data_exploration

December 13, 2017

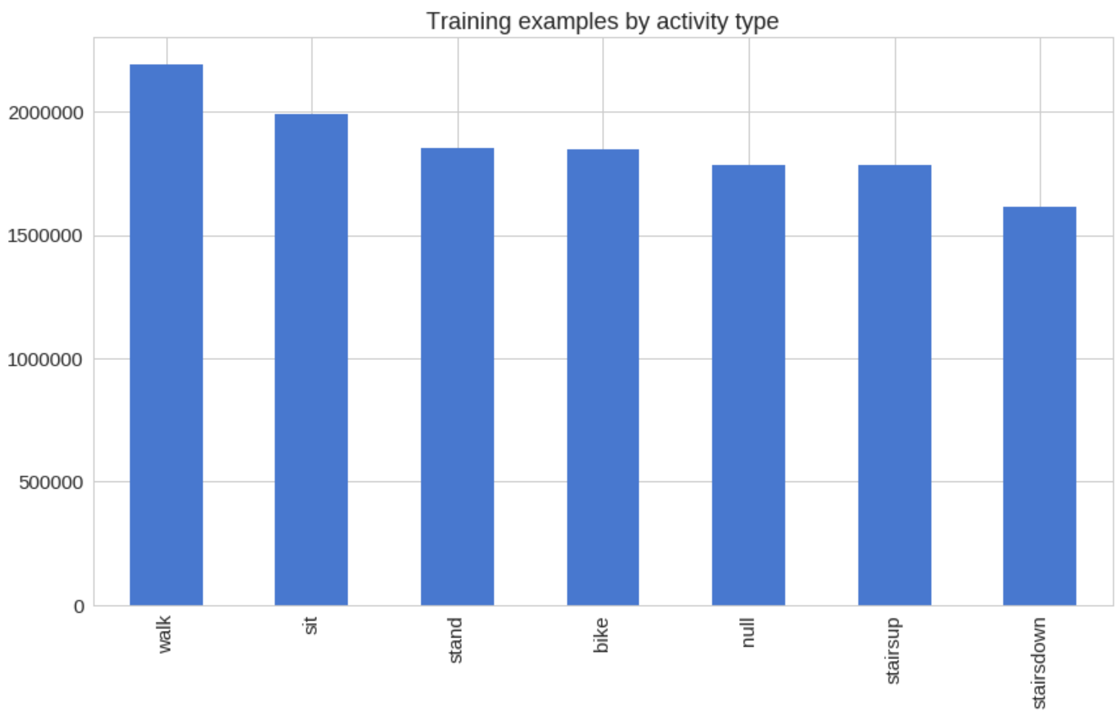
```
In [1]: import pandas as pd
        import numpy as np
        import pickle
        import matplotlib.pyplot as plt
        from scipy import stats
        import tensorflow as tf
        import seaborn as sns
        from pylab import rcParams
        from sklearn import metrics
        from sklearn.model_selection import train_test_split
        %matplotlib inline
        sns.set(style='whitegrid', palette='muted', font_scale=1.5)
        rcParams['figure.figsize'] = 14, 8
        RANDOM_SEED = 42
```

```
In [2]: df = pd.read_csv("../data/activity_data/Phones_accelerometer.csv")
        df.head()
```

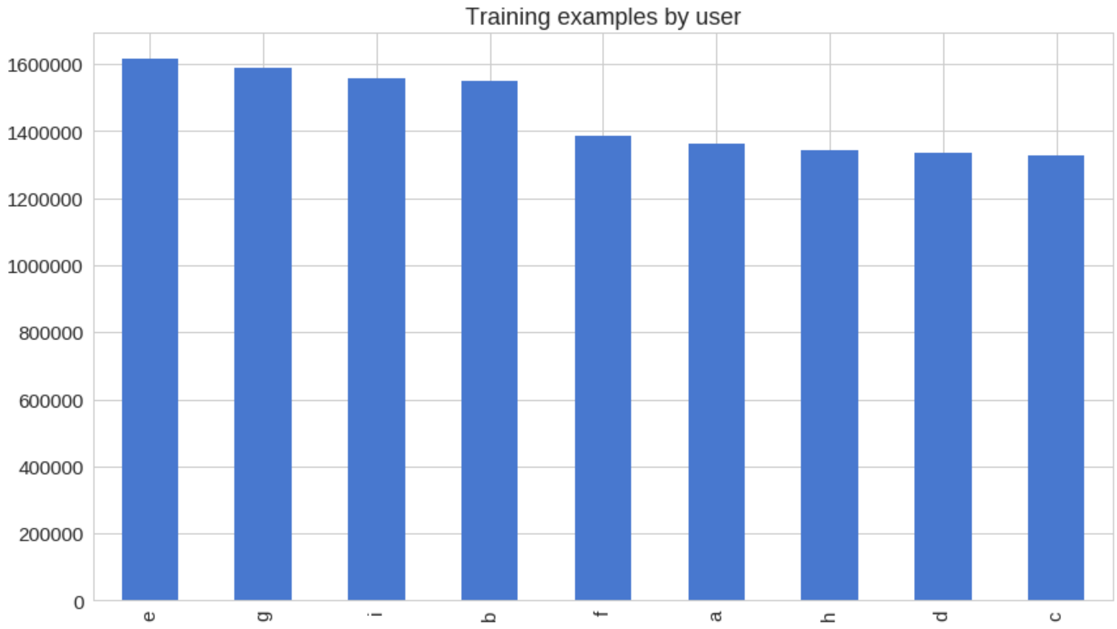
```
Out[2]:    Index  Arrival_Time        Creation_Time         x         y         z  \
        0      0  1424696633908  1424696631913248572 -5.958191  0.688065  8.135345
        1      1  1424696633909  1424696631918283972 -5.952240  0.670212  8.136536
        2      2  1424696633918  1424696631923288855 -5.995087  0.653549  8.204376
        3      3  1424696633919  1424696631928385290 -5.942718  0.676163  8.128204
        4      4  1424696633929  1424696631933420691 -5.991516  0.641647  8.135345

          User   Model    Device     gt
        0    a  nexus4  nexus4_1  stand
        1    a  nexus4  nexus4_1  stand
        2    a  nexus4  nexus4_1  stand
        3    a  nexus4  nexus4_1  stand
        4    a  nexus4  nexus4_1  stand
```

```
In [3]: df['gt'].value_counts().plot(kind='bar', title='Training examples by activity type');
```

