Configuration Manual

MSc Research Project
Msc. in Data Analytics

Melvin Akash Ambrose MohanDoss
Student ID: x22152601

School of Computing
National College of Ireland

Supervisor: Dr. Anu Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Melvin Akash AmbroseDoss
Student ID:	x22152601
Programme:	Msc in Data Analytics
Year:	2023 -2024
Module:	Msc Research Project
Lecturer:	Dr. Anu Sahni
Submission Due Date:	14/12/2023
Project Title:	Implementing Machine Learning Models for Predicting Road Accident Severity in Northern Ireland

Word Count: 730 **Page Count: 7**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Melvin Akash Ambrose MohanDoss

Date: 14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Melvin Akash Ambrose MohanDoss
Student ID: x22152601

1 Introduction

This Configuration Manual contains a list of all the requirements needed to replicate the study and its findings in a personal setting. All models constructed, data import and exploratory data analysis, data augmentation, and software and hardware requirements are covered.

2 System Specifications

This section covers Hardware and Software requirements.

2.1 Hardware Requirements

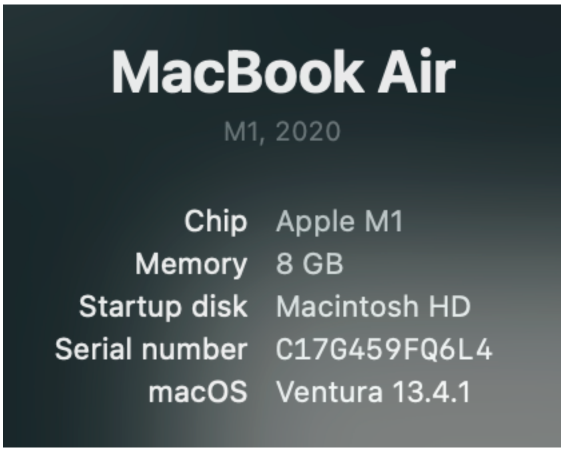

Fig 1: Hardware requirement

2.2 Software Requirements

- Jupyter Notebook (Version 6.5.2) or Google Colab
- Python (Version 3.10)
- MySql server (Version 8)
- Mysql Workbench

3 Data Collection

The Data is sourced from the UK government website.

Website link :
https://admin.opendatani.gov.uk/dataset?organization=police-service-of-northern-ireland&tags=injury+collisions

4 Data Pre-processing

The total 9 datasets from the years 2020, 2021, 2022 are merged using Mysql and Python.

First , the 9 datasets are loaded into MySql workbench and is merged year wise.

```sql
-- merging 3 datasets from 2020 (collision , casualty , vehicle)into a merged 2020 dataset, this
-- merged dataset has 37786 rows.
-- after running export as csv to jupyter notebook
SELECT *
FROM casualty2020
LEFT JOIN collision2020 ON casualty2020.a_ref = collision2020.a_ref
LEFT JOIN vehicle2020 ON casualty2020.a_ref = vehicle2020.a_ref;


-- merging 3 datasets from 2021 (collision , casualty , vehicle)into a merged 2021 dataset, this
-- merged dataset has 42495 rows.
-- after running export as csv to jupyter notebook
SELECT *
FROM casualty2021
LEFT JOIN collision2021 ON casualty2021.a_ref = collision2021.a_ref
LEFT JOIN vehicle2021 ON casualty2021.a_ref = vehicle2021.a_ref;

-- merging 3 datasets from 2022 (collision , casualty , vehicle)into a merged 2022 dataset, this
-- merged dataset has 46809 rows.
-- after running export as csv to jupyter notebook
SELECT *
FROM casualty2021
LEFT JOIN collision2021 ON casualty2021.a_ref = collision2021.a_ref
LEFT JOIN vehicle2021 ON casualty2021.a_ref = vehicle2021.a_ref;
```

Fig 2: Merging the datasets year wise.

The merged datasets from Mysql are then concatenated vertically in python.

In [2]:
```python
dataset_path_2020 = '/Users/melvinakash/Desktop/NCI/ric/datasets/merged_2020.csv'
dataset_path_2021 = '/Users/melvinakash/Desktop/NCI/ric/datasets/merged_2021.csv'
dataset_path_2022 = '/Users/melvinakash/Desktop/NCI/ric/datasets/merged_2022.csv'
```

In [3]:
```python
pd.set_option('display.max_columns',None)

data_2020 = pd.read_csv(dataset_path_2020)
data_2021 = pd.read_csv(dataset_path_2021)
data_2022 = pd.read_csv(dataset_path_2022)
```

In [4]:
```python
# Concatenate the DataFrames vertically (along rows)
df = pd.concat([data_2020, data_2021 , data_2022], ignore_index=True)
```

Fig 3: Concatenated vertically.

