National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Donovan Michael Jeremiah
Student ID: x18181562

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Donovan Michael Jeremiah
Student ID:	x18181562
Programme:	Data Analytics
Year:	2019-2020
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	371
Page Count:	13

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:	Donovan Michael Jeremiah
Date:	27th September 2020

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

Donovan Michael Jeremiah
x18181562

1 Introduction

The configuration manual is part of the project code for the paper titled 'Detecting Depression from Speech with Residual Learning'. It contains system setup, and the explanation of the preprocessing steps and neural network models that were implemented as part of the research project.

2 System Setup

The preprocessing steps, model execution and evaluation of the 3 models (Base-CNN, AlexNet, and ResNet-18) used in this research was executed on Google Colab as the local system environment (laptop) did not meet the recommended system requirements for the project.

2.1 Hardware

- **Environment:** Google Colaboratory's online cloud-based Jupyter notebook environment.
- **Processor:** NVIDIA Tesla P100 GPU, 16GB
- **RAM:** 25 GB
- **Operating System:** Ubuntu 18.04.3 LTS
- **Disk Storage:** 150 GB
- **Google Drive Storage:** 200 GB

2.2 Software

- Jupyter Notebook (Google Colab)
- Python Version: 3.6.9
- Keras Version: 2.4.3
- Tensorflow: 2.3.0

2.3 Main Packages Used

Preprocessing:

- import numpy as np
- import pandas as pd
- pyAudioAnalysis
- scipy.io.wavfile
- wave

Neural Network Models:

- keras
- tensorflow
- sklearn

3 Data Collection

The data[1] used in this project is the Wizard-of-Oz interviews from the Distress Analysis Interview Corpus (DAIC-WOZ). It was from obtained from the University of Southern California's Institute for Creative Technologies. The official documentation[2] for the DAIC-WOZ database is found at its homepage.

It consists of 189 .wav audio files of interview sessions between Ellie, a virtual interviewer (who is controlled by a human in another room) and the participant.

The data must be downloaded and placed in google drive. Its appropriate path must be placed in the AUDIO_FOLDER_PATH variable of the **config.py** file as seen below.

```
SRC_PATH = "/content/drive/My Drive/Projects/depression-detection"

AUDIO_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/audio"
INTERIM_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/interim"
PROCESSED_513_125_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processed/51
PROCESSED_513_513_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processed/51
PROCESSED_513_125_15P_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processe
PROCESSED_513_513_15P_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processe
PROCESSED_513_125_20P_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processe
PROCESSED_513_513_20P_FOLDER_PATH = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/processe

TRAIN_SPLIT_CSV = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/train_split_Depression_AVE
TEST_SPLIT_CSV = "/content/drive/My Drive/Projects/datasets/DAIC-WOZ/dev_split_Depression_AVEC20

PLOTS_DATA = "/content/drive/My Drive/Projects/depression-detection/plots"
```

Figure 1: config.py

[1] https://dcapswoz.ict.usc.edu/

[2] https://dcapswoz.ict.usc.edu/wwwutil_files/DAICWOZDepression_Documentation.pdf

