```
 1 # Sapan Sharma, 6352819, COSC 4P80
 2 import os
 3 import numpy as np
 4 import random
 5 from scipy.stats import shapiro
 6 from scipy.stats import ttest_ind
 7 from sklearn.utils import shuffle
 8 from sklearn.model_selection import train_test_split
 9 import matplotlib.pyplot as plt
10 import tensorflow as tf
11 from tensorflow import keras
12 import pandas as pd
13 import statsmodels.api as sm
14 from statsmodels.formula.api import ols
15
16 Data2019 = os.path.join(os.getcwd(), "Data2019")  # it will get path from current directory of folder
17 GridEyeData = os.path.join(os.getcwd(), "GridEyeData")
18
19 # loading all files from disk to numpy arrays to process data. --Warning-- only tested on a windows machine.
20 # numpy arrays are loaded from files without first 2 lines of header
21 # First column(0) of thermistor reading is removed from array on axis = 1
22 control_nothing = np.delete(np.genfromtxt(os.path.join(Data2019, "control" , "nothing") , skip_header = 2), 0, 1)
23 indoors_none_blank =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "none" , "blank.txt") , skip_header = 2), 0, 1)
24 indoors_single_1foot =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "single" , "1foot.txt") , skip_header = 2), 0,
   1)
25 indoors_single_3feet =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "single" , "3feet.txt") , skip_header = 2), 0,
   1)
26 indoors_single_8feet =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "single" , "8feet.txt") , skip_header = 2), 0,
   1)
27 indoors_two_3foot =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "two" , "3foot.txt") , skip_header = 2), 0, 1)
28 indoors_three_3feet =np.delete(np.genfromtxt(os.path.join(GridEyeData, "indoors" , "three" , "3feet.txt") , skip_header = 2), 0, 1)
29 distance_1p1ft =np.delete(np.genfromtxt(os.path.join(Data2019, "distance" , "1p1ft") , skip_header = 2), 0, 1)
30 control_1p3ft =np.delete(np.genfromtxt(os.path.join(Data2019, "control" , "1p3ft") , skip_header = 2), 0, 1)
31 distance_1p6ft =np.delete(np.genfromtxt(os.path.join(Data2019, "distance" , "1p6ft") , skip_header = 2), 0, 1)
32 qty_2p3ft = np.delete(np.genfromtxt(os.path.join(Data2019, "qty" , "2p3ft") , skip_header = 2), 0, 1)
33 qty_3p3ft = np.delete(np.genfromtxt(os.path.join(Data2019, "qty" , "3p3ft") , skip_header = 2), 0, 1)
34 outdoors_nothingo =np.delete(np.genfromtxt(os.path.join(Data2019, "outdoors" , "nothingo") , skip_header = 2), 0, 1)
35 outdoors_none_blank =np.delete(np.genfromtxt(os.path.join(GridEyeData, "outdoors" , "none" , "blank.txt") , skip_header = 2), 0, 1)
36 outdoors_1p1fto =np.delete(np.genfromtxt(os.path.join(Data2019, "outdoors" , "1p1fto") , skip_header = 2), 0, 1)
37 outdoors_single_1foot =np.delete(np.genfromtxt(os.path.join(GridEyeData, "outdoors" , "single" , "1foot.txt") , skip_header = 2), 0
   , 1)
38 outdoors_1p3fto =np.delete(np.genfromtxt(os.path.join(Data2019, "outdoors" , "1p3fto") , skip_header = 2), 0, 1)
39 outdoors_single_6feet =np.delete(np.genfromtxt(os.path.join(GridEyeData, "outdoors" , "single" , "6feet.txt") , skip_header = 2), 0
```

```
39 , 1)
40
41 # creating labels for each file according to the information provided in the file and storing into a ndarray
42 # labels are as:
43 #     "Location": ["Outdoor": 0, "Indoor": 1],
44 #     "Subject Quantity": ["None": 0, "Single":1, "Two": 2, "Three": 3],
45 #     "Presence": ["Absence": 0, "Presence": 1],
46 #     "Distance": ["Dist 1'": 0, "Dist 3'": 1, "Dist 6/8'": 2]
47
48 label_control_nothing = np.tile([1,0,0,np.nan], (control_nothing.shape[0],1))
49 label_indoors_none_blank = np.tile([1,0,0,np.nan], (indoors_none_blank.shape[0],1))
50 label_indoors_single_1foot = np.tile([1,1,1,0], (indoors_single_1foot.shape[0],1))
51 label_indoors_single_3feet = np.tile([1,1,1,1], (indoors_single_3feet.shape[0],1))
52 label_indoors_single_8feet = np.tile([1,1,1,2], (indoors_single_8feet.shape[0],1))
53 label_indoors_two_3foot = np.tile([1,2,1,1], (indoors_two_3foot.shape[0],1))
