```
from google.colab import drive
drive.mount('/content/drive')
%cd ./drive/My Drive
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
/content/drive/My Drive
```

```
import numpy as np
import pandas as pd
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.compat.v1.set_random_seed(42)
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

```
Using TensorFlow backend.
```

Binary Class

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'

# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]

# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)
```

```
# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = 'UCI_HAR_Dataset/'+subset+'/Inertial Signals/'+signal+'_'+subset+'.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'UCI_HAR_Dataset/'+subset+'/y_'+subset+'.txt'
    y = _read_csv(filename)[0]
    y[y<=3] = 0
    y[y>3] = 1

    return pd.get_dummies(y).as_matrix()

def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test

# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:32: FutureWarning: Metho
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:53: FutureWarning: Metho
```

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])

print(timesteps)
print(input_dim)
print(len(X_train))

print(X_train.shape)
```

```
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
128
9
7352
(7352, 128, 9)
(7352, 2)
(2947, 128, 9)
(2947, 2)
```

```
#Training with the best parameters

# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(32, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(2, activation='sigmoid'))
model.summary()

# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Training the model
model.fit(X_train,
          Y_train,
          batch_size=32,
          validation_data=(X_test, Y_test),
          epochs=15)

# Confusion Matrix
score = model.evaluate(X_test, Y_test)
score
```

```
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Train on 7352 samples, validate on 2947 samples
Epoch 1/15
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

7352/7352 [==============================] - 21s 3ms/step - loss: 0.3277 - acc: 0.873
Epoch 2/15
7352/7352 [==============================] - 20s 3ms/step - loss: 0.0671 - acc: 0.983
Epoch 3/15
7352/7352 [==============================] - 20s 3ms/step - loss: 0.0349 - acc: 0.994
Epoch 4/15
7352/7352 [==============================] - 20s 3ms/step - loss: 0.0290 - acc: 0.996
Epoch 5/15
7352/7352 [==============================] - 20s 3ms/step - loss: 0.0176 - acc: 0.997
Epoch 6/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0212 - acc: 0.997
Epoch 7/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0214 - acc: 0.996
Epoch 8/15
7352/7352 [==============================] - 20s 3ms/step - loss: 0.0079 - acc: 0.998
Epoch 9/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0073 - acc: 0.999
Epoch 10/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0089 - acc: 0.999
Epoch 11/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0057 - acc: 0.999
Epoch 12/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0045 - acc: 0.999
Epoch 13/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0054 - acc: 0.999
Epoch 14/15
7352/7352 [==============================] - 21s 3ms/step - loss: 0.0373 - acc: 0.997
Epoch 15/15
7352/7352 [==============================] - 22s 3ms/step - loss: 0.0257 - acc: 0.997
2947/2947 [==============================] - 1s 359us/step
[0.03120530384494252, 0.9888021717000339]
```

```
#save model
model.save('model_binary.h5')
```

Dynamic and Static classes

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'

# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]

# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
```

```
    signals_data = []

    for signal in SIGNALS:
        filename = 'UCI_HAR_Dataset/'+subset+'/Inertial Signals/'+signal+'_'+subset+'.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'UCI_HAR_Dataset/'+subset+'/y_'+subset+'.txt'
    y = _read_csv(filename)[0]

    return y #pd.get_dummies(y).as_matrix()

def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test

# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:32: FutureWarning: Metho
```

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(7352, 128, 9)
(7352,)
(2947, 128, 9)
(2947,)
```

```
a=Y_train.tolist()
b=Y_train.argsort()
b=b.tolist()
print(a.count(1))
```

```
print(a.count(2))
print(a.count(3))
print(a.count(4))
print(a.count(5))
print(a.count(6))

print("------------------------")

dyn=a.count(1)+a.count(2)+a.count(3)
print(dyn)
sta=a.count(4)+a.count(5)+a.count(6)
print(sta)

print("------------------------")

b_dyn=b[0:3285]
b_sta=b[3285:]
print(len(b_dyn))
print(len(b_sta))
```

```
1226
1073
986
1286
1374
1407
------------------------
3285
4067
------------------------
3285
4067
```

```
X_train_dyn=[]
X_train_sta=[]
Y_train_dyn=[]
Y_train_sta=[]

for i in b_dyn :
  m=X_train[i]
  X_train_dyn.append(m)

X_train_dyn=np.asarray(X_train_dyn)



for i in b_sta :
  n=X_train[i]
  X_train_sta.append(n)

X_train_sta=np.asarray(X_train_sta)


for i in b_dyn :
```

```
for i in b_dyn :
  p=Y_train[i]
  Y_train_dyn.append(p)

Y_train_dyn=np.asarray(Y_train_dyn)


for i in b_sta :
  q=Y_train[i]
  Y_train_sta.append(q)

Y_train_sta=np.asarray(Y_train_sta)



a=Y_test.tolist()
b=Y_test.argsort()
b=b.tolist()
print(a.count(1))
print(a.count(2))
print(a.count(3))
print(a.count(4))
print(a.count(5))
print(a.count(6))

print("------------------------")

dyn=a.count(1)+a.count(2)+a.count(3)
print(dyn)
sta=a.count(4)+a.count(5)+a.count(6)
print(sta)

print("------------------------")

b_dyn=b[0:1387]
b_sta=b[1387:]
print(len(b_dyn))
print(len(b_sta))
```

```
496
471
420
491
532
537
------------------------
1387
1560
------------------------
1387
1560
```

