

Solutions 4

Jumping Rivers

Predict a person based on accelerometer data from walking

We have accelerometer data on 15 individuals who all walked for a period of time. Each observation is a set of values from the x,y and z axis of an accelerometer in 5 seconds sampled at a frequency of 52Hz. One sample is 260 observations. We have between 300-600 observations on each individual

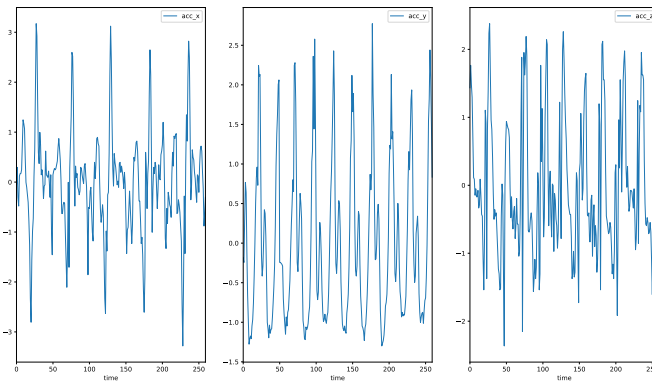
```
import jrpytensorflow
```

```
walking = jrpytensorflow.datasets.load_walking()
```

The following code will produce a visualisation of one sample

```
import matplotlib.pyplot as plt
import numpy as np
```

```
sub = walking[walking['sample'] == 400]
sub['time'] = np.arange(260)
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 10))
sub.plot(x='time', y='acc_x', ax=ax1)
sub.plot(x='time', y='acc_y', ax=ax2)
sub.plot(x='time', y='acc_z', ax=ax3)
plt.show()
```



At present this data is not in a particularly convenient structure for training my model. The following code will create the (n,260,3) (n observations of 260 inputs for 3 channels) input shape and class labels as separate array objects

```

dims = ['acc_x', 'acc_y', 'acc_z']
x = np.dstack([walking[[d]].values.reshape(-1, 260) for d in dims])
y = walking['person'].values[:,260] - 1

```

Each observation is a sequence of 260 values with 3 channels. This is the sort of data structure where we would use a convolutional neural network with 1d convolutions.

- Create a model structure that takes in the 260 input features for each of 3 data channels and returns 15 output features (one for each person). We will want a set of convolution and pooling layers to begin with before having some linear layers to get to the final output. The convolutions and pooling extract features from the 3 channels of sequences, the linear layers then map those features to the final output.

```

import tensorflow as tf

def convModel():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv1D(
            40, 30, strides=2,
            activation='relu', input_shape=(260, 3)
        ),

        tf.keras.layers.Conv1D(
            40, 10, activation='relu'
        ),

        tf.keras.layers.MaxPooling1D(2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(100, activation='relu'),
        tf.keras.layers.Dense(15, activation='sigmoid')
    ])
    return model

```

- Partition your data into training and test sets and binarize the labels (Optional: partition data into a validation set as well)

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

X_train, X_test, y_train, y_test = train_test_split(
    x, y, shuffle=True, test_size=0.2

```

```

)
prep = LabelBinarizer()
y_train_bin = prep.fit_transform(y_train)
y_test_bin = prep.transform(y_test)

```

- Compile and train your model

```

model = convModel()
model.compile(
    optimizer='Adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

hist = model.fit(
    X_train, y_train_bin, epochs=5,
    validation_data=[X_test, y_test_bin]
)

```

- Using the `.history` object, can you plot how the loss and accuracy changes over the training time?

```
import pandas as pd
```

```

acc = pd.Series(hist.history['accuracy'])
acc.plot()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()

```

- what is the loss and accuracy in your test set?

```
loss, acc = model.evaluate(X_test, y_test_bin, verbose=2)
```

- Try to view the model architecture in TensorBoard

```

%load_ext tensorboard
from tensorflow import keras

```

```

tensorBoardCallback = keras.callbacks.TensorBoard(
    log_dir="logs/fit/",
    histogram_freq=1
)

```

```
model = convModel()
```

```
model.compile(  
    optimizer='Adam',  
    loss='categorical_crossentropy',  
    metrics=['accuracy']  
)  
  
model.fit(  
    X_train, y_train_bin, epochs=5,  
    validation_data=(X_test, y_test_bin),  
    callbacks = [tensorBoardCallback]  
)
```

For me this gives 95% plus accuracy on classifying a person purely on the accelerometer data while walking. Pretty neat!