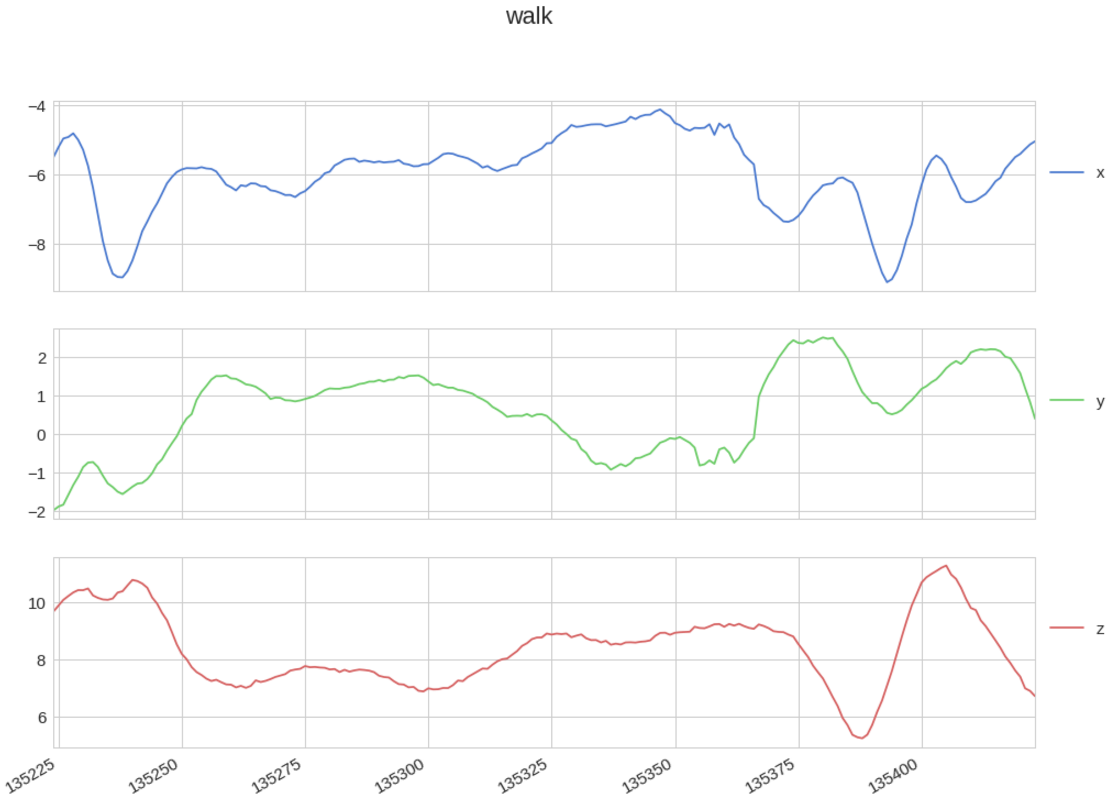


```
In [4]: df['User'].value_counts().plot(kind='bar', title='Training examples by user');
```