```
X_test_dyn=[]
X_test_sta=[]
Y_test_dyn=[]
Y_test_sta=[]
```

```
for i in b_dyn :
  m=X_test[i]
  X_test_dyn.append(m)

X_test_dyn=np.asarray(X_test_dyn)



for i in b_sta :
  n=X_test[i]
  X_test_sta.append(n)

X_test_sta=np.asarray(X_test_sta)


for i in b_dyn :
  p=Y_test[i]
  Y_test_dyn.append(p)

Y_test_dyn=np.asarray(Y_test_dyn)


for i in b_sta :
  q=Y_test[i]
  Y_test_sta.append(q)

Y_test_sta=np.asarray(Y_test_sta)



for n, i in enumerate(Y_train_dyn):
  if i == 1:
    Y_train_dyn[n]=0
  if i == 2:
    Y_train_dyn[n]=1
  if i == 3 :
    Y_train_dyn[n]=2

for n, i in enumerate(Y_test_dyn):
  if i == 1:
    Y_test_dyn[n]=0
  if i == 2:
    Y_test_dyn[n]=1
  if i == 3 :
    Y_test_dyn[n]=2


for n, i in enumerate(Y_train_sta):
  if i == 4:
    Y_train_sta[n]=0
  if i == 5:
    Y_train_sta[n]=1
  if i == 6 :
    Y_train_sta[n]=2
```

```
for n, i in enumerate(Y_test_sta):
  if i == 4:
    Y_test_sta[n]=0
  if i == 5:
    Y_test_sta[n]=1
  if i == 6 :
    Y_test_sta[n]=2

from keras.utils import to_categorical

Y_train_dyn = np.array(Y_train_dyn)
Y_test_dyn = np.array(Y_test_dyn )
Y_train_sta = np.array(Y_train_sta)
Y_test_sta = np.array(Y_test_sta)


# one hot encode
Y_train_dyn = to_categorical(Y_train_dyn)
Y_test_dyn  = to_categorical(Y_test_dyn)
Y_train_sta = to_categorical(Y_train_sta)
Y_test_sta = to_categorical(Y_test_sta )




print(X_train_dyn.shape)
print(Y_train_dyn.shape)

print(X_test_dyn.shape)
print(Y_test_dyn.shape)

print(X_train_sta.shape)
print(Y_train_sta.shape)

print(X_test_sta.shape)
print(Y_test_sta.shape)
```

```
(3285, 128, 9)
(3285, 3)
(1387, 128, 9)
(1387, 3)
(4067, 128, 9)
(4067, 3)
(1560, 128, 9)
(1560, 3)
```

```
#Dynamic class

#Training with the best parameters

# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(64, input_shape=(timesteps, input_dim),return_sequences=True))
# Adding a dropout layer
```

```
# Adding a dropout layer
model.add(Dropout(0.5))
model.add(LSTM(32))
# Adding a dense output layer with sigmoid activation
model.add(Dense(3, activation='sigmoid'))
model.summary()

# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Training the model
model.fit(X_train_dyn,
          Y_train_dyn,
          batch_size=32,
          validation_data=(X_test_dyn, Y_test_dyn),
          epochs=20)

# Confusion Matrix
score = model.evaluate(X_test_dyn, Y_test_dyn)
score
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 128, 64)           18944
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 64)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 32)                12416
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99
=================================================================
Total params: 31,459
Trainable params: 31,459
Non-trainable params: 0
_________________________________________________________________
Train on 3285 samples, validate on 1387 samples
Epoch 1/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.9351 - acc: 0.526
Epoch 2/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.6118 - acc: 0.782
Epoch 3/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.3949 - acc: 0.871
Epoch 4/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.2778 - acc: 0.921
Epoch 5/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.1517 - acc: 0.962
Epoch 6/20
3285/3285 [==============================] - 24s 7ms/step - loss: 0.1097 - acc: 0.973
Epoch 7/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0824 - acc: 0.978
Epoch 8/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0380 - acc: 0.992
Epoch 9/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0619 - acc: 0.983
Epoch 10/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0334 - acc: 0.992
Epoch 11/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0561 - acc: 0.988
Epoch 12/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0290 - acc: 0.993
Epoch 13/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0283 - acc: 0.993
Epoch 14/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0147 - acc: 0.996
Epoch 15/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0178 - acc: 0.996
Epoch 16/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0157 - acc: 0.996
Epoch 17/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0177 - acc: 0.995
Epoch 18/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0145 - acc: 0.995
Epoch 19/20
3285/3285 [==============================] - 23s 7ms/step - loss: 0.0087 - acc: 0.998
Epoch 20/20
3285/3285 [==============================] - 22s 7ms/step - loss: 0.0130 - acc: 0.997
1387/1387 [==============================] - 1s 1ms/step
[0.20508717492719, 0.9733237202595529]
```

