National College of Ireland

Deep learning approach to analyze Sleep & Apps usage pattern to predict Problematic Smartphone Usage (PSU) – Configuration Manual

MSc Research Project
Data Analytics

Ashok Kumar Ghunalan
Student ID: X22193561

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ashok Kumar Ghunalan
Student ID:	X22193561
Programme:	Master of Science in Data Analytics **Year:** 2023 - 2024
Module:	Research Project
Supervisor:	Ahmed Makki
Submission Due Date:	16–Sep-2024
Project Title:	Deep learning approach to analyse Sleep & Apps Usage pattern to predict Problematic Smartphone Usage (PSU)
Word Count:	1676 **Page Count:** 12

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:	Ashok Kumar Ghunalan
Date:	16–Sep-2024

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):	

Deep learning approach to analyze Sleep & Apps usage pattern to predict Problematic Smartphone Usage (PSU) – Configuration Manual

Ashok Kumar Ghunalan
X22193561

1 Introduction

This configuration manual outlines the research to predict Problematic Smartphone Usage (PSU) by analysing the sleep and apps usage pattern by employing deep learning techniques. The report contains the detailed information on the system configuration, instruction on how to download the dataset and sleep & apps usage analysis methodology followed in the research.

2 System Configuration

In this section the system configuration used for this research has been listed such as hardware, software and packages employed. This provides a clear view of the configuration requirement needed for research analysis.

2.1 Hardware configuration

For this research the author has used a basic hardware system containing 11th Generation Intel I3 processor with 8GB RAM and storage of 20GB. This simple configuration was selected due to cost effective and widely accessible environment to conduct the research.

2.2 Software configuration

The software's used for this research are Anaconda Navigator 2.5.0 installed in the local system to access Jupyter notebook and it efficiently handles the packages & libraries needed. Jupyter Notebook is used as the developing environment which known for interactive coding and data analysis. The primary programming language used for the research is Python 3.11.5 which is suitable for build machine learning models.

2.3 Packages used

Figure 1. shows the list of packages used in the research for model implementation.

Package	Description
Pandas	Used data manipulation and preprocessing
Numpy	Used for numerical operations and array handling
scikit-learn	Used for preprocessing and evaluation metrics
Imblearn	Uses SMOTE for handling data imbalance
Matplotlib & Seaborn	Used for building, training and evaluate the deep learning models.
Keras tuner	Used for hyperparameter tuning

Figure 1: Packages Used

3 Dataset

In this research the author has used two open-source datasets to analyze the sleep and apps usage pattern to predict PSU.

3.1 Sleep Dataset

The dataset used to analyze sleep pattern was downloaded from Kaggle, the dataset name is 'SleepQual and B.Health' (PSU_Dataset.csv) *Arora et al. (2022)* has collected the dataset using the wearable devices such as smartwatches. The dataset contains the information on physical activity, sleep duration pattern and smartphone usage metrics from 24 undergraduate students.

3.2 Apps Usage Dataset

The dataset used to analyze the apps usage pattern was downloaded from Denmark Technical University website (PSU_Apps.csv), Sapienza et al. (2023) collected the data from global smartphone and electronics company, it has 464,455 individual's smartphone apps usage details collected for 20 days.

4 Project Development for Sleep Data Analysis

The research has been developed in 2 parts due to dataset constraints, one set of the deep learning models has been developed to analyze the sleep data and other set of models has been developed to analyze the apps usage data.

4.1 Importing Libraries

Figure 2. shows the list of libraries used for building the deep learning models to analyze the sleep data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, LearningRateScheduler, ModelCheckpoint
import keras_tuner as kt
import tensorflow as tf
import random
import os
```

Figure 2: Libraries used - Sleep data analysis

4.2 Preprocessing

In this section the preprocessing steps shown in figure 3. such as selecting the features, employing feature engineering on night phone usage and phone unlock count. The code shows the list of selected variables, and the target variable is label, then the data is split into train and

test with test size 0.2 and random state is set for reproducibility. To balance the data, SMOTE has been applied and normalization to make the scales even for the all the variables used in the research.

