediction error:
plot_confusion_matrix(clf_default, x_test, y_test)
plt.figure()
plt.show()
print(classification_report(y_test, y_pred_default))
```



	precision	recall	f1-score	support
bus	0.92	0.88	0.90	139
opel	0.49	0.55	0.52	128
saab	0.43	0.47	0.45	131
van	0.81	0.69	0.74	132
accuracy			0.65	530
macro avg	0.66	0.64	0.65	530
weighted avg	0.67	0.65	0.66	530

iii. Tuning the hyperparameters and 10-fold cross validation:

The default model aims to fit the training dataset as perfectly as possible, which cause the tree becomes quite complicated and not able to accurately predict the target variable in the testing dataset. In the 'DecisionTreeClassifier()', there are several parameters worthy to be tuned, especially the last three shown below, which can prevent the overfitting issue of the model efficiently.

- 'criterion': In CART algorithm, the Gini Impurity (GI) (eq. 4) is used by default, but we can also choose Entropy to measure the purity in the subgroups of dataset.
- 'max_depth' is used to limit the maximum depth that the tree structure can reach.
- 'min_samples_split' is set to determine how many samples are required to have at least in the internal node before classifying that node.
- 'min_samples_leaf' is the minimum number of samples required in each leaf node.

Two models are created with different 'criterion' options and the same other parameters, based on the training dataset (TD). Then, the 10-fold cross validation will be performed on TD. In this process, TD is automatically split into 10 sets, and there is one random set considered as the validation, the other nine are training sets. After 10 times of validation the parameter value with the highest mean validation score will be the most appropriate for building the decision tree. In

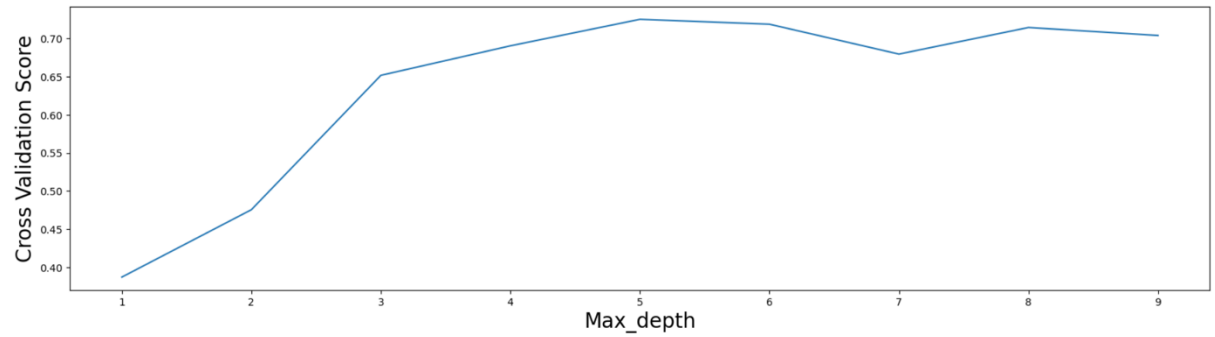
this case, according to the result (Gini score = 0.67; Entropy score = 0.7), ‘entropy’ is assumed as the best.

```
# Tune the 'criterion' parameter:
DT = DecisionTreeClassifier(criterion = 'gini')
score = cross_val_score(DT,x_train,y_train,cv=10).mean()
print('Gini Score: %.4f'%score)
DT = DecisionTreeClassifier(criterion = 'entropy')
score = cross_val_score(DT,x_train,y_train,cv=10).mean()
print('Entropy Score: %.4f'%score)
# The Entropy score is slightly higher than the Gini Score.
# So for the later tuning process, i will keep using the 'entropy'
as the criterion.
```

After that, to get the range of the best ‘max_depth’, a ‘for’ loop is used to make max_depth equal to 1, 2, 3, …, up to 10 in sequence, and compare the scores of 10-fold cross validations. And the graph below shows the change of score with different max_depth values.

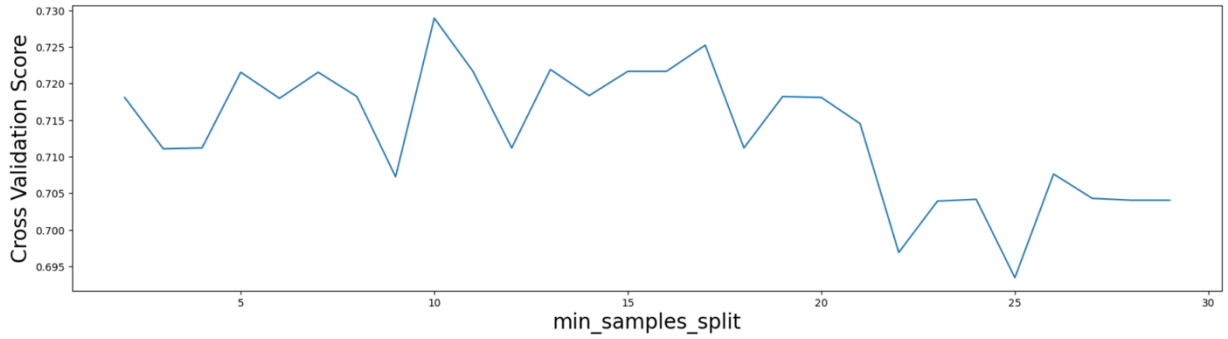
```
# Tuning the 'max_depth' with best the 'criterion':
ScoreAll = []
for i in range(1,10,1):
    DT = DecisionTreeClassifier(max_depth = i, criterion =
'entropy')
    score = cross_val_score(DT,x_train, y_train,cv=10).mean()
    ScoreAll.append([i,score])
ScoreAll = np.array(ScoreAll)

max_score = np.where(ScoreAll==np.max(ScoreAll[:,1]))[0][0]  #
find the indexes of the parameter with highest score
assumed_best_max_depth = int(ScoreAll[max_score][0])
print("The best max_depth: ", assumed_best_max_depth)
print("The score of the best max_depth: ",ScoreAll[max_score][1])
plt.figure(figsize=[20,5])
plt.plot(ScoreAll[:,0],ScoreAll[:,1], )
plt.xlabel("Max_depth", fontsize = 20)
plt.ylabel("Cross Validation Score", fontsize = 20)
plt.show()
```

```
The best max_depth:  5
The score of the best max_depth:  0.7251231527093596
```

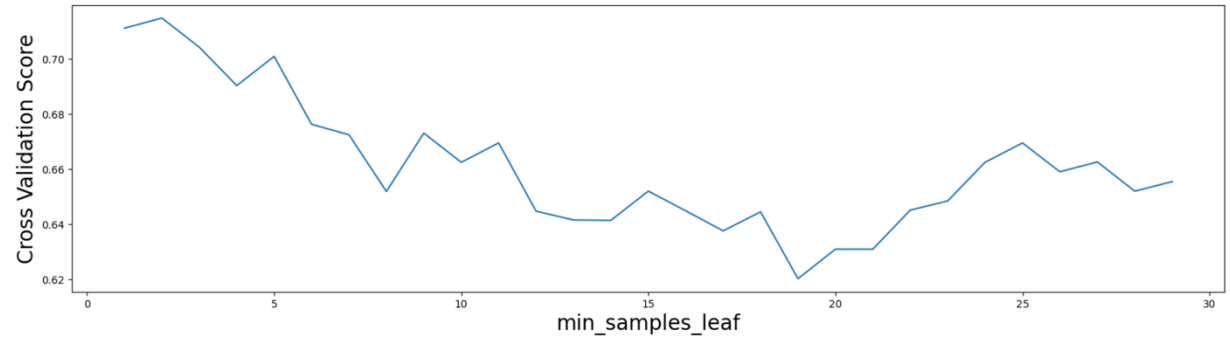


The graphs of validation score versus ‘min_samples_split’ and ‘min_samples_leaf’ can be also obtained in the same way, which are shown below:

```
The best min_samples_split:  10
The score of the best min_samples_split:  0.7289408866995075
```




```
The best min_samples_leaf:  2
The score of the best min_samples_leaf:  0.7147783251231528
```



Therefore, the appropriate ranges for max_depth, min_samples_split and min_samples_leaf are (6 ~ 9), (5 ~ 17) and (2 ~ 5), respectively. Now, it is going to tune these three hyperparameter together under the ranges found from the above graphs. In this step, the 10-fold cross validation is involved as well. Finally, a set of parameter values will be output according to the best validation score.

```
# Tuning the 'max_depth', 'min_samples_leaf' and 'min_samples_split' together!
param_grid = {
    'max_depth':np.arange(5, 9),
    'min_samples_leaf':np.arange(1, 6),
    'min_samples_split':np.arange(5, 18)
}

rfc = DecisionTreeClassifier(criterion = 'entropy')
GS = GridSearchCV(rfc,param_grid,cv=10)
GS.fit(x,y)
print(GS.best_params_)
print(GS.best_score_)
```

Result:

```
# Using the new hyperparameter which are tuned:
new_clf = DecisionTreeClassifier(criterion = 'entropy', max_depth = 8,
```

```
min_samples_leaf = 3, min_samples_split = 6)