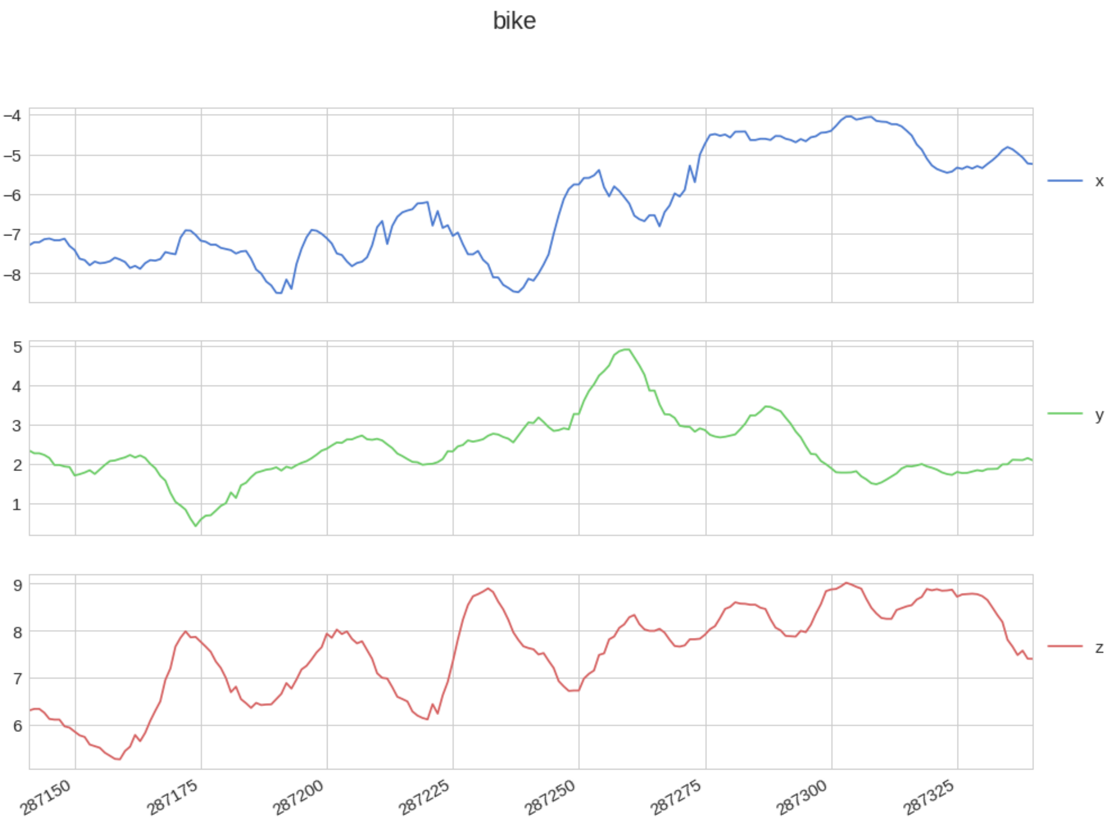


```
In [5]: def plot_activity(activity, df):
            data = df[df['gt'] == activity][['x', 'y', 'z']][:200]
```

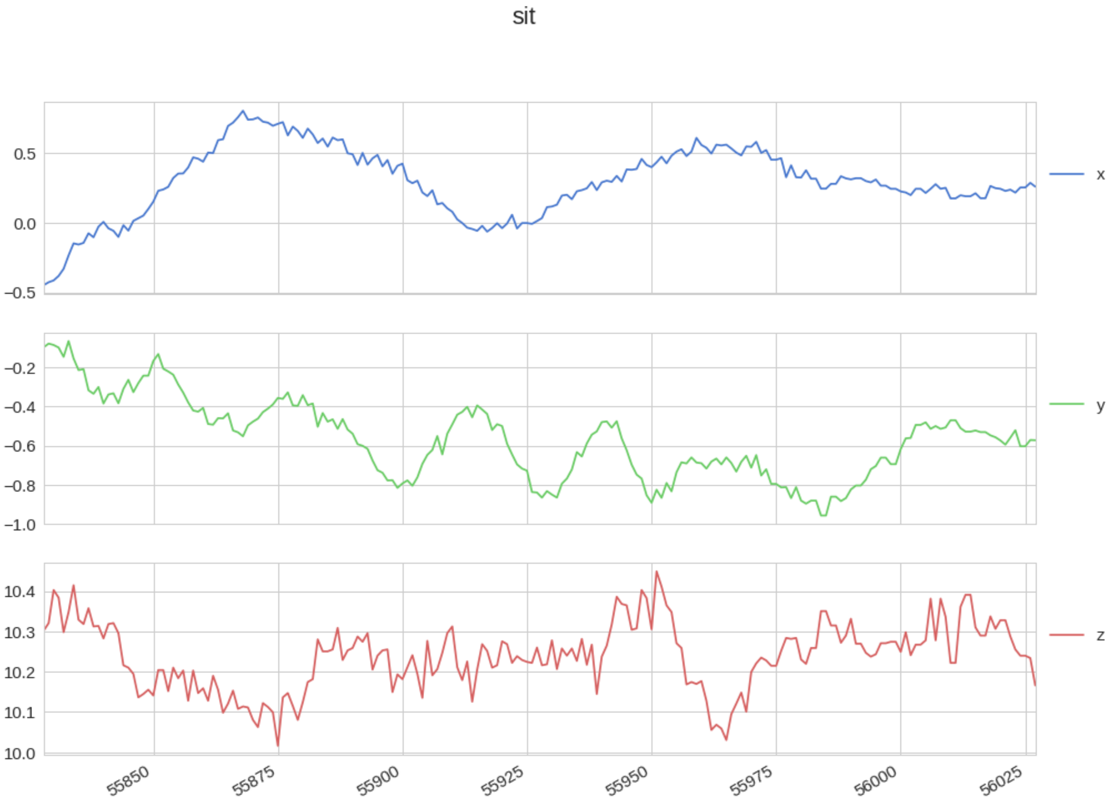
```
    axis = data.plot(subplots=True, figsize=(16, 12),
                     title=activity)
    for ax in axis:
        ax.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5))
plot_activity("walk", df)
```