```
# Feature Analysis
print("Features in the dataset:")
print(data.columns.tolist())

# Define the feature set and target variable
selected_features = ['awake_percentage', 'total_phone_usage_perday_minutes', 'sleep_onset_latency_minutes',
                     'in_bed_awake_duration_minutes',
                     'efficiency_%', 'light_sleep_duration_minutes', 'REM_duration_minutes',
                     'actual_sleep_duration_minutes','deep_sleep_duration_minutes',
                     'phone_usage_metric']  # Added this for advanced feature engineering

# Feature Engineering
data['phone_usage_metric'] = data['night_time_phone_usage_perday_minutes'] + data['phone_unlock_count_perday']

# Check for missing values
print("\nMissing values in each feature:")
print(data[selected_features].isnull().sum())

# Define features and target variable
X = data[selected_features]
y = data['label']

# Setting random seed
seed = 3561
np.random.seed(seed)
tf.random.set_seed(seed)
random.seed(seed)

# 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=seed, stratify=y)

# Apply SMOTE to balance the training dataset
smote = SMOTE(random_state=seed)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Standardize the features
scaler = StandardScaler()
X_train_resampled = scaler.fit_transform(X_train_resampled)
X_test = scaler.transform(X_test)
```

Figure 3: Preprocessing Sleep Data

4.3 Modelling

In the first part of the deep learning models shown below are developed to analyze the sleep data to find the hidden patterns based on smartphone usage and predict PSU.

4.3.1 Feedforward Neural Networks (FNN)

The figure 4. shows the implementation of FNN model where the balanced data is transformed to a format understandable by the neural network is sent to the model for training. During the training 5-fold cross validation with callbacks has been employed to get the highest accuracy.

The FNN model has been rebuilt with hyperparameter tuning as shown in figure 5. keras tuner is used to build the model and number of layers are set dynamically during the model building. Hyperband tuner will run for the various input and model is trained until it reaches highest accuracy, which is consider as the best model.

The figure 6. shows the code for evaluating the model's performance based on the accuracy, precision, recall and F1-score.

```python
# Model
def build_model(units1=256, units2=128, units3=64, dropout1=0.5, dropout2=0.5, dropout3=0.5, optimizer='adam'):
    model = Sequential([
        Dense(units1, activation='relu', input_shape=(X_train_resampled.shape[1],)),
        BatchNormalization(),
        Dropout(dropout1),
        Dense(units2, activation='relu'),
        BatchNormalization(),
        Dropout(dropout2),
        Dense(units3, activation='relu'),
        BatchNormalization(),
        Dropout(dropout3),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=7, min_lr=0.00001)
model_checkpoint = ModelCheckpoint('FNN_model.h5', save_best_only=True, monitor='val_loss', mode='min')

# cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
fold_accuracies = []
for train_index, val_index in skf.split(X_train_resampled, y_train_resampled):
    X_train_fold, X_val_fold = X_train_resampled[train_index], X_train_resampled[val_index]
    y_train_fold, y_val_fold = y_train_resampled[train_index], y_train_resampled[val_index]
    model = build_model()  # Rebuild the model for each fold
    history = model.fit(
        X_train_fold, y_train_fold, epochs=100, validation_data=(X_val_fold, y_val_fold),
        callbacks=[early_stopping, reduce_lr, model_checkpoint],
        verbose=1
    )
    val_loss, val_acc = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    fold_accuracies.append(val_acc)

print(f"\nFold accuracies: {fold_accuracies}")
print(f"Mean fold accuracy: {np.mean(fold_accuracies):.2f}")
print(f"Standard deviation of fold accuracies: {np.std(fold_accuracies):.2f}")
```

Figure 4: FNN Model for Sleep Data