# Train Decision Tree Classifer
new_clf = new_clf.fit(x_train,y_train)

# Prediction Accuracy:""
print("Accuracy:", new_clf.score(x_test, y_test))
```

The new decision tree is built based on the tuned hyperparameters (criterion = 'entropy', max_depth = 8, min_samples_leaf = 3, min_samples_split = 6). The overall prediction accuracy has increased from 64.72% to 70.38%, compared with the previous model (built with default parameter). Especially for classifications of Saab and Van types of vehicles, their accuracy rises by around 10%. Unfortunately, the classification of Opel and Saab vehicles is still quite poor, which might be resulted from the similar shapes of these two types of cars. Therefore, it could be very difficult for using this algorithm to distinguish these two classes of vehicle.

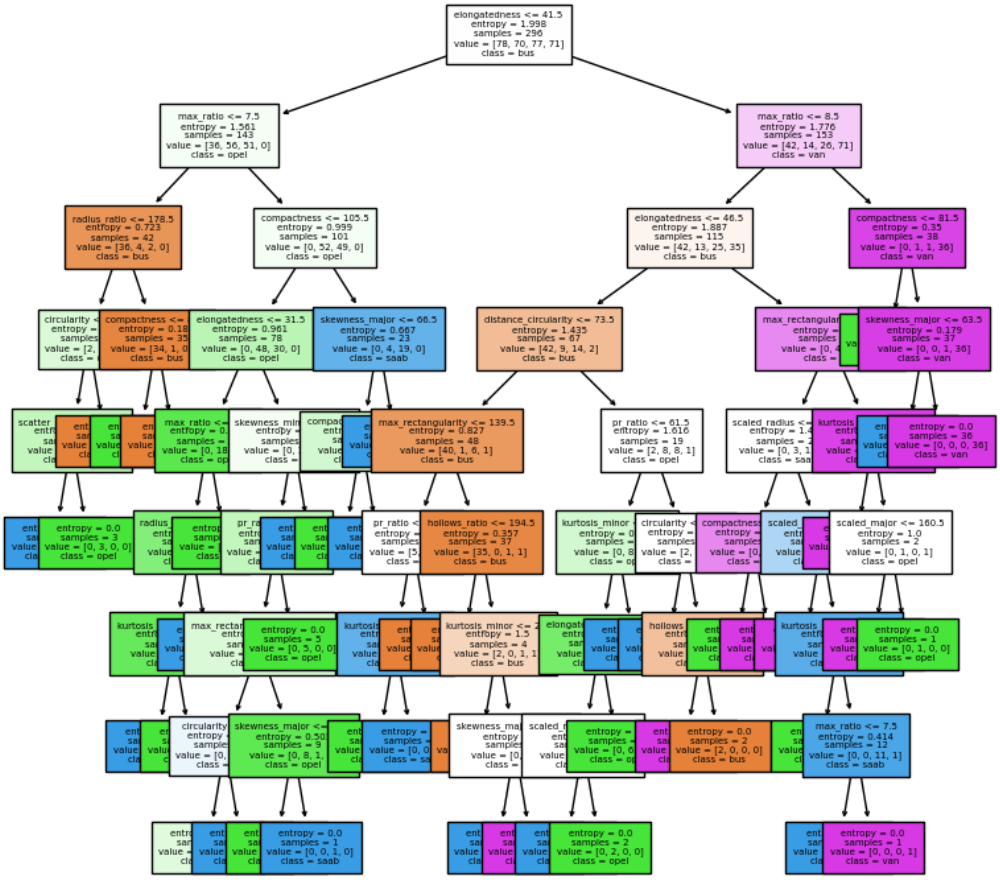
The classification report for the model with default hyperparameters:

	precision	recall	f1-score	support
bus	0.92	0.88	0.90	139
opel	0.49	0.55	0.52	128
saab	0.43	0.47	0.45	131
van	0.81	0.69	0.74	132
accuracy			0.65	530
macro avg	0.66	0.64	0.65	530
weighted avg	0.67	0.65	0.66	530

The classification report for the model with tuned hyperparameters:

	precision	recall	f1-score	support
bus	0.90	0.91	0.90	141
opel	0.47	0.38	0.42	125
saab	0.53	0.67	0.59	133
van	0.92	0.82	0.87	131
accuracy			0.70	530
macro avg	0.70	0.70	0.70	530
weighted avg	0.71	0.70	0.70	530

The final decision tree:



b. Instance-Based Learning

Instance-Based Learning (IBL) algorithms are a family of algorithms that are often called 'lazy' because they only perform computation when a new instance is observed. They store the training data in memory and use it to estimate the class of a new instance by locating the nearest training examples. These algorithms can handle both discrete and continuous inputs to predict discrete or continuous target functions.

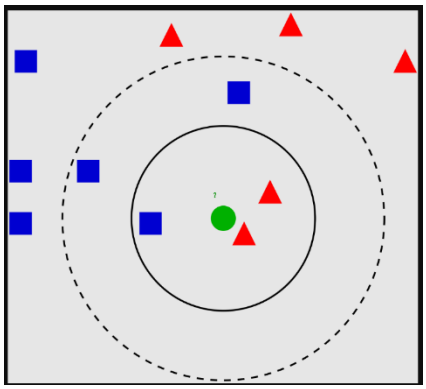
The three most common algorithms in the IBL family are Collaborative Filtering, Support Vector Machines, and K-Nearest Neighbours.

Collaborative Filtering is mainly used for recommendation systems, where historical user preferences are analyzed to estimate the preferences of individuals who are similar to them. The algorithm assumes that individuals who have agreed in the past will continue to do so in the future.

Support Vector Machines is a non-probabilistic binary classifier that determines the class of a new instance by viewing data through a p-dimensional vector space. It uses hyperplanes as decision boundaries to classify data points, whereas support vectors are data points that influence the orientation and positioning of a hyperplane. As a result, data points on either side of a p-dimensional hyperplane are identified as belonging to different classes.

K-Nearest Neighbours predicts the class for a new observation by calculating the distance between all the training data points and determining the probability of the instance belonging to a class of K training data. The steps in classifying an unseen observation involve finding the K-nearest neighbors in the training set, and assigning the class label that appears most frequently among these neighbors to the new observation, are:

- Choose the number of neighbors (N) to consider.
- Determine the distance between the new instance and each of the training data points using a distance metric such as Euclidean distance, Manhattan distance, or cosine distance.
- Select the N nearest neighbors based on the distance calculation in step 2 and count the number of data points in each class.
- Assign the most common class label among the N closest neighbors to the new instance.



Source: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

The figure above provides a visual representation of the KNN algorithm, where the number of neighbors is three. The three nearest neighbors to the new instance (circle) are selected, and as the number of triangles (2) is greater than the number of squares (1), the new instance is classified as a triangle.

There are various ways to calculate the distance between the new instance and the training data points in the KNN algorithm. Three common distance metrics are Euclidean distance, Manhattan distance, and cosine distance. The mathematical formulas for these distance metrics are presented below.

Manhattan Distance: The Manhattan distance (also known as taxicab distance or L1 distance) between two points (x1, y1) and (x2, y2) is calculated as the sum of the absolute differences of their coordinates:

Manhattan distance = |x1 - x2| + |y1 - y2|

Euclidean Distance: The Euclidean distance between two points (x1, y1) and (x2, y2) is the straight-line distance between them, calculated using the Pythagorean theorem:

Euclidean distance = sqrt((x1 - x2)^2 + (y1 - y2)^2)

Cosine distance: Cosine distance is a measure of similarity between two vectors, often used in text classification and recommendation systems. Given two vectors A and B, their cosine similarity is calculated as the dot product of the two vectors divided by the product of their magnitudes:

cosine similarity = dot product(A, B) / (magnitude(A) * magnitude(B))

The K-Nearest Neighbours (KNN) algorithm was chosen for this report because of its simplicity in implementation using the sklearn package.

Some advantages of KNN include:

- Easy to implement as the learning process happens at the time of prediction.
- New data can be added easily without the need for retraining.

However, there are also some drawbacks to consider:

- KNN may struggle with large datasets, as the distance between neighbors increases, making it difficult to calculate distances across multiple dimensions.
- KNN is sensitive to missing values and outliers in the dataset.
- Feature scaling (such as standardization) is required to improve the accuracy of predictions.

Preprocessing: Since the algorithm requires feature scaling, the dataset has been standardised using StandardScaler()

```
# Load the training and testing sets from CSV files
X_train = pd.read_csv('train_features.csv')
X_test = pd.read_csv('test_features.csv')
Y_train = pd.read_csv('train_target.csv')
Y_test = pd.read_csv('test_target.csv')

# Apply feature scaling to the training and testing sets
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
Y_train_change = Y_train.values.ravel()
```

Implementation: The model was first trained for the default parameters of the KNN classifier(). The default parameters are obtained below using `(knn.get_params()).`

{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}

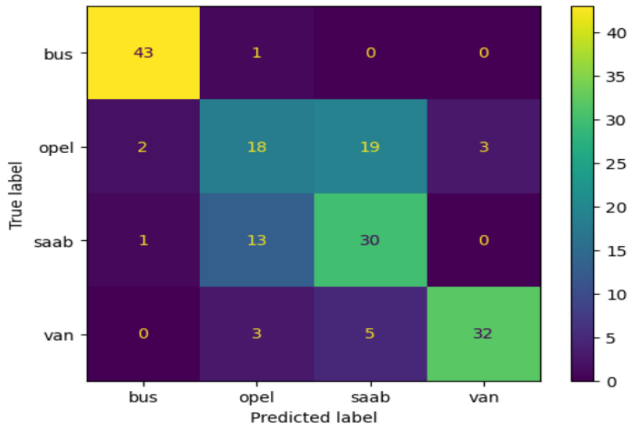
```
primary_knn = KNeighborsClassifier()
primary_knn.fit(X_train_scaled, Y_train_change)
```

The default classifier was then used to predict the test dataset.

```
predict_y_initial = primary_knn.predict(X_test_scaled)
```

The confusion matrix, accuracy, and precision score were obtained using the codes below along with the results.

```
ConfusionMatrixDisplay.from_estimator(primary_knn, X_test_scaled, Y_test)
plt.show()

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
print('Test accuracy of kNN', accuracy_score(Y_test, predict_y_initial))
print('Test precision of kNN:', precision_score(Y_test, predict_y_initial, average='weighted'))
```



Test accuracy of kNN
0.7235294117647059

Test precision of kNN:
0.7279194576381277

	A (actual)	B (actual)	C (actual)	D (actual)
A (predicted)	TP	Fopel	Fsaab	Fvan
B (predicted)	Fbus	TP	Fsaab	Fvan
C (predicted)	Fbus	Fopel	TP	Fvan
D (predicted)	Fbus	Fopel	Fsaab	TP

This table indicates the meaning of the above matrix.

TP = True positive, the diagonal elements, which are correctly classified. Fxyz, where xyz represents a class falsely(F) classified as a different class.

From the confusion matrix, class – bus has the highest True positives, and opel has the lowest true positives. Also, more instances of classes opel and saab are falsely classified as the other class.

Hyper Parameter tuning: The hyperparameters are tuned and cross-validated using the code below.

```
param_grid = [{
    'weights': ["uniform", "distance"],
    'n_neighbors': range(1, 25),
    'metric':['euclidean', 'manhattan', 'cosine'],}]

knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=2)
grid_search.fit(X_train_scaled, Y_train_change)
```

Weights, n_neighbors, and metrics are the three hyperparameters tuned for the range/ values as shown above. GridSearchCV is a method that allows us to explore a predefined grid of hyperparameters in order to find the best combination of parameters for a given estimator or model using the training set. The parameter n_fold is set to 5, which means that the cross-validation process will use 5 folds or subsets of the data to evaluate the model.

The cross-validation result is displayed as data frame using the code below.

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

The best score and corresponding hyperparameter details are obtained using the following.

