homework4

October 1, 2019

NAME: Zhi Li
SECTION #: 11352395
CS 5970: Machine Learning Practices

1 Homework 4: Linear Regression

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code.
For all plots, make sure all necessary axes and curves are clearly and accurately labeled. Include figure/plot titles appropriately as well.

1.1.1 Task

For this assignment you will work with different training set sizes, constructing regression models from these sets, and evaluating the training and test performance of these models. Additionally, it is good practice to have a high level understanding of the data one is working with, thus upon loading the data, general information is also displayed and/or plotted.

1.1.2 Data set

The BMI (Brain Machine Interface) data consists of several files prefixed with 'MI', 'theta', 'dtheta', 'torque', or 'time'.
* *MI* files contain data with the number of activations for 48 neurons, at mutliple time points, for a single fold. There are 20 folds (20 files), where each fold consists of over 1000 times points (the rows). At each time point, we record the number of activations for each neuron for 20 bins. Therefore, each time point has 48 * 20 = 960 columns.
* *theta* files record the angular position of the shoulder (in column 0) and the elbow (in column 1) for each time point.
* *dtheta* files record the angular velocity of the shoulder (in column 0) and the elbow (in column 1) for each time point.
* *torque* files record the torque of the shoulder (in column 0) and the elbow (in column 1) for each time point.
* *time* files record the actual time stamp of each time point.

A fold is essentially a subset of data, useful for adjusting training, validation, and test sets sizes, to access the generality of a model.

This assignment utilizes code examples and concepts from the lectures on regression

1.1.3 Objectives

- Understand the impact of the training set size
- Linear Regression
  - Prediction
  - Multiple Regression
  - Performance Evaluation
- Do not save work within the ml_practices folder
  - create a folder in your home directory for assignments, and copy the templates there

1.1.4 General References

- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Summary of matplotlib
- Pandas DataFrames
- Sci-kit Learn Linear Models
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Leatn Model Selection
- Torque

[1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import os, re, fnmatch
import matplotlib.pyplot as plt
import matplotlib.patheffects as peffects

from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR

FIGWIDTH = 5
FIGHEIGHT = 5
FONTSIZE = 12
```

```
plt.rcParams['figure.figsize'] = (FIGWIDTH, FIGHEIGHT)
plt.rcParams['font.size'] = FONTSIZE

plt.rcParams['xtick.labelsize'] = FONTSIZE
plt.rcParams['ytick.labelsize'] = FONTSIZE

%matplotlib inline
```

2 LOAD DATA

[2]:
```
""" PROVIDED """
def read_bmi_file_set(directory, filebase):
    '''
    Read a set of CSV files and append them together
    :param directory: The directory in which to scan for the CSV files
    :param filebase: A file specification that potentially includes wildcards
    :returns: A list of Numpy arrays (one for each fold)
    '''

    # The set of files in the directory
    files = fnmatch.filter(os.listdir(directory), filebase)
    files.sort()

    # Create a list of Pandas objects; each from a file in the directory that␣
↪matches filebase
    lst = [pd.read_csv(directory + "/" + file, delim_whitespace=True).values␣
↪for file in files]

    # Concatenate the Pandas objects together.  ignore_index is critical here␣
↪so that
    # the duplicate row indices are addressed
    return lst
```

[3]:
```
""" TODO
Load the BMI data from all the folds, using read_bmi_file_set()
"""
dir_name = 'ml_practices/imports/datasets/bmi/DAT6_08/'
# TODO: finish loading the MI data folds
MI_folds = read_bmi_file_set(dir_name, 'MI_fold*')# TODO
theta_folds = read_bmi_file_set(dir_name, 'theta_fold*')
dtheta_folds = read_bmi_file_set(dir_name, 'dtheta_fold*')
torque_folds = read_bmi_file_set(dir_name, 'torque_fold*')
time_folds = read_bmi_file_set(dir_name, 'time_fold*')
```

```
alldata_folds = zip(MI_folds, theta_folds, dtheta_folds, torque_folds,␣
 ↪time_folds)

nfolds = len(MI_folds)
nfolds
```

[3]: 20

[4]:
```
""" TODO
Print out the shape of all the data for each fold
"""
# TODO: finish by including shape of time data
for i, (MI, theta, dtheta, torque, time) in enumerate(alldata_folds):
    print("FOLD %2d " % i, MI.shape, theta.shape,
          dtheta.shape, torque.shape, time.shape) # TODO
```

