BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604 Machine Learning

Coursework Report

Module Leader: Mr. Sahan Priyanayana

RGU Student ID	:	2330906
IIT Student ID	:	20230145
Student Name	:	P. K. S. D. C. Bandara

Table of Contents

Introduction

This report demonstrates the outcomes and analysis of the machine learning module coursework. Binary classification problem which predicts whether a client will subscribe to a term deposit based on historical bank marketing data is the primary goal of this project. The dataset called "Bank Marketing" from UCI machine learning repository was used as the primary dataset for the classification training and testing.

To achieve the above objective, Random Forest and Neural Network machine learning models were implemented and evaluated. This report provides a clear overview on data preprocessing, model training and evaluation.

Corpus Preparation

Initially dataset was imported using Pandas library and separated to columns

1. **Handling "Unknown" values**

Dataset did not have any null values. But it had values which were named as "unknown" to represent null values. First values that was unknown was converted to null values. All together there were 12718 rows that contains null values. All null rows were dropped except rows that contain null values in the default column.

2. **Handling Duplicates**

There were 12 duplicate rows in the dataset. All duplicate rows were dropped.

3. **Handling Outliers**

Outliers was identified by examining boxplots of numeric columns. After the inspection was done outliers were identified in 3 features (age, campaign, duration). Outliers were removed respect to the 1st and 3rd quantiles of each feature.

4. **Standardization**

Numeric features were transformed to a constant format across the dataset. StandardScaler (z-Score) was used in transforming data.

5. **Encoding Categorical columns**

All the categorical columns were transformed into numerical format using encoding. One-hot encoding and label encoding was used in this process.

6. **Handling Default column**

Default column was dropped due to the extreme imbalance. Default column was an extremely imbalanced column which has Boolean values. Because of the extreme imbalance this column will not help in resulting the final outcome in the training process. Considering those, Default column was dropped.

7. **Handling Duration column**

In the text file of the dataset it is given that the attribute highly affects the output target (duration=0 -> output =0). Such scenario is not suitable for realistic predictive model. Dataset contained 4 rows that duration was equal to 0. Those rows were removed.

8. **Handling the target variable imbalance**

Target variable y also had a high imbalance which was 88.73% and 11.26% this might cause a biasness towards one class. Oversampling method SMOTE was used in this scenario to balance the dataset.

9. **Removing Correlated features**

Removed features that the correlation is higher than 0.9 for better performance of neural network though the correlativity does not affect random forest models

Solution Methodology

Well known supervised learning algorithms (***Random Forest*** & ***Neural networks***) were used for this project solution. Both of these models has their own strengths in these kind of classification problems.

Random Forest

Random Forest is made out of many decision trees where the mean output of each tree is considered as the prediction. Such model is suitable for this kind of project for several reasons.

- Handling overfitting
- Nonlinear approach
- High predictive accuracy with minimal hyper parameter tuning

Neural Network

Neural network is a combination of neurons which has a weight, bias and an activation function. In this specific problem rectified linear unit and Sigmoid functions are used as activation functions of hidden layers and output layer.

- Effective for High-Dimensional Data
- Ability to Model Complex Patterns
- Normalization Benefits

Evaluation Methodology

After the model training, evaluation is one of the most important tasks that has to follow. Metrics such as accuracy, precision, recall and f1-Score is used in evaluating the project

Accuracy: Total number of true negatives and true positives are divided by total number of predictions

Accuracy = (TN+ TP) / (TP + TN + FP + FN)

Precision: Precision measures out of all the positive predictions how much were actually positive. This is calculated by dividing the total number of true positives by total number of predicted positives

Precision = TP / (TP + FP)

Recall: Measures how well model has picked actual positives from the dataset. This is calculated by dividing true positives by actual number of positives.

Recall = TP / (TP + FN)

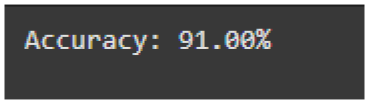
F1 Score: Harmonic mean of precision and recall is known as f1 score.