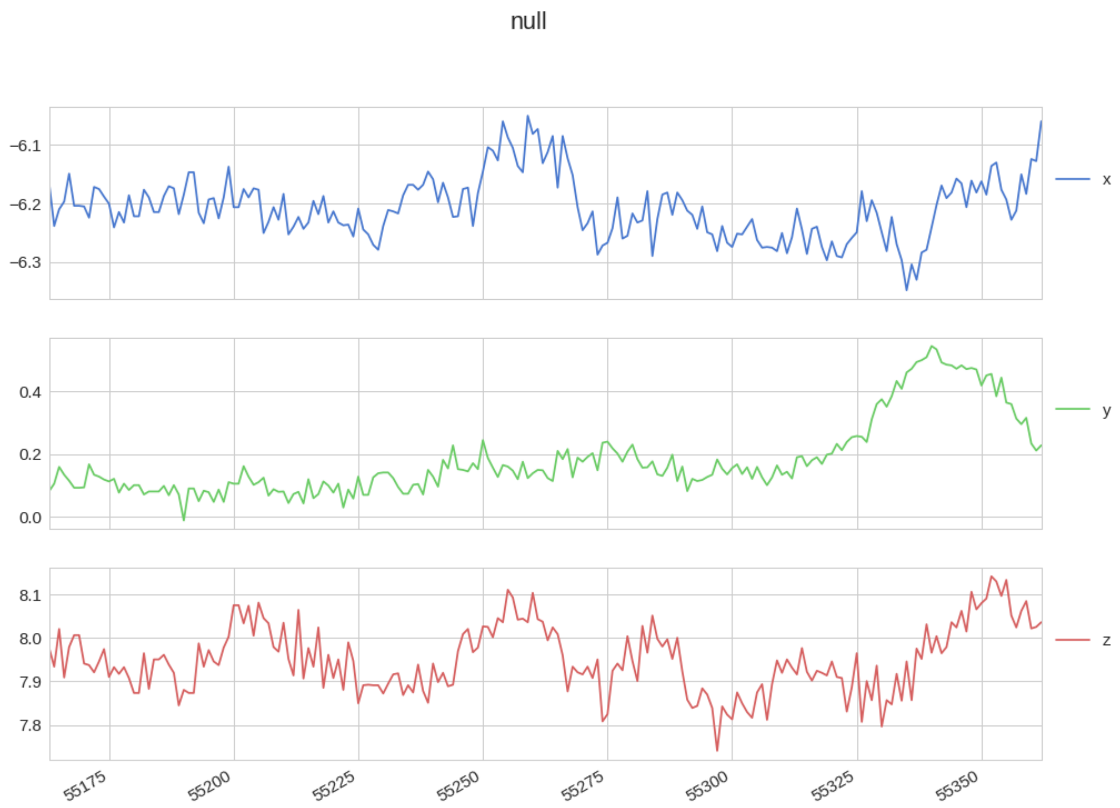
```
In [6]: plot_activity("bike", df)
```



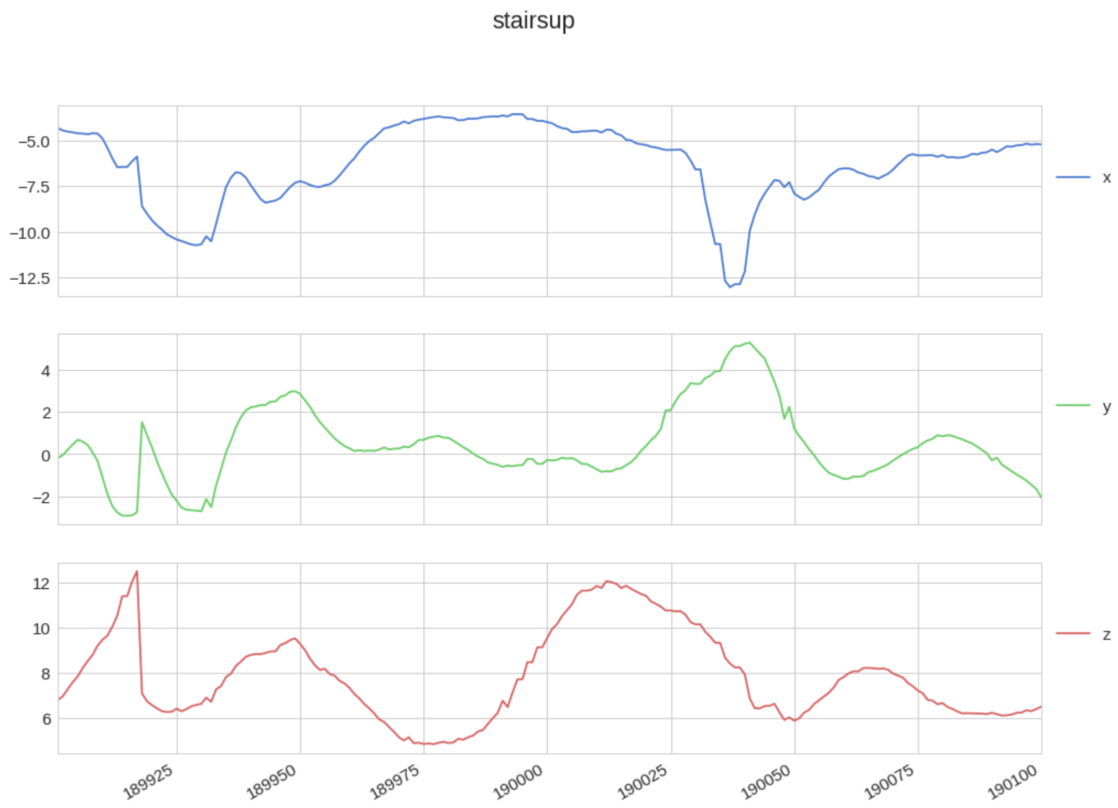
```
In [7]: plot_activity("sit", df)
```



