UNIL | Université de Lausanne

Machine Learning Final Project

Machine learning in Business Analytics 2024, Group C

Lodrik ADAM — Student id: 18418871
Stefan FAVRE — Student id: 20402277
Jeff MACARAEG — Student id: 19322429

2024-05-08

Table of contents

Introduction

• The context and background: course, company name, business context.

During our 1st master year as students in Management - orientation Business Analytics, we have had the opportunity to attend some lectures of Machine Learning for Business Analytics. In content of this class, we have seen multiple machine learning techniques for business context, mainly covering supervised (regressions, trees, support vector machine, neural networks) and unsupervised methods (clustering, PCA, FAMD, Auto-Encoder) but also other topics such as data splitting, ensemble methods and metrics.

• Aim of the investigation: major terms should be defined, the question of research (more generally the issue), why it is of interest and relevant in that context.

In the context of this class, me and my group have had the opportunity to work on an applied project. From scratch, we had to look for some potential dataset for using on real cases what we have learned in class. Thus, we had found an interesting dataset concerning vehicule MPG, range, engine stats and more, for more than 100 brands. The goal of our research was to predict the make of the car according to its characteristics (Consumption, range, fuel type, …) thanks to a model that we would have trained (using RF, ANN or Trees). As some cars could have several identical characteristics, but could differentiate on various other ones, we thought that it would be pertinent to have a model that was able to predict a car brand, from its features.

• Description of the data and the general material provided and how it was made available (and/or collected, if it is relevant). Only in broad terms however, the data will be further described in a following section. Typically, the origin/source of the data (the company, webpage, etc.), the type of files (Excel files, etc.), and what it contains in broad terms (e.g. “a file containing weekly sales with the factors of interest including in particular the promotion characteristics”).

The csv dataset has been found on data.world, a data catalog platform that gather various open access datasets online. The file contains more than 45’000 rows and 26 columns, each colomn concerning one

feature (such as the year of the brand, the model, the consumption per barrel, the highway mpg per fuel type and so on).

• The method that is used, in broad terms, no details needed at this point. E.g. "Model based machine learning will help us quantifying the important factors on the sales".

Among these columns, we have had to find a machine learning model that could help us to quantify the importance of the features in predicting the make of the car. (To do so, we tried to use ANN model and RF)…

• An outlook: a short paragraph indicating from now what will be treated in each following sections/chapters. E.g. "in Section 3, we describe the data. Section 4 is dedicated to the presentation of the text mining methods…" In the following sections, you will find 1st the description in the data, then in Section 2 the method used, in Section 3 the results, in Section 4 our conclusion and recommendations and finally in Section 5 our references.

Data description

Description of the data file format (xlsx, csv, text, video, etc.)

On the webpage of the dataset, we have found a CSV and xlsx format. We have decided then to download both.

The features or variables: type, units, the range (e.g. the time, numerical, in weeks from January 1, 2012 to December 31, 2015), their coding (numerical, the levels for categorical, etc.), etc.

The features or variables: type, units,…

In the original dataset, we had the following 26 columns, each one corresponding to a a feature. Here is a quick overview of the types of variables. (MAKE TABLE)

The instances: customers, company, products, subjects, etc.

In a basic instance, each row is concerning one car. We can find in order the ID of the car corresponding to a precise feature observation, then the features as seen in the table before.

Missing data pattern: if there are missing data, if they are specific to some features, etc.

Any modification to the initial data: aggregation, imputation in replacement of missing data, recoding of levels, etc.

If only a subset was used, it should be mentioned and explained; e.g. inclusion criteria. Note that if inclusion criteria do not exist and the inclusion was an arbitrary choice, it should be stated as such. One should not try to invent unreal justifications.

Classificatin Tree

First, we load the necessary packages and the dataset. We then prepare the data by handling missing values and encoding categorical variables. We select a subset of features for modeling and split the data into training and testing sets. Finally, we fit a Decision Tree model and evaluate its performance on the train and test sets.

We can first load pandas for data manipulation and the necessary modules for the Decision Tree model. We also import our dataset from a csv file.