5 Project Development

5.1 Importing Libraries

List of python libraries used :

In [29]:
```
import pandas as pd
import numpy as np
import missingno as msno
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Fig 4: Libraries used

Some of the main Libraries used in this project were Pandas , Numpy, Matplotlib and SMOTE.

5.2 Processing

- **Treatment of Missing Values:** The pattern is MCAR and 90% data were missing so the columns were deleted. .
- **Feature selection:** Generated histograms of all the data , and removed features which were unbalanced and could potentially lead to biasing.
- **Encoding :** One hot encoding and ordinal encoding are done to two variables a_District and a_wkday.
- The **final subset** of the filtered variables is shown in Figure 5.

In [10]:
```
#taking subset of necessary data after eval of columns
data_subset = df[[ 'a_ref', 'a_District', 'a_type', 'a_veh', 'a_cas', 'a_wkday',
        'a_day', 'a_month', 'a_hour', 'a_min', 'a_gd1', 'a_gd2', 'a_ctype',
        'a_speed', 'c_class', 'c_sex', 'c_agegroup',
      'c_school', 'c_vtype', 'v_type', 'v_tow', 'v_man',
        'v_loc',  'v_impact',
        'v_sex', 'v_agegroup', 'v_hitr']]
```
Fig 5: Subset of variables to ML models.

.

- **Correlation** was checked with the target variable a_type.

The Data is split into train and test with a 80:20 split.

In [6]:
```
#over sampling using smote
```

In [7]:
```
pd.set_option('display.max_columns', None)

# Extract features and target variable
X = data.drop('a_type', axis=1)  # Features
y = data['a_type']  # Target variable

# 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, random_state=42)

# Apply SMOTE only to the training data
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_synthetic, y_train_synthetic = smote.fit_resample(X_train, y_train)

# Combine the synthetic training data with the original training data
X_train_combined = pd.concat([X_train, X_train_synthetic])
y_train_combined = pd.concat([y_train, y_train_synthetic])


```

Fig 5: SMOTE oversampler and test, train split.

- The packages or libraries to perform the above tasks are shown in the Figure 4.

5.3 Modelling

- **Oversampling and test, train split:** The target variable is imbalanced , so we take samples of minority class and oversamples it .This is done using the SMOTE oversampler.

- The following code snippets contains implementation of four machine learning and one deep learning models.

- Each model is tuned with the best hyperparameters.

5.3.1 Case Study 1 : Random Forest Classifier.

```
# RandomForestClassifier
model = RandomForestClassifier()

# Defining the hyperparameters and their possible values for grid search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Performing Grid Search
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_combined, y_train_combined)

# Getting the best hyperparameters
best_params = grid_search.best_params_

# Training the model with the best hyperparameters
best_model = RandomForestClassifier(**best_params)
best_model.fit(X_train_combined, y_train_combined)

# Making predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with the best hyperparameters: {accuracy:.4f}")

# Performing Randomized Search (alternative to Grid Search)
random_search = RandomizedSearchCV(model, param_distributions=param_grid, n_iter=10, cv=5,
                                   scoring='accuracy', random_state=42)
random_search.fit(X_train_combined, y_train_combined)

# best hyperparameters from randomized search
best_params_random = random_search.best_params_

# Train the model with the best hyperparameters from randomized search
best_model_random = RandomForestClassifier(**best_params_random)
best_model_random.fit(X_train_combined, y_train_combined)

# Making predictions on the test set
y_pred_random = best_model_random.predict(X_test)

# Evaluating the model
accuracy_random = accuracy_score(y_test, y_pred_random)
report_random = classification_report(y_test, y_pred_random)


```

```
Accuracy with the best hyperparameters: 0.9849
```

Fig 6: Code snippet of Random Forest Classifier.

```
#Confusion Matrix for RandomForestClassifier():
confusion_mat = confusion_matrix(y_test, y_pred_random)
print(f"Confusion Matrix for RandomForestClassifier():\n")
# Visualize the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Class 1', 'Class 2', 'Class 3'],
            yticklabels=['Class 1', 'Class 2', 'Class 3'])
plt.title(f'Confusion Matrix for RandomForestClassifier()')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('=' * 40)
```

Fig 7: Code snippet of Confusion matrix.