```
FOLD  0  (1193, 960) (1193, 2) (1193, 2) (1193, 2) (1193, 1)
FOLD  1  (1104, 960) (1104, 2) (1104, 2) (1104, 2) (1104, 1)
FOLD  2  (1531, 960) (1531, 2) (1531, 2) (1531, 2) (1531, 1)
FOLD  3  (1265, 960) (1265, 2) (1265, 2) (1265, 2) (1265, 1)
FOLD  4  (1498, 960) (1498, 2) (1498, 2) (1498, 2) (1498, 1)
FOLD  5  (1252, 960) (1252, 2) (1252, 2) (1252, 2) (1252, 1)
FOLD  6  (1375, 960) (1375, 2) (1375, 2) (1375, 2) (1375, 1)
FOLD  7  (1130, 960) (1130, 2) (1130, 2) (1130, 2) (1130, 1)
FOLD  8  (1247, 960) (1247, 2) (1247, 2) (1247, 2) (1247, 1)
FOLD  9  (1257, 960) (1257, 2) (1257, 2) (1257, 2) (1257, 1)
FOLD 10  (1265, 960) (1265, 2) (1265, 2) (1265, 2) (1265, 1)
FOLD 11  (1146, 960) (1146, 2) (1146, 2) (1146, 2) (1146, 1)
FOLD 12  (1225, 960) (1225, 2) (1225, 2) (1225, 2) (1225, 1)
FOLD 13  (1238, 960) (1238, 2) (1238, 2) (1238, 2) (1238, 1)
FOLD 14  (1570, 960) (1570, 2) (1570, 2) (1570, 2) (1570, 1)
FOLD 15  (1359, 960) (1359, 2) (1359, 2) (1359, 2) (1359, 1)
FOLD 16  (1579, 960) (1579, 2) (1579, 2) (1579, 2) (1579, 1)
FOLD 17  (1364, 960) (1364, 2) (1364, 2) (1364, 2) (1364, 1)
FOLD 18  (1389, 960) (1389, 2) (1389, 2) (1389, 2) (1389, 1)
FOLD 19  (1289, 960) (1289, 2) (1289, 2) (1289, 2) (1289, 1)
```

[5]:
```
""" PROVIDED
Print out the first few rows and columns of the MI data
for a few folds
"""
for i, MI in enumerate(MI_folds[:3]):
    print("FOLD %2d" % i)
    print(MI[:5,:20])
```

```
FOLD  0
[[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
FOLD  1
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]]
FOLD  2
[[0 0 0 0 0 1 0 1 2 1 0 0 1 0 0 0 1 2 0 0]
 [0 0 0 0 1 0 1 2 1 0 0 1 0 0 0 1 2 0 0 0]
 [0 0 0 1 0 1 2 1 0 0 1 0 0 0 1 2 0 0 0 0]
 [0 0 1 0 1 2 1 0 0 1 0 0 0 1 2 0 0 0 0 0]
 [0 1 0 1 2 1 0 0 1 0 0 0 1 2 0 0 0 0 0 0]]
```

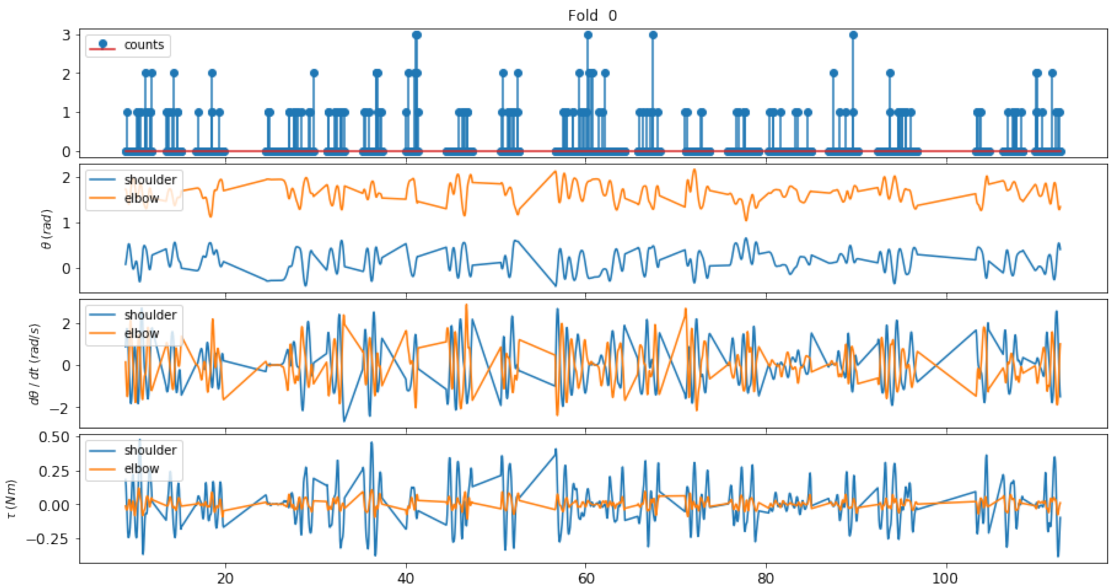
[6]:
```python
""" TODO
Check the data for any NaNs
"""
def anynans(X):
    return np.isnan(X).any()

alldata_folds = zip(MI_folds, theta_folds, dtheta_folds, torque_folds,␣
 ↪time_folds)