Variable Name	Explanation
ID	Number corresponding to the precise combination of the features
Model Year	Year of the model of the car
Make	The brand of the car, character
Model	The model of the car, character
Estimated Annual Petroleum Consumption (Barrels)	Consumption in Barrels, numerical
Fuel Type 1	Regular Gasoline, Diesel,
City MPG (Fuel Type 1)	Score on each SDG except SDG 14 (16 variables)
Highway MPG (Fuel Type 1)	Population of the country
Combined MPG (Fuel Type 1)	d
Fuel Type 2	d
City MPG (Fuel Type 2)	d
Highway MPG (Fuel Type 2)	d
Combined MPG (Fuel Type 2)	d
Engine Cylinders	d
Engine Displacement	d
Drive	d
Engine Description	d
Transmission	d
Vehicle Class	d
Time to Charge EV (hours at 120v)	d
Time to Charge EV (hours at 240v)	d
Range (for EV)	d
City Range (for EV - Fuel Type 1)	d
City Range (for EV - Fuel Type 2)	d
Hwy Range (for EV - Fuel Type 1)	d
Hwy Range (for EV - Fuel Type 2)	d

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from pathlib import Path
# !pip3 install pyprojroot ----- TO INSTALL IF MISSING
from pyprojroot.here import here

# Load the dataset
file_path = here("data/Vehicle MPG - 1984 to 2023.csv")
data = pd.read_csv(file_path)
```

Now, that we have loaded the data, we can prepare it by handling missing values and encoding categorical variables. We have selected the "Engine Displacement", "Drive", "Transmission", "Vehicle Class", "Fuel Type 1", and "Model Year" columns as features for our model. We encode the categorical variables using one-hot encoding and label encoding for the "Make" column. We make sure to store our target and features in separate variables.

```
# Handle missing values: dropping rows where specific required columns are missing
required_columns = ['Engine Displacement', 'Drive', 'Transmission', 'Vehicle Class',
                    'Fuel Type 1', 'Model Year']
data_clean = data.dropna(subset=required_columns)

# Encode categorical variables
categorical_features = ['Fuel Type 1', 'Drive', 'Transmission', 'Vehicle Class']
data_clean = pd.get_dummies(data_clean, columns=categorical_features, drop_first=True)

# Label encoding for 'Make' column
le = LabelEncoder()
data_clean['Make'] = le.fit_transform(data_clean['Make'])

# Selecting a subset of features for modeling
features = data_clean[['Engine Displacement', 'Model Year']]
features = pd.concat([features, data_clean.filter(like='Fuel Type 1_'),
                      data_clean.filter(like='Drive_'), data_clean.filter(like='Transmiss
                      data_clean.filter(like='Vehicle Class_')], axis=1)
target = data_clean['Make']
```

We then split our data into training and testing sets using a 90-10 split ratio.

```
# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.1, rand
```

Fitting the Decision Tree Model

We can now dive into the modeling part by fitting a Decision Tree model to the training data and evaluating its performance on the train and test sets.

```
# Initialize and fit the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
# Checking the initial performance
train_score = round(model.score(X_train, y_train)*100, 2)
test_score = round(model.score(X_test, y_test)*100, 2)

# Printing the result
print("Trainset accuracy: ", train_score, "%","\nTestset accuracy: ", test_score, "%", se
```

```
Trainset accuracy: 74.27%
Testset accuracy: 55.04%
```

By looking at the accuracy of our model on the training and test set, we can see a clear indication of overfitting as the model performs well on the training set but poorly on the test set. To address this issue, we can prune the decision tree by limiting the maximum depth of the tree and setting a minimum number of samples required to be at a leaf node.

We set here 10 as the maximum depth of the tree, and 50 as the minimum number of samples required to be at a leaf node.

Fitting the Decision Tree Model aftre pruning the tree

```
# We'll limit the maximum depth of the tree and set a minimum number of samples required

pruned_model = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_samples_l
pruned_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, random_state=42)
```

```
# Evaluating the pruned model
pruned_train_score = round(pruned_model.score(X_train, y_train)*100, 2)
pruned_test_score = round(pruned_model.score(X_test, y_test)*100, 2)

print("Trainset accuracy: ", pruned_train_score, "%","\nTestset accuracy: ", pruned_test_
```

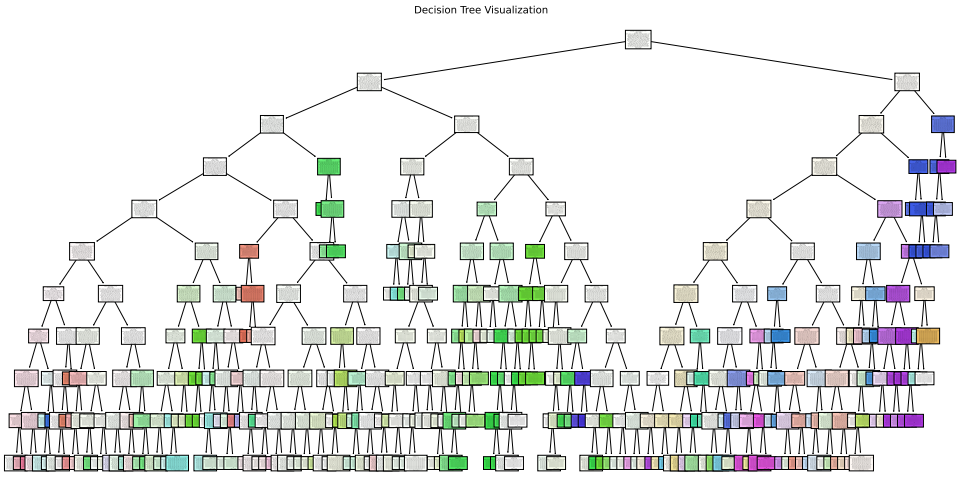
```
Trainset accuracy: 37.76%
Testset accuracy: 35.22%
```

We can see that the pruned model performs better on the test set compared to the initial model, indicating that pruning has helped reduce overfitting. But this has been at the expense of a noticable drop in the overall accuracy.

Here is what the Tree looks like after pruning: ::: {.cell}