```
# printing the best accuracy score and hyperparameters

print("Best Score is: ", grid_search.best_score_)
print("It was achieved using the following hyper-parameters: ")
print(grid_search.best_params_)
```

```
Result:
Best Score is:  0.7338235294117647
It was achieved using the following hyper-parameters:
{'metric': 'cosine', 'n_neighbors': 13, 'weights': 'uniform'}
```

The classifier is trained with the above hyperparameters.

```
# model with best hyperparameters
model_tuned = KNeighborsClassifier(metric = 'cosine',
                        n_neighbors = 13,
                        weights = 'uniform')

model_tuned.fit(X_train_scaled, Y_train_change)
```

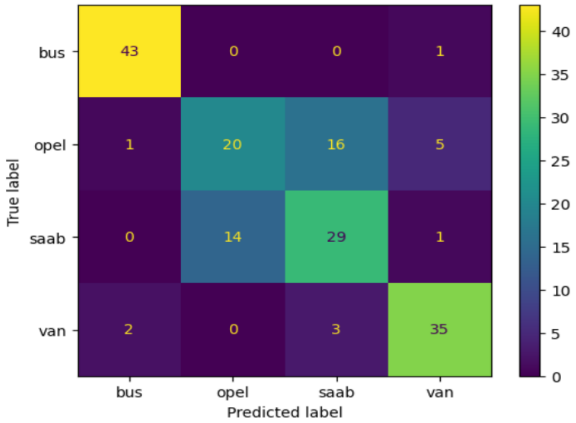
The tuned classifier was then used to predict the test dataset.

```
predict_y_test = model_tuned.predict(X_test_scaled)

print('Test accuracy of kNN', accuracy_score(Y_test, predict_y_test))
```

```
print('Test precision of kNN:', precision_score(Y_test, predict_y_test, average='weighted'))
```

The confusion matrix, accuracy, and precision score were obtained using the codes as shown.

```
ConfusionMatrixDisplay.from_estimator(model_tuned, X_test_scaled, Y_test)
plt.show()
```



Test accuracy of kNN
0.7470588235294118

Test precision of kNN:
0.7397234341306855

From the above confusion matrix, class – bus has the highest True positives, and opel has the lowest true positives. But after hyperparameter tuning the number of TPs of opel class has been increased and the number of instances of classes opel and saab falsely classified as the other class has been reduced and contributed to the overall increase in accuracy and precision of the performance of the learning task.

c. Bayesian Learning

General introduction

Bayesian learning is a probabilistic classification technique based on the Bayes theorem:

$$P(Y|D) = \frac{P(D|Y)P(Y)}{P(D)}$$

that is, it combines prior knowledge with observed data to determine the final probability of a hypothesis.

The brute force bayes concept learning calculates the posterior probability of each possible hypothesis given the training dataset D and outputs the most probably one or one with the highest probability. However, this becomes computationally expensive to calculate for each possible hypothesis.

To go around the problem, the Naïve Bayes classifier makes a (naïve) assumption that all the features $(x_1 x_2 \dots x_n)$ are conditionally independent given the class label, so that the following holds true.

$$P(y|x_1 x_2 \dots x_n) = \frac{P(x_1 x_2 \dots x_n|y)P(y)}{P(x_1 x_2 \dots x_n)} \quad \text{(from the Bayes theorem)}$$

$$= \frac{[P(x_1|y)\,P(x_2|y)\,P(x_3|y)\ldots P(x_n|y)]P(y)}{P(x_1x_2\ldots x_n)} \quad \text{(from conditional independence)}$$

$$= \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1x_2\ldots x_n)}$$

The classifier will output the hypothesis $\hat{y}$ that has the highest posterior probability. As the denominator of the above is constant, the above is proportional to the following.

$$P(y)\prod_{i=1}^{n} P(x_i|y)$$

Thus, the Naïve Bayes classifier solves the following:

$$\hat{y}_{NB} = \arg\max(P(y)\prod_{i=1}^{n} P(x_i|y))$$

The specific one chosen for the data set

We chose the Gaussian Naïve Bayes classifier, in particular, because all of the 18 attributes in our dataset were continuous variables. In addition to the conditional independence assumption, the Gaussian Naïve Bayes assumes that each feature value follows a Gaussian distribution (or a normal distribution). Thus, we assume that $P(x_i|y)$ follows the normal distribution, for each I, and thus the classifier's task is as follows.

$$\hat{y}_{NB} = \arg\max(P(y)\prod_{i=1}^{n} P(x_i|y))$$

$$= \arg\max(P(y)\prod_{i=1}^{n} \mathrm{N}(x_i|y))$$

Where N refers to a normal (Gaussian) distribution.

Preprocessing

As already shown in the earlier exploratory data analysis section, some of the features were highly correlated with others. We removed four features in order to hold the conditional independence assumption (but also because the initial model with all the 18 features had poor accuracy of roughly 45%).

```
x_train_small = x_train.drop(['circularity', 'scatter ratio',
'pr.axis_rectangularity', 'scaled_variance.1'], axis=1)
x_test_small = x_test.drop(['circularity', 'scatter ratio',
'pr.axis_rectangularity', 'scaled_variance.1'], axis=1)
```

Furthermore, some of the features were not normally distributed, thus violating the normal distribution assumption for the Gaussian Naïve Beyes. In order to go around the problem, we log-transformed the features with high skewness, acknowledging that they might not become completely Gaussian distributions.

```
x_train_small['pr.axis aspect ratio log'] =
np.log(x_train_small['pr.axis_aspect_ratio'])
x_train_small['max.length_aspect_ratio  log'] =
np.log(x_train_small['max.length_aspect_ratio'])
x_train_small['scaled_radius_of_gyration.1 log'] =
np.log(x_train_small['scaled_radius_of_gyration.1'])
x_test_small['pr.axis aspect ratio log'] =
np.log(x_test_small['pr.axis_aspect_ratio'])
x_test_small['max.length_aspect_ratio  log'] =
np.log(x_test_small['max.length_aspect_ratio'])
x_test_small['scaled_radius_of_gyration.1 log'] =
np.log(x_test_small['scaled_radius_of_gyration.1'])
```

Implementation

As the training and testing split was carried out at the preprocessing stage as explained above (with 80% for training and 20% testing), data was read from the csv files first. Training was then conducted by calling the fit() method after creating an instance of the GaussianNB class.

```
gaussian = GaussianNB()
gaussian.fit(x_train_sml_log, y_train.values.ravel())
```

We did not change the prior, but used the priors provided by the dataset as below.

```
Priors used for this model:  [0.25739645 0.25147929 0.25591716 0.2352071 ]
```

(We considered changing the priors to make some of the classes more likely than others (such as increasing the prior probability of saab) but it did not change results much.) The accuracy of the training set was obtained by calling the score() method as below and we obtained the accuracy of 58.28%.

```
acc_gaussian_train = round(gaussian.score(x_train_sml_log, y_train) * 100, 2)
```

We then conducted a 10-fold cross validation using the cross_val_score function with the training dataset as below.

```
scores = cross_val_score(gaussian, x_train_sml_log,y_train.values.ravel(),
cv=10, scoring="accuracy")
```

We obtained the mean accuracy across the 10 different splits by calling the mean() method as below and it was 56.96%.

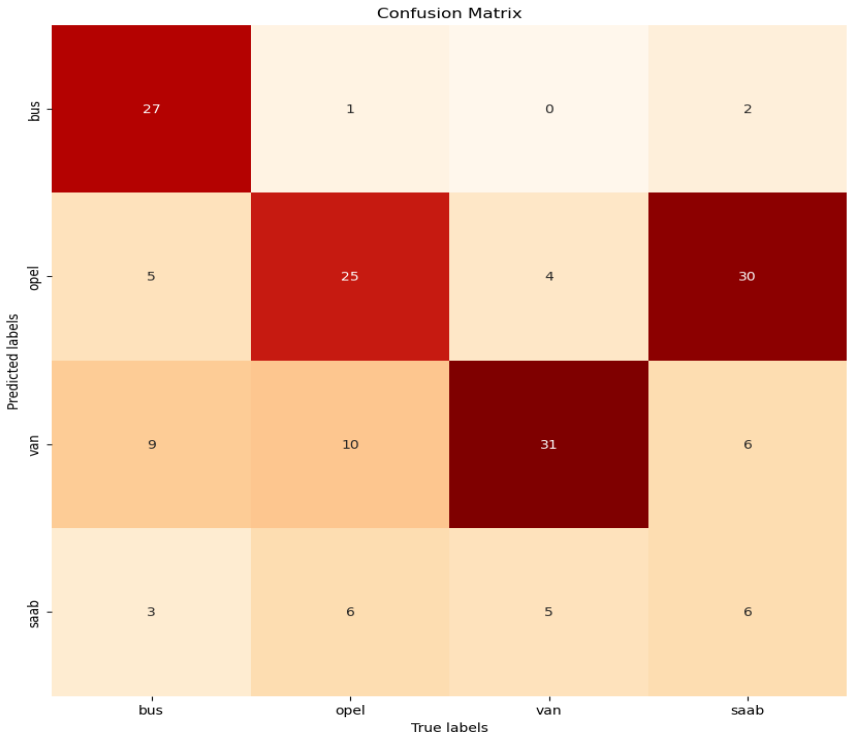
```
meanScore= scores.mean()
```

Finally, we used the testing dataset to predict the class for each instance and checked the accuracy by using the accuracy_score method of the sklearn.metric package as below.