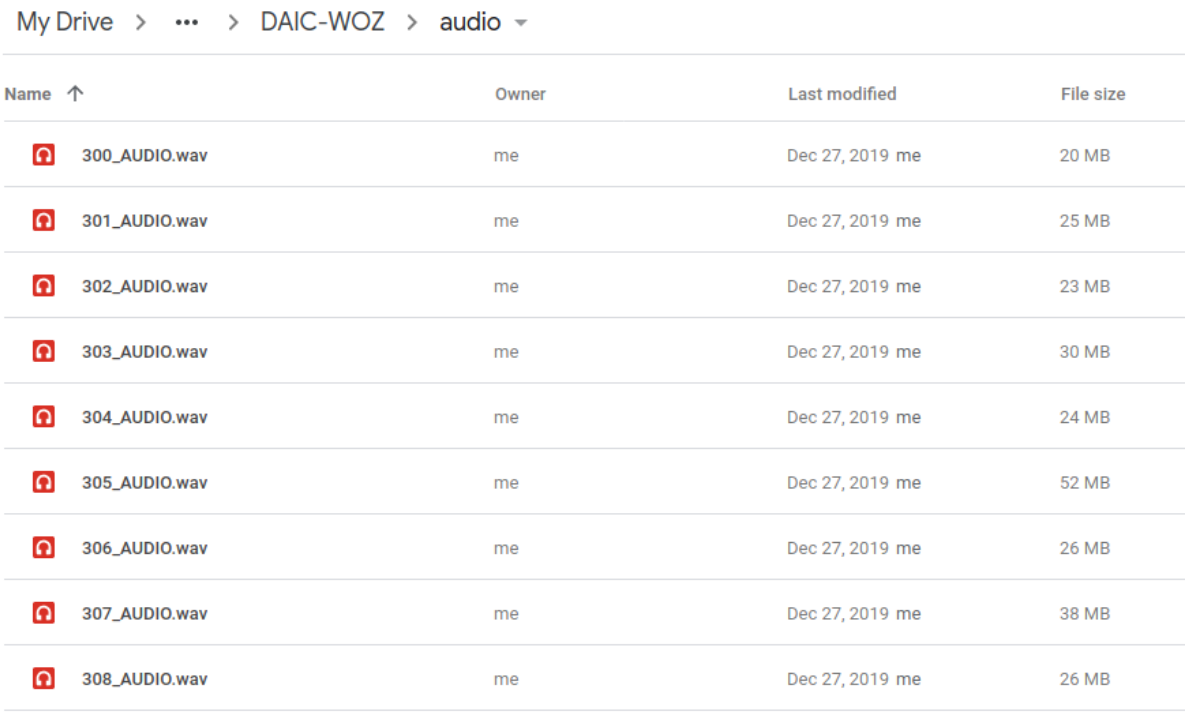

Figure 2: Audio files from DAIC-WOZ.

4 Preprocessing

4.1 Audio Segmentation

```
partic_id = 'P' + filename.split('/')[-1].split('_')[0]  # PXXX
if is_segmentable(partic_id):
    # create participant directory for segmented wav files
    participant_dir = os.path.join(out_dir, partic_id)
    if not os.path.exists(participant_dir):
        os.makedirs(participant_dir)

    os.chdir(participant_dir)

    [Fs, x] = aIO.read_audio_file(filename)
    segments = aS.silence_removal(x, Fs, 0.020, 0.020,
                                  smooth_window=smoothing,
                                  weight=weight,
                                  plot=plot)

    for s in segments:
        seg_name = "{:s}_{:.2f}-{:.2f}.wav".format(partic_id, s[0], s[1])
        wavfile.write(seg_name, Fs, x[int(Fs * s[0]):int(Fs * s[1])])

    # concatenate segmented wave files within participant directory
    concatenate_segments(participant_dir, partic_id)
```

Figure 3: segmentation.py

4.2 Spectrogram Extraction

```
def stft(sig, frameSize, overlapFac=0.5, window=np.hanning):
    """
    Short-time Fourier transform of audio signal.
    """
    #print("framesize: " + str(frameSize)) # DJ
    win = window(frameSize)
    hopSize = int(frameSize - np.floor(overlapFac * frameSize))
    # zeros at beginning (thus center of 1st window should be for sample n
    # Change: DJ - 9/7/2020
    #samples = np.append(np.zeros(np.floor(frameSize/2)), sig)
    samples = np.append(np.zeros(int(np.floor(frameSize/2))), sig)
    # cols for windowing
    # Change: DJ - 9/7/2020
    #cols = np.ceil((len(samples) - frameSize) / float(hopSize)) + 1
    cols = int(np.ceil((len(samples) - frameSize) / float(hopSize))) + 1
    # zeros at end (thus samples can be fully covered by frames)
    samples = np.append(samples, np.zeros(frameSize))

    frames = stride_tricks.as_strided(samples, shape=(cols, frameSize),
                                      strides=(samples.strides[0]*hopSize,
                                      samples.strides[0])).copy()

    # Change: DJ - 9/7/2020 - Added below line
    # Src: https://stackoverflow.com/questions/28272322/typeerror-slice-in
    frames = frames.astype(int)

    # Change: DJ - 26/7/2020
    # Src: https://stackoverflow.com/questions/38673531/numpy-cannot-cast-
    #frames *= win
    frames = frames * win

    return np.fft.rfft(frames)
```