```
# Plot the decision tree
plt.figure(figsize=(20,10))  # Set the figure size for better readability
plot_tree(pruned_model, filled=True, feature_names=X_train.columns, class_names=True)
plt.title('Decision Tree Visualization')
```

```
plt.show()
```



:::

Neural Network

First, we load the necessary packages and the dataset. We then prepare the data by handling missing values and encoding categorical variables. We select a subset of features for modeling and split the data into training and testing sets. Finally, we fit a Neural Network model and evaluate its performance on the train and test sets.

We can first load the necessary modules for the Decision Tree model. We also import our dataset from a csv file.

```
import sys
# !pip3 install pandas ----- TO INSTALL IF MISSING
# !pip3 install tensorflow ----- TO INSTALL IF MISSING
# !pip3 install scikit-learn ----- TO INSTALL IF MISSING
# !pip3 install matplotlib ----- TO INSTALL IF MISSING
# !pip3 install pyprojroot ----- TO INSTALL IF MISSING
from pyprojroot.here import here
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import pandas as pd

# Load the dataset
file_path = here("data/Vehicle MPG - 1984 to 2023.csv")
data = pd.read_csv(file_path)
```

We can then concentrate on dealing with missing values and encoding categorical variables. We will select the necessary columns for our model and split the data into training and testing sets.

```
# Drop the 'Model' column
data_dropped = data.drop('Model', axis=1)

# Check for missing values
missing_data_summary = data_dropped.isnull().sum()

# Determine the types of each column to identify categorical vs numeric columns
column_types = data_dropped.dtypes

missing_data_summary, column_types
```

```
# Drop the specified columns
data_cleaned = data_dropped.drop(['Fuel Type 2', 'Engine Description'], axis=1)

# Fill missing values for 'Engine Cylinders' and 'Engine Displacement' with their median
# since these are numerical and median is a robust measure for central tendency
for column in ['Engine Cylinders', 'Engine Displacement']:
    median_value = data_cleaned[column].median()
    data_cleaned[column].fillna(median_value, inplace=True)

# Drop rows with any remaining missing values (mostly in 'Drive' and 'Transmission' colum
data_cleaned.dropna(inplace=True)

# Check again for missing values to ensure clean dataset
final_missing_data_summary = data_cleaned.isnull().sum()
final_missing_data_summary
```

```
# Select categorical columns for one-hot encoding
categorical_columns = ['Make', 'Fuel Type 1', 'Drive', 'Transmission', 'Vehicle Class']
one_hot_encoder = OneHotEncoder()
encoded_categorical = one_hot_encoder.fit_transform(data_cleaned[categorical_columns]).to

# Create a DataFrame from the encoded categorical data
encoded_categorical_df = pd.DataFrame(encoded_categorical, columns=one_hot_encoder.get_fe
encoded_categorical_df.index = data_cleaned.index  # To align indices

# Carefully select and verify the numeric columns
numeric_columns = [col for col in data_cleaned.columns if col not in categorical_columns
scaler = StandardScaler()
normalized_numeric = scaler.fit_transform(data_cleaned[numeric_columns])

# Create a DataFrame from the normalized numeric data
normalized_numeric_df = pd.DataFrame(normalized_numeric, columns=numeric_columns)
normalized_numeric_df.index = data_cleaned.index  # To align indices

# Combine the encoded and normalized data
final_preprocessed_data = pd.concat([encoded_categorical_df, normalized_numeric_df], axis
```

```
final_preprocessed_data.head()
```

We then split our data into training and testing sets using an 80-20 split ratio. We build a neural network model with two hidden layers and compile it with the Adam optimizer and categorical cross-entropy loss function. Here is a summary of the model architecture:

```
# Prepare the target data (y) and feature data (X)
X = final_preprocessed_data.drop([col for col in final_preprocessed_data.columns if 'Make
y = final_preprocessed_data[[col for col in final_preprocessed_data.columns if 'Make_' in

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the neural network model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(y_train.shape[1], activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Overview of the model
model.summary()
```