```python
# Adjusted Hyperparameter Tuning
def model_builder(hp):
    model = Sequential([
        Dense(units=hp.Int('units1', min_value=128, max_value=512, step=64), activation='relu',
              input_shape=(X_train_resampled.shape[1],)),
        BatchNormalization(),
        Dropout(hp.Float('dropout1', 0.3, 0.6, step=0.1)),
        Dense(units=hp.Int('units2', min_value=64, max_value=256, step=32), activation='relu'),
        BatchNormalization(),
        Dropout(hp.Float('dropout2', 0.3, 0.6, step=0.1)),
        Dense(units=hp.Int('units3', min_value=32, max_value=128, step=16), activation='relu'),
        BatchNormalization(),
        Dropout(hp.Float('dropout3', 0.3, 0.6, step=0.1)),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer=hp.Choice('optimizer', values=['adam', 'sgd']),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = kt.Hyperband(
    model_builder,
    objective='val_accuracy',
    max_epochs=150,
    hyperband_iterations=3,
    directory='my_dir',
    project_name='FNN_Hyperparameter',
    seed=seed
)

tuner.search(X_train_fold, y_train_fold, epochs=150, validation_data=(X_val_fold, y_val_fold))
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"\nBest hyperparameters for the fold: {best_hps.values}")

# Build and train the model with the best hyperparameters
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(X_train_fold, y_train_fold, epochs=150, validation_data=(X_val_fold, y_val_fold), verbose=1,
                         callbacks=[early_stopping, reduce_lr, model_checkpoint])
```

Figure 5: FNN model for Sleep data – Hyperparameter Tuning

```
# Evaluate the model
model.load_weights('FNN_model.h5')
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest loss with selected features: {test_loss:.4f}")
print(f"Test accuracy with selected features: {test_acc:.2f}")

# Make predictions with the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = y_test.values

# Classification Report
report = classification_report(y_true_classes, y_pred_classes, target_names=['Good', 'Moderate', 'Poor'])
print("\nClassification Report with Selected Features:")
print(report)

# Confusion Matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Good', 'Moderate', 'Poor'],
            yticklabels=['Good', 'Moderate', 'Poor'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix with Selected Features')
plt.show()
```

Figure 6: Evaluation Metrics - Sleep Data

4.3.2 Convolutional Neural Networks (CNN)

```
# Reshape data for CNN
X_train_resampled = np.expand_dims(X_train_resampled, axis=2)
X_test = np.expand_dims(X_test, axis=2)

# Define and build the CNN model
def build_cnn_model():
    model = Sequential()
    model.add(Conv1D(filters=256, kernel_size=3, activation='relu', input_shape=(X_train_resampled.shape[1],
                                                                                 X_train_resampled.shape[2])))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Initialize the model
model = build_cnn_model()

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=7, min_lr=0.0001)
model_checkpoint = ModelCheckpoint('best_cnn_model.h5', save_best_only=True, monitor='val_loss', mode='min')

# cross-validation
skf = StratifiedKFold(n_splits=5)
histories = []  # To store histories of each fold
for train_index, val_index in skf.split(X_train_resampled, y_train_resampled):
    X_train_fold, X_val_fold = X_train_resampled[train_index], X_train_resampled[val_index]
    y_train_fold, y_val_fold = y_train_resampled[train_index], y_train_resampled[val_index]
    history = model.fit(
        X_train_fold, y_train_fold, epochs=100, validation_data=(X_val_fold, y_val_fold),
        callbacks=[early_stopping, reduce_lr, model_checkpoint],
        verbose=1
    )
    histories.append(history)
```

Figure 7: CNN model for Sleep data

The figure 7. shows the code used for building CNN model for sleep data it has different layers and functions used in extracting the sleep pattern. The output from these layers is flatted and sent to the dense layer to get the hidden patterns present in the data. In this model various

callback function and 5-fold cross validation has been added to train the model to get more accuracy, and the test data will be evaluated on the best trained model.

```
# Hyperparameter Tuning
def cnn_model_builder(hp):
    model = Sequential()
    model.add(Conv1D(filters=hp.Int('filters1', min_value=32, max_value=64, step=32),
                     kernel_size=hp.Choice('kernel_size1', values=[3]), activation='relu',
                     input_shape=(X_train_resampled.shape[1], X_train_resampled.shape[2])))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(hp.Float('dropout1', 0.2, 0.3, step=0.1)))
    model.add(Conv1D(filters=hp.Int('filters2', min_value=64, max_value=128, step=64),
                     kernel_size=hp.Choice('kernel_size2', values=[3]), activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(hp.Float('dropout2', 0.2, 0.3, step=0.1)))
    model.add(Flatten())
    model.add(Dense(units=hp.Int('units1', min_value=32, max_value=64, step=32), activation='relu'))
    model.add(Dropout(hp.Float('dropout3', 0.2, 0.3, step=0.1)))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer=hp.Choice('optimizer', values=['adam']),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = kt.Hyperband(cnn_model_builder,
                     objective='val_accuracy',
                     max_epochs=150,
                     hyperband_iterations=3,
                     directory='my_dir',
                     project_name='CNN_Hyperparameter',
                     seed=seed)

tuner.search(X_train_resampled, y_train_resampled, epochs=150, validation_split=0.2)
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"\nBest hyperparameters: {best_hps.values}")

