Annex U. Python code that implements the OneVsRest classiﬁcation strategy

import matplotlib.pyplot as plt

import pandas as pd

import pyodbc

# Environment settings:

pd.set\_option('display.max\_column', None)

pd.set\_option('display.max\_rows', None)

pd.set\_option('display.max\_seq\_items', None)

pd.set\_option('display.max\_colwidth', 500)

pd.set\_option('expand\_frame\_repr', True)

defaults.to\_csv('base\_table\_' + name\_string + '.csv', index=False)

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.decomposition import PCA

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import roc\_auc\_score, confusion\_matrix

from sklearn.metrics import f1\_score

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

pd.set\_option('display.max\_rows', 500)

pd.set\_option('display.max\_columns', 500)

# mlp for multi-label classification

from numpy import mean

from numpy import std

from sklearn.datasets import make\_multilabel\_classification

from sklearn.model\_selection import RepeatedKFold

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import accuracy\_score

# get the dataset

def get\_dataset():

X = defaults.drop(columns = ['vie', 'du', 'tr', 'ket', 'pen', 'ses', 'sep', 'ast'])

X.shape

y = defaults.drop(columns = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay'

,'Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

y.shape

return X, y

# load dataset

X, y = get\_dataset()

X.describe()

y.describe()

# create dataframe from file

dataframe = X

# use corr() method on dataframe to

# make correlation matrix

matrix = dataframe.corr()

# print correlation matrix

print("Correlation Matrix is : ")

print(matrix)

corr = dataframe.corr()

corr.style.background\_gradient(cmap='coolwarm')

from keras.models import Sequential

from keras.layers import Dense, Activation

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.datasets import make\_multilabel\_classification

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.losses import binary\_crossentropy

from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import label\_binarize

n\_inputs, n\_outputs = X.shape[1], y.shape[1]

print(n\_inputs, n\_outputs)

# Binarize the output

y = label\_binarize(y, classes=['vie', 'du', 'tr', 'ket', 'pen', 'ses', 'sep', 'ast'])

n\_classes = y.shape[1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42)

from sklearn.linear\_model import LogisticRegression

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.svm import LinearSVC

from sklearn.naive\_bayes import MultinomialNB

from lightgbm import LGBMClassifier

from sklearn.multiclass import OneVsRestClassifier

from sklearn.svm import SVC

import matplotlib.pylab as pl

from sklearn.metrics import roc\_curve, auc

# Learn to predict each class against the other

classif = OneVsRestClassifier(SVC(kernel='linear'))

classif =classif.fit(X\_train, y\_train)

yhat = classif.predict(X\_test)

print(yhat)

y\_score = classif.decision\_function(X\_test)

print(y\_score)

# Compute ROC curve and ROC area for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = y.shape[1]

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:, i], y\_score[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test.ravel(), y\_score.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[4], tpr[4], label='ROC curve (area = %0.2f)' % roc\_auc[4])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[5], tpr[5], label='ROC curve (area = %0.2f)' % roc\_auc[5])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot ROC curve

plt.figure()

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average ROC curve (area = {0:0.2f})'

''.format(roc\_auc["micro"]))

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Some extension of Receiver operating characteristic to multi-class')

plt.legend(loc="lower right")

plt.show()

from sklearn.metrics import hamming\_loss, accuracy\_score

y\_true = y\_test

print (y\_true)

y\_pred = yhat

print (y\_pred)

print("accuracy\_score:", accuracy\_score(y\_true, y\_pred))

print("Hamming\_loss:", hamming\_loss(y\_true, y\_pred))

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.metrics import f1\_score

m = MultiLabelBinarizer().fit(y\_true)

f1\_score(m.transform(y\_true),

m.transform(y\_pred),

average='macro')

from sklearn.metrics import precision\_recall\_fscore\_support

precision\_recall\_fscore\_support(y\_true, y\_pred, average='macro')

precision\_recall\_fscore\_support(y\_true, y\_pred, average='micro')

precision\_recall\_fscore\_support(y\_true, y\_pred, average='weighted')

precision\_recall\_fscore\_support(y\_true, y\_pred)

from sklearn.metrics import classification\_report

print(classification\_report(y\_true, y\_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7 ]))

from sklearn.linear\_model import Perceptron

clf= OneVsRestClassifier(Perceptron(tol=1e-3, random\_state=0))

clf=clf.fit(X\_train, y\_train)

yhatt = clf.predict(X\_test)

print(yhatt)