```
In [25]: print(f"Best Hyperparameters for Decision Tree: {best_params}")
```

```
Best Hyperparameters for Decision Tree: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimat
ors': 200}
```

Fig 8: Code snippet of Best Hyperparameters.

5.3.2 Case Study 2 : Decision Tree Classifier.

```
In [12]:
# Decision Tree
dt_model = DecisionTreeClassifier()

# Defining hyperparameters for Decision Tree
param_grid_dt = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Performing Grid Search for Decision Tree
grid_search_dt = GridSearchCV(dt_model, param_grid_dt, cv=5, scoring='accuracy')
grid_search_dt.fit(X_train_combined, y_train_combined)

# Getting the best hyperparameters for Decision Tree
best_params_dt = grid_search_dt.best_params_

# Training the Decision Tree model with the best hyperparameters
best_dt_model = DecisionTreeClassifier(**best_params_dt)
best_dt_model.fit(X_train_combined, y_train_combined)

# Making predictions on the test set using Decision Tree
y_pred_dt = best_dt_model.predict(X_test)

# Evaluating the Decision Tree model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
report_dt = classification_report(y_test, y_pred_dt)

#  Evaluating the model
print(f"Accuracy for Decision Tree with hyperparameter tuning: {accuracy_dt:.4f}")
print(f"Best Hyperparameters for Decision Tree: {best_params_dt}")
print(f"Classification Report for Decision Tree with hyperparameter tuning:\n{report_dt}")

```

```
Accuracy for Decision Tree with hyperparameter tuning: 0.9697
```

Fig 9: Code snippet of Decision Tree Classifier.

5.3.3 Case Study 3 : K-Nearest Neighbors

```
In [13]:
# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier()

# Defining hyperparameters for KNN
param_grid_knn = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

# Performing Grid Search for KNN
grid_search_knn = GridSearchCV(knn_model, param_grid_knn, cv=5, scoring='accuracy')
grid_search_knn.fit(X_train_combined, y_train_combined)

# Get the best hyperparameters for KNN
best_params_knn = grid_search_knn.best_params_

# Training the KNN model with the best hyperparameters
best_knn_model = KNeighborsClassifier(**best_params_knn)
best_knn_model.fit(X_train_combined, y_train_combined)

# Making predictions on the test set using KNN
y_pred_knn = best_knn_model.predict(X_test)

# Evaluating the KNN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

# Evaluating the model
print(f"Accuracy for K-Nearest Neighbors with hyperparameter tuning: {accuracy_knn:.4f}")
print(f"Best Hyperparameters for K-Nearest Neighbors: {best_params_knn}")
print(f"Classification Report for K-Nearest Neighbors with hyperparameter tuning:\n{report_knn}")

Accuracy for K-Nearest Neighbors with hyperparameter tuning: 0.9901
```

Fig 10: Code snippet of K-Nearest Neighbors.

5.3.4 Case Study 4 : Artificial Neural Network

```
In [14]:
# Artificial Neural Network (ANN)
ann_model = MLPClassifier()

# Defining hyperparameters for ANN
param_grid_ann = {
    'hidden_layer_sizes': [(50, 25), (100, 50), (150, 75)],
    'alpha': [0.0001, 0.001, 0.01]
}

# Performing Grid Search for ANN
grid_search_ann = GridSearchCV(ann_model, param_grid_ann, cv=5, scoring='accuracy')
grid_search_ann.fit(X_train_combined, y_train_combined)

# Getting the best hyperparameters for ANN
best_params_ann = grid_search_ann.best_params_

# Train the ANN model with the best hyperparameters
best_ann_model = MLPClassifier(**best_params_ann)
best_ann_model.fit(X_train_combined, y_train_combined)

# Making predictions on the test set using ANN
y_pred_ann = best_ann_model.predict(X_test)

# Evaluating the ANN model
accuracy_ann = accuracy_score(y_test, y_pred_ann)
report_ann = classification_report(y_test, y_pred_ann)

# Evaluating the model
print(f"Accuracy for Artificial Neural Network with hyperparameter tuning: {accuracy_ann:.4f}")
print(f"Best Hyperparameters for Artificial Neural Network: {best_params_ann}")
print(f"Classification Report for Artificial Neural Network with hyperparameter tuning:\n{report_ann}")

Accuracy for Artificial Neural Network with hyperparameter tuning: 0.8382
```

Fig 11: Code snippet of ANN.