# Build the model with the best hyperparameters
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(X_train_resampled, y_train_resampled, epochs=100, validation_split=0.2, verbose=1)
```

Figure 8: CNN model for Sleep data – Hyperparameter Tuning

The figure 8. Shows the code used for building the CNN model with hyperparameter tuning. This process helps in the finding the best hyperparameter to build the model with higher accuracy.

4.3.3 Recurrent Neural Networks Long-Short Term Memory (RNN-LSTM)

The figure 9. shows the code used for building the RNN-LSTM model, it is the advanced version of RNN. This model is used for analysing the long-term dependencies in the sequential data. In the input layer the data is transformed to a format which is understandable by the RNN-LSTM model. Then the data is passed through 3 LSTM layers where the return_sequences is set true to in the first 2 layers to make sequential data to be passed for the consecutive layers, each LSTM layer is followed by batch normalization and dropout layers. Then this data will be sent to dense layer to convert the data into classification output. Various callback and 5-fold cross validation techniques are employed to build a model with higher accuracy.

The figure 10. shows the code used for building the RNN-LSTM model with hyperparameter tuning, in this model the parameters are set to pass dynamically during the model training by the tuner search and the best model with higher accuracy will be saved. Finally, the model is built using the best hyperparameter. This process helps in finding the optimal hyperparameter which helps to increase the model's accuracy.

```python
# Reshape the data for LSTM input
X_train_resampled = X_train_resampled.reshape(X_train_resampled.shape[0], 1, X_train_resampled.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Define the model
def build_model(units1=256, units2=128, units3=64, dropout1=0.5, dropout2=0.5, dropout3=0.5, optimizer='adam'):
    model = Sequential([
        LSTM(units1, activation='relu', input_shape=(X_train_resampled.shape[1], X_train_resampled.shape[2]),
             return_sequences=True),
        BatchNormalization(),
        Dropout(dropout1),
        LSTM(units2, activation='relu', return_sequences=True),
        BatchNormalization(),
        Dropout(dropout2),
        LSTM(units3, activation='relu'),
        BatchNormalization(),
        Dropout(dropout3),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001)
model_checkpoint = ModelCheckpoint('LSTM_model.h5', save_best_only=True, monitor='val_loss', mode='min')

# Train the model with cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
fold_accuracies = []
for train_index, val_index in skf.split(X_train_resampled, y_train_resampled):
    X_train_fold, X_val_fold = X_train_resampled[train_index], X_train_resampled[val_index]
    y_train_fold, y_val_fold = y_train_resampled[train_index], y_train_resampled[val_index]
    model = build_model()  # Rebuild the model for each fold
    history = model.fit(
        X_train_fold, y_train_fold, epochs=100, validation_data=(X_val_fold, y_val_fold),
        callbacks=[early_stopping, reduce_lr, model_checkpoint],verbose=1)
    val_loss, val_acc = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    fold_accuracies.append(val_acc)

print(f"\nFold accuracies: {fold_accuracies}")
print(f"Mean fold accuracy: {np.mean(fold_accuracies):.2f}")
print(f"Standard deviation of fold accuracies: {np.std(fold_accuracies):.2f}")
```

Figure 9: RNN-LSTM model for Sleep data