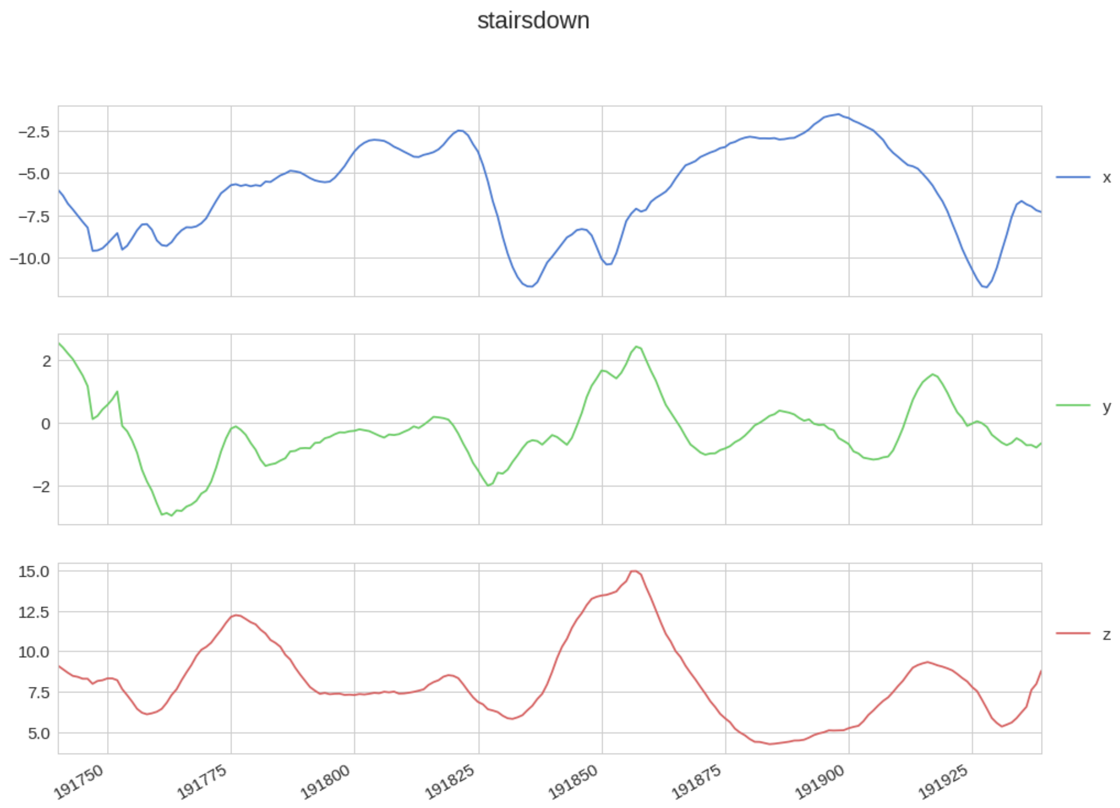
```
In [8]: plot_activity("null", df)
```



```
In [9]: plot_activity("stairsup", df)
```



