National College of Ireland

Configuration Manual

MSc Research Project
Research in Computing

Ananya Kachawa
Student ID: X21136751

School of Computing
National College of Ireland

Supervisor: Mr. Taimur Hafees

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ananya Kachawa
Student ID:	X21136751
Programme:	Master of Science in Data Analytics Information
Year:	2023
Module:	MSc Research Project
Supervisor:	Mr. Taimur Hafeez
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	206
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:	Ananya Kachawa
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

Ananya Kachawa
x21136751

1 Introduction

The setup manual contains all of the conditions that must be met in order to reproduce the research experiment and the results of the experiment on each specific setting.The prerequisites for the software and hardware are included in this document, together with the code for Data Import and Preprocessing, Exploratory Data Analysis, all models that were developed, and Evaluation.

2 Hardware and Software Requirments

Opertating System	Windows 10 or later
Processor	Core I5 or later
Memory	8 GB RAM
Storage	100 GB free disk Space
Python Environment	Python 3.6
Tool	Jupyter Notebook

3 Data Collection

The dataset has been taken from the Kaggle. Below are links for all three datasets:

https://www.kaggle.com/blastchar/telco-customer-churn/download
https://www.kaggle.com/becksddf/churn-in-telecoms-dataset/download
https://www.kaggle.com/c/customer-churn-prediction-2020/data

4 Importing libraries

```
]: import pandas as pd
   import matplotlib.pyplot as plt
   import seaborn as sns
   from sklearn.preprocessing import StandardScaler, OneHotEncoder
   from sklearn.impute import SimpleImputer
   from sklearn.compose import ColumnTransformer
   from sklearn.pipeline import Pipeline
   from sklearn.model_selection import train_test_split
   from sklearn.linear_model import LogisticRegression
   from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
   from sklearn.metrics import classification_report, accuracy_score
```

5 Importing datasets

In [2]:
```
# Load the first dataset
telco_data = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')

# Display the first few rows of the dataset
telco_data.head()
```

In [3]:
```
# Load the second dataset
bigml_data = pd.read_csv('bigml_59c28831336c6604c800002a.csv')

# Display the first few rows of the dataset
bigml_data.head()
```

In [4]:
```
# Load the third dataset
train_data = pd.read_csv('churntrain.csv')

# Display the first few rows of the dataset
train_data.head()
```

6 Data pre-processing and exploratory data analysis

In [5]:
```
# Check for missing values in each dataset
missing_telco = telco_data.isnull().sum()
missing_bigml = bigml_data.isnull().sum()
missing_train = train_data.isnull().sum()

missing_telco, missing_bigml, missing_train
```

Out[5]:
```
(customerID          0
 gender              0
 SeniorCitizen       0
 Partner             0
 Dependents          0
 tenure              0
 PhoneService        0
 MultipleLines       0
 InternetService     0
 OnlineSecurity      0
 OnlineBackup        0
 DeviceProtection    0
 TechSupport         0
 StreamingTV         0
 StreamingMovies     0
 Contract            0
 PaperlessBilling    0
 PaymentMethod       0
 MonthlyCharges      0
 TotalCharges        0
 Churn               0
 dtype: int64,
 state                     0
 account length            0
 area code                 0
 phone number              0
 international plan        0
 voice mail plan           0
```

There were no missing values were found in all three datasets.

```
dtype: int64)
```

In [6]:
```
# Check for duplicate rows in each dataset
duplicates_telco = telco_data.duplicated().sum()
duplicates_bigml = bigml_data.duplicated().sum()
duplicates_train = train_data.duplicated().sum()

duplicates_telco, duplicates_bigml, duplicates_train
```

Out[6]:
```
(0, 0, 0)
```

There were no duplicates were found in all three datasets.

In [7]:
```
# Standardize the churn column for all datasets
telco_data['Churn'] = telco_data['Churn'].replace({'Yes': 'Yes', 'No': 'No'})
bigml_data['churn'] = bigml_data['churn'].replace({True: 'Yes', False: 'No'})
train_data['churn'] = train_data['churn'].replace({'yes': 'Yes', 'no': 'No'})

# Confirm the unique values in the churn column for each dataset
churn_values_telco = telco_data['Churn'].unique()
churn_values_bigml = bigml_data['churn'].unique()
churn_values_train = train_data['churn'].unique()

churn_values_telco, churn_values_bigml, churn_values_train
```

Out[7]:
```
(array(['No', 'Yes'], dtype=object),
 array(['No', 'Yes'], dtype=object),
 array(['No', 'Yes'], dtype=object))
```

```
# Function to plot churn distribution for each dataset
def plot_churn_distribution(data, title):
    ax = data.value_counts().plot(kind='bar', color=['skyblue', 'salmon'], alpha=0.8)
    plt.title(title)
    plt.xlabel('Churn')
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    for p in ax.patches:
        ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', xytext=(0, 5), textcoords='offset points')
    plt.show()

# Plot churn distribution for each dataset
plot_churn_distribution(telco_data['Churn'], 'Churn Distribution - Telco Data')
plot_churn_distribution(bigml_data['churn'], 'Churn Distribution - BigML Data')
plot_churn_distribution(train_data['churn'], 'Churn Distribution - Train Data')
```

The above is used to visualize churn distribution of each dataset.

```
In [9]: # Function to plot distribution of numerical features
        def plot_feature_distribution(data, column, title):
            plt.hist(data[column], bins=30, color='skyblue', alpha=0.8, edgecolor='black')
            plt.title(title)
            plt.xlabel(column)
            plt.ylabel('Frequency')
            plt.grid(axis='y')
            plt.show()

        # Plot distributions of selected features
        plot_feature_distribution(telco_data, 'tenure', 'Distribution of Tenure - Telco Data')
        plot_feature_distribution(bigml_data, 'total day minutes', 'Distribution of Total Day Minutes - BigML Data')
        plot_feature_distribution(train_data, 'total_eve_minutes', 'Distribution of Total Eve Minutes - Train Data')
```

The above code is used to plot the distribution of numerical features.
Similarly, many visualization has been done for showing the insight of data.

7 Data transformation

```
# Convert 'TotalCharges' to numeric
telco_data['TotalCharges'] = pd.to_numeric(telco_data['TotalCharges'], errors='coerce')

# Identify categorical and numerical columns
categorical_columns = telco_data.select_dtypes(include=['object']).columns.tolist()
categorical_columns.remove('customerID')  # Exclude the customerID as it's not a feature
numerical_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']

# Imputer for handling missing values
numeric_imputer = SimpleImputer(strategy='mean')

# Apply one-hot encoding to categorical variables
categorical_encoder = OneHotEncoder(drop='first')

# Scale numerical features
numeric_scaler = StandardScaler()

# Creating a column transformer to transform categorical and numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_imputer, numerical_columns),
        ('cat', categorical_encoder, categorical_columns)
    ])

# Applying the transformations to the telco_data
telco_data_processed = preprocessor.fit_transform(telco_data.drop(columns=['customerID']))

# The result is a numpy array. We can convert it to a DataFrame for better readability
# Also, we need to get the feature names for the transformed columns
cat_features = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_columns)
processed_columns = numerical_columns + cat_features.tolist()

# Creating the processed DataFrame
telco_data_processed_df = pd.DataFrame(telco_data_processed, columns=processed_columns)

# Showing the first few rows of the processed DataFrame
telco_data_processed_df.head()
```

In [27]:
```
# Identifying categorical and numerical columns for bigml_data and train_data
categorical_columns_bigml = bigml_data.select_dtypes(include=['object']).columns.tolist()
categorical_columns_bigml.remove('phone number')  # Exclude phone number as it's not a feature
numerical_columns_bigml = bigml_data.select_dtypes(include=['float64', 'int64']).columns.tolist()

categorical_columns_train = train_data.select_dtypes(include=['object']).columns.tolist()
categorical_columns_train.remove('area_code')  # Assuming area_code to be numerical
numerical_columns_train = train_data.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Creating a column transformer for bigml_data
preprocessor_bigml = ColumnTransformer(
    transformers=[
        ('num', numeric_imputer, numerical_columns_bigml),
        ('cat', categorical_encoder, categorical_columns_bigml)
    ])

# Applying the transformations to the bigml_data
bigml_data_processed = preprocessor_bigml.fit_transform(bigml_data)

# Creating the processed DataFrame for bigml_data
cat_features_bigml = preprocessor_bigml.named_transformers_['cat'].get_feature_names_out(categorical_columns_bigml)
processed_columns_bigml = numerical_columns_bigml + cat_features_bigml.tolist()
#bigml_data_processed_df = pd.DataFrame(bigml_data_processed, columns=processed_columns_bigml)

# Creating a column transformer for train_data
preprocessor_train = ColumnTransformer(
    transformers=[
        ('num', numeric_imputer, numerical_columns_train),
        ('cat', categorical_encoder, categorical_columns_train)
    ])

# Applying the transformations to the train_data
train_data_processed = preprocessor_train.fit_transform(train_data)

# Creating the processed DataFrame for train_data
cat_features_train = preprocessor_train.named_transformers_['cat'].get_feature_names_out(categorical_columns_train)
processed_columns_train = numerical_columns_train + cat_features_train.tolist()
#train_data_processed_df = pd.DataFrame(train_data_processed, columns=processed_columns_train)
```

In [28]:
```
# Inspecting the shape of the processed data and the length of the feature names list
processed_data_shape = bigml_data_processed.shape
expected_columns_length = len(processed_columns_bigml)

processed_data_shape, expected_columns_length
```

Out[28]:
```
((3333, 69), 69)
```

In [29]:
```
# Checking the lengths of numerical and categorical feature lists
len_numerical_columns_bigml = len(numerical_columns_bigml)
len_categorical_features_bigml = len(cat_features_bigml)

len_numerical_columns_bigml, len_categorical_features_bigml, len_numerical_columns_bigml + len_categorical_features_bigml
```

Out[29]:
```
(16, 53, 69)
```

In [30]:
```
# Reconstructing the DataFrame for bigml_data
bigml_data_processed_df = pd.DataFrame(bigml_data_processed.toarray(), columns=processed_columns_bigml)

# Showing the first few rows of the reconstructed DataFrame
bigml_data_processed_df.head()
```

In [31]:
```
# Generating feature names for the encoded categorical columns in train_data
cat_features_train = preprocessor_train.named_transformers_['cat'].get_feature_names_out(categorical_columns_train)
processed_columns_train = numerical_columns_train + cat_features_train.tolist()

# Reconstructing the DataFrame for train_data
train_data_processed_df = pd.DataFrame(train_data_processed.toarray(), columns=processed_columns_train)

# Showing the first few rows of the reconstructed DataFrame for train_data
train_data_processed_df.head()
```

8 Data Preparation

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score

# Splitting the telco_data into features (X) and target variable (y)
X_telco = telco_data_processed_df.drop('Churn_Yes', axis=1)
y_telco = telco_data_processed_df['Churn_Yes']

# Splitting into training and testing sets (70% train, 30% test)
X_train_telco, X_test_telco, y_train_telco, y_test_telco = train_test_split(X_telco, y_telco, test_size=0.3, random_state=42)
```

9 Model implementation and evaluation

Logistic Regression

In [33]:
```
# Initializing the models
logreg_telco = LogisticRegression(max_iter=1000)
rf_telco = RandomForestClassifier()
gb_telco = GradientBoostingClassifier()
```

In [39]:
```
# Training Logistic Regression
logreg_telco.fit(X_train_telco, y_train_telco)
y_pred_logreg_telco = logreg_telco.predict(X_test_telco)


# Evaluating the models
logreg_report_telco = classification_report(y_test_telco, y_pred_logreg_telco)

logreg_accuracy_telco = accuracy_score(y_test_telco, y_pred_logreg_telco)


print(logreg_report_telco)
print('Accuracy: ',logreg_accuracy_telco)
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.90      0.88      1539
         1.0       0.69      0.58      0.63       574

    accuracy                           0.81      2113
   macro avg       0.77      0.74      0.75      2113
weighted avg       0.81      0.81      0.81      2113

Accuracy:  0.8140085186938003
```

Random Forest

In [41]:
```
# Training Random Forest
rf_telco.fit(X_train_telco, y_train_telco)
y_pred_rf_telco = rf_telco.predict(X_test_telco)



# Evaluating the models
rf_report_telco = classification_report(y_test_telco, y_pred_rf_telco)


rf_accuracy_telco = accuracy_score(y_test_telco, y_pred_rf_telco)
print(rf_report_telco)
print('Accuracy: ',rf_accuracy_telco)
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.90      0.86      1539
         1.0       0.64      0.46      0.53       574

    accuracy                           0.78      2113
   macro avg       0.73      0.68      0.70      2113
weighted avg       0.77      0.78      0.77      2113

Accuracy:  0.7823000473260767
```

Gradient Boosting

In [42]:
```
# Training Gradient Boosting
gb_telco.fit(X_train_telco, y_train_telco)
y_pred_gb_telco = gb_telco.predict(X_test_telco)

# Evaluating the models
gb_report_telco = classification_report(y_test_telco, y_pred_gb_telco)

gb_accuracy_telco = accuracy_score(y_test_telco, y_pred_gb_telco)
print(gb_report_telco)
print('Accuracy: ',gb_accuracy_telco)
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.91      0.87      1539
         1.0       0.68      0.53      0.59       574

    accuracy                           0.80      2113
   macro avg       0.76      0.72      0.73      2113
weighted avg       0.79      0.80      0.80      2113

Accuracy:  0.8035967818267865
```