```
Model: "sequential"

 Layer (type)                Output Shape              Param #

 dense (Dense)               (None, 128)               13,440

 dense_1 (Dense)             (None, 64)                8,256

 dense_2 (Dense)             (None, 139)               9,035

Total params: 30,731 (120.04 KB)
Trainable params: 30,731 (120.04 KB)
Non-trainable params: 0 (0.00 B)
```

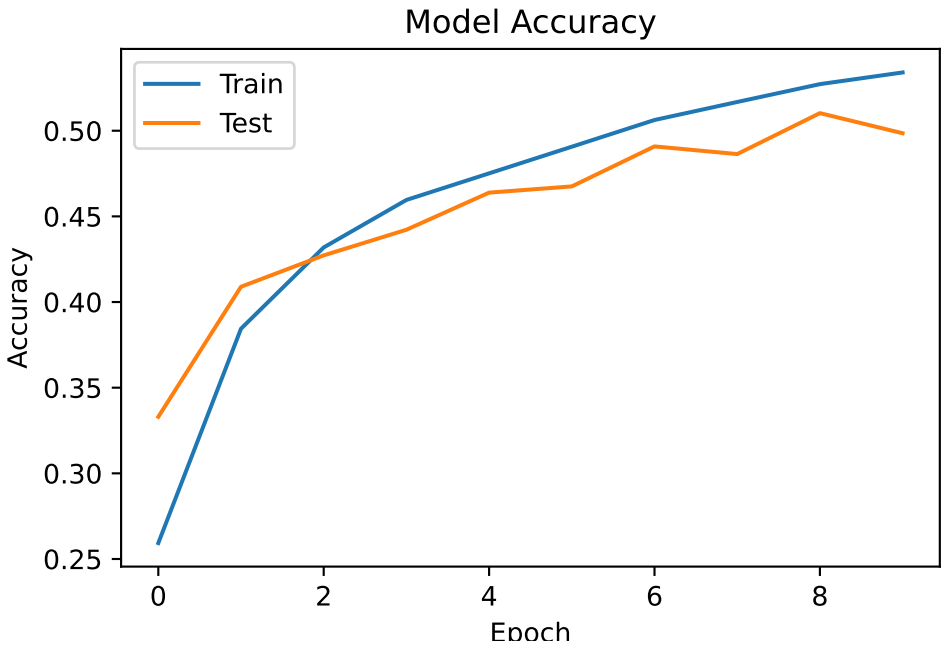
We can now train the model. We fit the model to the training data for 10 epochs with a batch size of 32.

```
# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2, batch_size=32)
```

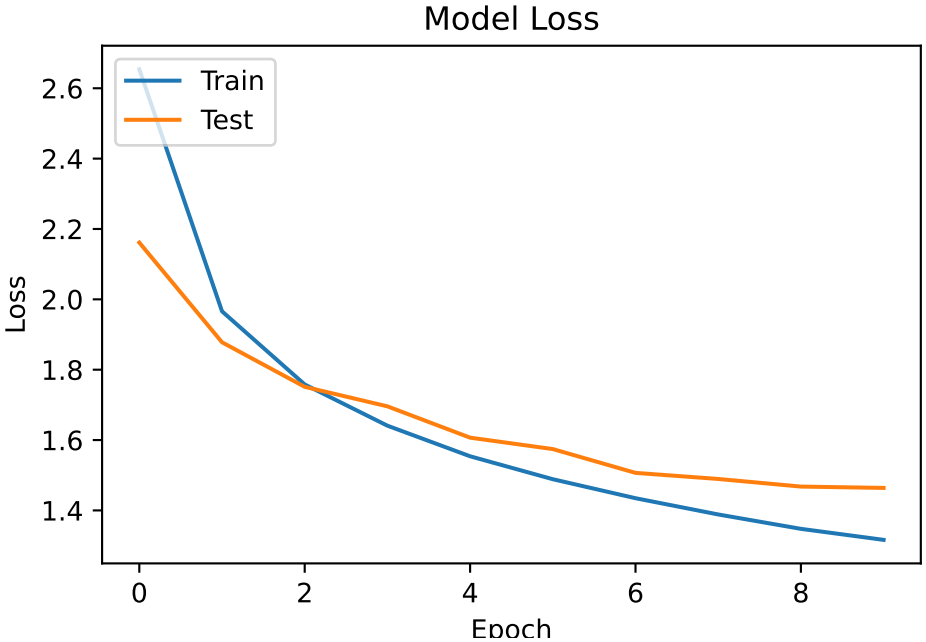
We can now plot the training and validation accuracy and loss values to visualize the model's performance.

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```


Model Loss
Train
Test
Loss
2.6
2.4
2.2
2.0
1.8
1.6
1.4
0
2
4
6
8
Epoch