# TODO: finish by checking the MI data for any NaNs
for i, (MI, theta, dtheta, torque, time) in enumerate(alldata_folds):
    print("FOLD %2d " % i, anynans(MI), anynans(theta), # TODO
          anynans(dtheta), anynans(torque), anynans(time))
```

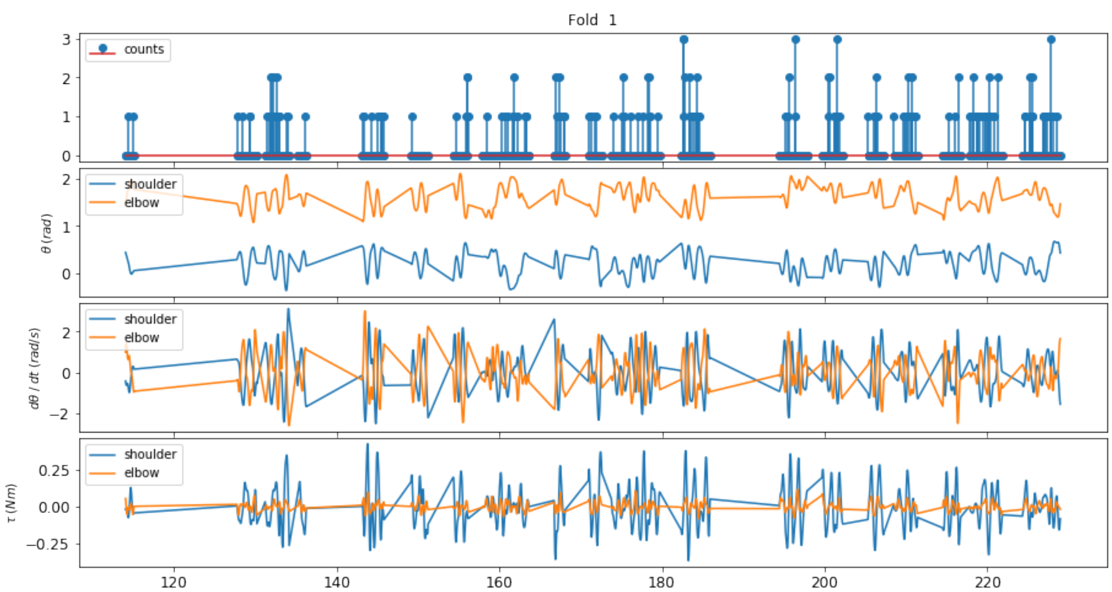
```
FOLD  0  False False False False False
FOLD  1  False False False False False
FOLD  2  False False False False False
FOLD  3  False False False False False
FOLD  4  False False False False False
FOLD  5  False False False False False
FOLD  6  False False False False False
FOLD  7  False False False False False