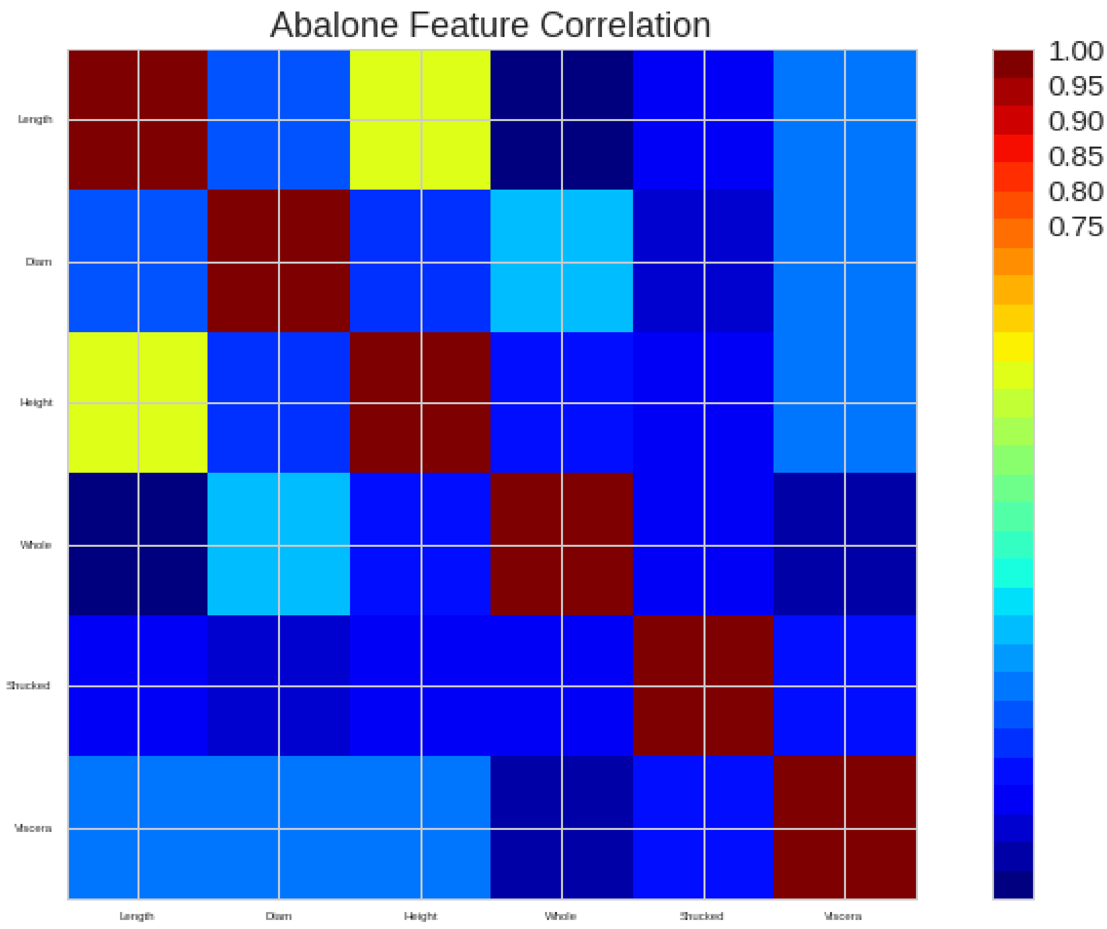
```
In [10]: plot_activity("stairsdown", df)
```



```
In [11]: def correlation_matrix(df):
             from matplotlib import pyplot as plt
             from matplotlib import cm as cm

             fig = plt.figure()
             ax1 = fig.add_subplot(111)
             cmap = cm.get_cmap('jet', 30)
             cax = ax1.imshow(df.corr(), interpolation="nearest", cmap=cmap)
             ax1.grid(True)
             plt.title('Abalone Feature Correlation')
             labels=['Sex','Length','Diam','Height','Whole','Shucked','Viscera','Shell','Rings',
             ax1.set_xticklabels(labels,fontsize=6)
             ax1.set_yticklabels(labels,fontsize=6)
             # Add colorbar, make sure to specify tick locations to match desired ticklabels
             fig.colorbar(cax, ticks=[.75,.8,.85,.90,.95,1])
             plt.show()

         correlation_matrix(df)
```



```
In [12]: N_TIME_STEPS = 200
         N_FEATURES = 3
         step = 200
         segments = []
         labels = []
         for i in range(0, len(df) - N_TIME_STEPS, step):
             xs = df['x'].values[i: i + N_TIME_STEPS]
             ys = df['y'].values[i: i + N_TIME_STEPS]
             zs = df['z'].values[i: i + N_TIME_STEPS]
             label = stats.mode(df['gt'][i: i + N_TIME_STEPS])[0][0]
             segments.append([xs, ys, zs])
             labels.append(label)
```

```
/home/limmen/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:253: RuntimeWarning: The
  "values. nan values will be ignored.", RuntimeWarning)
```

```
In [13]: np.array(segments).shape
```

```
Out[13]: (65312, 3, 200)
```

```
In [14]: reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, N_TIME_STEPS, N
         labels = np.asarray(pd.get_dummies(labels), dtype = np.float32)
```

```
In [15]: reshaped_segments.shape
```

```
Out[15]: (65312, 200, 3)
```

```
In [16]: labels[0]
```

```
Out[16]: array([ 0.,  0.,  0.,  0.,  0.,  1.,  0.], dtype=float32)
```

```
In [17]: X_train, X_test, y_train, y_test = train_test_split(
                 reshaped_segments, labels, test_size=0.2, random_state=RANDOM_SEED)
```

```
In [18]: N_CLASSES = 7
         N_HIDDEN_UNITS = 64
         def create_LSTM_model(inputs):
             W = {
                 'hidden': tf.Variable(tf.random_normal([N_FEATURES, N_HIDDEN_UNITS])),
                 'output': tf.Variable(tf.random_normal([N_HIDDEN_UNITS, N_CLASSES]))
             }
             biases = {
                 'hidden': tf.Variable(tf.random_normal([N_HIDDEN_UNITS], mean=1.0)),
                 'output': tf.Variable(tf.random_normal([N_CLASSES]))
             }

             X = tf.transpose(inputs, [1, 0, 2])
             X = tf.reshape(X, [-1, N_FEATURES])
             hidden = tf.nn.relu(tf.matmul(X, W['hidden']) + biases['hidden'])
             hidden = tf.split(hidden, N_TIME_STEPS, 0)

             # Stack 2 LSTM layers
             lstm_layers = [tf.contrib.rnn.BasicLSTMCell(N_HIDDEN_UNITS, forget_bias=1.0) for _
             lstm_layers = tf.contrib.rnn.MultiRNNCell(lstm_layers)

             outputs, _ = tf.contrib.rnn.static_rnn(lstm_layers, hidden, dtype=tf.float32)

             # Get output for the last time step
             lstm_last_output = outputs[-1]

             return tf.matmul(lstm_last_output, W['output']) + biases['output']
```

```
In [19]: tf.reset_default_graph()

         X = tf.placeholder(tf.float32, [None, N_TIME_STEPS, N_FEATURES], name="input")
         Y = tf.placeholder(tf.float32, [None, N_CLASSES])
```

In [20]:
```
tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, N_TIME_STEPS, N_FEATURES], name="input")
Y = tf.placeholder(tf.float32, [None, N_CLASSES])
```

In [21]:
```
pred_Y = create_LSTM_model(X)

pred_softmax = tf.nn.softmax(pred_Y, name="y_")
```

In [22]:
```
L2_LOSS = 0.0015

l2 = L2_LOSS * \
    sum(tf.nn.l2_loss(tf_var) for tf_var in tf.trainable_variables())

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = pred_Y, labels =
```

In [23]:
```
LEARNING_RATE = 0.0025

optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE).minimize(loss)

correct_pred = tf.equal(tf.argmax(pred_softmax, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, dtype=tf.float32))
```