54 label_indoors_three_3feet = np.tile([1,3,1,1], (indoors_three_3feet.shape[0],1))
55 label_distance_1p1ft = np.tile([np.nan,1,1,0], (distance_1p1ft.shape[0],1))
56 label_control_1p3ft = np.tile([np.nan,1,1,1], (control_1p3ft.shape[0],1))
57 label_distance_1p6ft = np.tile([np.nan,1,1,2], (distance_1p6ft.shape[0],1))
58 label_qty_2p3ft = np.tile([np.nan,2,1,1], (qty_2p3ft.shape[0],1))
59 label_qty_3p3ft = np.tile([np.nan,3,1,1], (qty_3p3ft.shape[0],1))
60 label_outdoors_nothingo = np.tile([0,0,0,np.nan], (outdoors_nothingo.shape[0],1))
61 label_outdoors_none_blank = np.tile([0,0,0,np.nan], (outdoors_none_blank.shape[0],1))
62 label_outdoors_1p1fto = np.tile([0,1,1,0], (outdoors_1p1fto.shape[0],1))
63 label_outdoors_single_1foot = np.tile([0,1,1,0], (outdoors_single_1foot.shape[0],1))
64 label_outdoors_1p3fto = np.tile([0,1,1,1], (outdoors_1p3fto.shape[0],1))
65 label_outdoors_single_6feet = np.tile([0,1,1,2], (outdoors_single_6feet.shape[0],1))
66
67 # dictionary to find out class name from a given attribute like distance or location
68 # print(class_categories_names["Distance"][1])#instead of 1 goes prediction of neural net using model.predict_classes(X)
69 class_categories_names = {
70     "Location": ["Outdoor", "Indoor"],
71     "Subject Quantity": ["None", "Single", "Two", "Three"],
72     "Presence": ["Absence", "Presence"],
73     "Distance": ["Dist 1'", "Dist 3'", "Dist 6/8'"]
74 }
75
76 # creating datasets and labels to use for MLP models. Datasets and labels are created according to the identified
77 # files which answers the required questions. like dist_in_1_dataset uses all files which have 1 object indoors with
78 # different distance so it is used to classify distance of object indoors.
79
80 # concatenates different datasets along 0 axis
81 # concatenates different labels along 0 axis and choose only label which answers question and is required from label
```

```
 82 # arr and converts to integer as previously these arrays had np.nan values. (conversion for cross entropy loss function)
 83
 84 dist_in_1_dataset = np.concatenate((indoors_single_1foot,indoors_single_3feet,indoors_single_8feet), axis = 0)
 85 dist_in_1_label = np.concatenate((label_indoors_single_1foot, label_indoors_single_3feet, label_indoors_single_8feet), axis = 0
    )[:,3].astype(int)
 86 dist_unknown_dataset = np.concatenate((distance_1p1ft, control_1p3ft, distance_1p6ft), axis = 0)
 87 dist_unknown_label = np.concatenate((label_distance_1p1ft, label_control_1p3ft, label_distance_1p6ft), axis = 0)[:,3].astype(int)
    #gets only distance label
 88 dist_out_dataset = np.concatenate((outdoors_1p1fto, outdoors_single_1foot, outdoors_1p3fto, outdoors_single_6feet), axis = 0)
 89 dist_out_label = np.concatenate((label_outdoors_1p1fto, label_outdoors_single_1foot, label_outdoors_1p3fto,
    label_outdoors_single_6feet), axis = 0)[:,3].astype(int)
 90 presence_in_dataset = np.concatenate((control_nothing, indoors_none_blank, indoors_single_1foot,indoors_single_3feet,
    indoors_single_8feet, indoors_two_3foot, indoors_three_3feet), axis = 0)
 91 presence_in_label = np.concatenate((label_control_nothing, label_indoors_none_blank, label_indoors_single_1foot,
    label_indoors_single_3feet, label_indoors_single_8feet, label_indoors_two_3foot, label_indoors_three_3feet), axis = 0)[:,2].astype
    (int)
 92 presence_out_dataset = np.concatenate((outdoors_nothingo, outdoors_none_blank, outdoors_1p1fto, outdoors_single_1foot,
    outdoors_1p3fto, outdoors_single_6feet), axis = 0)
 93 presence_out_label = np.concatenate((label_outdoors_nothingo, label_outdoors_none_blank, label_outdoors_1p1fto,
    label_outdoors_single_1foot, label_outdoors_1p3fto, label_outdoors_single_6feet), axis = 0)[:,2].astype(int)
 94 qty_in_dataset = np.concatenate((control_nothing, indoors_none_blank, indoors_single_1foot,indoors_single_3feet,
    indoors_single_8feet, indoors_two_3foot, indoors_three_3feet), axis = 0)
 95 qty_in_label = np.concatenate((label_control_nothing, label_indoors_none_blank, label_indoors_single_1foot,