Figure 4: spectrograms.py

```python
def logscale_spec(spec, sr=44100, factor=20.):
    """
    Scale frequency axis logarithmically.
    """
    timebins, freqbins = np.shape(spec)

    scale = np.linspace(0, 1, freqbins) ** factor
    scale *= (freqbins-1)/max(scale)
    scale = np.unique(np.round(scale))

    # create spectrogram with new freq bins
    newspec = np.complex128(np.zeros([timebins, len(scale)]))
    # Change: DJ - 9/7/2020 - Added below line
    # Src: https://stackoverflow.com/questions/28272322/typeerror-slice-i
    scale = scale.astype(int)

    for i in range(0, len(scale)):
        if i == len(scale)-1:
            newspec[:, i] = np.sum(spec[:, scale[i]:], axis=1)
        else:
            newspec[:, i] = np.sum(spec[:, scale[i]:scale[i+1]], axis=1)

    # list center freq of bins
    allfreqs = np.abs(np.fft.fftfreq(freqbins*2, 1./sr)[:freqbins+1])
    freqs = []
    for i in range(0, len(scale)):
        if i == len(scale)-1:
            freqs += [np.mean(allfreqs[scale[i]:])]
        else:
            freqs += [np.mean(allfreqs[scale[i]:scale[i+1]])]

    return newspec, freqs
```

Figure 5: Log scaling of spectrograms (spectrograms.py).

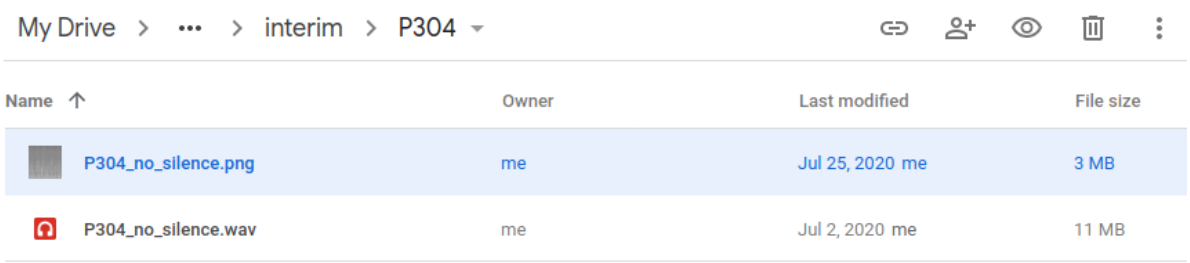

Figure 6: Segmented audio from previous step and spectrograms.

4.3 Exclusion of Shorter Interviews

```
start = time.time()
# uncomment when ready to use...
#depressed_dict, normal_dict = spd.build_class_dictionaries(cfg.INTERIM_FOLDER
end = time.time()
print("*** Time Taken (in mins): " + str(round((end - start)/60,2)))

# Compute Quartiles to check if we can omit some (short) interviews
# in an attempt to increase the equal no. of samples taken from
# each participant.
df_n_len = pd.DataFrame(columns=['id', 'length'])
for x in normal_dict.items():
  row = {'id':x[0], 'length':x[1].shape[1]}
  #print("P: " + str(x[0]) + " | Length: " + str(x[1].shape[1]))
  df_n_len = df_n_len.append(row, ignore_index=True)
df_n_len.length.quantile([0.15,0.25,0.50,0.75,0.90])

# for depressed participants...
df_d_len = pd.DataFrame(columns=['id', 'length'])
for x in depressed_dict.items():
  row = {'id':x[0], 'length':x[1].shape[1]}
  #print("P: " + str(x[0]) + " | Length: " + str(x[1].shape[1]))
  df_d_len = df_d_len.append(row, ignore_index=True)

print("Note: Approx... 31.2 length = 1 second in interview clip.")
print("**** Intervew length quartiles for normal participants: ")
print(df_n_len.length.quantile([0.15,0.20,0.25,0.50,0.75,0.90]))
print("15% of normal participants with the shortest interviews: ")
```