```python
# Adjusted Hyperparameter Tuning with K-Fold Cross-Validation
def model_builder(hp):
    model = Sequential([
        LSTM(units=hp.Int('units1', min_value=128, max_value=512, step=64), activation='relu',
             input_shape=(X_train_resampled.shape[1], X_train_resampled.shape[2]), return_sequences=True),
        BatchNormalization(),
        Dropout(hp.Float('dropout1', 0.3, 0.6, step=0.1)),
        LSTM(units=hp.Int('units2', min_value=64, max_value=256, step=32), activation='relu', return_sequences=True),
        BatchNormalization(),
        Dropout(hp.Float('dropout2', 0.3, 0.6, step=0.1)),
        LSTM(units=hp.Int('units3', min_value=32, max_value=128, step=16), activation='relu'),
        BatchNormalization(),
        Dropout(hp.Float('dropout3', 0.3, 0.6, step=0.1)),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer=hp.Choice('optimizer', values=['adam', 'sgd']),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = kt.Hyperband(
    model_builder,
    objective='val_accuracy',
    max_epochs=150,
    hyperband_iterations=3,
    directory='my_dir',
    project_name='RNN_Hyperparameter',
    seed=seed)

tuner.search(X_train_fold, y_train_fold, epochs=150, validation_data=(X_val_fold, y_val_fold))

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"\nBest hyperparameters for the fold: {best_hps.values}")

# Build and train the model with the best hyperparameters
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(X_train_fold, y_train_fold, epochs=150, validation_data=(X_val_fold, y_val_fold),
                         verbose=1, callbacks=[early_stopping, reduce_lr, model_checkpoint])
```

Figure 10: RNN-LSTM model for Sleep data – Hyperparameter Tuning

4.3.4 Prediction for Sleep Data

To check the model's ability to classify the sleep quality a function has been written as shown in figure 11. It accepts the input as subject id and provides the output in the form of 'Good', 'Moderate' and 'Poor' as the sleep quality based on the smartphone usage.

```
##### Analyze a specific subject's sleep quality
def analyze_subject(subject):
    subject_data = data[data['subject'] == subject]
    if subject_data.empty:
        print(f"Subject {subject} not found in the dataset.")
        return
    print(f"\nAnalyzing Sleep Quality for Subject {subject}:")

    # Extract the subject's data
    subject_row = subject_data[selected_features]
    subject_data_scaled = scaler.transform(subject_row)
    prediction = model.predict(subject_data_scaled)
    predicted_class = np.argmax(prediction, axis=1)[0]
    predicted_label = ['Good', 'Moderate', 'Poor'][predicted_class]
    print(f"\nPredicted Sleep Quality for Subject {subject}: {predicted_label}")

# Example usage of the analyze_subject function
analyze_subject(24)  # Change the subject ID to analyze different subjects
```

Figure 11: Prediction function - Sleep data

5 Project Development for Apps Usage analysis

This is the second set of deep learning models built to analyze the apps usage pattern to predict PSU. In this section, the author has explained the libraries used, preprocessing, modelling, evaluation and predictions techniques used in this research.

5.1 Importing Libraries

The figure 12. shows the list of libraries used in this research to analyze the apps usage pattern using the deep learning techniques like FNN, CNN and RNN-LSTM.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.utils.class_weight import compute_class_weight
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from collections import Counter
```