```
Y_pred = gaussian.predict(x_test_sml_log)
accuracy_gaussian=round(accuracy_score(y_test,Y_pred)* 100, 2)
```

We obtained that the accuracy was about 52% and the precision was about 54%.

Below is the confusion matrix for the final model. It is clear that the bus and the van were classified correctly in the majority of the instances but the saab and the opel suffered greatly from misclassification.



d. Neural Networks

General introduction

Neural networks are a method of artificial intelligence used to teach computers to process data in a way inspired by the human brain. It is a machine learning process, called deep learning, which uses interconnected nodes or neurons in a hierarchical structure similar to that of the human brain. It creates adaptive systems that computers use to learn from their mistakes and to continually improve. As a result, neural networks can attempt to solve complex problems such as machine vision and voice recognition.

The neural network is divided into three layers:

- Input layer

Information from the external world enters the neural network through the input layer. The input nodes process, analyse or classify the data and then pass it on to the next layer.

- Hidden layer

The hidden layer takes its input from the input layer or from other hidden layers. Neural networks can have a large number of hidden layers. Each hidden layer analyses and further processes the output from the previous layer and then passes it on to the next layer.

- Output layer

The output layer provides the final result of the processing of all the data by the neural network. It can contain single or multiple nodes. For example, if we are solving a binary (yes/no) classification problem, the output layer contains a single output node which will provide a result of 1 or 0. However, if we are solving a multi-class classification problem, the output layer may consist of more than one output node.

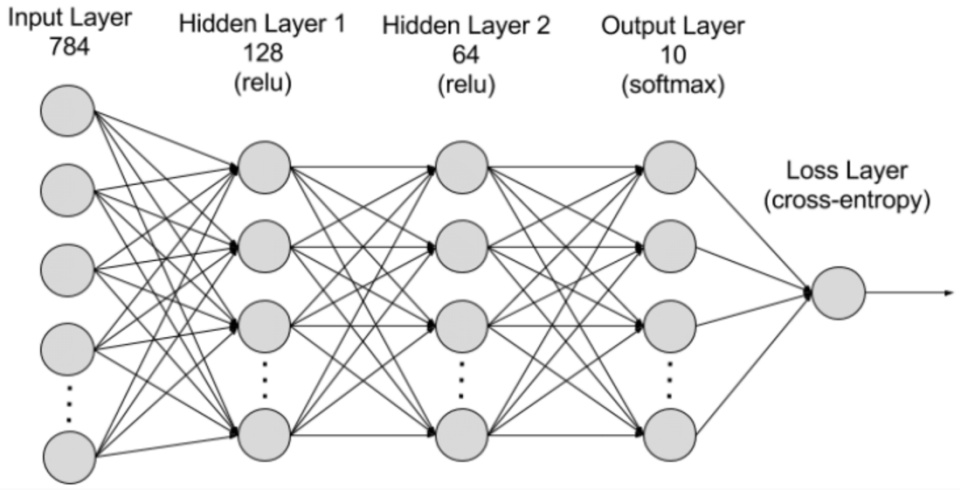


Figure: Neural Network [1]

Neural network training is the process of teaching a neural network to perform a task. Neural networks learn by first processing several large sets of labelled or unlabelled data. The specific learning method varies depending on the structure of the network and the model. There are many types of neural networks: feedforward neural networks, back propagation algorithms, convolutional neural networks (CNN). By using the training result, they can handle unknown inputs more accurately.

The specific algorithm I picked:

In this project, we used Deep Neural Networks (DNN). Neural networks are based on extensions of perceptrons, while DNNs can be understood as neural networks with many hidden layers. Multi-layer neural networks and DNNs refer to the same thing, and DNNs are sometimes called multi-layer perceptron (MLP). DNN is a feedforward neural network with input, hidden and output layers, and is a type of deep learning algorithm that uses back propagation to train the algorithm.

Pre-processing steps:

In this project, we used Keras to develop the neural network. Keras is an open-source python library to build deep learning models based on TensorFlow.

- Importing data

We firstly imported the data from the pre-processed dat file into dataframe format to prepare the data for later processing. And then we viewed this dataset. The first 18 columns are the input variables, and the last column is the label classification of the data (i.e., the output). After importing the data, we split the data into two groups, input and output, for cross-checking purposes.

```
X = dataset.iloc[:, 0:18].values
y = dataset.iloc[:, 18].values
```

In the dataframe, the last column of the classification is shown as string. To do machine learning later, we need to perform a neural network approach to data labelling, using LabelEncoder to transform the discontinuous label codes into vectors.

```
encoder = LabelEncoder()
y_class = encoder.fit_transform(y)
Y = pd.get_dummies(y_class).values
```

From the output we can see that:

bus - 0

opel - 1

saab - 2

van - 3

```
array(['van', 'van', 'saab', 'van', 'bus', 'bus', 'bus', 'van', 'van',
       'saab', 'van', 'saab', 'bus', 'van', 'bus', 'opel', 'van', 'bus',
       'saab', 'opel', 'bus', 'van', 'bus', 'bus', 'saab', 'van', 'saab',
       'saab', 'bus', 'saab', 'van', 'saab', 'opel', 'opel', 'opel',
       'van', 'bus', 'van', 'saab', 'bus', 'opel', 'van', 'van', 'saab',
       'saab', 'van', 'van', 'bus', 'van', 'saab', 'saab', 'saab', 'opel',
       'bus', 'bus', 'van', 'saab', 'van', 'opel', 'van', 'opel', 'opel',
       'van', 'bus', 'bus', 'opel', 'bus', 'opel', 'van', 'bus', 'opel',
       'opel', 'opel', 'opel', 'van', 'opel', 'saab', 'saab', 'bus',
       'bus', 'bus', 'bus', 'van', 'opel', 'bus', 'bus', 'van', 'van',
       'bus', 'opel', 'saab', 'opel', 'saab', 'van', 'bus', 'opel',
       'saab', 'bus', 'opel', 'bus', 'bus', 'van', 'van', 'van', 'bus',
       'saab', 'opel', 'opel', 'bus', 'bus', 'van', 'van', 'opel', 'opel',
       'van', 'van', 'opel', 'saab', 'bus', 'bus', 'saab', 'van', 'van',
       'saab', 'van', 'van', 'bus', 'bus', 'van', 'bus', 'saab', 'saab',
       'saab', 'van', 'opel', 'van', 'van', 'van', 'saab', 'van', 'saab',
       'bus', 'opel', 'bus', 'opel', 'opel', 'van', 'bus', 'saab', 'van',
       'saab', 'bus', 'opel', 'saab', 'van', 'bus', 'opel', 'van', 'saab',
       'opel', 'opel', 'opel', 'saab', 'saab', 'opel', 'opel', 'saab',
       'saab', 'bus', 'van', 'opel', 'bus', 'van', 'bus', 'van', 'bus',
       'opel', 'bus', 'bus', 'van', 'saab', 'opel', 'bus', 'saab', 'saab',
       'bus', 'bus', 'opel', 'opel', 'opel', 'opel', 'van', 'saab', 'bus',
       'saab', 'bus', 'saab', 'bus', 'bus', 'bus', 'van', 'saab', 'opel',
       'opel', 'van', 'van', 'opel', 'bus', 'bus', 'opel', 'opel', 'van',
       'bus', 'bus', 'opel', 'opel', 'saab', 'opel', 'bus', 'van', 'opel',
...
       'saab', 'bus', 'saab', 'opel', 'opel', 'bus', 'van', 'bus', 'bus',
       'van', 'saab', 'opel', 'saab', 'saab', 'opel', 'van', 'opel',
       'opel', 'opel', 'opel', 'opel', 'van', 'van', 'bus', 'bus', 'saab',
       'saab', 'saab', 'bus', 'van', 'saab', 'van', 'van', 'opel', 'saab',
       'van', 'saab', 'saab', 'van'], dtype=object)
```

```
array([3, 3, 2, 3, 0, 0, 0, 3, 3, 2, 3, 2, 0, 3, 0, 1, 3, 0, 2, 1, 0, 3,
       0, 0, 2, 3, 2, 2, 0, 2, 3, 2, 1, 1, 1, 3, 0, 3, 2, 0, 1, 3, 3, 2,
       2, 3, 3, 0, 3, 2, 2, 2, 1, 0, 0, 3, 2, 3, 1, 3, 1, 1, 3, 0, 0, 1,
       0, 1, 3, 0, 1, 1, 1, 1, 3, 1, 2, 2, 0, 0, 0, 3, 1, 0, 0, 3, 3, 3,
       0, 1, 2, 1, 2, 3, 0, 1, 2, 0, 1, 0, 0, 3, 3, 3, 0, 2, 1, 1, 0, 0,
       3, 3, 1, 1, 3, 3, 1, 2, 0, 0, 2, 3, 3, 2, 3, 3, 0, 0, 3, 0, 2, 2,
       2, 3, 1, 3, 3, 3, 2, 3, 2, 0, 1, 0, 1, 1, 3, 0, 2, 3, 2, 0, 1, 2,
       3, 0, 1, 3, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 0, 3, 1, 0, 3, 0, 3, 0,
       1, 0, 0, 3, 2, 1, 0, 2, 2, 0, 0, 1, 1, 1, 1, 3, 2, 0, 2, 0, 2, 0,
       0, 0, 3, 2, 1, 1, 3, 3, 1, 0, 0, 1, 1, 3, 0, 0, 1, 1, 2, 1, 0, 3,
       1, 3, 0, 0, 2, 3, 2, 3, 2, 2, 3, 2, 0, 2, 3, 3, 0, 0, 1, 0, 1, 1,
       0, 2, 3, 3, 0, 2, 1, 2, 3, 1, 3, 0, 3, 2, 2, 1, 2, 1, 2, 2, 3, 3,
       2, 0, 0, 2, 3, 3, 0, 2, 3, 3, 0, 0, 0, 1, 2, 0, 1, 0, 0, 2, 3, 2,
       0, 3, 1, 2, 3, 3, 0, 3, 0, 0, 0, 2, 2, 0, 2, 1, 0, 3, 0, 0, 2, 1,
       1, 1, 2, 3, 0, 1, 0, 3, 3, 2, 3, 0, 2, 0, 0, 2, 2, 0, 1, 1, 3, 2,
       3, 1, 1, 0, 0, 2, 3, 0, 3, 0, 0, 1, 2, 1, 1, 0, 2, 1, 3, 1, 1, 2,
       3, 0, 2, 1, 0, 2, 1, 1, 2, 2, 2, 3, 1, 2, 3, 2, 0, 1, 0, 0, 3, 0,
       3, 0, 2, 2, 2, 1, 1, 0, 0, 1, 1, 3, 2, 3, 3, 1, 0, 3, 3, 0, 2, 1,
       0, 0, 0, 3, 2, 1, 1, 3, 1, 0, 1, 2, 0, 2, 2, 1, 0, 1, 3, 1, 0, 0,
       0, 2, 3, 1, 2, 0, 1, 1, 1, 1, 2, 3, 2, 0, 2, 0, 2, 3, 3, 1, 1, 2,
       2, 3, 3, 1, 0, 3, 2, 1, 0, 3, 0, 1, 1, 0, 2, 0, 3, 3, 1, 2, 3, 1,
       1, 1, 0, 0, 0, 1, 2, 3, 3, 1, 0, 1, 1, 2, 1, 3, 0, 0, 2, 1, 1, 3,
       3, 0, 1, 2, 1, 1, 2, 2, 0, 1, 0, 0, 1, 3, 3, 1, 1, 1, 1, 2, 1, 2,
       2, 1, 3, 1, 0, 1, 1, 2, 3, 1, 1, 2, 2, 3, 2, 3, 2, 3, 1, 2, 2, 1,
       0, 0, 2, 0, 2, 3, 3, 3, 2, 3, 1, 0, 0, 3, 2, 2, 0, 3, 3, 0, 0, 2,
...
       3, 2, 2, 3, 0, 2, 3, 3, 2, 2, 0, 3, 1, 2, 3, 1, 2, 0, 2, 0, 1, 2,
       0, 1, 1, 0, 3, 0, 2, 0, 2, 3, 2, 3, 1, 2, 3, 0, 2, 3, 2, 2, 0, 3,
       1, 0, 0, 3, 1, 2, 1, 0, 3, 1, 1, 0, 1, 0, 2, 2, 0, 2, 1, 1, 0, 3,
       0, 0, 3, 2, 1, 2, 2, 1, 3, 1, 1, 1, 1, 1, 3, 3, 0, 0, 2, 2, 2, 0,
       3, 2, 3, 3, 1, 2, 3, 2, 2, 3])
```

We then split all the data into training and test sets by 8:2, and then split the training set into training and validation sets by 8:2.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)
X_train, x_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=0)
```