F1 = 2 * ((Precision * Recall) / (Precision + Recall))

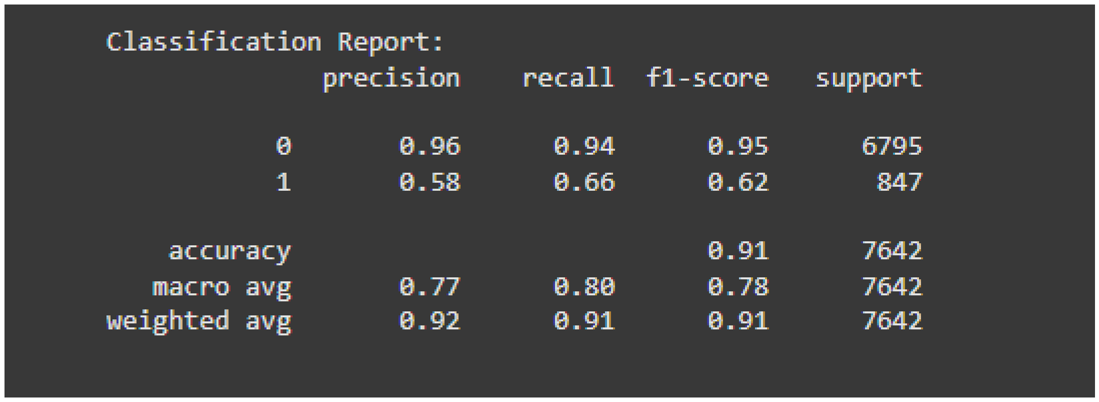
Evaluation of the solution

Random Forest

Classification Report

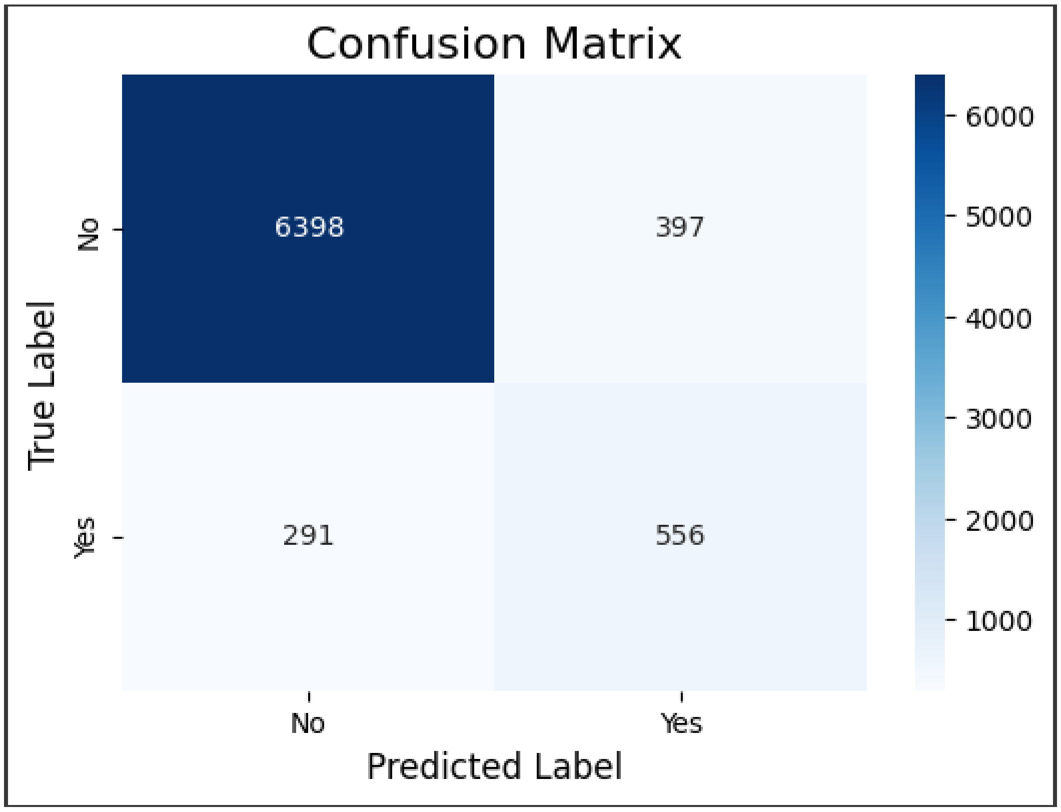


```
Accuracy: 91.00%
```



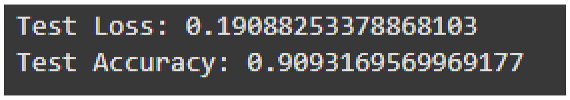
```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      6795
           1       0.58      0.66      0.62       847

    accuracy                           0.91      7642
   macro avg       0.77      0.80      0.78      7642
weighted avg       0.92      0.91      0.91      7642
```

Confusion matrix

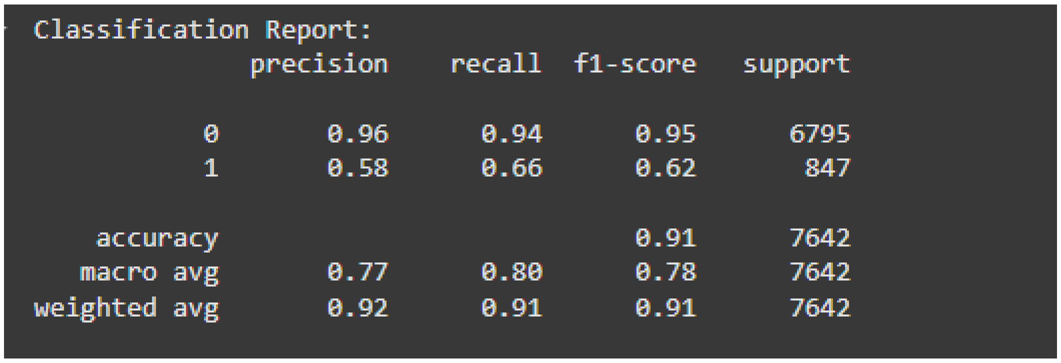


Neural Network

Classification Report

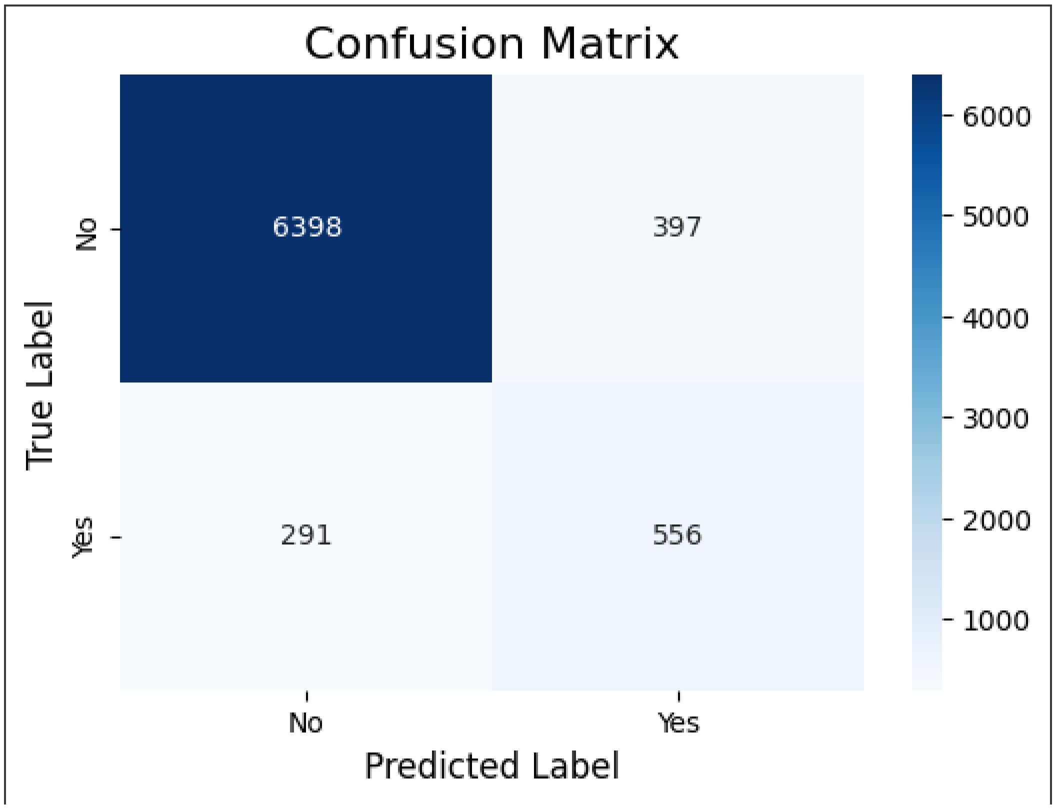


```
Test Loss: 0.19088253378868103
Test Accuracy: 0.9093169569969177
```



```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      6795
           1       0.58      0.66      0.62       847

    accuracy                           0.91      7642
   macro avg       0.77      0.80      0.78      7642
weighted avg       0.92      0.91      0.91      7642
```

Confusion matrix



Limitations

Data Related Limitations

- **Class Imbalance**: Dataset has a significant class imbalance in target variable. Even though methods such as SMOTE is used, it may not perfectly reflect real world scenario
- **Irrelevant or Noisy Features**: Some features (duration) have direct relations with the target and might leak information about the target variable. Such instances cannot be used in real world predictive models.

Model Related Limitations

- **Scalability**: Random forest face performance issues for larger datasets (Memory and Training time)
- **Overfitting**: samples could lead to overfitting, particularly in Neural Networks

Enhancements

Since this is a dataset with large number of dimensions, Dimensionality Reduction can be performed using a technique like PCA. Also experimenting with calculating and removing less important features can also be done. Can perform a hyperparameter search using GridSearchCV or RandomizedSearchCV for parameters rather than randomly assigning values to those parameters.

GITHUB Repository

Link: https://github.com/daham-13/CM2604-Coursework

Reference list

Bhavesh Bhatt (2019). *SMOTE (Synthetic Minority Oversampling Technique) for Handling Imbalanced Datasets*. [online] YouTube. Available at: https://www.youtube.com/watch?v=U3X98xZ4_no [Accessed 30 Dec. 2024].

GeeksforGeeks (2019). *Neural Networks | A beginners guide*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/.

IBM Technology (2022). *Neural Networks Explained in 5 minutes*. *YouTube*. Available at: https://www.youtube.com/watch?v=jmmW0F0biz0 [Accessed 31 Dec. 2024].

Six Sigma Pro SMART (2024). *Association of Categorical Variables | Cramer's V*. [online] YouTube. Available at: https://www.youtube.com/watch?v=wZeCM8ordWk [Accessed 20 Dec. 2024].

Starmer, J. (2020). *StatQuest: Random Forests Part 2: Missing data and clustering*. *YouTube*. Available at: https://www.youtube.com/watch?v=sQ870aTKqiM.

Source code

```python
#import pandas
import pandas as pd

#Fetch data
url = "https://raw.githubusercontent.com/daham-13/CM2604-
Coursework/refs/heads/main/Data/bank-additional/bank-additional-full.csv"
data = pd.read_csv(url, sep=";")
print(data.head())

"""# Data Preprocessing

### Data Overview
"""

#Basic information of data

# General Information
print(data.info())

# Summary Statistics
print(data.describe(include="all"))

# Shape of the Data
print(f"Dataset contains {data.shape[0]} rows and {data.shape[1]} columns.")

#Null values

# Replace "unknown" with NaN to treat them as missing values
data = data.replace("unknown", pd.NA)

#Finding the total number of null values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)

missing_presentage = (data.isnull().sum() / len(data)) * 100
print("\nMissing Values Percentage:\n", missing_presentage)

print("All null rows: ",missing_values.sum())

#Duplicate Data

# Count Duplicate Rows
```

```python
duplicates = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

#Feature Analization

# Identify numerical columns
numerical_columns = data.select_dtypes(include=["int64", "float64"]).columns
print("Numerical Columns:\n", numerical_columns)

# Summary Statistics
print(data[numerical_columns].describe())

# Correlation Heatmap
from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8))
sns.heatmap(data[numerical_columns].corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()

# Identify categorical columns
categorical_columns = data.select_dtypes(include=["object", "category"]).columns
print("Categorical Columns:\n", categorical_columns)

# Unique values and frequency
for col in categorical_columns:
    print(f"{col}: {data[col].nunique()} unique values")
    print(data[col].value_counts())
    print("\n")

#Cramér's V correlation for categorical variables
import numpy as np
from scipy.stats import chi2_contingency

# Function to calculate C
def cramers_v(x, y):
    contingency_table = pd.crosstab(x, y)
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    return np.sqrt(chi2 / (len(x) * (min(contingency_table.shape) - 1)))

# Create an empty correlation matrix
corr_matrix = pd.DataFrame(np.zeros((len(categorical_columns),
len(categorical_columns))),
                            columns=categorical_columns,
index=categorical_columns)

# Fill the correlation matrix with Cramér's V values
for i in range(len(categorical_columns)):
    for j in range(i, len(categorical_columns)):
        corr_matrix.iloc[i, j] = cramers_v(data[categorical_columns[i]],
data[categorical_columns[j]])
        corr_matrix.iloc[j, i] = corr_matrix.iloc[i, j]


plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=0, vmax=1)
plt.title("Cramér's V Heatmap")
plt.show()

# Plot histogram for numerical columns
data[numerical_columns].hist(bins=15, figsize=(15, 10))
plt.suptitle("Numerical Features Distributions")
plt.show()

# Plot bar plots for categorical columns
for col in categorical_columns:
    data[col].value_counts().plot(kind='bar', figsize=(7.5, 5),
title=f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.show()

# Imbalance Ratio
class_distribution = data["y"].value_counts(normalize=True) * 100
print("Class Distribution (%):\n", class_distribution)

"""### Preprocess"""

# Columns to drop based on low percentage of missing values
columns_to_check = ['job', 'marital', 'education', 'housing', 'loan']

# Drop rows where any of the selected columns have null values
dataframe = data.dropna(subset=columns_to_check)
dataframe = dataframe.drop('default', axis=1)
print("Cleaned DataFrame (rows with nulls in specified columns dropped):")
print(dataframe.info())

#Null values
```

```
#Finding the total number of null values
missing_values = dataframe.isnull().sum()
print("Missing Values:\n", missing_values)

# Remove duplicates
dataframe = dataframe.drop_duplicates()

#Remove Outliers
#Identifying Outliers

# Convert 'pdays' to a binary column: 1 if previously contacted, 0 if not
dataframe['pdays'] = (dataframe['pdays'] != 999).astype(int)

numerical_columns = dataframe.select_dtypes(include=["int64", "float64"]).columns


for col in numerical_columns:
  sns.boxplot(data=dataframe, x=col)
  plt.title(f"Boxplot of {col}")
  plt.show()

import seaborn as sns
import matplotlib.pyplot as plt

# Define the columns to check
columns_to_check = ['age', 'campaign', 'duration']
multipliers = (2.8,17,15)
def remove_outliers(col, multiplier):
    # Calculate Q1, Q3, and IQR
    global dataframe
    Q1 = dataframe[col].quantile(0.25)
    Q3 = dataframe[col].quantile(0.75)
    IQR = Q3 - Q1

    # Calculate bounds for outliers
    lower_bound = Q1 - multiplier * IQR
    upper_bound = Q3 + multiplier * IQR

    # Identify outliers