#Predict margin (libsvm name for this is predict\_values)

y\_scoree = clf.decision\_function(X\_test)

# Compute ROC curve and ROC area for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = y.shape[1]

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:, i], y\_scoree[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test.ravel(), y\_scoree.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[4], tpr[4], label='ROC curve (area = %0.2f)' % roc\_auc[4])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[5], tpr[5], label='ROC curve (area = %0.2f)' % roc\_auc[5])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot ROC curve

plt.figure()

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average ROC curve (area = {0:0.2f})'

''.format(roc\_auc["micro"]))

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Some extension of Receiver operating characteristic to multi-class')

plt.legend(loc="lower right")

plt.show()

from sklearn.metrics import hamming\_loss, accuracy\_score

y\_true = y\_test

print (y\_true)

y\_pred = yhatt

print (y\_pred)

print("accuracy\_score:", accuracy\_score(y\_true, y\_pred))

print("Hamming\_loss:", hamming\_loss(y\_true, y\_pred))

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.metrics import f1\_score

m = MultiLabelBinarizer().fit(y\_true)

f1\_score(m.transform(y\_true),

m.transform(y\_pred),

average='macro')

from sklearn.metrics import precision\_recall\_fscore\_support

precision\_recall\_fscore\_support(y\_true, y\_pred, average='macro')

precision\_recall\_fscore\_support(y\_true, y\_pred, average='micro')

precision\_recall\_fscore\_support(y\_true, y\_pred, average='weighted')

precision\_recall\_fscore\_support(y\_true, y\_pred)

from sklearn.metrics import classification\_report

print(classification\_report(y\_true, y\_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7 ]))

from sklearn.linear\_model import LogisticRegression

clf= OneVsRestClassifier(LogisticRegression(random\_state=0, max\_iter=600))

clf=clf.fit(X\_train, y\_train)

yhattt = clf.predict(X\_test)

print(yhatt)

#Predict margin (libsvm name for this is predict\_values)

y\_scoreee = clf.decision\_function(X\_test)

# Compute ROC curve and ROC area for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = y.shape[1]

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:, i], y\_scoreee[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test.ravel(), y\_scoreee.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[4], tpr[4], label='ROC curve (area = %0.2f)' % roc\_auc[4])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot of a ROC curve for a specific class

#Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.

plt.figure()

plt.plot(fpr[5], tpr[5], label='ROC curve (area = %0.2f)' % roc\_auc[5])

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()

# Plot ROC curve

plt.figure()

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average ROC curve (area = {0:0.2f})'

''.format(roc\_auc["micro"]))

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Some extension of Receiver operating characteristic to multi-class')

plt.legend(loc="lower right")

plt.show()

from sklearn.metrics import hamming\_loss, accuracy\_score

y\_true = y\_test

print (y\_true)

y\_pred = yhattt

print (y\_pred)

print("accuracy\_score:", accuracy\_score(y\_true, y\_pred))

print("Hamming\_loss:", hamming\_loss(y\_true, y\_pred))

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.metrics import f1\_score

m = MultiLabelBinarizer().fit(y\_true)

f1\_score(m.transform(y\_true),

m.transform(y\_pred),

average='macro')

from sklearn.metrics import precision\_recall\_fscore\_support

precision\_recall\_fscore\_support(y\_true, y\_pred, average='macro')

from sklearn.metrics import precision\_recall\_fscore\_support

precision\_recall\_fscore\_support(y\_true, y\_pred, average='micro')

from sklearn.metrics import precision\_recall\_fscore\_support

precision\_recall\_fscore\_support(y\_true, y\_pred, average='weighted')

precision\_recall\_fscore\_support(y\_true, y\_pred)

from sklearn.metrics import classification\_report

print(classification\_report(y\_true, y\_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7 ]))

from sklearn.naive\_bayes import MultinomialNB

from sklearn.linear\_model import SGDClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.multiclass import OneVsRestClassifier