- Defining the model:

The Keras model consists of layers: we build a Sequential model, adding neurons layer by layer. The first step is to make sure that the number of input layers is correct: this is determined when creating the model with the input_shape parameter. For example, for this project there are 18 input variables, so set it to 18.

The fully connected layers are defined using the Dense class: the first parameter is the number of neurons in this layer, followed by the initialisation method and the activation function. The initialisation method here is a continuous uniform distribution from 0 to 0.05, which is also the default method in Keras.

At first we set up only two hidden layers, but later we found that increasing the number of hidden layers could make the model training more effective. Therefore, we ended up with 7 hidden layers. The activation function for the first seven hidden layers is a linear rectification function (relu) and the activation function for the last layer is softmax function. The first seven hidden layers have 128 or 64 neurons respectively, and the last layer is 4 neurons (van, saab, bus or opel).

```
model = Sequential([
        Dense(64, activation='relu', input_shape=(18,)),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(64, activation='relu'),
        Dense(64, activation='relu'),
        Dense(4, activation='softmax')
    ])
```

- Compiling the model

We need to define the loss function and the optimisation algorithm, as well as the data that needs to be collected. We use categorical_crossentropy, the logarithm of the error, as the loss function and adam as the optimisation algorithm. Since this problem is a classification problem, we collect the accuracy of each round.

```
model.compile(Adam(learning_rate=0.001),           'categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

Keras prints out a summary of the model we have just built:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1216

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 64)                4160

 dense_6 (Dense)             (None, 64)                4160

 dense_7 (Dense)             (None, 4)                 260

=================================================================
Total params: 59,396
Trainable params: 59,396
Non-trainable params: 0
_________________________________________________________________
```

Figure: Summary of the model

EarlyStopping is a type of callbacks used to stop training early, as continuing training will result in a decrease in accuracy on the test set. Specifically, it can be achieved by stopping training when the loss on the training set is no longer decreasing (i.e., decreasing by less than a certain threshold). The early stop method is intended to solve the problem of having to set the number of epochs manually. It can also be considered as a regularisation method to avoid overfitting the neural network (similar to the L1/L2 weight decay and discard method). We set the patience to 50, i.e., if the accuracy does not improve after 50 iterations, then we stop training.

```
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=50,
    min_delta=0.001,
    mode='max'
)
```

- Training the model

Training can be started by calling the fit() method of the model. We fit the training and validation data into the model. Neural network is trained in rounds, controlled by the *epochs* parameter. The data fed in each time (batch size) can be controlled with the *batch_size* parameter. Here we only run 500 rounds, 8 data at a time. This is the result after several tests we conducted. We use the control variable method, adjusting one parameter at a time. Firstly, we set the batch size manually, starting from the default value of 32, and finally found that the model trained best with a batch

size of 8, and obtained the highest test accuracy. Similarly, we adjusted the values of epochs and early_stop patience.

```
history = model.fit(X_train, y_train, validation_data=(x_val,y_val), epochs=500, batch_size=8, callbacks=[early_stopping])
```

- Testing the model

We call the evaluation() method of the model, passing in the data from the training. The output is the average value, including the average loss and accuracy. We also generate the “Training and validation loss” and “Training and validation accuracy” plots.

```
Training loss: 51.57%
```

```
Training accuracy: 77.59%
```

Figure: Training and validation loss

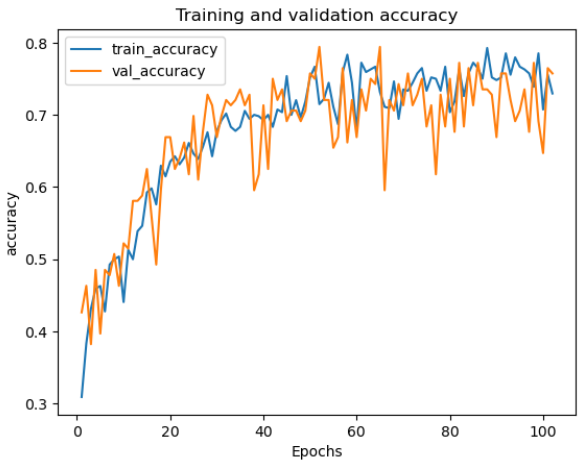


Figure: Training and validation accuracy

Afterwards we take the test data and test the model to get the testing loss and accuracy.

```
Test loss: 52.82%
```

```
Test accuracy: 75.88%
```

Then we use the test data to generate the prediction for drawing the confusion matrix. We can see from below that 0 (van) and 3 (bus) has high prediction accuracy, while 1 (opel) and 2 (saab) have low accuracy.

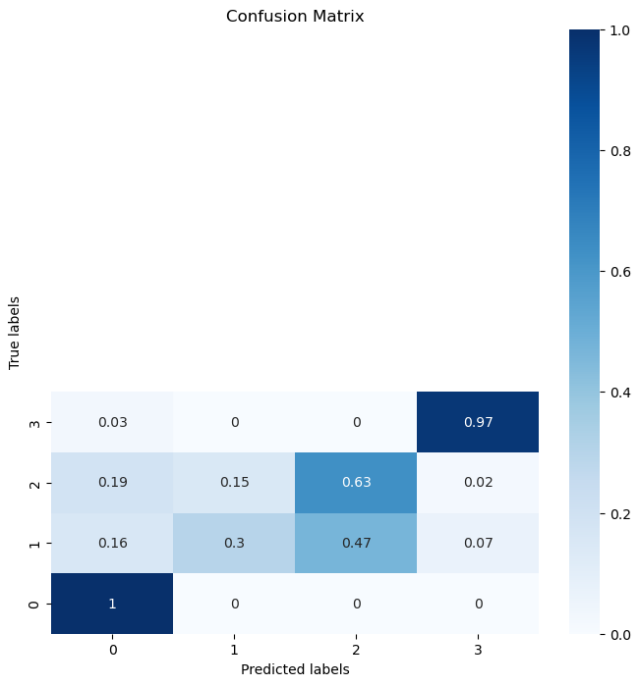


Figure: Confusion Matrix

After several training sessions (because each time the training set was randomly split from the test set and therefore different answers were obtained), we obtained an accuracy of over 70% for the test set.

e. Model Ensemble

General introduction

Ensemble learning is a machine learning technique to combine the predictions of diverse models, to improve the overall prediction performance. It can be achieved in different ways, for example, through using different algorithms or different training datasets. Through ensemble modelling, the predictions of previous models are aggregated, as the input for the final prediction, to improve the original possibly biased or weakly training individual learners. It can be useful because using the common procedure of selecting the model with the least error on training data can be flawed. In other case, if two models have the same pseudo generalized performance on the validation set, we will come to the question of which one to choose [2]. Therefore, if we use ensemble learning, we can combine the outputs of the tested models instead of choosing just one. This can be simply done, for example, by averaging the output, but it can reduce the chance of selecting a poorly performing classifiers [2]. However, it is also worth noting that the combined classifiers do not necessarily outperform the original classifier, but it has the undeniable advantage of reducing risk, so that we have a less chance to select a poorly performing model.

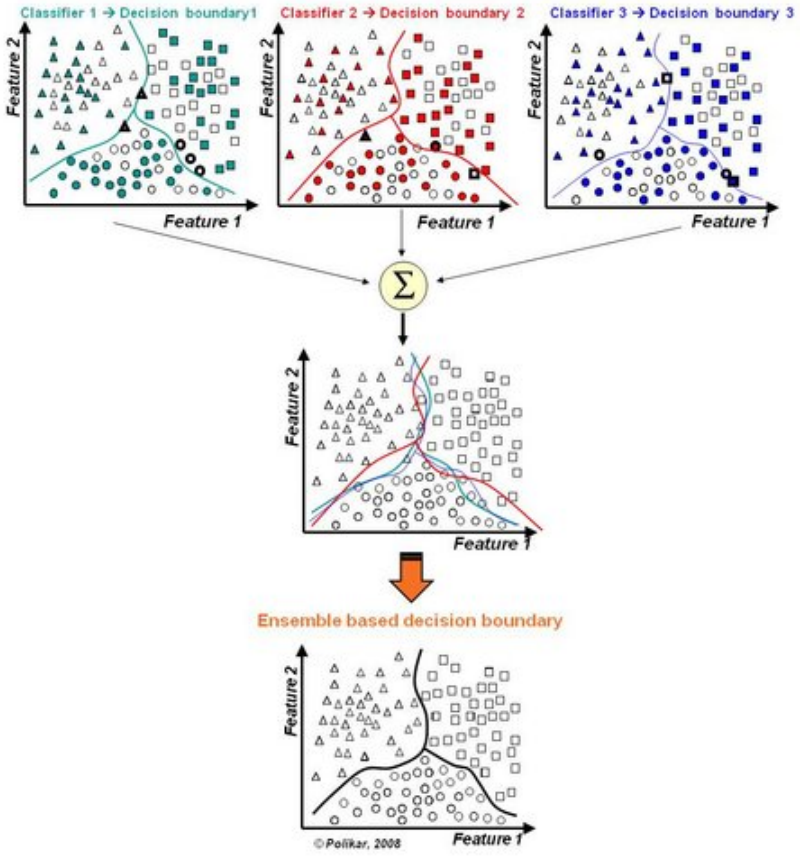


Figure: Combining classifiers to reduce classification error, reduce risk in model selection [2]

For model ensemble strategies, there are unlimited ways to achieve this concept, but three classes of ensemble learning are mostly discussed and commonly used. They are bagging, boosting, and stacking.