```
model.save('model_dynamic.h5')

#Training with the best parameters

# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(64, input_shape=(timesteps, input_dim),return_sequences=True))
# Adding a dropout layer
model.add(Dropout(0.5))
model.add(LSTM(32))
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(3, activation='softmax'))
model.summary()

# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

# Training the model
model.fit(X_train_sta,
          Y_train_sta,
          batch_size=32,
          validation_data=(X_test_sta, Y_test_sta),
          epochs=15)

# Confusion Matrix
score = model.evaluate(X_test_sta, Y_test_sta)
score
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_4 (LSTM)                (None, 128, 64)           18944     
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 64)           0         
_________________________________________________________________
lstm_5 (LSTM)                (None, 32)                12416     
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99        
=================================================================
Total params: 31,459
Trainable params: 31,459
Non-trainable params: 0
_________________________________________________________________
Train on 4067 samples, validate on 1560 samples
Epoch 1/15
4067/4067 [==============================] - 29s 7ms/step - loss: 0.4482 - acc: 0.815
Epoch 2/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.3178 - acc: 0.885
Epoch 3/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.2507 - acc: 0.912
Epoch 4/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.2297 - acc: 0.917
Epoch 5/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.2243 - acc: 0.917
Epoch 6/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.2494 - acc: 0.904
Epoch 7/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.2137 - acc: 0.917
Epoch 8/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.2086 - acc: 0.914
Epoch 9/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.2747 - acc: 0.910
Epoch 10/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.2041 - acc: 0.919
Epoch 11/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.1998 - acc: 0.918
Epoch 12/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.1967 - acc: 0.917
Epoch 13/15
4067/4067 [==============================] - 28s 7ms/step - loss: 0.1965 - acc: 0.918
Epoch 14/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.2130 - acc: 0.916
Epoch 15/15
4067/4067 [==============================] - 27s 7ms/step - loss: 0.1990 - acc: 0.922
1560/1560 [==============================] - 2s 1ms/step
[0.35382931890365005, 0.8826923076923077]
```

```
model.save('model_static.h5')
```

Prediction

```
from keras.models import load_model
import pickle
model_binary = load_model('model_binary.h5')
model_dynamic = load_model('model_dynamic.h5')
model_static = load_model('model_static.h5')


# Data directory
DATADIR = 'UCI_HAR_Dataset'

# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]

# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = 'UCI_HAR_Dataset/'+subset+'/Inertial Signals/'+signal+'_'+subset+'.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'UCI_HAR_Dataset/'+subset+'/y_'+subset+'.txt'
```

```
    filename = 'UCI_HAR_Dataset/'+subset+'/y_'+subset+'.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()


def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test

# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:32: FutureWarning: Metho
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:51: FutureWarning: Metho
```

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(7352, 128, 9)
(7352, 6)
(2947, 128, 9)
(2947, 6)
```

```
#predict binary
a=model_binary.predict_classes(X_test)
b=np.argsort(a)
a=a.tolist()
print(a.count(0))
print(a.count(1))
```

```
1408
1539
```

```
#separate dynamic and static classes
dyn=b[0:1408]
sta=b[1408:]

print(dyn.shape)
print(sta.shape)
```

```
(1408,)
(1539,)
```

```
x_test_dyn=[]
```

```
for i in dyn :
    a=X_test[i]
    x_test_dyn.append(a)
x_test_dyn=np.asarray(x_test_dyn)

x_test_sta=[]

for i in sta :
    a=X_test[i]
    x_test_sta.append(a)
x_test_sta=np.asarray(x_test_sta)


dyn_pre=model_dynamic.predict_classes(x_test_dyn)
sta_pre=model_static.predict_classes(x_test_sta)


a=sta_pre
a=a.tolist()
print(a.count(0))
print(a.count(1))
print(a.count(2))
```

```
406
613
520
```

```
for n, i in enumerate(a):
    if i == 0:
        a[n] = 3
    if i == 1:
        a[n]=4
    if i == 2 :
        a[n]=5


print(a.count(3))
print(a.count(4))
print(a.count(5))
```

```
406
613
520
```

```
sta_pre=a
sta_pre=np.asarray(sta_pre)


print(type(dyn))
print(type(sta))
print(type(dyn_pre))
print(type(sta_pre))

print(dyn.shape)
print(sta.shape)
print(dyn_pre.shape)
```

```
print(sta_pre.shape)
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(1408,)
(1539,)
(1408,)
(1539,)
```

```
dyn_zip=list(zip(dyn, dyn_pre))
sta_zip=list(zip(sta,sta_pre))

d_s=dyn_zip+sta_zip


y_pred=[]

for i in range (0,2947):
    y_pred.append(i)


for i in d_s:
    a=i[0]
    b=i[1]
    y_pred[a]=b


from keras.utils import to_categorical

y_pred = np.array(y_pred)
print(y_pred)
# one hot encode
y_pred = to_categorical(y_pred)
print(y_pred)
```

```
[4 4 4 ... 1 1 1]
[[0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0.]
 ...
 [0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]]
```

```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test, y_pred)
```

```
0.9141499830335935
```