from sklearn.svm import LinearSVC

from sklearn.metrics import hamming\_loss

from sklearn.linear\_model import Perceptron

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import PassiveAggressiveClassifier

def hamming\_score(y\_true, y\_pred, normalize=True, sample\_weight=None):

acc\_list = []

for i in range(y\_true.shape[0]):

set\_true = set( np.where(y\_true[i])[0] )

set\_pred = set( np.where(y\_pred[i])[0] )

tmp\_a = None

if len(set\_true) == 0 and len(set\_pred) == 0:

tmp\_a = 1

else:

tmp\_a = len(set\_true.intersection(set\_pred))/float(len(set\_true.union(set\_pred)))

acc\_list.append(tmp\_a)

return np.mean(acc\_list)

def print\_score(y\_pred, clf):

print("Clf: ", clf.\_\_class\_\_.\_\_name\_\_)

print("Hamming loss: {}".format(hamming\_loss(y\_pred, y\_test)))

print("Hamming score: {}".format(hamming\_score(y\_pred, y\_test)))

print("---")

nb\_clf = MultinomialNB()

sgd = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random\_state=42, max\_iter=6, tol=None)

lr = LogisticRegression(random\_state=0, max\_iter=900)

mn = LinearSVC(random\_state=0,max\_iter=130000, tol=1e-5)

prc = Perceptron(tol=1e-3, random\_state=0)

bst = GradientBoostingClassifier(n\_estimators=200, learning\_rate=1.0, max\_depth=1, random\_state=0)

pag =PassiveAggressiveClassifier(max\_iter=1000, random\_state=0,tol=1e-5)

for classifier in [nb\_clf, sgd, lr, mn, prc,bst,pag]:

clf = OneVsRestClassifier(classifier)

clf.fit(X\_train, y\_train)

y\_predd = clf.predict(X\_test)

print\_score(y\_predd, classifier)

*binarisation.py*

import matplotlib.pyplot as plt

import pandas as pd

import pyodbc

pd.set\_option('display.max\_seq\_items', None)

pd.set\_option('display.max\_colwidth', 500)

pd.set\_option('expand\_frame\_repr', True)

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.decomposition import PCA

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import roc\_auc\_score, confusion\_matrix

from sklearn.metrics import f1\_score

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

pd.set\_option('display.max\_rows', 500)

pd.set\_option('display.max\_columns', 500)

from numpy import mean

from numpy import std

from sklearn.datasets import make\_multilabel\_classification

from sklearn.model\_selection import RepeatedKFold

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import accuracy\_score

def get\_dataset():

X = defaults.drop(columns = ['vie','du','tr','ket' ,'pen','ses','sep' ,'ast'])

X.shape

y = defaults.drop(columns = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay'

,'Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

y.shape

return X, y

def get\_model(n\_inputs, n\_outputs):

model = Sequential()

model.add(Dense(12, input\_dim=n\_inputs, kernel\_initializer='he\_uniform', activation='relu'))

model.add(Dense(8, input\_dim=n\_inputs, kernel\_initializer='he\_uniform', activation='relu'))

model.add(Dense(8, input\_dim=n\_inputs, kernel\_initializer='he\_uniform', activation='relu'))

model.add(Dense(n\_outputs, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam')

return model

def evaluate\_model(X, y):

results = list()

n\_inputs, n\_outputs = X.shape[1], y.shape[1]

print(n\_inputs, n\_outputs)

cv = RepeatedKFold(n\_splits=10, n\_repeats=5, random\_state=1)

for train\_ix, test\_ix in cv.split(X):

X\_train, X\_test = X[train\_ix], X[test\_ix]

y\_train, y\_test = y[train\_ix], y[test\_ix]

print(X\_train, X\_test)

print(y\_train, X\_test)

model = get\_model(n\_inputs, n\_outputs)

model.fit(X\_train, y\_train, verbose=0, epochs=600)

yhat = model.predict(X\_test)

yhat = yhat.round()

acc = accuracy\_score(y\_test, yhat)

print('>%.3f' % acc)

results.append(acc)

return results

X, y = get\_dataset()

X.describe()

y.describe()

dataframe = y

matrix = dataframe.corr()

print("Correlation Matrix is : ")

print(matrix)

corr = dataframe.corr()

corr.style.background\_gradient(cmap='coolwarm')

dataframe = X

matrix = dataframe.corr()

print("Correlation Matrix is : ")

print(matrix)

corr = dataframe.corr()

corr.style.background\_gradient(cmap='coolwarm')

import matplotlib.pyplot as plt

plt.matshow(dataframe.corr())

plt.show()

from keras.models import Sequential

from keras.layers import Dense, Activation

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.datasets import make\_multilabel\_classification

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.losses import binary\_crossentropy

from tensorflow.keras.optimizers import Adam

n\_inputs, n\_outputs = X.shape[1], y.shape[1]

print(n\_inputs, n\_outputs)

model = get\_model(n\_inputs, n\_outputs)

results = evaluate\_model(X.values, y.values)

print('MAE: %.3f (%.3f)' % (mean(results), std(results)))

get\_ipython().system('pip install imblearn ')

from imblearn.pipeline import make\_pipeline

from collections import Counter

pipeline = make\_pipeline( model)

predictions = model.predict(X.values)

print(predictions)

from sklearn.metrics import multilabel\_confusion\_matrix

y\_pred=predictions

print(y\_pred)

y\_pred=y\_pred.round()

print(y\_pred)

y\_test=y

print(y\_test)

cm = multilabel\_confusion\_matrix(y\_test, y\_pred)

print(cm)

from sklearn.metrics import classification\_report

label\_names = ['sensing', 'intuitive', 'visual', 'verbal', 'active', 'reflective', 'sequential', 'global']

print(classification\_report(y.values, predictionsfinalr,target\_names=label\_names))

from sklearn.metrics import hamming\_loss, accuracy\_score

y\_true = y\_test

print (y\_true)

print (y\_pred)

print("Hamming\_loss:", hamming\_loss(y\_true, y\_pred))

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

from sklearn.metrics import multilabel\_confusion\_matrix

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42)

y\_test.reset\_index(drop=True)

model.fit(X\_train.values, y\_train.values, verbose=0, epochs=400)

y\_pred = model.predict(X\_test.values)

y\_pred=y\_pred.round()

print(y\_pred)

print(np.array(y\_test))

cm = multilabel\_confusion\_matrix(np.array(y\_test), y\_pred)

print(cm)

from sklearn import metrics

from sklearn.metrics import recall\_score, f1\_score, precision\_score

recall\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average='weighted')

precision\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average='weighted')

f1\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average='weighted')

f1\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average=None)

recall\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average=None)

precision\_score(y\_true=np.array(y\_test), y\_pred=y\_pred, average=None)

from sklearn.metrics import roc\_curve, auc

fpr = dict()

tpr = dict()

roc\_auc = dict()

print (np.array(y\_test))

for i in range(8):

fpr[i], tpr[i], \_ = roc\_curve( np.array(y\_test)[:, i], y\_pred[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

predictionsfinal = model.predict(X.values)

print(predictionsfinal)

predictionsfinalr=predictionsfinal.round()

recall\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average='weighted')

precision\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average='weighted')

f1\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average='weighted')

recall\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average=None)

precision\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average=None)

f1\_score(y\_true=np.array(y), y\_pred=predictionsfinalr, average=None)

from sklearn.metrics import roc\_curve

roc\_auc\_score(y.values, predictionsfinal, average=None)

from sklearn.metrics import classification\_report

label\_names = ['sensing', 'intuitive', 'visual', 'verbal', 'active', 'reflective', 'sequential', 'global']

print(classification\_report(y.values, predictionsfinalr,target\_names=label\_names))

ging performed on the data:

row = [5, 12, 15, 20, 5, 5, 1, 7, 6, 2, 6, 17]

print(row)

newX = np.asarray([row])

yhat = model.predict(newX)

print('Predicted: %s' % yhat[0])

row = [1, 2, 20, 20, 20, 20, 20, 20, 20, 20, 20, 1]

print(row)

newX = np.asarray([row])

yhat = model.predict(newX)