- Bagging

Bagging stands for bootstrap aggregating. Instead of the original training data, in bagging, a diversity of classifiers is obtained by applying bootstrapped replicas of the training data [2]. Therefore, different training subsets are randomly drawn. In the training process, multiple instances of the same model on different random subsets of the training data are trained, the predictions are then averaged to make a final prediction. Since the training datasets may greatly overlap, it is also possible to adopt additional measures to increase diversity. For example, using individual subsets of data to train each classifier [2]. The following shows the structure of bagging.

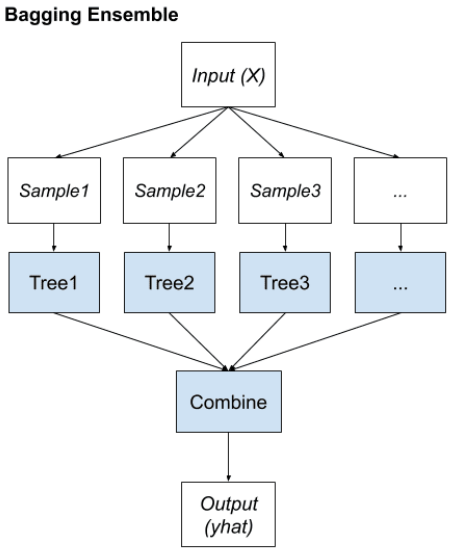


Figure: Structure of Bagging [3]

- Boosting

Boosting is another ensemble strategy that aims to change the training data, so that it can focus on examples that previous models have predicted wrongly in the training set [3]. The key identity of boosting is the concept of “correcting prediction errors”. The models are fit and added, in a way that the next attempt corrects the prediction of the previous one, and so on. Its goal is to develop a “strong-learner” from many “weak-learners”. In general, it bias training data towards those that are hard to predict, add ensemble models to correct predictions of prior models and lastly combine their predictions with a weighted average of models [3]. The following shows how boosting works.

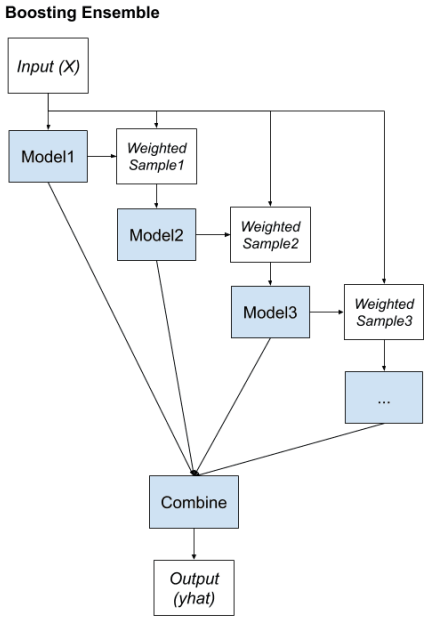


Figure: Structure of Boosting [3]

- Stacking

Stacking is another model ensemble method that first trains several different models and then combining their predictions with a meta-model, such as a logistic regression or voting mechanism. Firstly, models are trained in the first layer, a set of classifiers. They are usually weak learners. And their predictions are used as the input of the meta-model, so that their predictions can be combined, and a final prediction is made by the meta-classifier, the meta-classifier learns how to best combine the predictions in the first layer. Instead of having only the original layer of classifiers and the meta-model, we can also make several layers of models. The concept is that the output of the previous layer will be input into the next layer for combining and learning. Stacking has the following features: (1) the training dataset is not modified, (2) different machine learning algorithms can be used in the classifiers, (3) using machine learning model to learn how to combine predictions [3]. Therefore, it is useful in combining a set of models that are constructed in different ways or concept. With their different assumptions, less correlated prediction errors will result [3]. The following shows the standard stacking structure, with the set of first level classifier and final level meta-learner.

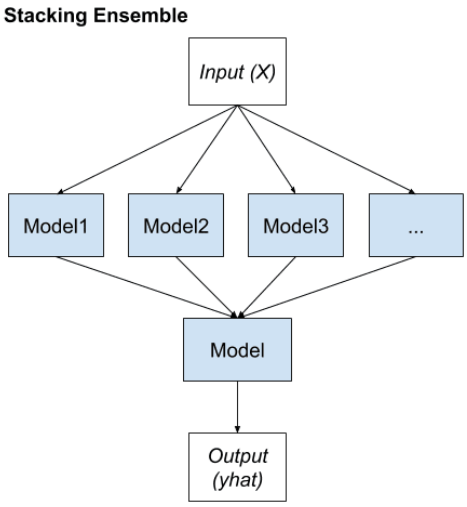


Figure: Structure of Stacking [3]

The above-mentioned strategies all belong to the family of model ensemble. However, they have different advantages and usage, so that they can be useful in different situations as described in the general description. In this assignment, the concept of stacking is mainly used to perform integration of the previous models trained: decision tree, KNN, naive bayes and neural network. It is because they are classifiers constructed in different ways and concepts, as mentioned, one of the advantages of stacking is that it facilitates combing models of different concepts by inputting their predictions into next layer for further learning. Furthermore, it does not improve learning the training set by modifying the training dataset itself, instead, it aims to use more models to improve the performance. The structure of the stacking classifier chosen is a stacking classifier with cross-validation. And the model used is StackingCVClassifier by mlxtend [4]. It extends from the standard stacking algorithm such as the one shown above in the figure with the standard structure

of stacking. It uses cross-validation to prepare the input data for the meta classifier. It is used because in the standard stacking process, the first level classifiers and the meta classifier are fit with the same training set, so a problem caused by this is the possibility of overfitting [4]. But if we use the cross-validation stacking classifier, “the dataset is split into k folds, and in k successive rounds, k-1 folds are used to fit the first level classifier; in each round, the first-level classifiers are then applied to the remaining 1 subset that was not used for model fitting in each iteration” [4]. After that, the predictions of the classifiers are stacked and input to the meta classifier. The whole structure of the cross-validation stacking is shown in below. It is tested that this method strikes a balance between precision and overfitting. Since the dataset has a relatively small numbers of instances, with approximately more than 800 rows, if an excessive amount of correction is used, although the result of the classifier in training set is improved, the result of the classifier in the testing and validation set is not improved. For example, adaboost of boosting was originally adopted, however abandoned due to the existence of overfitting in testing and validation. Next, the implementation of the model and its result will be explained.

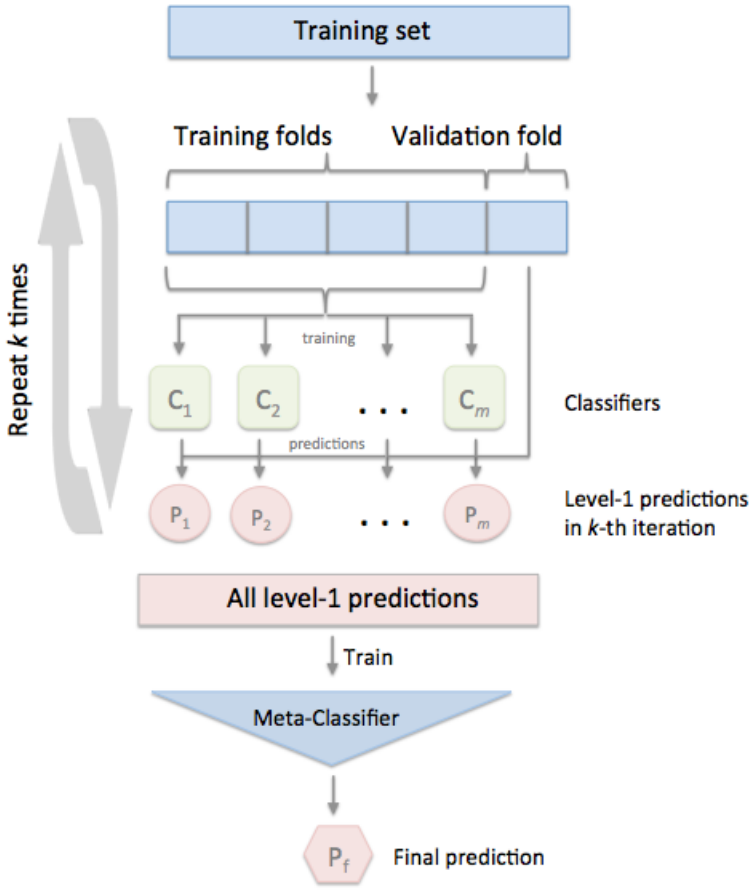


Figure: Structure of the cross-validation stacking classifier [4]

Algorithm 19.8 Stacking with K-fold Cross Validation

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier H

Step 1: Adopt cross validation approach in preparing a training set for second-level classifier
Randomly split $\mathcal{D}$ into K equal-size subsets: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K\}$
for $k \leftarrow 1$ to K **do**
    Step 1.1: Learn first-level classifiers
    for $t \leftarrow 1$ to T **do**
        Learn a classifier h_{kt} from $\mathcal{D} \setminus \mathcal{D}_k$
    end for
    Step 1.2: Construct a training set for second-level classifier
    for $\mathbf{x}_i \in \mathcal{D}_k$ **do**
        Get a record $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_{k1}(\mathbf{x}_i), h_{k2}(\mathbf{x}_i), \ldots, h_{kT}(\mathbf{x}_i)\}$
    end for
end for
Step 2: Learn a second-level classifier
Learn a new classifier h' from the collection of $\{\mathbf{x}'_i, y_i\}$
Step 3: Re-learn first-level classifiers
for $t \leftarrow 1$ to T **do**
    Learn a classifier h_t based on $\mathcal{D}$
end for
return $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

Figure: Summary of the cross-validation process [5]