    label_indoors_single_3feet, label_indoors_single_8feet, label_indoors_two_3foot, label_indoors_three_3feet), axis = 0)[:,1].astype
    (int)
 96 qty_unknown_dataset = np.concatenate((control_1p3ft, qty_2p3ft, qty_3p3ft), axis = 0)
 97 qty_unknown_label = np.concatenate((label_control_1p3ft, label_qty_2p3ft, label_qty_3p3ft), axis = 0)[:,1].astype(int)
 98 location_dataset = np.concatenate((control_nothing, indoors_none_blank, indoors_single_1foot,indoors_single_3feet,
    indoors_single_8feet, indoors_two_3foot, indoors_three_3feet, outdoors_nothingo, outdoors_none_blank, outdoors_1p1fto,
    outdoors_single_1foot, outdoors_1p3fto, outdoors_single_6feet), axis = 0)
 99 location_label = np.concatenate((label_control_nothing, label_indoors_none_blank, label_indoors_single_1foot,
    label_indoors_single_3feet, label_indoors_single_8feet, label_indoors_two_3foot, label_indoors_three_3feet,
    label_outdoors_nothingo, label_outdoors_none_blank, label_outdoors_1p1fto, label_outdoors_single_1foot, label_outdoors_1p3fto,
    label_outdoors_single_6feet), axis = 0)[:,0].astype(int)
100
101
102 # pre process data before passing to neural network.
103 def preProcess(X,y):
104
105     # pre-thresholding(clipping)- any data above 1000 is divided by 10 to prevent overexposure to radiation,
106     # overexposure affects the ability to adjust weights (find features) in the dataset.
107     np.divide(X, 10, out = X, where = X > 1000)
108     # normalization- perform min max feature scaling on data to bring them in a range of [0,1]
```

```
109     max_v = np.max(X)
110     min_v = np.min(X)
111     X = np.array([(s - min_v) / (max_v - min_v) for s in X])
112     # shuffle the dataset in unison so that the labels and samples remain in same order
113     # shuffling will help neural network to process different instance as files are concatenated.
114     X, y = shuffle(X, y, random_state = 66)
115     # split dataset and labels into train(75%) and test(25%) for each using train test split function from sklearn.
116     X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False)
117
118     return X_train, X_test, y_train, y_test
119
120
121 # get train test split of dataset and labels, get data after clipping, get data after normalization and shuffle. pheww--
122 dist_in_1_dataset_train, dist_in_1_dataset_test, dist_in_1_label_train, dist_in_1_label_test = preProcess(dist_in_1_dataset,
    dist_in_1_label)
123 dist_unknown_dataset_train, dist_unknown_dataset_test, dist_unknown_label_train, dist_unknown_label_test = preProcess(
    dist_unknown_dataset, dist_unknown_label)
124 dist_out_dataset_train, dist_out_dataset_test, dist_out_label_train, dist_out_label_test = preProcess(dist_out_dataset,
    dist_out_label)
125 presence_in_dataset_train, presence_in_dataset_test, presence_in_label_train, presence_in_label_test = preProcess(
    presence_in_dataset, presence_in_label)
126 presence_out_dataset_train, presence_out_dataset_test, presence_out_label_train, presence_out_label_test = preProcess(
    presence_out_dataset, presence_out_label)
127 qty_in_dataset_train, qty_in_dataset_test, qty_in_label_train, qty_in_label_test = preProcess(qty_in_dataset, qty_in_label)
128 qty_unknown_dataset_train, qty_unknown_dataset_test, qty_unknown_label_train, qty_unknown_label_test = preProcess(
    qty_unknown_dataset, qty_unknown_label)
129 location_dataset_train, location_dataset_test, location_label_train, location_label_test = preProcess(location_dataset,
    location_label)
130
131
132 # sparse categorical cross entropy loss and softmax output activation function unless different values passed
133 # sigmoid output activation function and binary crossentropy if labels are binary (yes or no/ presence or absence)
134 # softmax output activation function for the multi-classes which are exclusive and sparse categorical crossentropy loss function
135 # for sparse labels with target class index like 0,1,2,3 for multi-class output.