print('Predicted: %s' % yhat[0])

import shap

import ipywidgets

from ipywidgets import IntProgress

shap.initjs()

explainer = shap.KernelExplainer(model.predict,X)

print(X.shape)

shap\_values = explainer.shap\_values(X)

print (shap\_values )

pd.DataFrame(shap\_values[0])

pd.DataFrame(shap\_values[1])

pd.DataFrame(shap\_values[2])

pd.DataFrame(shap\_values[3])

pd.DataFrame(shap\_values[4])

pd.DataFrame(shap\_values[5])

pd.DataFrame(shap\_values[6])

pd.DataFrame(shap\_values[7])

pd.DataFrame(shap\_values[7][0,:])

pd.DataFrame(shap\_values[7]).head()

# Shap values show how much a given feature changed our prediction (compared to if we made that prediction at some baseline value (pradinė reikšmė) of that feature)

shap.summary\_plot(shap\_values,X\_test,feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay'

,'Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'],class\_names= ['sensing', 'intuitive', 'visual', 'verbal', 'active', 'reflective', 'sequential', 'global'])

#The SHAP value plot can further show the positive and negative relationships of the predictors with the target variable.

shap.summary\_plot(shap\_values, X, feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'],class\_names= ['sensing', 'intuitive', 'visual', 'verbal', 'active', 'reflective', 'sequential', 'global'])

shap.summary\_plot(shap\_values, X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[0], X.values, feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'],class\_names= ['sensing', 'intuitive', 'visual', 'verbal', 'active', 'reflective', 'sequential', 'global'])

shap.summary\_plot(shap\_values[1], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[2], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[3], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[4], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[5], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[6], X.values, feature\_names = X.columns)

shap.summary\_plot(shap\_values[7], X.values, feature\_names = X.columns)

#summary\_plot of a specific class for X\_test

shap.summary\_plot(shap\_values, X.values, feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[0], shap\_values[0], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[1], shap\_values[1], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[2], shap\_values[2], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[2], shap\_values[2], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[3], shap\_values[3], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[4], shap\_values[4], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[5], shap\_values[5], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[6], shap\_values[6], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[7], shap\_values[7], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[0], shap\_values[0][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[1], shap\_values[1][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[2], shap\_values[2][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[3], shap\_values[3][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[4], shap\_values[4][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[5], shap\_values[5][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[6], shap\_values[6][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[6], shap\_values[6][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[7], shap\_values[7][0,:], feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

# make a prediction for new data

row = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,1, 1]

print(row)

newX = np.asarray([row])

yhat = model.predict(newX)

print('Predicted: %s' % yhat[0])

shap\_values = explainer.shap\_values(newX)

print(shap\_values)

pd.DataFrame(shap\_values[0])

pd.DataFrame(shap\_values[1])

pd.DataFrame(shap\_values[2])

pd.DataFrame(shap\_values[3])

pd.DataFrame(shap\_values[4])

pd.DataFrame(shap\_values[5])

pd.DataFrame(shap\_values[6])

pd.DataFrame(shap\_values[7])

shap.force\_plot(explainer.expected\_value[1], shap\_values[1][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

# make a prediction for new data

row = [10, 0, 1, 5, 9, 12, 10, 1, 1, 1,1, 1]

print(row)

newX = np.asarray([row])

yhat = model.predict(newX)

print('Predicted: %s' % yhat[0])

shap\_values = explainer.shap\_values(newX)

pd.DataFrame(shap\_values[0])

pd.DataFrame(shap\_values[1])

shap.force\_plot(explainer.expected\_value[0], shap\_values[0][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[1], shap\_values[1][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[2], shap\_values[2][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[3], shap\_values[3][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[4], shap\_values[4][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[5], shap\_values[5][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[6], shap\_values[6][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])

shap.force\_plot(explainer.expected\_value[7], shap\_values[7][0,:],feature\_names = ['Navigation\_deep','Navigation\_skip\_overview','Forum\_visit','Forum\_post','Video\_pictures' ,'Content\_text\_stay','Feedback\_no','NO\_connections\_links','Quiz\_revisions','Ques\_detail','Ques\_facts','Ques\_concepts'])