Figure 12: Libraries used – Apps Usage analysis

5.2 Preprocessing

In this section the preprocessing steps carried out shown in figure 13. which includes heuristic feature selection to find a user is affected by PSU or not, here the author created a list which contains the apps categories which are the major contributor of PSU and set the sum of app usage value goes beyond 30% the user is affected by PSU. The target variable has used binary conversion to give output in the form of 0 (Not affected by PSU) or 1 (affected by PSU) using one hot encoding. The data has been split into train and test for further processing and SMOTE technique has been used to overcome the imbalance.

```python
# Define a heuristic to determine if a user is affected by PSU
major_contributors = ['Browsing', 'Communication', 'Entertainment', 'Game', 'Movie/TV', 'Music', 'Shopping', 'Social']
data['PSU_Affected'] = data[major_contributors].sum(axis=1) > 0.30  # Change 0.30 based on your specific heuristic

# Convert PSU_Affected to binary values
data['PSU_Affected'] = data['PSU_Affected'].astype(int)

# Define the features and target variable
feature_columns = [
    'median_screen_time','median_n_apps',
    'Books', 'Browsing', 'Business', 'Camera/Album', 'Communication',
    'Game', 'Health_and_Fitness', 'Maps_and_Navigation',
    'Movie/TV', 'Music', 'News', 'Other', 'Productivity', 'Shopping',
    'Social', 'Tools', 'Travel_and_Local', 'Weather'
]

X = data[feature_columns]
y = data['PSU_Affected']  # Target variable

# Encode categorical variables
preprocessor = ColumnTransformer(
    transformers=[
        ('scaler', StandardScaler(), [
            'median_screen_time','median_n_apps',
            'Books', 'Browsing', 'Business', 'Camera/Album', 'Communication',
            'Game', 'Health_and_Fitness', 'Maps_and_Navigation',
            'Movie/TV', 'Music', 'News', 'Other', 'Productivity', 'Shopping',
            'Social', 'Tools', 'Travel_and_Local', 'Weather'
        ])
    ],
    remainder='passthrough'
)

# Preprocess the data
X_processed = preprocessor.fit_transform(X)

# Convert the target variable to categorical (one-hot encoding)
y_categorical = to_categorical(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y_categorical, test_size=0.2, random_state=3561)
```

```python
# Check the distribution of classes in the training set
print("Class distribution in y_train:", Counter(np.argmax(y_train, axis=1)))

# Apply SMOTE to the training data if there are more than 1 class
smote = SMOTE(sampling_strategy='auto', random_state=3561, k_neighbors=5)
X_resampled, y_resampled = smote.fit_resample(X_train, np.argmax(y_train, axis=1)) if len(np.unique(np.argmax(y_train, axis=1)))
y_resampled = to_categorical(y_resampled)  # Convert back to one-hot encoding

# Check the distribution of classes after resampling
print("Class distribution after resampling:", Counter(np.argmax(y_resampled, axis=1)))

# Calculate class weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(np.argmax(y_resampled, axis=1)),
                                     y=np.argmax(y_resampled, axis=1))
class_weights_dict = dict(enumerate(class_weights))
print("Class weights:", class_weights_dict)
```

Figure 13: Preprocessing Apps usage data

5.3 Modelling

In this phase deep learning models has been built for analysing the apps usage data to find the hidden patterns present in the data and predicts the PSU.

5.3.1 Feedforward Neural Networks (FNN)

```python
# Build the FNN model
model = Sequential()
model.add(Dense(128, input_shape=(X_resampled.shape[1],), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the FNN model
history = model.fit(X_resampled, y_resampled, epochs=25, batch_size=64, validation_split=0.2, verbose=1,
                    class_weight=class_weights_dict, callbacks=[early_stopping])
```

Figure 14: FNN Model for Apps usage data

For analysing the apps usage data same model building techniques has been used same as sleep data. figure 14. shows the code used for building the FNN model.

5.3.2 Convolutional Neural Networks (CNN)

Figure 15. shows the code used for building the CNN model for analysing the apps usage data, here also the same parameters used for sleep data analysis is used.

```
# Reshape Data for CNN
X_train_cnn = X_resampled.reshape(X_resampled.shape[0], X_resampled.shape[1], 1)
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Build the CNN model
model = Sequential()
model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv1D(128, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the CNN model
history = model.fit(X_train_cnn, y_resampled, epochs=25, batch_size=64, validation_split=0.2, verbose=1,
                    class_weight=class_weights_dict, callbacks=[early_stopping])
```

Figure 15: CNN Model for Apps usage data

5.3.3 Recurrent Neural Networks Long-Short Term Memory (RNN-LSTM)

Figure 16. shows the code used for building the RNN-LSTM model for analysing the apps usage data, here also the same parameters used for sleep data analysis is used.

```
# Reshape Data for LSTM input
X_train_rnn = X_resampled.reshape(X_resampled.shape[0], 1, X_resampled.shape[1])
X_test_rnn = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2]), return_sequences=True))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(LSTM(64))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the LSTM model
history = model.fit(X_train_rnn, y_resampled, epochs=25, batch_size=64, validation_split=0.2,
                    verbose=1, class_weight=class_weights_dict, callbacks=[early_stopping])
```