136
137 # user can input most hyperparameters to the function
138
139 def build_model_GD(n_hidden, n_neurons_list, activation_function, user_learning_rate, user_momentum, n_outputs, user_loss="
    sparse_categorical_crossentropy", output_activation="softmax"):
140     tf.keras.backend.clear_session() # to clear previous values and weights from tf backend
141     i = 0 # index to get value of neurons from neuron list
142     model = keras.models.Sequential() # adds layers sequentially
143     model.add(keras.layers.InputLayer(input_shape=[64,])) # input layer with shape (none, 64) where none is a batch axis
```

```
144     for layer in range(n_hidden): # to create user given number of hidden layers
145         model.add(keras.layers.Dense(n_neurons_list[i], activation=activation_function))
146         i += 1
147     model.add(keras.layers.Dense(n_outputs, activation=output_activation)) # output layer with output function sigmoid or softmax depending on class
148     # initializing stochastic gradient descent optimizer which computes gradient descent step in its tape tree.
149     optimizer = keras.optimizers.SGD(learning_rate=user_learning_rate, momentum=user_momentum)
150     # if binary label classification calculate false positives and false negatives
151     if output_activation == "sigmoid":
152         metrics_list = ["accuracy", tf.keras.metrics.FalsePositives(), tf.keras.metrics.FalseNegatives()]
153     else:
154         metrics_list = ["accuracy"] # otherwise just accuracy
155     # compiling the model where random weights and bias weights are set loss function, gradient descent is initialized and metrics to calculate are loaded
156     model.compile(loss=user_loss, metrics=metrics_list, optimizer=optimizer)
157     return model # returns model object
158
159
160 # sparse categorical cross entropy loss and softmax output activation function unless different values passed
161 # sigmoid output activation function and binary crossentropy if labels are binary (yes or no/ presence or absence)
162 # softmax output activation function for the multi-classes which are exclusive and sparse categorical crossentropy loss function
163 # for sparse labels with target class index like 0,1,2,3 for multi-class output.
164
165 # user can input most hyperparameters to the function
166
167 def build_model_RMS(n_hidden, n_neurons_list, activation_function, user_learning_rate, user_momentum, n_outputs, user_rho = 0.9, user_loss="sparse_categorical_crossentropy", output_activation="softmax"):
168     tf.keras.backend.clear_session() # to clear previous values and weights from tf backend
169     j = 0
170     model = keras.models.Sequential()
171     model.add(keras.layers.InputLayer(input_shape=[64,]))
172     for layer in range(n_hidden):
173         model.add(keras.layers.Dense(n_neurons_list[j], activation=activation_function))
174         j += 1
175     model.add(keras.layers.Dense(n_outputs, activation=output_activation))
176     # RMSprop optimizer which is a variation for learning which maintains a moving average of square of gradients and divide the gradient by the root of that average.
177     # this adaptive learning rate method of gradient descent provides an alternative to classical stochastic gradient descent implemented above.
178     optimizer = keras.optimizers.RMSprop(learning_rate=user_learning_rate, momentum=user_momentum, rho=user_rho)
179     # Accuracy calculates accuracy according to the context of loss. If loss is binary cross entropy then it calculates how often predictions match binary labels.
180     # If loss is sparse catagorical cross entropy then it calculates how often predictions matches integer labels.
```

```
    # FalsePositives class calculates the number of false positives and FalseNegatives class calculates the number of false negatives.
    # if binary label classification calculate false positives and false negatives
    if output_activation == "sigmoid":
        metrics_list = ["accuracy", tf.keras.metrics.FalsePositives(), tf.keras.metrics.FalseNegatives()]
    else:
        metrics_list = ["accuracy"] # otherwise just accuracy
    model.compile(loss=user_loss, metrics=metrics_list, optimizer=optimizer)
    return model

def main():
    print("Please ignore red warning, those are for GPU which is not connected in the env. And make sure inputs are correct as there is no try catch block.")
    print("Sample input for Gradient Descent: 1,2,20,10,relu,0.1,0,9,20..  Sample input for RMSprop: 2,2,20,10,relu,0.009,0.9,0.9,20..")
    test_no = int(input("Do you want to perform ANOVA Test on different hyperparameters, if no enter 1. Otherwise, Enter number of variants: "))
    df_GD = {}
    df_RMS = {}
    for i in range(test_no):
        print()
        model_no = int(input("Enter 1 for Gradient Descent, and 2 for RMSprop: "))
        n_hidden_v = int(input("Enter number of hidden layer for test: "))
        n_neurons_list_v = []
        for i in range(n_hidden_v):
            s = int(input("Enter number of neurons for "+str(i+1)+" hidden layer: "))
            n_neurons_list_v.append(s)

        activation_function_v = input("Activation function for hidden layer: ")
        if model_no == 1:
            user_learning_rate_GD = float(input("Enter learning rate for Gradient Descent: "))
            user_momentum_GD = float(input("Enter momentum for Gradient Descent: "))

        if model_no == 2:
            user_learning_rate_RMS = float(input("Enter learning rate for RMSprop (optimum: 0.001-0.01): "))
            user_momentum_RMS = float(input("Enter momentum for RMSprop (optimum: 0.3-0.9): "))
            user_rho_v = float(input("Enter rho value for RMSprop (optimum:0.7-0.9): "))

        epochs_v = int(input("Enter number of epochs (give 20): "))

        # model_dist_in_1_dataset classifies 3 classes Dist 1': 0; Dist 3': 1; Dist 8': 2 for predicting distance indoor with 1 subject
        # model_dist_unknown_dataset classifies 3 classes Dist 1': 0; Dist 3': 1; Dist 6': 2 for predicting distance unknown
```

```
location with 1 subject
        # model_dist_out_dataset classifies 3 classes Dist 1': 0; Dist 3': 1; Dist 6': 2 for predicting distance outdoor with different subject
        # model_presence_in_dataset classifies binary label classes Presence: 0; Absence: 1; predicts if subject is present or absent indoors
        # model_presence_out_dataset classifies binary label Presence: 0; Absence: 1; predicts if subject is present or absent outdoors
        # model_qty_in_dataset classifies 4 classes none: 0; single: 1; two:2; three:3; for predicting quantity of subjects inside
        # model_qty_unknown_dataset classifies 3 classes single: 1; two:2; three:3; for predicting quantity of subjects in unknown location
        # model_location_dataset classifies binary label classes Outdoor:0; Indoor:1; for predicting the location of the subject irrespective of quantity or distance

        if model_no == 1:
            model_dist_in_1_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=3,user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_in_1_dataset_GD")
            history_dist_in_1_dataset_GD = model_dist_in_1_dataset_GD.fit(dist_in_1_dataset_train, dist_in_1_label_train, verbose=2,epochs=epochs_v)
            print()
            model_dist_unknown_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=3,user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_unknown_dataset_GD")
            history_dist_unknown_dataset_GD = model_dist_unknown_dataset_GD.fit(dist_unknown_dataset_train, dist_unknown_label_train,verbose=2,epochs=epochs_v)
            print()
            model_dist_out_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=3,user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_out_dataset_GD")
            history_dist_out_dataset_GD = model_dist_out_dataset_GD.fit(dist_out_dataset_train, dist_out_label_train,verbose=2, epochs=epochs_v)
            print()
            model_presence_in_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=1,user_loss="binary_crossentropy",output_activation="sigmoid")
            print("Training Stats of presence_in_dataset_GD")
            history_presence_in_dataset_GD = model_presence_in_dataset_GD.fit(presence_in_dataset_train, presence_in_label_train, verbose=2,epochs=epochs_v)
            print()
            model_presence_out_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function
```

```
243 =activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=1,user_loss="
    binary_crossentropy",output_activation="sigmoid")
244             print("Training Stats of presence_out_dataset_GD")
245             history_presence_out_dataset_GD = model_presence_out_dataset_GD.fit(presence_out_dataset_train,
    presence_out_label_train, verbose=2,epochs=epochs_v)
246             print()
247             model_qty_in_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=
    activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=4,user_loss= "
    sparse_categorical_crossentropy", output_activation="softmax")
248             print("Training Stats of qty_in_dataset_GD")
249             history_qty_in_dataset_GD = model_qty_in_dataset_GD.fit(qty_in_dataset_train, qty_in_label_train, verbose=2,epochs=
    epochs_v)
250             print()
251             model_qty_unknown_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=
    activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=4,user_loss= "
    sparse_categorical_crossentropy", output_activation="softmax")
252             print("Training Stats of qty_unknown_dataset_GD")
253             history_qty_unknown_dataset_GD = model_qty_unknown_dataset_GD.fit(qty_unknown_dataset_train, qty_unknown_label_train,
    verbose=2,epochs=epochs_v)
254             print()
255             model_location_dataset_GD = build_model_GD(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=
    activation_function_v,user_learning_rate=user_learning_rate_GD,user_momentum=user_momentum_GD,n_outputs=1,user_loss="
    binary_crossentropy",output_activation="sigmoid")
256             print("Training Stats of location_dataset_GD:")
257             history_location_dataset_GD = model_location_dataset_GD.fit(location_dataset_train, location_label_train, verbose=2,
    epochs=epochs_v)
258             print()
259             print("Model evaluations on test data set: ")
260             evaluate_dist_in_1_dataset_train_GD = model_dist_in_1_dataset_GD.evaluate(dist_in_1_dataset_test,
    dist_in_1_label_test)
261             evaluate_dist_unknown_dataset_train_GD = model_dist_unknown_dataset_GD.evaluate(dist_unknown_dataset_test,
    dist_unknown_label_test)
262             evaluate_dist_out_dataset_train_GD = model_dist_out_dataset_GD.evaluate(dist_out_dataset_test,  dist_out_label_test)
263             evaluate_presence_in_dataset_train_GD = model_presence_in_dataset_GD.evaluate(presence_in_dataset_test,
    presence_in_label_test)
264             evaluate_presence_out_dataset_train_GD = model_presence_out_dataset_GD.evaluate(presence_out_dataset_test,
    presence_out_label_test)
265             evaluate_qty_in_dataset_train_GD = model_qty_in_dataset_GD.evaluate(qty_in_dataset_test, qty_in_label_test)
266             evaluate_qty_unknown_dataset_train_GD = model_qty_unknown_dataset_GD.evaluate(qty_unknown_dataset_test,
    qty_unknown_label_test)
267             evaluate_location_dataset_train_GD = model_location_dataset_GD.evaluate(location_dataset_test,location_label_test)
268             print()
269             print("Hyperparameters for this run: Model"+str(model_no)+"_Hidden:"+str(n_hidden_v)+"_Neurons"+str(n_neurons_list_v)+
```

```
"_LR"+str(user_learning_rate_GD)+"_M"+str(user_momentum_GD))

        if model_no == 2:
            model_dist_in_1_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v, n_outputs=3, user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_in_1_dataset_RMS")
            history_dist_in_1_dataset_RMS = model_dist_in_1_dataset_RMS.fit(dist_in_1_dataset_train, dist_in_1_label_train, verbose=2,epochs=epochs_v)
            print()
            model_dist_unknown_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v, activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=3,user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_unknown_dataset_RMS")
            history_dist_unknown_dataset_RMS = model_dist_unknown_dataset_RMS.fit(dist_unknown_dataset_train, dist_unknown_label_train, verbose=2,epochs=epochs_v)
            print()
            model_dist_out_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=3, user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of dist_out_dataset_RMS")
            history_dist_out_dataset_RMS = model_dist_out_dataset_RMS.fit(dist_out_dataset_train, dist_out_label_train, verbose=2, epochs=epochs_v)
            print()
            model_presence_in_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v, activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=1,user_loss="binary_crossentropy",output_activation="sigmoid")
            print("Training Stats of presence_in_dataset_RMS")
            history_presence_in_dataset_RMS = model_presence_in_dataset_RMS.fit(presence_in_dataset_train,presence_in_label_train , verbose=2,epochs=epochs_v)
            print()
            model_presence_out_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v, activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=1,user_loss="binary_crossentropy",output_activation="sigmoid")
            print("Training Stats of presence_out_dataset_RMS")
            history_presence_out_dataset_RMS = model_presence_out_dataset_RMS.fit(presence_out_dataset_train, presence_out_label_train, verbose=2,epochs=epochs_v)
            print()
            model_qty_in_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=4, user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
            print("Training Stats of qty_in_dataset_RMS")
            history_qty_in_dataset_RMS = model_qty_in_dataset_RMS.fit(qty_in_dataset_train, qty_in_label_train, verbose=2,epochs=
```

```
294 epochs_v)
295             print()
296             model_qty_unknown_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,
    activation_function=activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=
    user_rho_v,n_outputs=4,user_loss= "sparse_categorical_crossentropy", output_activation="softmax")
297             print("Training Stats of qty_unknown_dataset_RMS")
298             history_qty_unknown_dataset_RMS = model_qty_unknown_dataset_RMS.fit(qty_unknown_dataset_train, qty_unknown_label_train
    , verbose=2,epochs=epochs_v)
299             print()
300             model_location_dataset_RMS = build_model_RMS(n_hidden=n_hidden_v,n_neurons_list=n_neurons_list_v,activation_function=
    activation_function_v,user_learning_rate=user_learning_rate_RMS,user_momentum=user_momentum_RMS, user_rho=user_rho_v,n_outputs=1,
    user_loss="binary_crossentropy",output_activation="sigmoid")
301             print("Training Stats of location_dataset_RMS:")
302             history_location_dataset_RMS = model_location_dataset_RMS.fit(location_dataset_train, location_label_train, verbose=2,
    epochs=epochs_v)
303             print()
304             print("Model evaluations on test data set: ")
305             evaluate_dist_in_1_dataset_train_RMS = model_dist_in_1_dataset_RMS.evaluate( dist_in_1_dataset_test,
    dist_in_1_label_test)
306             evaluate_dist_unknown_dataset_train_RMS = model_dist_unknown_dataset_RMS.evaluate(dist_unknown_dataset_test,
    dist_unknown_label_test)
307             evaluate_dist_out_dataset_train_RMS = model_dist_out_dataset_RMS.evaluate(dist_out_dataset_test, dist_out_label_test)
308             evaluate_presence_in_dataset_train_RMS = model_presence_in_dataset_RMS.evaluate( presence_in_dataset_test,
    presence_in_label_test)
309             evaluate_presence_out_dataset_train_RMS = model_presence_out_dataset_RMS.evaluate(presence_out_dataset_test,
    presence_out_label_test)
310             evaluate_qty_in_dataset_train_RMS = model_qty_in_dataset_RMS.evaluate(qty_in_dataset_test, qty_in_label_test)
311             evaluate_qty_unknown_dataset_train_RMS = model_qty_unknown_dataset_RMS.evaluate(qty_unknown_dataset_test,
    qty_unknown_label_test)
312             evaluate_location_dataset_train_RMS = model_location_dataset_RMS.evaluate(location_dataset_test, location_label_test)
313             print()
314             print("Hyperparameters for this run: Model"+str(model_no)+"_Hidden:"+str(n_hidden_v)+"_Neurons"+str(n_neurons_list_v)+
    "_LR"+str(user_learning_rate_RMS)+"_M"+str(user_momentum_RMS))
315
316         if test_no > 1:
317             if model_no == 1:
318                 test_name = "GD"+"_"+str(n_hidden_v)+"_"+str(n_neurons_list_v)+"_"+str(user_learning_rate_GD)+"_"+str(
    user_momentum_GD)
319                 GD_1 = (evaluate_dist_in_1_dataset_train_GD[1],
320                             evaluate_dist_unknown_dataset_train_GD[1],
321                             evaluate_dist_out_dataset_train_GD[1],
322                             evaluate_presence_in_dataset_train_GD[1],
323                             evaluate_presence_out_dataset_train_GD[1],
```

```
                                evaluate_qty_in_dataset_train_GD[1],
                                evaluate_qty_unknown_dataset_train_GD[1],
                                evaluate_location_dataset_train_GD[1],)
                df_GD[test_name] = GD_1
            if model_no == 2:
                test_name = "RMS"+"_"+str(n_hidden_v)+"_"+str(n_neurons_list_v)+"_"+str(user_learning_rate_RMS)+"_"+str(
    user_momentum_RMS)
                RMS_1 = (evaluate_dist_in_1_dataset_train_RMS[1],
                        evaluate_dist_unknown_dataset_train_RMS[1],
                        evaluate_dist_out_dataset_train_RMS[1],
                        evaluate_presence_in_dataset_train_RMS[1],
                        evaluate_presence_out_dataset_train_RMS[1],
                        evaluate_qty_in_dataset_train_RMS[1],
                        evaluate_qty_unknown_dataset_train_RMS[1],
                        evaluate_location_dataset_train_RMS[1],)
                df_RMS[test_name] = RMS_1

    # doing statistical tests
    if model_no == 1:
        experiment1 = pd.DataFrame(df_GD)
        experiment1.index.name = 'Dataset NO'
        print()
        print("Accuracy of all tests in- Gradient_hiddenLayers_neuronsList_LearningRate_Momentum: ")
        print()
        print(experiment1)
        print()
        print(experiment1.describe()) # mean, std. dev, IQ ranges, min, max
        print()
        fig1, ax1 = plt.subplots() # boxplot for initial reference
        ax1.set_title('Plot')
        ax1.boxplot(experiment1)
        plt.show()

        # shapiro test
        # testing null hypothesis if data is normally distributed.
        # if p-value greater than 0.05 it is, otherwise reject the hypothesis
        print("Shapiro test")
        for col, val in experiment1.iteritems():
            for col, val in experiment1.iteritems():
                print(col, end=' ')
                stat, p = shapiro(val)
                print('stats=%.4f, p=%.4f' % (stat,p))

```

```
366         # ANOVA test
367         # Hypothesis that mean of all datasets are not statistically different from each other.
368         # Hypothesis holds if p-value>0.05
369         # Otherwise there exist a better mean of all which can be found with T-test
370         # last minute bug
371         print()
372         print("ANOVA test")
373         ana_model = ols('experiment1.iloc[:,0]~experiment1.iloc[:,1]+experiment1.iloc[:,2]', experiment1).fit()
374         anova_table = sm.stats.anova_lm(ana_model, typ=2)
375         print(anova_table)
376 
377         # t-test, our hypothesis that means are same no statistical advantage over any parameter.
378         # p-value > 0.05 we don't have statistical evidence to 2 comparisons have different means, hypothesis stands
379         # if p-value less than 0.05 there is statistical evidence that one of the mean is better, hypothesis is rejected.
380         print()
381         print("t-tests")
382         t1 = ttest_ind(experiment1.iloc[:,0],experiment1.iloc[:,1], equal_var=False)
383         t2 = ttest_ind(experiment1.iloc[:,1],experiment1.iloc[:,2], equal_var=False)
384         t3 = ttest_ind(experiment1.iloc[:,0],experiment1.iloc[:,2], equal_var=False)
385         print(t1)
386         print(t2)
387         print(t3)
388         print()
389 
390         # then whichever average value of accuracy is higher if hypothesis is rejected is choosen so we get better hyper-
    parameters
391         if t1[1]< 0.005 or t2[1]< 0.005 or t3[1]< 0.005:
392 
393             print("Statistically better mean available")
394         else:
395             print("No statistically better mean")
396 
397 
398     if model_no == 2:
399         experiment2 = pd.DataFrame(df_RMS)
400         experiment2.index.name = 'Dataset NO'
401         print()
402         print("Accuracy of all tests in- Gradient_hiddenLayers_neuronsList_LearningRate_Momentum: ")
403         print()
404         print(experiment2)
405         print()
406         print(experiment2.describe())
407         print()
```

```
408         fig1, ax1 = plt.subplots()
409         ax1.set_title('Plot')
410         ax1.boxplot(experiment2)
411         plt.show()
412 
413         # shapiro test
414         # testing null hypothesis if data is normally distributed.
415         # if p-value greater than 0.05 it is, otherwise reject the hypothesis
416         print("Shapiro test")
417         for col, val in experiment2.iteritems():
418             for col, val in experiment2.iteritems():
419                 print(col, end=' ')
420                 stat, p = shapiro(val)
421                 print('stats=%.4f, p=%.4f' % (stat,p))
422 
423         # ANOVA test
424         # Hypothesis that mean of all datasets are not statistically different from each other.
425         # Hypothesis holds if p-value>0.05
426         # Otherwise there exist a better mean of all which can be found with T-test
427         # last minute bug
428         print()
429         print("ANOVA test")
430         ana_model = ols('experiment2.iloc[:,0]~experiment2.iloc[:,1]+experiment2.iloc[:,2]', experiment2).fit()
431         anova_table = sm.stats.anova_lm(ana_model, typ=2)
432         print(anova_table)
433         # t-test, our hypothesis that means are same no statistical advantage over any parameter.
434         # p-value > 0.05 we don't have statistical evidence to 2 comparisons have different means, hypothesis stands
435         # if p-value less than 0.05 there is statistical evidence that one of the mean is better, hypothesis is rejected.
436         print()
437         print("t-tests")
438         t1 = ttest_ind(experiment2.iloc[:,0],experiment2.iloc[:,1], equal_var=False)
439         t2 = ttest_ind(experiment2.iloc[:,1],experiment2.iloc[:,2], equal_var=False)
440         t3 = ttest_ind(experiment2.iloc[:,0],experiment2.iloc[:,2], equal_var=False)
441         print(t1)
442         print(t2)
443         print(t3)
444         print()
445         # then whichever average value of accuracy is higher if hypothesis is rejected is choosen so we get better hyper-
    parameters
446         if t1[1]< 0.005 or t2[1]< 0.005 or t3[1]< 0.005:
447             print("Statistically better mean available: ")
448         else:
449             print("No statistically better mean")
```

```
450
451
452 if __name__ == '__main__' :
453     main()
454
```