In [24]:
```
N_EPOCHS = 50
BATCH_SIZE = 1024
saver = tf.train.Saver()

history = dict(train_loss=[],
                     train_acc=[],
                     test_loss=[],
                     test_acc=[])

sess=tf.InteractiveSession()
sess.run(tf.global_variables_initializer())

train_count = len(X_train)

for i in range(1, N_EPOCHS + 1):
    for start, end in zip(range(0, train_count, BATCH_SIZE),
                          range(BATCH_SIZE, train_count + 1,BATCH_SIZE)):
        sess.run(optimizer, feed_dict={X: X_train[start:end],
                                       Y: y_train[start:end]})

    _, acc_train, loss_train = sess.run([pred_softmax, accuracy, loss], feed_dict={
                                            X: X_train, Y: y_train})

    _, acc_test, loss_test = sess.run([pred_softmax, accuracy, loss], feed_dict={
                                            X: X_test, Y: y_test})
```

```
             history['train_loss'].append(loss_train)
             history['train_acc'].append(acc_train)
             history['test_loss'].append(loss_test)
             history['test_acc'].append(acc_test)

             if i != 1 and i % 10 != 0:
                 continue

             print(f'epoch: {i} test accuracy: {acc_test} loss: {loss_test}')

         predictions, acc_final, loss_final = sess.run([pred_softmax, accuracy, loss], feed_dict

         print()
         print(f'final results: accuracy: {acc_final} loss: {loss_final}')
```

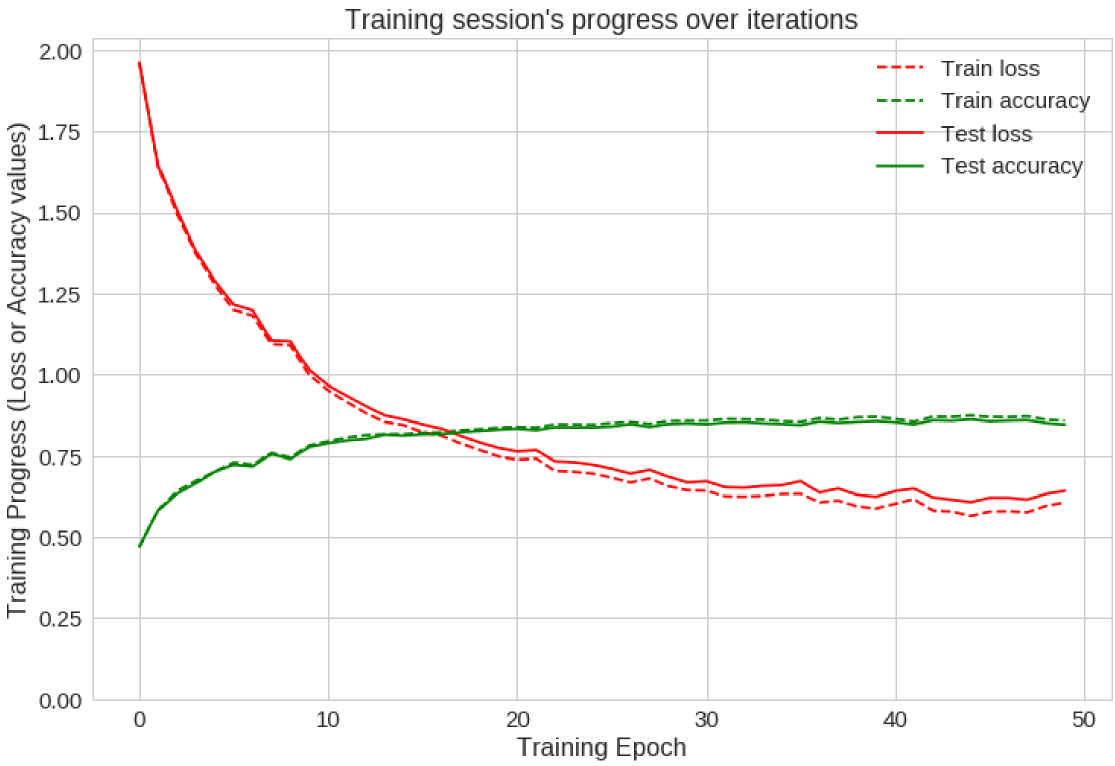
```
epoch: 1 test accuracy: 0.47133123874664307 loss: 1.961308479309082
epoch: 10 test accuracy: 0.7769271731376648 loss: 1.0142518281936646
epoch: 20 test accuracy: 0.8307433128356934 loss: 0.7741105556488037
epoch: 30 test accuracy: 0.8497282266616821 loss: 0.668204665184021
epoch: 40 test accuracy: 0.8571537733078003 loss: 0.6235238313674927
epoch: 50 test accuracy: 0.8459771871566772 loss: 0.6423799991607666

final results: accuracy: 0.8459771871566772 loss: 0.6423799991607666
```