FOLD  8  False False False False False
FOLD  9  False False False False False
FOLD 10  False False False False False
FOLD 11  False False False False False
FOLD 12  False False False False False
FOLD 13  False False False False False
FOLD 14  False False False False False
FOLD 15  False False False False False
FOLD 16  False False False False False
```

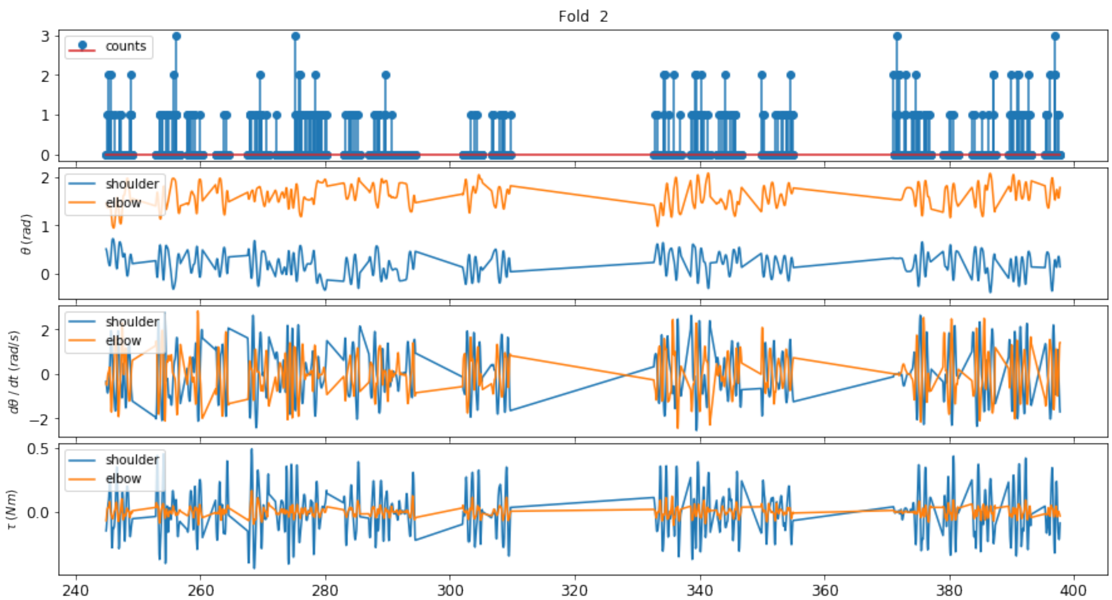
```
FOLD 17  False False False False False
FOLD 18  False False False False False
FOLD 19  False False False False False
```

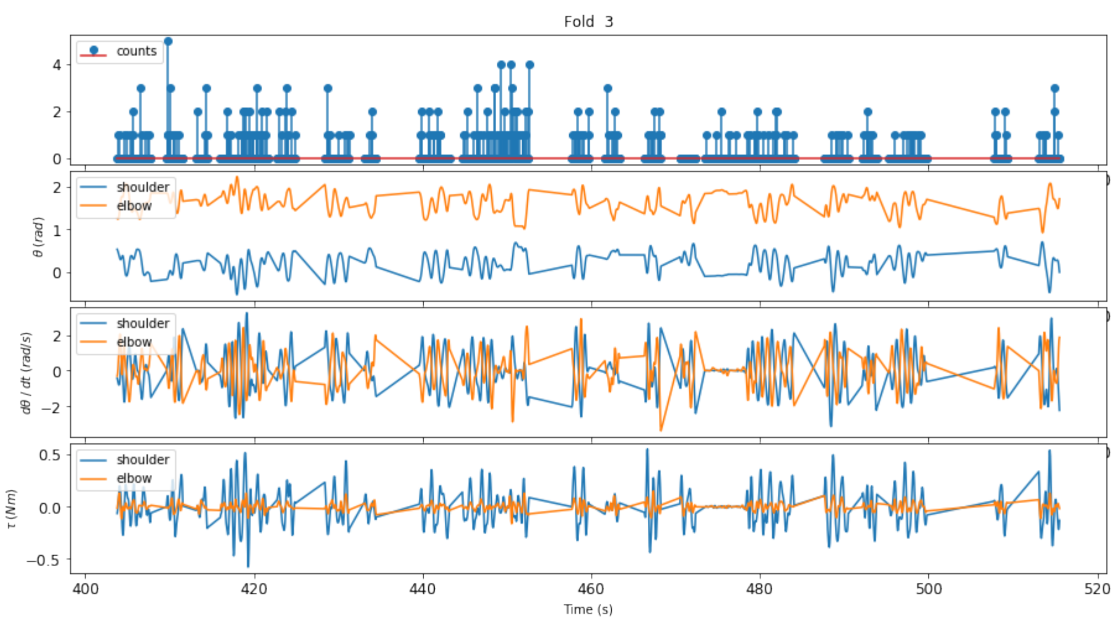
[7]:

```python
""" PROVIDED
For several folds, plot the data for the elbow and shoulder
and from one neuron
"""
f = 4
data_folds = zip(MI_folds[:f], theta_folds[:f], dtheta_folds[:f],
                 torque_folds[:f], time_folds[:f])

for i, (MI, theta, dtheta, torque, time) in enumerate(data_folds):
    fig, axs = plt.subplots(4, 1, figsize=(FIGWIDTH*3,8))
    fig.subplots_adjust(hspace=.05)
    axs = axs.ravel()

    # Neural Activation Counts
    axs[0].stem(time, MI[:,0], label='counts')
    axs[0].set_title("Fold %2d" % i)
    axs[0].legend(loc='upper left')

    lgnd = ['shoulder', 'elbow']

    # Position
    axs[1].plot(time, theta)
    axs[1].set_ylabel(r"$\theta \;(rad)$")
    axs[1].legend(lgnd, loc='upper left')

    # Velocity
    axs[2].plot(time, dtheta)
    axs[2].set_ylabel(r"$d\theta\; /\; dt \;(rad/s)$")
    axs[2].legend(lgnd, loc='upper left')

    # Torque
    axs[3].plot(time, torque)
    axs[3].set_ylabel(r"$\tau \;(Nm)$")
    axs[3].legend(lgnd, loc='upper left')
    if i == (f-1):
        axs[3].set_xlabel('Time (s)')
```


Fold 0
counts
shoulder
elbow
θ (rad)
dθ / dt (rad/s)
τ (Nm)


Fold 1
counts
shoulder
elbow
θ (rad)
dθ / dt (rad/s)
τ (Nm)


Fold 2
counts
shoulder
elbow
θ (rad)
dθ / dt (rad/s)
τ (Nm)

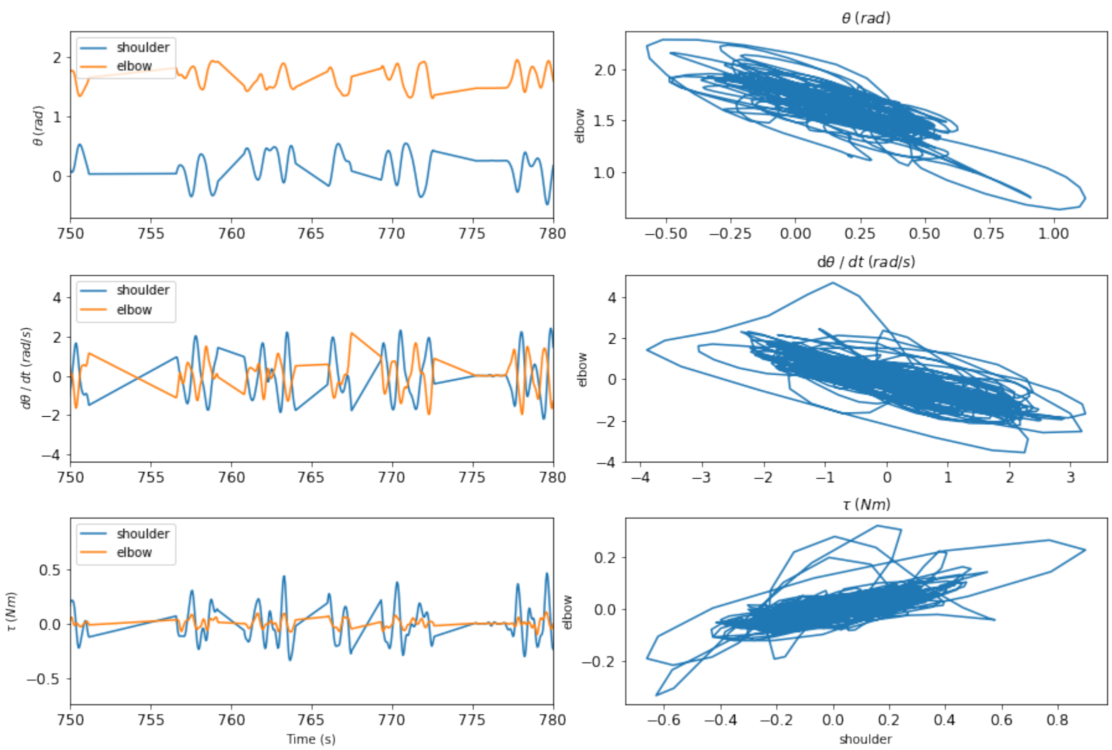

Fold 3
counts
shoulder
elbow
θ (rad)
dθ / dt (rad/s)
τ (Nm)
Time (s)

3 MODEL OUTPUTS

[8]:
```python
""" PROVIDED
For the sixth fold, visualize the correlation between the shoulder
and elbow for the angular position, the angular velocity, and the
torque
"""
f = 5

y_pos = theta_folds[f]
y_vel = dtheta_folds[f]
y_tor = torque_folds[f]
time = time_folds[f]

nrows = 3
ncols = 2
fig, axs = plt.subplots(nrows, ncols, figsize=(FIGWIDTH*3,FIGHEIGHT*2))
fig.subplots_adjust(wspace=.15, hspace=.3)
axs = axs.ravel()
xlim = [750, 780]

# POSITION
p = 0
axs[p].plot(time, y_pos)
axs[p].set_ylabel(r'$\theta \;(rad)$')
#axs[p].set_title(r'$\theta \;(rad)$')
axs[p].legend(['shoulder', 'elbow'], loc='upper left')
axs[p].set_xlim(xlim)

p = 1
axs[p].plot(y_pos[:,0], y_pos[:,1])
axs[p].set_ylabel('elbow')
axs[p].set_title(r'$\theta \; (rad)$')

# VELOCITY
p = 2
axs[p].plot(time, y_vel)
axs[p].set_ylabel(r'$d\theta\;/\;dt\;(rad/s)$')
#axs[p].set_title(r'$d\theta\;/\;dt\;(rad/s)$')
axs[p].legend(['shoulder', 'elbow'], loc='upper left')
axs[p].set_xlim(xlim)

p = 3
axs[p].plot(y_vel[:,0], y_vel[:,1])
axs[p].set_ylabel('elbow')
axs[p].set_title(r'd$\theta\;/\;dt\;(rad/s)$')
```

```python
# TORQUE
p = 4
axs[p].plot(time, y_tor)
axs[p].set_ylabel(r'$\tau \;(Nm)$')
#axs[p].set_title(r'$\tau$')
axs[p].legend(['shoulder', 'elbow'], loc='upper left')
axs[p].set_xlabel('Time (s)')
axs[p].set_xlim(xlim)

p = 5
axs[p].plot(y_tor[:,0], y_tor[:,1])
axs[p].set_xlabel('shoulder')
axs[p].set_ylabel('elbow')
axs[p].set_title(r'$\tau \;(Nm)$')
```

[8]: Text(0.5, 1.0, '$\\tau \\;(Nm)$')



4 REGRESSION

Predict torque of the shoulder and the elbow from the neural activations

[9]:
```
""" TODO
Evaluate the training performance of an already trained model
"""
def mse_rmse(trues, preds):
    '''
    Compute MSE and rMSE for each column separately.
    '''
    mse = np.sum(np.square(trues - preds), axis=0) / trues.shape[0]
    rmse_rads = np.sqrt(mse)
    rmse_degs = rmse_rads * 180 / np.pi
    return mse, rmse_rads, np.reshape(rmse_degs, (1, -1))