Implementation

Since the input of the layer one classifier above is different, in the data preprocessing, all columns are first added to the dataset, so that in the set of classifiers in the stackingCVclassifier, I can select the input columns for each classifier using the make_pipeline tool in sci-kit learn. So, after the data processing, the x_train data frame for the training features became 22 features, with the original 18 features and 4 log transformed features from the naïve bayes.

```
# New column order so that we can specify the features in stacking for each
classifiers
col = ['compactness','distance circularity', 'radius ratio', 'pr.axis aspect
ratio', 'max length aspect ratio', 'elongatedness', 'max length
rectangularity', 'scaled variance MAJOR AXIS',
       'scaled radious of gyration', 'skewness about major axis', 'skewness
about minor axis', 'kurtosis about minor axis', 'kurtosis about major axis',
'hollows ratio',
       'circularity', 'scatter ratio', 'pr axis rectangularity',  'scaled
variance MINOR AXIS',
       'pr.axis aspect ratio log', 'scaled variance MAJOR AXIS log',
'skewness about major axis log', 'skewness about minor axis log', 'vehicle
class']

# A dataframe with all features including the original ones and log ones
df1 = df_small[col]
dfall = df1
dfall
```

To perform fitting other classifiers with the small dataset with 22 features, a function col_selection is created to select the features for each model. X_train_knn_dt_nn selects the features for

decision tree, KNN and neural network. While X_train_gnb selects the features for the naïve bayes.

```
# A function to select columns for fitting, before using the sklearn
pipeline
def col_selection(x_set, columns):
    return x_set.iloc[:, columns]

# Training set columns selection
X_train_knn_dt_nn = col_selection(x_train, list(range(0, 18)))
X_train_gnb = col_selection(x_train, list(range(0, 14)) + [18] + [19] + [20]
+ [21])

# Testing set columns selection
X_test_knn_dt_nn = col_selection(x_test, list(range(0, 18)))
X_test_gnb = col_selection(x_test, list(range(0, 14)) + [18] + [19] + [20] +
[21])
```

Next, since the original classification columns are categorical data, we use label encoder in sci-kit learn to transform the vector into integer data. So that the classifiers can process the data. And specifical, since neural network requires one-hot output prediction type, the training, testing, validation output column of the dataset are also transformed. This step is shown as below.

```
# Encode target variable from categories to int for the classifier
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
y_val_encoded = label_encoder.transform(y_val)

# Convert target variable to one-hot encoding for neural network
y_train_onehot = np_utils.to_categorical(y_train_encoded)
y_test_onehot = np_utils.to_categorical(y_test_encoded)
y_val_onehot = np_utils.to_categorical(y_val_encoded)
```

After that, we define the specific models. Since the neural network model was trained with the keras package in the previous section, and the output of it is different from other sci-kit learn packages, a function of create_nn_model allows integration of neural network with other models, by setting the create_nn_model as the input of KerasClassifier. Since the predict output of the KerasClassifier will be the same as other classifiers – the encoded output, the set of the classifiers is ready to be put in the first layer of the stacking model.

```
# A function to create a classifier for neural network, since keras
classifier cannot be directly used in stackingCVclassifier
def create_nn_model():
    nn = Sequential([
        Dense(64, activation='relu', input_shape=(18,)),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(64, activation='relu'),
        Dense(64, activation='relu'),
```

```
        Dense(4, activation='softmax')
    ])

    nn.compile(Adam(learning_rate=0.001), 'categorical_crossentropy',
metrics=['accuracy'])
    return nn

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=50,
    min_delta=0.001,
    mode='max')
```

Moving on to the stacking model, pipelines of classifiers to select the specific columns are used. The pipeline construct is shown below.

```
# Use column selector to select the processing columns, and then make them
into pipelines for the stacking classifier
pipeline1 = make_pipeline(ColumnSelector(cols=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17)), knn)
pipeline2 = make_pipeline(ColumnSelector(cols=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17)), dt)
pipeline3 = make_pipeline(ColumnSelector(cols=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 18, 19, 20, 21)), gnb)
pipeline4 = make_pipeline(ColumnSelector(cols=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17)), nn)
```

The stacking classifier is then constructed by the pipelines. The meta classifier used is logistic regression and it makes prediction based on the class probabilities output by the previous classifier pipelines. The ‘use_probas’ is set as ‘True’ to facilitate this. It basically instructs the meta classifier to use class probabilities instead of hard voting, so the prediction can be better with a better test prediction result. The times of cross-validation folds, which is ‘cv’ is set as 5. It means the training data is split into 5 parts, and during each training cycle, the layer one classifiers are trained with 4 parts and use the remaining one as validation. With the cross-validation set, overfitting is reduced, and a more accurate estimation can be achieved.

```
# Define the StackingCVClassifier with Logistic Regression as the meta-
classifier, use_probas=True
stacking_classifier = StackingCVClassifier(classifiers=[pipeline1,
pipeline2, pipeline3, pipeline4], meta_classifier=LogisticRegression(),
use_probas=True, cv=5)

# Train the stacking classifier
stacking_classifier.fit(x_train, y_train_encoded)
```

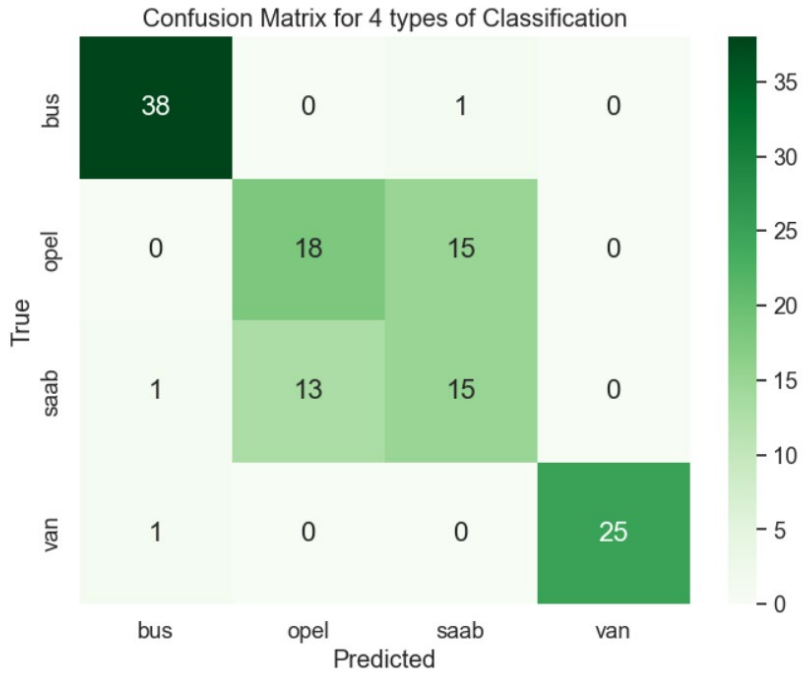
After fitting with the training data, testing set were used to evaluate the performance of the stacking classifier. The following shows the accuracy, precision, classification report and confusion matrix of the model. Other than that, a validation set (unseen data) is preserved to test the model's performance with unseen data. The validation set is not used during any of the training process, so it can provide an unbiased evaluation of the model's generalization ability, particularly in unseen data.

```
Testing Set accuracy: 0.7559055118110236
Validation Set accuracy: 0.8153846153846154
Precision: 0.8553098723139209
Classification Report:
              precision    recall  f1-score   support

         bus       0.95      0.97      0.96        39
        opel       0.58      0.55      0.56        33
        saab       0.48      0.52      0.50        29
         van       1.00      0.96      0.98        26

    accuracy                           0.76       127
   macro avg       0.75      0.75      0.75       127
weighted avg       0.76      0.76      0.76       127
```

From the above result, the testing set accuracy is about 0.76. This indicates that the model is not prone to the problem of overfitting. When testing with unseen data, the validation set, the accuracy is about 0.82. The precision also shows an improved result from previous models, with about 0.86 in the test set. The classifier has a better ability to correctly identify only the positive instances out of all the instances it predicted as positive. From the classification report, it shows that the model accuracy level may be affected by the problem of similarity in 'opel' and 'saab' as mentioned in the data exploration. Since the features of the two types are too similar, it is difficult to classify them with learners. Another reason causing this issue might be the limitation of the number of rows in the dataset. Since the training set has a low number of training data, the result is not favorable. Lastly, the confusion matrix of the model ensemble is shown in the following. From the matrix, it is shown that the classifier seems to struggle with 'opel' and 'saab' as discussed above. It shows that the stacking classifier also misclassifies many cases between the two classes. However, 'bus' and 'van' generally have correct predictions. Furthermore, compared to the previous models, the classifications of these two classes still show a slight improvement.

Although the performance does not improve much, it is still worth ensemble the classifiers for calibrating each other. It is shown in the improved classification report and confusion matrix. Also, a great advantage of it is that it is more useful to predict unseen data. This is shown in the testing

with the preserved validation set.



6. Conclusion

To conclude, the classifiers show high accuracy and precision in classifying ‘bus’ and ‘van’ but have problems in distinguishing between ‘saab’ and ‘opel’. As discussed in the data exploration process, it is found that many of the attributes of the two classes appear to share similar characteristics, which is likely to explain the misclassification between Saab. Moreover, the classifiers show a similarity in accuracy in the testing set with above 70% except for the naïve bayes algorithm. This phenomenon echoes the aforementioned observation - highly correlated features, and the hypothesis that it is hindering the accuracy. Since the naïve bayes classifier assumes features are independent of each other, correlating features violate the assumption, lead to inaccurate probability estimation and therefore poor classification performance. Another assumption of the Naïve Bayes that each feature is normally distributed is likely to be violated, once again leading to the poor performance of the naïve bayes.

Another phenomenon observed is as mentioned that most classifiers have a similar accuracy score and a precision score of above 70%. It also applies the stacking classifier which incorporates weak learners for a better training result. It does not improve the overall performance much. We hypothesize that it is due to insufficient training data, causing the algorithms’ accuracy to be capped at this level and cannot be much improved with the ensemble method. On the other hand, the classifiers: decision tree, KNN and neural network capture a similar pattern in the dataset, which causes the fact that using a stacking classifier to incorporate their results does not

add much value. However, there is still worth using the final ensemble method, since it also very much improves precision and allows the model to have better accuracy in predicting unseen data.