    outliers = dataframe[(dataframe[col] < lower_bound) | (dataframe[col] >
upper_bound)]

    # Boxplot to visualize the data
    plt.figure(figsize=(10, 6))
```

```python
    sns.boxplot(data=dataframe, x=col)
    plt.title(f"Boxplot for {col} (Outliers Highlighted)")

    # Highlight outliers in red on the boxplot
    outlier_values = outliers[col].values
    for outlier_value in outlier_values:
      plt.scatter(x=outlier_value, y=0, color='red', s=50, label='Outlier')
    plt.show()

    # Print the bounds and count of outliers
    print(f"Bounds for {col} - Lower: {lower_bound}, Upper: {upper_bound}")
    print(f"Number of Outliers: {len(outliers)}\n")

    dataframe.drop(outliers.index, inplace=True)

for col, multiplier in zip(columns_to_check, multipliers):
    remove_outliers(col, multiplier)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
dataframe[['age', 'campaign', 'pdays',  'previous', 'emp.var.rate',
'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']] =
scaler.fit_transform(dataframe[['age', 'campaign', 'pdays',   'previous',
'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']])
print(dataframe)
dataframe.shape

#Convert categorical variables into numerical

encode_features = ['job', 'marital', 'education', 'contact', 'month',
'day_of_week', 'poutcome']
map_features = ['housing', 'loan', 'y']

def encode(columns1, columns2):
    global dataframe
    for column in columns1:
      dataframe = pd.get_dummies(dataframe, columns=[column], drop_first=True)

    for column in columns2:
      dataframe[column] = dataframe[column].map({'yes': 1, 'no': 0})

encode(encode_features, map_features)

# Count rows where duration is 0
```

```
zero_duration_count = dataframe[dataframe['duration'] == 0].shape[0]
print(f"Number of rows with duration = 0: {zero_duration_count}")
 # Drop rows where duration is 0
dataframe = dataframe[dataframe['duration'] != 0]

# Confirm the rows have been dropped
print(f"Remaining rows after dropping duration = 0: {dataframe.shape[0]}")

dataframe.info()

"""# Split"""

from sklearn.model_selection import train_test_split

# Separate features and target
X = dataframe.drop("y", axis=1)
y = dataframe["y"]

# Split 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)

#Sample dataset

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

smote = SMOTE(random_state=42, sampling_strategy="auto")
X_train_sampled, y_train_sampled = smote.fit_resample(X_train, y_train)

"""## Random Forest

### Train
"""

from sklearn.ensemble import RandomForestClassifier

randomForestModel = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=5,
    class_weight='balanced',
    random_state=42
)
```

```
randomForestModel.fit(X_train_sampled, y_train_sampled)

"""### Test and Evaluate  """

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

y_pred = randomForestModel.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"],
yticklabels=["No", "Yes"])

# Add labels, title, and formatting
plt.title("Confusion Matrix", fontsize=16)
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.show()

"""## Nural Network"""

# Remove Correlated Column
corr_matrix = dataframe.corr()
correlated_features = set()
for i in range(len(corr_matrix.columns)):
    for j in range(i):
        if abs(corr_matrix.iloc[i, j]) > 0.85:
            colname = corr_matrix.columns[i]
            correlated_features.add(colname)

dataframe = dataframe.drop(columns=correlated_features)
```

```python
"""### Train"""

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Separate features and target
X = dataframe.drop("y", axis=1)
y = dataframe["y"]

# Split 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 to balance the training data
smote = SMOTE(random_state=42, sampling_strategy = 0.85)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import classification_report

# Define the Neural Network
model = Sequential([
    Dense(64, activation='relu', input_dim=X_train_smote.shape[1]),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(X_train_smote, y_train_smote, validation_data=(X_train_smote,
y_train_smote),
                    epochs=20, batch_size=32, verbose=1)

# Predict on the test set
y_pred_neural = model.predict(X_test)
y_pred_neural = (y_pred > 0.7).astype(int)

"""### Test and Evalurate"""

test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```python
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_neural))

# Generate a confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_neural))

cm = confusion_matrix(y_test, y_pred_neural)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"],
yticklabels=["No", "Yes"])

# Add labels, title, and formatting
plt.title("Confusion Matrix", fontsize=16)
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.show()
```