Figure 7: Code to see what interviews to exclude (main.ipynb).

4.4 Random Undersampling

```
def get_random_samples(matrix, n_samples, crop_width):
    """
    Get N random samples with width of crop_width from the numpy matrix
    representing the participant's audio spectrogram.
    """
    # crop full spectrogram into segments of width = crop_width
    clipped_mat = matrix[:, (matrix.shape[1] % crop_width):]
    n_splits = clipped_mat.shape[1] / crop_width
    cropped_sample_ls = np.split(clipped_mat, n_splits, axis=1)

    # get random samples
    samples = random.sample(cropped_sample_ls, n_samples)
    return samples
```

Figure 8: Fixed no. of random samples taken from each participant's spectrogram (random$_s$*ampling.py*).

4.5 Preprocess input for Neural Networks

```
# retrieve spectrogram matrices from .npz files...
#path = cfg.PROCESSED_513_125_20P_FOLDER_PATH
path = cfg.PROCESSED_513_125_FOLDER_PATH # change path accordingly to use diff
# PROCESSED_513_513_15P_FOLDER_PATH
X_train = np.load(path + '/train_samples.npz')
y_train = np.load(path + '/train_labels.npz')
X_test = np.load(path + '/test_samples.npz')
y_test = np.load(path + '/test_labels.npz')
X_train, y_train, X_test, y_test = \
        X_train['arr_0'], y_train['arr_0'], X_test['arr_0'], y_test['arr_0']

nb_classes = 2

# normalize data and prep for Keras
print('Processing images for Keras...')
X_train, X_test, y_train, y_test = cnn.prep_train_test(X_train, y_train,
                                                       X_test, y_test,
                                                       nb_classes=nb_classes)

# 513x125x1 for spectrogram with crop size of 125 pixels
img_rows, img_cols, img_depth = X_train.shape[1], X_train.shape[2], 1

# reshape image input for Keras
X_train, X_test, input_shape = cnn.keras_img_prep(X_train, X_test, img_depth,
                                                  img_rows, img_cols)
```

Figure 9: Preprocess the input for Keras models (Base-CNN, AlexNet, ResNet-18) (main.ipynb).

5 Neural Network Models

5.1 Base-CNN

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding='valid', strides=1,
                 input_shape=input_shape, activation='relu'))

model.add(MaxPooling2D(pool_size=(4, 3), strides=(1, 3)))

model.add(Conv2D(32, (1, 3), padding='valid', strides=1,
          input_shape=input_shape, activation='relu'))

model.add(MaxPooling2D(pool_size=(1, 3), strides=(1, 3)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(nb_classes))
#model.add(Activation('softmax'))
model.add(Activation('sigmoid')) # DJ

model.compile(loss='binary_crossentropy',
              #optimizer='adadelta',
              optimizer=Adam(lr=0.001),
              metrics=['accuracy'])
lrr = ReduceLROnPlateau(monitor='val_accuracy', factor=.01, pati

history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0,
                    validation_data=(X_test, y_test),
                    callbacks=[lrr])
```

Figure 10: Implementation of Base-CNN (cnn.py)