# TODO: finish implementation
def predict_score_eval(model, X, y):
    '''
    Compute the model predictions and corresponding scores.
    PARAMS:
        X: feature data
        y: corresponding output
    RETURNS:
        mse: mean squared error for each column
        rmse_rads: rMSE in radians
        rmse_deg: rMSE in degrees
        score: score computed by the models score() method
        preds: predictions of the model from X
    '''
    preds = model.predict(X)# TODO: use the model to predict the outputs from␣
 ↪the input data
    score = model.score(X, y)# TODO: use the model to compute the score
    # for the LinearRegression model, this is the coefficient of determination:␣
 ↪R^2
    # see the Sci-kit Learn documentation for LinearRegression for more details
    mse, rmse_rads, rmse_deg = mse_rmse(y, preds)# TODO: use mse_rmse() to␣
 ↪compute the mse and rmse
    return mse, rmse_rads, rmse_deg, score, preds
```

4.0.1 Training

[10]:
```
""" TODO
Extract the MI data from fold 5 as input and the torque data from
fold 5 as the output, for a multiple linear regression model (i.e.
the model will simultaneously predict shoulder and elbow torque).
Create a LinearRegression() model and train it using fit() on the
data from fold 5
"""
```

```
f = 5
X = MI_folds[f]# TODO
y = torque_folds[f]# TODO
time = time_folds[f]# TODO

model = LinearRegression()# TODO
model.fit(X,y)# TODO
```

```
[10]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
               normalize=False)
```

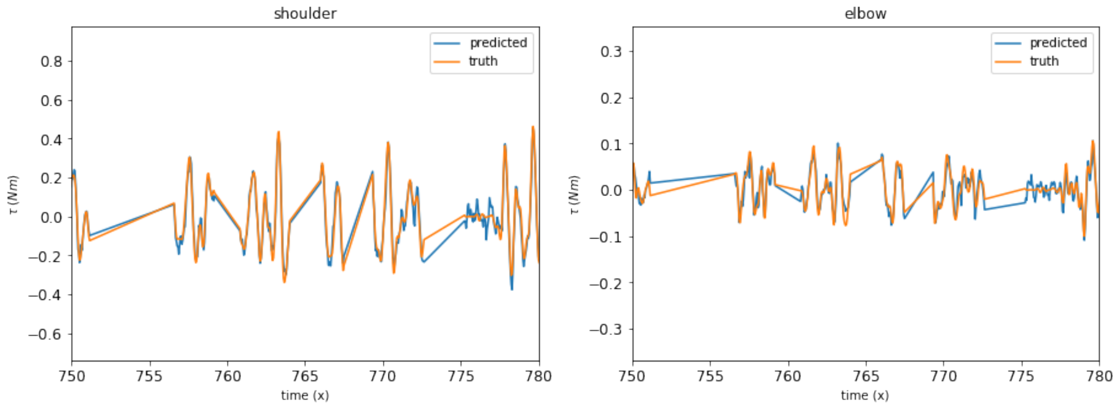
```
[11]: """ TODO
Evaluate the training performace of the model, using predict_score_eval()
Print the results displaying MSE, rMSE in rads and degrees, and the
correlation
"""
# TODO: call predict_score_eval() and get the corresponding outputs
mse, rmse_rads, rmse_deg, score, preds= predict_score_eval(model, X, y)
# TODO: print the results of predict_score_eval()
print('mse: %.4f %.4f\nrmse_rads: %.2f %.2f\n rmse_deg: %.2f %.2f\nscore %.
 ↪2f'%(mse[0], mse[1], rmse_rads[0], rmse_rads[1], rmse_deg[0,0],␣
 ↪rmse_deg[0,1], score))
```

```
mse: 0.0016 0.0002
rmse_rads: 0.04 0.02
 rmse_deg: 2.30 0.88
score 0.95
```

```
[12]: """ TODO
Plot the true torque and the predicted torque for the shoulder and
elbow, over time. Use 2 subplots (one subplot per output).

Focus on the time range 750 to 780 seconds
"""
titles = ['Shoulder', 'Elbow']
xlim = [750, 780]# TODO

# TODO: Generate the plots
fig, ax= plt.subplots(1,2,figsize=(FIGWIDTH*3, FIGHEIGHT))
ax[0].plot(time, preds[:,0], label= 'predicted', c='C0')
ax[0].plot(time, y[:, 0], label='truth', c='C1')
ax[0].set_title('shoulder')
ax[0].set_xlabel('time (x)')
ax[0].set_ylabel(r'$\tau \;(Nm)$')
ax[0].set_xlim(xlim)
ax[0].legend()
```