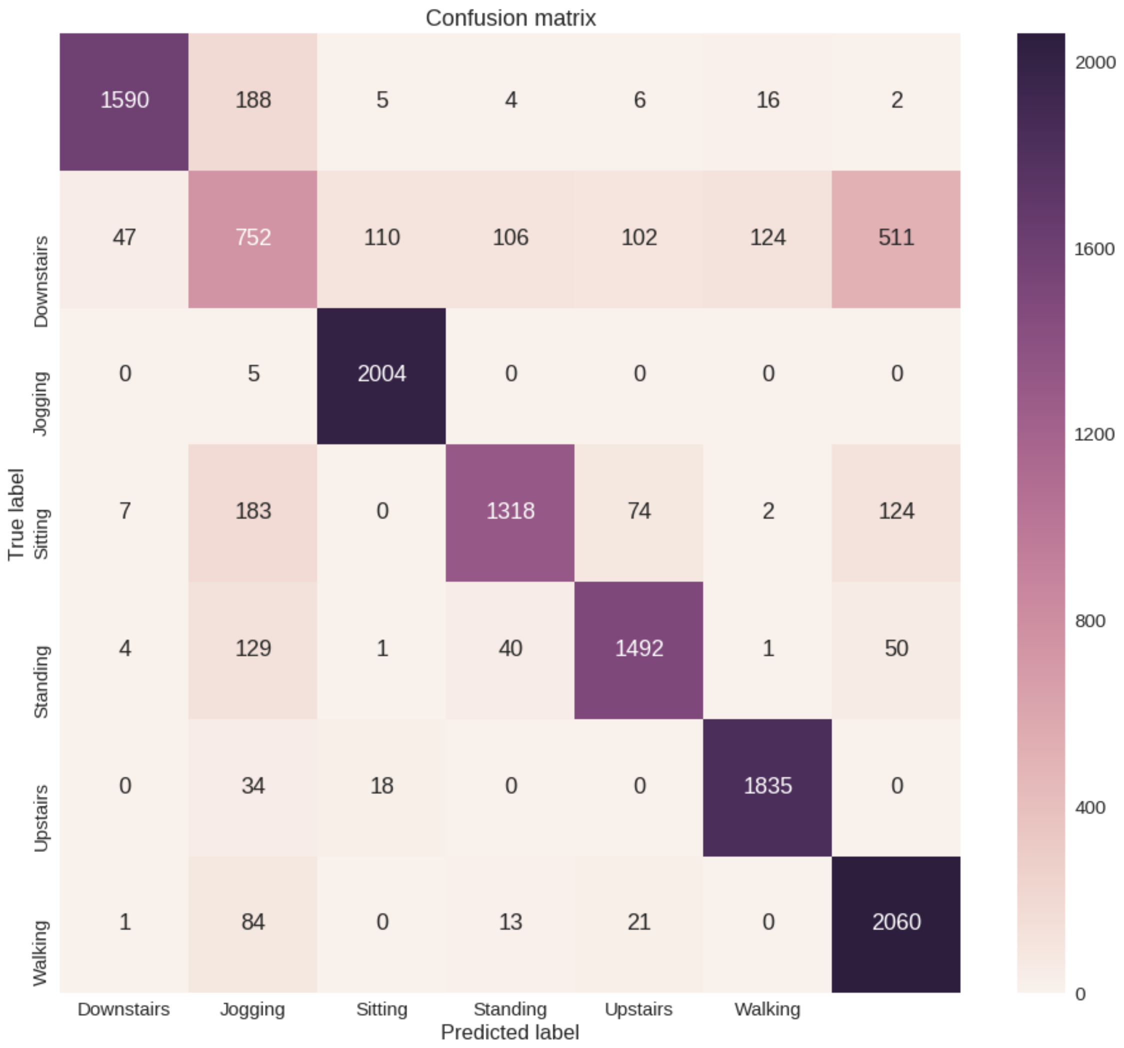
```
In [26]: pickle.dump(predictions, open("predictions.p", "wb"))
         pickle.dump(history, open("history.p", "wb"))
         tf.train.write_graph(sess.graph_def, '.', './checkpoint/har.pbtxt')
         saver.save(sess, save_path = "./checkpoint/har.ckpt")
         sess.close()
```

```
In [27]: history = pickle.load(open("history.p", "rb"))
         predictions = pickle.load(open("predictions.p", "rb"))
```

```
In [28]: plt.figure(figsize=(12, 8))
         plt.plot(np.array(history['train_loss']), "r--", label="Train loss")
         plt.plot(np.array(history['train_acc']), "g--", label="Train accuracy")
         plt.plot(np.array(history['test_loss']), "r-", label="Test loss")
         plt.plot(np.array(history['test_acc']), "g-", label="Test accuracy")
         plt.title("Training session's progress over iterations")
         plt.legend(loc='upper right', shadow=True)
         plt.ylabel('Training Progress (Loss or Accuracy values)')
         plt.xlabel('Training Epoch')
         plt.ylim(0)
         plt.show()
```



```
In [29]: LABELS = ['Downstairs', 'Jogging', 'Sitting', 'Standing', 'Upstairs', 'Walking']
         max_test = np.argmax(y_test, axis=1)
         max_predictions = np.argmax(predictions, axis=1)
         confusion_matrix = metrics.confusion_matrix(max_test, max_predictions)
         plt.figure(figsize=(16, 14))
         sns.heatmap(confusion_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="
         plt.title("Confusion matrix")
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
         plt.show();
```



```
In [30]: from tensorflow.python.tools import freeze_graph

         MODEL_NAME = 'har'

         input_graph_path = 'checkpoint/' + MODEL_NAME+'.pbtxt'
         checkpoint_path = './checkpoint/' +MODEL_NAME+'.ckpt'
         restore_op_name = "save/restore_all"
         filename_tensor_name = "save/Const:0"
         output_frozen_graph_name = 'frozen_'+MODEL_NAME+'.pb'

         freeze_graph.freeze_graph(input_graph_path, input_saver="",
                                   input_binary=False, input_checkpoint=checkpoint_path,
                                   output_node_names="y_", restore_op_name="save/restore_all",
                                   filename_tensor_name="save/Const:0",
                                   output_graph=output_frozen_graph_name, clear_devices=True, in
```

```
INFO:tensorflow:Restoring parameters from ./checkpoint/har.ckpt
INFO:tensorflow:Froze 8 variables.
Converted 8 variables to const ops.
6862 ops in the final graph.
```

```
In [ ]:
```