Figure 16: RNN-LSTM Model for Apps usage data

Figure 17. shows the code used for evaluating the performance metrics of the models built for analysing the apps usage data.

```
# Evaluate the LSTM model
loss, accuracy = model.evaluate(X_test_rnn, y_test, verbose=1)
print(f'Test loss: {loss:.4f}')
print(f'Test accuracy: {accuracy:.4f}')

# Make predictions on the test set
y_pred_probs = model.predict(X_test_rnn)
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_true_classes, y_pred_classes, target_names=['Not Affected by PSU', 'Affected by PSU']))

print("Confusion Matrix:")
cm = confusion_matrix(y_true_classes, y_pred_classes)
print(cm)

# Plot the confusion matrix
def plot_confusion_matrix(y_true, y_pred, classes):
    disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true, y_pred), display_labels=classes)
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()
plot_confusion_matrix(y_true_classes, y_pred_classes, classes=['Not Affected by PSU', 'Affected by PSU'])
```

Figure 17: Evaluation Metrics - Apps usage data

5.3.4 Predictions for Apps Usage Data

The below function has been written to predict model's ability to classify the data as shown in figure 18. Here some random samples from the dataset are passed to the function to evaluate the model and provides the output in the form of 'Affected by PSU' or 'Not affected by PSU' based on the smartphone usage.

```
# Define the prediction function
def make_prediction_from_csv(csv_file_path, model, preprocessor):
    new_data = pd.read_csv(csv_file_path)
    required_columns = feature_columns
    missing_columns = [col for col in required_columns if col not in new_data.columns]
    if missing_columns:
        raise ValueError(f"The CSV file is missing the following required columns: {', '.join(missing_columns)}")

    X_new = new_data[feature_columns].copy()

    # Ensure the major_contributors columns are in the DataFrame
    for col in major_contributors:
        if col not in X_new.columns:
            X_new[col] = 0
    X_new_processed = preprocessor.transform(X_new)
    X_new_lstm = X_new_processed.reshape(X_new_processed.shape[0], 1, X_new_processed.shape[1])
    y_pred_probs = model.predict(X_new_lstm)
    y_pred_classes = np.argmax(y_pred_probs, axis=1)

    # Convert numeric class labels to human-readable labels
    predictions = ['Affected by PSU' if pred == 1 else 'Not Affected by PSU' for pred in y_pred_classes]

    # Find the most used app category for each user among the major PSU contributors
    X_new_psu = X_new[major_contributors]
    most_used_apps = X_new_psu.idxmax(axis=1).map({
        'Browsing': 'Browsing',
        'Communication': 'Communication',
        'Entertainment': 'Entertainment',
        'Game': 'Game',
        'Movie/TV': 'Movie/TV',
        'Music': 'Music',
        'Shopping': 'Shopping',
        'Social': 'Social'
    }).tolist()

    # Extract median screen times
    median_screen_times = X_new['median_screen_time'].tolist()

    # Print the results
    for i in range(len(predictions)):
        print(f"User {i+1}: {predictions[i]}, Most used app: {most_used_apps[i]}, Median Screen Time: {median_screen_times[i]}")
    return predictions, most_used_apps, median_screen_times

# Example usage of the prediction function
csv_file_path = 'PSU_apps_predict.csv'  # Update this path to the CSV file you want to predict on
predictions, most_used_apps, median_screen_times = make_prediction_from_csv(csv_file_path, model, preprocessor)
```

Figure 18: Prediction function – Apps Usage data

References

Arora, A., Chakraborty, P. and Bhatia, M.P.S., 2022, January. SleepQual and B. health: Smartwatch and Smartphone based behavioral Datasets of youth. In 2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 340-344). IEEE.

Sapienza, A., Lítlá, M., Lehmann, S. and Alessandretti, L., 2023. Exposure to urban and rural contexts shapes smartphone usage behavior. PNAS nexus, 2(11), p.pgad357.

Sleep Quality dataset has been downloaded from website https://www.kaggle.com/datasets/anshika1011/sleepqual-and-bhealth-dataset(Last accessed on 11-08-2024)

Apps usage dataset has been downloaded from website https://data.dtu.dk/articles/dataset/Data_for_Exposure_to_urban_and_rural_contexts_shapes_smartphone_usage_behavior_/24316516/1?file=42886558 (Last accessed on 11-08-2024)