```
ax[1].plot(time, preds[:, 1], label= 'predicted', c='C0')
ax[1].plot(time, y[:, 1], label='truth', c='C1')
ax[1].set_title('elbow')
ax[1].set_xlabel('time (x)')
ax[1].set_ylabel(r'$\tau \;(Nm)$')
ax[1].legend()
ax[1].set_xlim(xlim)
```

[12]: (750, 780)



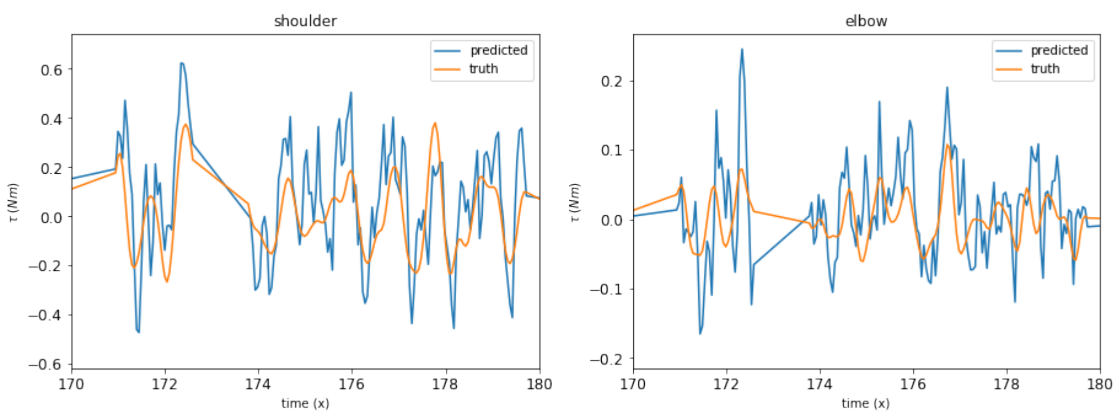
4.0.2 Testing

[13]:
```
""" TODO
Evaluate the performace of the model on unseen data from fold 1.
Recall that your model was trained using data from fold 5.
Print the results displaying MSE, rMSE in rads and degrees, and
the correlation
"""
ft = 1
Xtest = MI_folds[ft] # TODO
ytest = torque_folds[ft]# TODO
time_tst = time_folds[ft]# TODO

# TODO: call predict_score_eval() and get the corresponding outputs
mse_test, rmse_rads_test, rmse_deg_test, score_test, preds_test=␣
 ↪predict_score_eval(model, Xtest, ytest)
# TODO: print the results of predict_score_eval()
print('mse: %.4f %.4f\nrmse_rads: %.2f %.2f\n rmse_deg: %.2f %.2f\nscore %.
 ↪2f'%(mse_test[0], mse_test[1], rmse_rads_test[0], rmse_rads_test[1],␣
 ↪rmse_deg_test[0,0], rmse_deg_test[0,1], score_test))
```

```
mse: 0.0318 0.0042
rmse_rads: 0.18 0.06
 rmse_deg: 10.22 3.72
score -0.68
```

[14]:
```python
""" TODO
Plot the true torque and the predicted torque over time, for the
shoulder and the elbow. Use 2 subplots (one for the shoulder and
the other for the elbow)

Focus on the time range 170 to 180 seconds
"""
titles = ['Shoulder', 'Elbow']
xlim = (170, 180)# TODO

# TODO: Generate the plots
fig, ax= plt.subplots(1,2,figsize=(FIGWIDTH*3, FIGHEIGHT))
ax[0].plot(time_tst, preds_test[:,0], label= 'predicted', c='C0')
ax[0].plot(time_tst, ytest[:, 0], label='truth', c='C1')
ax[0].set_title('shoulder')
ax[0].set_xlabel('time (x)')
ax[0].set_ylabel(r'$\tau \;(Nm)$')
ax[0].set_xlim(xlim)
ax[0].legend()

ax[1].plot(time_tst, preds_test[:, 1], label= 'predicted', c='C0')
ax[1].plot(time_tst, ytest[:, 1], label='truth', c='C1')
ax[1].set_title('elbow')
ax[1].set_xlabel('time (x)')
ax[1].set_ylabel(r'$\tau \;(Nm)$')
ax[1].legend()
ax[1].set_xlim(xlim)
```

[14]: (170, 180)



4.0.3 Training Size Sensitivity

For this section, you will be training the model on a different number of folds, each time testing it on the same unseen data from another fold not used in the training procedure.

[15]:
```
""" TODO
Fill in the missing lines of code
"""
def training_set_size_loop(model, X, y, folds_inds, val_fold_idx):
    '''
    Train a model on multiple training set sizes

    PARAMS:
        model: object to train
        X: input data
        y: output data
        folds_inds: list of the number of folds to use for different
                    training sets
        val_fold_idx: fold index to use as the validation set
    RETURNS:
        rmse: dict of train and validation RMSE lists
        corr: dict of train and validation R^2 lists
    '''
    # Initialize log of performance metrics
    ncats = y[0].shape[1]
    rmse = {'train':np.empty((0, ncats)), 'val':np.empty((0, ncats))}
    corr = {'train':[], 'val':[]}