```
for train, val in kfold.split(np.zeros(ZX_train.shape[0]), Zy_label):
    print("********** FOLD NO: " + str(fold_no) + " **********")
    #if fold_no != 4:
        #fold_no = fold_no + 1
        #continue
    fold_no = fold_no + 1
    # create fold to use
    train_X, val_X = ZX_train[train], ZX_train[val]
    train_y, val_y = Zy_train[train], Zy_train[val]
    print(f'Train** N: {str(np.count_nonzero(Zy_label[train] == 0))} D:
    print(f'Valid** N: {str(np.count_nonzero(Zy_label[val] == 0))} D: {s

    input_shape = (train_X.shape[1], train_X.shape[2], 1)

    start = time.time()
    model, history = cnn.cnn(train_X, train_y, val_X, val_y, batch_size,
                      nb_classes, epochs, input_shape)
    end = time.time()
    print("*** Time Taken (in mins): " + str(round((end - start)/60,2)))

    # Evaluate accuracy on test and train sets
    e.accuracy(model, train_X, val_X, train_y, val_y, met)
    y_test_pred_proba, conf_matrix = e.performance(model, train_X, val_X
    e.additional_metrics(conf_matrix, met)
```

Figure 11: Base-CNN with K-fold (main.ipynb)

5.2 AlexNet

```python
def AlexNet(input_shape, nb_classes, padding='same'):
  #Instantiation
  AlexNet = Sequential()

  #1st Convolutional Layer
  AlexNet.add(Conv2D(filters=96, input_shape=input_shape, kernel_size=(11,11), strides=
  AlexNet.add(BatchNormalization())
  AlexNet.add(Activation('relu'))
  AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding=padding))

  #2nd Convolutional Layer
  AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding=padding))
  AlexNet.add(BatchNormalization())
  AlexNet.add(Activation('relu'))
  AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding=padding))

  #3rd Convolutional Layer
  AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding=padding))
  AlexNet.add(BatchNormalization())
  AlexNet.add(Activation('relu'))

  #4th Convolutional Layer
  AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding=padding))
  AlexNet.add(BatchNormalization())
  AlexNet.add(Activation('relu'))

  #5th Convolutional Layer
  AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding=padding))
  AlexNet.add(BatchNormalization())
  AlexNet.add(Activation('relu'))
  AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding=padding))

  #Passing it to a Fully Connected layer
```

Figure 12: Implementation of AlexNet ($\text{alexnet}_1.py$)

```python
fold_no = 1
for train, val in kfold.split(np.zeros(ZX_train.shape[0]), Zy_label):
    print("********** FOLD NO: " + str(fold_no) + " **********")
    # create fold to use
    train_X, val_X = ZX_train[train], ZX_train[val]
    train_y, val_y = Zy_train[train], Zy_train[val]
    # model
    model = AlexNet(input_shape, nb_classes, padding='valid')
    model.compile(loss='binary_crossentropy', optimizer=Adam(lr=learning_rate), metrics=["accuracy"])
    lrr = ReduceLROnPlateau(monitor='val_accuracy', factor=.01, patience=3, min_lr=1e-5)

    start = time.time()
    history = model.fit(train_X, train_y,
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=(val_X, val_y),
                  shuffle=True,
                  verbose=0,
                  callbacks=[lrr])
    end = time.time()
    print("*** Time Taken (in mins): " + str(round((end - start)/60,2)))
    fold_no = fold_no + 1

    # Evaluate accuracy on test and train sets
    e.accuracy(model, train_X, val_X, train_y, val_y)
    y_test_pred_proba, conf_matrix = e.performance(model, train_X, val_X, train_y, val_y, isSequential
```

Figure 13: AlexNet with K-fold (main.ipynb)

5.3 ResNet-18

```
    if K.image_data_format() == 'channels_last':
        input_shape = (input_shape[1], input_shape[2], input_shape[0])

    # Load function from str if needed.
    block_fn = _get_block(block_fn)

    input = Input(shape=input_shape)
    conv1 = _conv_bn_relu(filters=64, kernel_size=(7, 7), strides=(2, 2))(input)
    pool1 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(conv1)

    block = pool1
    filters = 64
    for i, r in enumerate(repetitions):
        block = _residual_block(block_fn, filters=filters, repetitions=r, is_first_l
        filters *= 2

