This documentation provides a detailed explanation of the two provided scripts: one for preprocessing data and training a Random Forest classifier, and another that integrates deep learning for feature extraction and classification using a Gradient Boosting Classifier.

**Script 1: Random Forest Classifier**

**Imports and Libraries**

* pandas and numpy: For data manipulation and numerical operations.
* sklearn modules:
  + SimpleImputer: To handle missing values.
  + LabelEncoder: For encoding categorical variables.
  + train\_test\_split: To split the dataset into training and testing subsets.
  + RandomForestClassifier: A machine learning model for classification tasks.
  + accuracy\_score, classification\_report: To evaluate model performance.

**Functions**

**1. load\_and\_preprocess\_data(file\_paths, chunk\_size=100000)**

Processes multiple CSV files in chunks to handle large datasets efficiently.

* **Input**:
  + file\_paths: List of paths to CSV files.
  + chunk\_size: Number of rows to process at a time.
* **Steps**:
  + Reads files in chunks to avoid memory issues.
  + Handles missing values using mean imputation for numeric columns.
  + Encodes categorical columns using LabelEncoder.
  + Concatenates all processed chunks.
  + Splits features (X) and labels (y), downsampling if necessary.
  + Splits data into training and testing subsets.
* **Output**:
  + X\_train, X\_test, y\_train, y\_test: Processed datasets for training and testing.

**2. train\_and\_evaluate\_models(X\_train, X\_test, y\_train, y\_test)**

Trains a Random Forest model and evaluates its performance.

* **Steps**:
  + Initializes a RandomForestClassifier with 100 trees.
  + Fits the model on training data.
  + Predicts labels for the test set.
  + Prints accuracy and a classification report.

**Code is :**

**import pandas as pd**

**import numpy as np**

**from sklearn.impute import SimpleImputer**

**from sklearn.preprocessing import LabelEncoder**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score, classification\_report**

**def load\_and\_preprocess\_data(file\_paths, chunk\_size=100000):**

**dataframes = []**

**for file in file\_paths:**

**# Read CSV file in chunks**

**chunk\_iter = pd.read\_csv(file, chunksize=chunk\_size)**

**for chunk in chunk\_iter:**

**# Handle missing values for numeric columns**

**numeric\_cols = chunk.select\_dtypes(include=[np.number]).columns**

**imputer = SimpleImputer(strategy='mean')**

**chunk[numeric\_cols] = imputer.fit\_transform(chunk[numeric\_cols])**

**# Handle categorical columns**

**non\_numeric\_cols = chunk.select\_dtypes(exclude=[np.number]).columns**

**for col in non\_numeric\_cols:**

**encoder = LabelEncoder()**

**chunk[col] = encoder.fit\_transform(chunk[col].astype(str))**

**dataframes.append(chunk)**

**data = pd.concat(dataframes, ignore\_index=True)**

**target\_column = 'Label' if 'Label' in data.columns else 'Class'**

**X = data.drop(target\_column, axis=1)**

**y = data[target\_column]**

**# Downsample the data if it's too large**

**max\_samples = 100000**

**if len(X) > max\_samples:**

**sampled\_indices = np.random.choice(len(X), max\_samples, replace=False)**

**X = X.iloc[sampled\_indices]**

**y = y.iloc[sampled\_indices]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**return X\_train, X\_test, y\_train, y\_test**

**def train\_and\_evaluate\_models(X\_train, X\_test, y\_train, y\_test):**

**rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)**

**rf\_model.fit(X\_train, y\_train)**

**y\_pred = rf\_model.predict(X\_test)**

**print("Accuracy:", accuracy\_score(y\_test, y\_pred))**

**print("Classification Report:\n", classification\_report(y\_test, y\_pred))**

**file\_paths = [**

**'C:\\Users\\ascom\\CTU-13\\1\\capture20110810.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\2\\capture20110811.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\3\\capture20110812.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\4\\capture20110815.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\5\\capture20110815-2.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\6\\capture20110816.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\7\\capture20110816-2.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\8\\capture20110816-3.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\9\\capture20110817.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\10\\capture20110818.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\11\\capture20110818-2.binetflow',**