    # Data used for validation
    Xval = X[val_fold_idx]
    yval = y[val_fold_idx]

    # Loop over the different experiments
    for f in folds_inds:
        # Construct training set
        Xtrain = np.concatenate(X[:f])
        ytrain = np.concatenate(y[:f])

        # Build the model
        model.fit(Xtrain, ytrain)

        # TODO: call predict_score_eval using the training data
        _, _, rmse_degs, score, _= predict_score_eval(model, Xtrain, ytrain)#␣
 ↪TODO
        # TODO: call predict_score_eval using the validation data
```

```
        _, _, rmse_degs_val, score_val, _ = predict_score_eval(model, Xval,␣
↪yval)# TODO

        print('training folds: %d, training rmse: %.2f, validating rmse %.2f␣
↪total error: %.2f'%(f,
                                                                              ␣
↪                   rmse_degs[0][0],
                                                                              ␣
↪                   rmse_degs_val[0][0],
                                                                              ␣
↪                  rmse_degs[0][0]+rmse_degs_val[0][0]))

        # Record the performance metrics for this experiment
        rmse['train'] = np.append(rmse['train'], rmse_degs, axis=0)
        corr['train'].append(score)
        rmse['val'] = np.append(rmse['val'], rmse_degs_val, axis=0)
        corr['val'].append(score_val)

    return rmse, corr
```

[17]:
```
""" TODO
Create a new linear model and train the model on different training set sizes,
using training_set_size_loop() with training set sizes of folds 1 through 17
and use 18 as the val_fold_idx.
The input data is the MI data and the output data is the torque for both the
shoulder and elbow.
"""
val_fold = 18
folds = range(1,val_fold)

# TODO: Create a new LinearRegression model
model= LinearRegression()

# TODO: get the list of rMSE and correlation values per training set size, by
#       using training_set_size_loop
rmse, corr= training_set_size_loop(model, MI_folds, torque_folds, folds,␣
 ↪val_fold)
```

```
training folds: 1, training rmse: 1.27, validating rmse 7.19 total error: 8.46
training folds: 2, training rmse: 2.28, validating rmse 4.05 total error: 6.34
training folds: 3, training rmse: 2.74, validating rmse 3.46 total error: 6.20
training folds: 4, training rmse: 3.04, validating rmse 3.43 total error: 6.47
training folds: 5, training rmse: 3.80, validating rmse 3.50 total error: 7.29
training folds: 6, training rmse: 4.06, validating rmse 3.50 total error: 7.56
training folds: 7, training rmse: 4.15, validating rmse 3.56 total error: 7.70
training folds: 8, training rmse: 4.65, validating rmse 3.78 total error: 8.43
training folds: 9, training rmse: 4.61, validating rmse 3.73 total error: 8.34
```

```
training folds: 10, training rmse: 4.55, validating rmse 3.65 total error: 8.20
training folds: 11, training rmse: 4.50, validating rmse 3.64 total error: 8.14
training folds: 12, training rmse: 4.44, validating rmse 3.61 total error: 8.06
training folds: 13, training rmse: 4.39, validating rmse 3.55 total error: 7.94
training folds: 14, training rmse: 4.35, validating rmse 3.49 total error: 7.84
training folds: 15, training rmse: 4.28, validating rmse 3.43 total error: 7.71
training folds: 16, training rmse: 4.30, validating rmse 3.44 total error: 7.75
training folds: 17, training rmse: 4.26, validating rmse 3.44 total error: 7.70
```

[19]:

```
""" TODO
Plot rMSE as a function of the training set size for
the shoulder and the elbow; also plot correlation as
a function of training set size. Use three subplots
(one for the shoulder rMSE, one for the elbow rMSE, and
one with the correlation)
"""
fig, ax= plt.subplots(1,3,figsize=(FIGWIDTH*4, FIGHEIGHT*1))
ax[0].plot(folds, rmse['train'][:,0], label='training', c='C0')
ax[0].plot(folds, rmse['val'][:,0], label='validation', c='C1')
ax[0].set_xlabel('# of training samples')
ax[0].set_ylabel('rmse')
ax[0].set_title('shoulder', fontsize=FONTSIZE)
ax[1].plot(folds, rmse['train'][:,1], label='training', c='C0')
ax[1].plot(folds, rmse['val'][:,1], label='validation', c='C1')
ax[1].set_xlabel('# of training samples')
ax[1].set_ylabel(r'rmse')
ax[1].set_title('elbow', fontsize=FONTSIZE)
ax[2].plot(folds, corr['train'], label='training', c='C0')
ax[2].plot(folds, corr['val'], label='validation', c='C1')
ax[2].set_xlabel('# of training samples')
ax[2].set_ylabel(r'correlation coefficient')
ax[2].set_title('correlation', fontsize=FONTSIZE)
plt.legend()
```

[19]: <matplotlib.legend.Legend at 0x7fcabb3cffd0>