    # Last activation
    block = _bn_relu(block)

    # Classifier block
    block_shape = K.int_shape(block)
    pool2 = AveragePooling2D(pool_size=(block_shape[ROW_AXIS], block_shape[COL_AXIS]
                             strides=(1, 1))(block)
    flatten1 = Flatten()(pool2)
    dense = Dense(units=num_outputs, kernel_initializer="he_normal",
                  activation="sigmoid")(flatten1)

    model = Model(inputs=input, outputs=dense)
    return model

@staticmethod
def build_resnet_18(input_shape, num_outputs):
    return ResnetBuilder.build(input_shape, num_outputs, basic_block, [2, 2, 2, 2])
```

Figure 14: Implementation of ResNet-18 ($\text{resnet}_1.py$)

```
fold_no = 1
for train, val in kfold.split(np.zeros(ZX_train.shape[0]), Zy_label):
    print("********** FOLD NO: " + str(fold_no) + " **********")
    fold_no = fold_no + 1
    # create fold to use
    train_X, val_X = ZX_train[train], ZX_train[val]
    train_y, val_y = Zy_train[train], Zy_train[val]
    print(f'Train** N: {str(np.count_nonzero(Zy_label[train] == 0))} D:
    print(f'Valid** N: {str(np.count_nonzero(Zy_label[val] == 0))} D: {

    input_shape = (1, train_X.shape[1], train_X.shape[2])
    optimizer = Adam(lr=learning_rate)
    #optimizer = RMSprop(lr=learning_rate)
    lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1), cooldown=0, pat
    # must give input shape as (img_channels, img_rows, img_cols), resn
    model = resnet_1.ResnetBuilder.build_resnet_18(input_shape, nb_clas
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])

    start = time.time()
    history = model.fit(train_X, train_y,
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=(val_X, val_y),
                  verbose=0,
```

Figure 15: ResNet-18 with K-fold (main.ipynb)

6 Evaluation

```
        y_test_pred_proba = model.predict(X_test)
        y_train_pred_proba = model.predict(X_train)
        y_test_pred = np.argmax(y_test_pred_proba,axis=1)
        y_train_pred = np.argmax(y_train_pred_proba,axis=1)
    # Converting y_test back to 1-D array for confusion matrix computation
    y_test_1d = y_test[:, 1]
    # Computing confusion matrix for test dataset
    conf_matrix = standard_confusion_matrix(y_test_1d, y_test_pred)

    return y_test_pred_proba, conf_matrix

def additional_metrics(conf_matrix, met=None):
    # Custom evaluation metrics
    # Explanation on precision and recall:
    # Src: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
    accuracy = round(float(conf_matrix[0][0] + conf_matrix[1][1]) / np.sum(conf_matrix), 2)
    precision = round(float(conf_matrix[0][0]) / (conf_matrix[0][0] + conf_matrix[0][1]), 2)
    recall = round(float(conf_matrix[0][0]) / (conf_matrix[0][0] + conf_matrix[1][0]), 2)
    f1_score = round(2 * (precision * recall) / (precision + recall), 2)
    # Accuracy: (TP + TN) / (TP + FP + TN + FN)
    print("Accuracy: {}".format(accuracy))
    # Precision: TP / (TP + FP)
    print("Precision: {}".format(precision))
    # Recall: TP / (TP + FN)
    print("Recall: {}".format(recall))
    # F Measure: 2*(precision*recall)/(precision+recall)
    print("F1-Score: {}".format(f1_score))
    if met is not None:
        met.avg['precision'] += precision
        met.avg['recall'] += recall
        met.avg['f1_score'] += f1_score

def standard_confusion_matrix(y_test, y_test_pred):
    """
```

Figure 16: Calculates Confusion Matrix and metrics like precision, recall, and f1-score (evaluate.py)