**'C:\\Users\\ascom\\CTU-13\\12\\capture20110819.binetflow',**

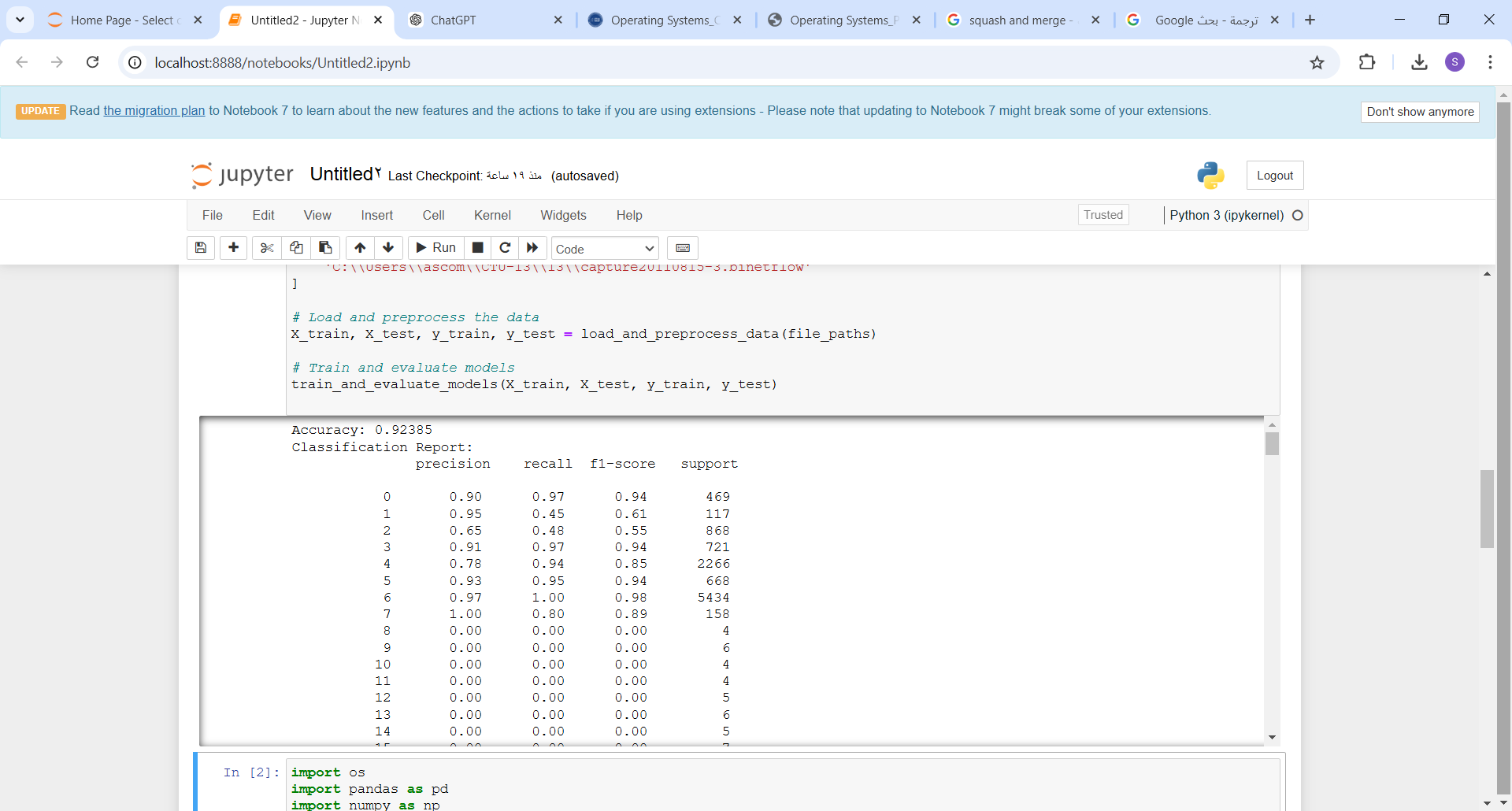
**'C:\\Users\\ascom\\CTU-13\\13\\capture20110815-3.binetflow'**

**]**

**X\_train, X\_test, y\_train, y\_test = load\_and\_preprocess\_data(file\_paths)**

**train\_and\_evaluate\_models(X\_train, X\_test, y\_train, y\_test)**

**Output is:**

****

**Script 2: Deep Learning and Gradient Boosting Classifier**

**Imports and Libraries**

* os, pandas, numpy: For file handling, data manipulation, and numerical operations.
* sklearn modules:
  + train\_test\_split: To split the dataset.
  + GradientBoostingClassifier: A machine learning model for classification tasks.
  + classification\_report: To evaluate model performance.
* keras modules:
  + Sequential: For building deep learning models.
  + Dense: For defining fully connected layers.

**Functions**

**1. load\_and\_preprocess\_data(file\_paths)**

Loads and preprocesses data for deep learning.

* **Input**:
  + file\_paths: List of file paths.
* **Steps**:
  + Reads files in chunks, keeping only numerical features and labels.
  + Drops rows with missing values.
  + Converts labels to binary values (e.g., Botnet = 1, otherwise = 0).
  + Normalizes features to a 0-1 range.
  + Splits data into training and testing subsets.
* **Output**:
  + X\_train, X\_test, y\_train, y\_test: Preprocessed datasets.

**2. build\_autoencoder(input\_dim)**

Defines an autoencoder model for dimensionality reduction.

* **Input**:
  + input\_dim: Number of input features.
* **Structure**:
  + Encoder: Reduces the input dimension to 16 through multiple layers.
  + Decoder: Reconstructs input from encoded features.
* **Output**:
  + autoencoder: A Keras Sequential model combining encoder and decoder.

**3. extract\_features\_with\_dnn(X\_train, X\_test)**

Uses the autoencoder to extract features.

* **Steps**:
  + Builds and trains the autoencoder on training data.
  + Uses the encoder to transform training and testing data.
* **Output**:
  + X\_train\_features, X\_test\_features: Feature representations from the encoder.

**4. train\_and\_evaluate\_model(X\_train, X\_test, y\_train, y\_test)**

Trains a Gradient Boosting Classifier and evaluates its performance.

* **Steps**:
  + Fits a GradientBoostingClassifier on the training data.
  + Predicts labels for the test set.
  + Prints a classification report.

**Dataset**

* **CTU-13 Dataset**: A public dataset for botnet detection.
* **Features Used**:
  + Dur, TotPkts, TotBytes, SrcBytes, and Label.
* **Labels**:
  + Binary classification: Botnet (1) vsNon-Botnet (0).

**Code is :**

**import os**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import GradientBoostingClassifier**

**from sklearn.metrics import classification\_report**

**from keras.models import Sequential**

**from keras.layers import Dense**

**def load\_and\_preprocess\_data(file\_paths):**

**dataframes = []**

**for file\_path in file\_paths:**

**for chunk in pd.read\_csv(file\_path, chunksize=10000):**

**columns\_to\_keep = ['Dur', 'TotPkts', 'TotBytes', 'SrcBytes', 'Label']**

**chunk = chunk[columns\_to\_keep]**

**chunk.dropna(inplace=True)**

**# Convert categorical labels to binary**

**chunk['Label'] = chunk['Label'].apply(lambda x: 1 if 'Botnet' in x else 0)**

**dataframes.append(chunk)**

**data = pd.concat(dataframes, ignore\_index=True)**

**X = data.drop('Label', axis=1)**

**y = data['Label']**

**# Normalize features**

**X = (X - X.min()) / (X.max() - X.min())**

**return train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**def build\_autoencoder(input\_dim):**

**encoder = Sequential([**

**Dense(64, activation='relu', input\_dim=input\_dim),**

**Dense(32, activation='relu'),**

**Dense(16, activation='relu')**

**])**

**decoder = Sequential([**

**Dense(32, activation='relu', input\_dim=16),**

**Dense(64, activation='relu'),**

**Dense(input\_dim, activation='sigmoid')**

**])**

**autoencoder = Sequential([encoder, decoder])**

**return autoencoder**

**def extract\_features\_with\_dnn(X\_train, X\_test):**

**input\_dim = X\_train.shape[1]**

**model = build\_autoencoder(input\_dim)**

**model.compile(optimizer='adam', loss='mean\_squared\_error')**

**model.fit(X\_train, X\_train, epochs=10, batch\_size=32, verbose=0)**

**encoder = model.layers[0]**

**return encoder.predict(X\_train), encoder.predict(X\_test)**

**def train\_and\_evaluate\_model(X\_train, X\_test, y\_train, y\_test):**

**gbc = GradientBoostingClassifier()**

**gbc.fit(X\_train, y\_train)**

**y\_pred = gbc.predict(X\_test)**

**print(classification\_report(y\_test, y\_pred))**

**if \_\_name\_\_ == "\_\_main\_\_":**

**dataset\_dir = r"C:\Users\ascom\CTU-13"**

**file\_paths = [**

**os.path.join(dataset\_dir, subdir, file)**

**for subdir in os.listdir(dataset\_dir)**

**for file in os.listdir(os.path.join(dataset\_dir, subdir))**

**if file.endswith('.binetflow')**

**]**

**X\_train, X\_test, y\_train, y\_test = load\_and\_preprocess\_data(file\_paths)**

**X\_train\_features, X\_test\_features = extract\_features\_with\_dnn(X\_train, X\_test)**

**train\_and\_evaluate\_model(X\_train\_features, X\_test\_features, y\_train, y\_test)**

**Output is:**