7. Git Log

commit 7068e0e013478d3f0d089a7e629cee4a19d8aeeb
Author: kjy000329 <jiayun.kang2000@gmail.com>
Date: Sun Mar 26 15:32:23 2023 +0100

Add new print

commit 6e304707795758b75eb5da7833dbd0387fe8a250
Author: kjy000329 <jiayun.kang2000@gmail.com>
Date: Sun Mar 26 15:06:40 2023 +0100

delete cleaned_data

commit 3f85403312129a9c12fcdb2a2837b8bac41479b6
Merge: c65d335 f3d76f8
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Sun Mar 26 14:37:33 2023 +0100

Merge pull request #6 from CEGE0004/Gagan_BA

instance based learning + data exploration

commit c65d335c84c1d0a3ead2db190803b899745d5d63
Merge: 879603c 8b9f510
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Sun Mar 26 14:35:42 2023 +0100

Merge pull request #5 from CEGE0004/jiayun_kang

neural network

commit 879603c0e57ade242f3a51cb36578d58f30a1b14
Merge: e5d13fe cc194ce
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Sun Mar 26 14:34:54 2023 +0100

Merge pull request #4 from CEGE0004/James_Yik_Kien_Tse

Model ensembles

commit e5d13fe534e2f76d48d8ba14a658561633393e4b
Merge: e57a334 0b92234
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Sun Mar 26 14:33:44 2023 +0100

    Merge pull request #3 from CEGE0004/Qianxi-Feng

    Decision Tree

commit cc194ce20e9061d3654fee2fcd38b07007a3526f (origin/James_Yik_Kien_Tse)
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Sun Mar 26 13:52:55 2023 +0100

    Add files via upload

commit 8bd15fe954685f286cdee74193dbb3fd9e4adfa1
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Sun Mar 26 13:52:45 2023 +0100

    Delete Model ensembles.ipynb

commit 17a8248d0160394bcaf077557c8fee7702fd8a06
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Sun Mar 26 13:52:39 2023 +0100

    Delete README.md

commit cc9730a3e9c64e7f8b27f8dde41f3ab262b65abf
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Sun Mar 26 13:52:33 2023 +0100

    Delete original data.csv

commit 5a32341ed117fca400b1ebd094a9707a904f228d
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Sun Mar 26 13:52:22 2023 +0100

    Delete model_ensemble.pkl

commit 0b9223476b7f34db2152b70d533ecc62b1711e16 (origin/Qianxi-Feng)
Author: kjy000329 <jiayun.kang2000@gmail.com>
Date: Sun Mar 26 13:49:29 2023 +0100

```
    update the name of decision tree

commit 30995668b453b192295f5d9ec4d8d767f25366ad
Author: kjy000329 <jiayun.kang2000@gmail.com>
Date:   Sun Mar 26 13:47:19 2023 +0100

    Update confusion matrix

commit 4d9bf598552d94e9fbe036e2bd7a0c11c03916fc
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date:   Sun Mar 26 13:40:09 2023 +0100

    Delete Machine Learning CW1.ipynb

commit e57a334349c97cf9f8548d5bc7765d68773ce9f0
Author: atsumi-ucl <117816866+atsumi-ucl@users.noreply.github.com>
Date:   Sun Mar 26 13:39:49 2023 +0100

    pca updated

commit b4529b01dd414986798b5ac7b0c7ade89df600b5
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date:   Sun Mar 26 13:39:40 2023 +0100

    Add files via upload

commit 678680ffa8bd2a21f2668d3df1ac59d461737218
Author: atsumi-ucl <117816866+atsumi-ucl@users.noreply.github.com>
Date:   Sun Mar 26 13:37:00 2023 +0100

    data visualisation pca

commit 95e23b32298c867e881bf30133a8477c37759f70
Author: atsumi-ucl <117816866+atsumi-ucl@users.noreply.github.com>
Date:   Sun Mar 26 12:48:45 2023 +0100

    naive bayes using 'cleaned data'

commit eb06e22c5acdb53a60fc51a87637f587409e698f
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Sun Mar 26 00:41:25 2023 +0000
```

```
    Add files via upload

commit 1f4cb0ce3373a9e115ae615fbe2c4a75ed698a87
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Sun Mar 26 00:40:59 2023 +0000

    Delete Decision_Tree.ipynb

commit a4409ddeada2947da44bb315ca020a580d370366
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Sun Mar 26 00:40:44 2023 +0000

    Delete final_data_cleaned.csv

commit b63e422b07e9d357a4971d82d302edfe79581d3f
Merge: 64a0d28 f194d4c
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Sat Mar 25 17:14:58 2023 +0000

    Merge pull request #2 from CEGE0004/Qianxi-Feng

    Qianxi_Feng

commit f194d4c853a5c578db8d3585f6a3cc40b9dcb5c1
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Fri Mar 24 05:12:14 2023 +0000

    Add files via upload

commit 64a0d2827b410a4531b7a4d70efa7205ff1f9c1a
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Fri Mar 24 05:10:47 2023 +0000

    Delete Machine Learning CW1.ipynb

commit 6a54b9abaa9c2ba821b8812c6396a2ed499a3d19
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date:   Fri Mar 24 05:09:39 2023 +0000

    Delete Decision_Tree directory

commit 8b9f510700b84d3f01c2c6ca899c1f43468ac57a (origin/jiayun_kang)
Author: kjy000329 <jiayun.kang2000@gmail.com>
```

Date: Thu Mar 23 19:18:52 2023 +0000

    Add a confusion matrix

commit f3d76f8f82e9c3f5ca0fecaf0367a9b4e42ac185 (origin/Gagan_BA)
Author: gagankumarba12 <116358685+gagankumarba12@users.noreply.github.com>
Date: Wed Mar 22 23:56:20 2023 +0000

    Add files via upload

    Output files of the notebooks

commit aee0869787b58f75871e461932d7579c13bb4d96
Author: gagankumarba12 <ucesgkb@ucl.ac.uk>
Date: Wed Mar 22 23:51:54 2023 +0000

    Modification to outlier handling and KNN

commit 1823d498fb19c51e6eae6908de94d44ab574086c
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date: Wed Mar 22 05:06:02 2023 +0000

    Delete Machine Learning CW1.ipynb

commit 3149083af3a5eec35adadcd5b050113fcc5b456f
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date: Wed Mar 22 05:04:59 2023 +0000

    Add files via upload

commit 69e0c703ec1044e3c54477a29c7ee0df78567ec5
Author: Qianxi Feng <117454031+ucesqf0@users.noreply.github.com>
Date: Wed Mar 22 05:02:00 2023 +0000

    Delete Ensemble learning directory

commit d9a084a02c9cafc181cb0f75057af82d6af25c7d
Author: gagankumarba12 <ucesgkb@ucl.ac.uk>
Date: Tue Mar 21 22:48:53 2023 +0000

    revision to data exploration methods

commit 977433b72eaee88f070e29d344c4790392df693a

Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Mon Mar 13 20:09:44 2023 +0000

    Add files via upload

commit 9fbc64946a312183c513bd4dbf908ab7f19ceee9
Author: James <114963128+jamestse1114@users.noreply.github.com>
Date: Mon Mar 13 20:09:21 2023 +0000

    Update README.md

commit ebb8ea3daad199357495a3d3ea4303aea75b7833
Author: James <jamestse1114@gmail.com>
Date: Mon Mar 13 17:44:08 2023 +0000

    removed ensemble learning file, later put it in another branch

commit faeae5f003ab5b953d2d5d5952bb58cd6518d1ad
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Mon Mar 13 17:39:57 2023 +0000

    upload nn

commit ccc88d8d42431f235ed3cd3e88f141ea60a2cf77
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Mon Mar 13 17:39:35 2023 +0000

    Delete neural network.ipynb

commit e026dca57a553f359a17e5deb75f3349700e9a42
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Mon Mar 13 17:38:55 2023 +0000

    upload nn

commit 746329113d762e9bd88cba1e49303b96f338386b
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Mon Mar 13 17:38:39 2023 +0000

    Delete neural network.ipynb

commit 25c998d0462351684777a9c80346a9dc0049be5c
Author: James <114963128+jamestse1114@users.noreply.github.com>

Date: Mon Mar 13 17:38:24 2023 +0000

    Add files via upload

commit f09d5888a995d6f1abd501ff1c7bbf2a186edfd2
Author: Jiayun Kang <87413952+kjy000329@users.noreply.github.com>
Date: Mon Mar 13 17:36:45 2023 +0000

    Upload nn

commit 1e500b87170c76e48ffaaab14cc67632d64278b0
Author: gagankumarba12 <ucesgkb@ucl.ac.uk>
Date: Sat Mar 11 17:44:22 2023 +0000

    Data cleaning, preprocessing, visualzation and KNN task

commit 0ce6678c7d5dd7a197ed6a751746b4bbd62556f0
Author: James <jamestse1114@gmail.com>
Date: Mon Mar 6 15:56:03 2023 +0000

    added folder of ensemble learning part with the code and merged data, it is the trial version 1 much need to be added such as indicating accuracy and modifying the parameters for each of the classifiers in the stacking method

commit 9d30319a347840615a7ad2d0f9e72478d687bc0c
Merge: 7beff9f d748d26
Author: gagankumarba12 <116358685+gagankumarba12@users.noreply.github.com>
Date: Mon Jan 30 14:10:55 2023 +0000

    Merge pull request #1 from CEGE0004/Gagan

    Commit-2: Data Preprocessing

commit d748d26893b0e60ae2473f106eb0e11f636aed9d
Author: gagankumarba12 <116358685+gagankumarba12@users.noreply.github.com>
Date: Mon Jan 30 14:08:21 2023 +0000

    Commit-1: Data Preprocessing

commit 7beff9f64a34451143dadb38500580cdf8ac1522
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date: Mon Jan 16 16:30:12 2023 +0000

Initial commit
(END)

Reference

[1] “What is a neural network? A guide to Artificial Intelligence (AI) and Machine Learning (ML) - AWS,” *Amazon Web Services, Inc.* https://aws.amazon.com/cn/what-is/neural-network/

[2] Robi Polikar (2009). Ensemble learning. Scholarpedia, 4(1):2776. doi:10.4249/scholarpedia.2776

[3] Jason Brownlee (2021). A Gentle Introduction to Ensemble Learning Algorithms. https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/

[4] mlxtend. StackingCVClassifier: Stacking with cross-validation. http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/

[5] Tang, J., S. Alelyani, and H. Liu. (2015). "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press, 498-500.

[6].A. Ajanki (2007) ‘Example of k-nearest neighbour classification’. Available at: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg