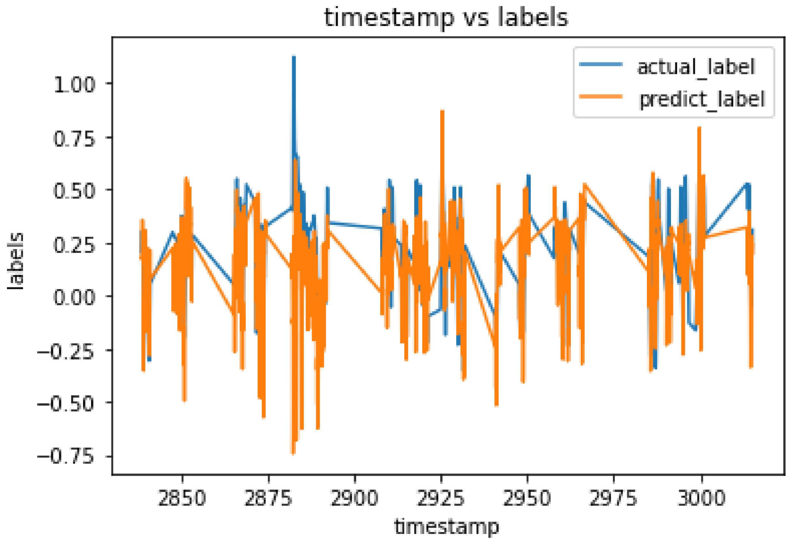
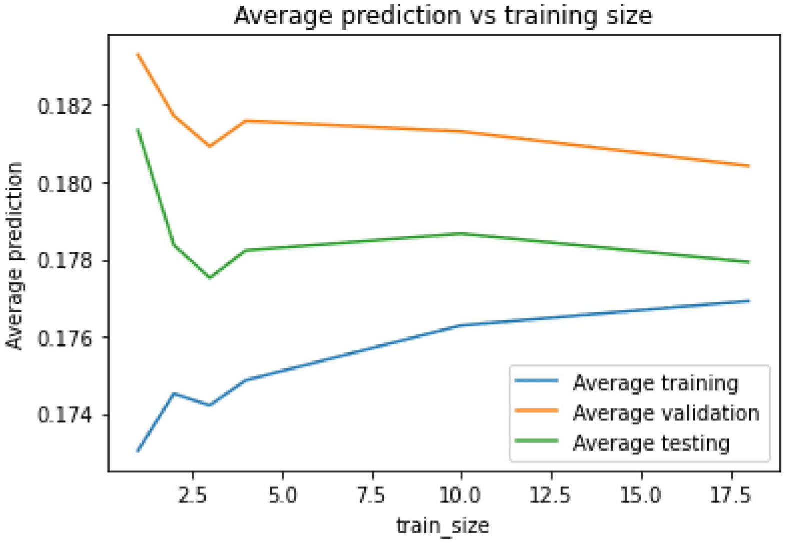
In [22]:

```
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow import keras
import os
import fnmatch
import time
import pickle
from tensorflow.keras.layers import InputLayer, Dense
from tensorflow.keras.models import Sequential

#opening the file
file_1 = open("bmi_dataset.pkl", "rb")
f_open_1 = pickle.load(file_1)
file_1.close()

#opening the file
file_2 = open("./results/bmi_theta_0_hidden_10_5_JI_Ntraining_1_rotation_0_results.pkl
f_open_2 = pickle.load(file_2)
file_2.close()

# finding the actual testing labels
ins = f_open_1['MI']
Nfolds = len(ins)
folds_testing = (np.array([Nfolds-1]) + 0) % Nfolds
outs = f_open_1['theta']
outs_testing = np.concatenate(np.take(outs, folds_testing))
actual_testing = outs_testing[:,[0]]

# Getting the predicted testing labels
predict_testing = f_open_2['predict_testing']

## Getting the timestamp for predicted labels
timestamp = f_open_2['time_testing']

#plotting figure 1
plt.plot(timestamp,actual_testing)
plt.plot(timestamp,predict_testing)


plt.ylabel('labels')
plt.xlabel('timestamp')
plt.title('timestamp vs labels')
plt.legend(['actual_label','predict_label'])
#saving the figure1
plt.savefig("figure1.png")
plt.show()
plt.close()

new_results = []

def read_all_rotations(dirname, filebase):
    '''Read results from dirname from files matching filebase'''

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


    # Loop over matching files
    for f in files:
        fp = open("%s/%s"%(dirname,f), "rb")
        r = pickle.load(fp)
        fp.close()
        results.append(r)
    return results

# matching the files
train = [1,2,3,4,10,18]
for t in train:
    filebase = "bmi_theta_0_hidden_10_5_JI_Ntraining_"+str(t)+"_rotation_*_results.pkl
    new_results.append( read_all_rotations("results", filebase))

avg_train = []
avg_validate = []
avg_test = []

temp_1 = []
temp_2 = []
temp_3 = []

# calculating the average
for i in range(len(train)):
    for j in range(len(new_results[0][0])):
        temp_1.append(np.mean(new_results[i][j]['predict_training']))
        temp_2.append(np.mean(new_results[i][j]['predict_validation']))
        temp_3.append(np.mean(new_results[i][j]['predict_testing']))

    avg_train.append(sum(temp_1)/len(temp_1))
    avg_validate.append(sum(temp_2)/len(temp_2))
    avg_test.append(sum(temp_3)/len(temp_3))

#plotting figure 2
plt.plot(train,avg_train)
plt.plot(train,avg_validate)
plt.plot(train,avg_test)


plt.ylabel('Average prediction')
plt.xlabel('train_size')
plt.title('Average prediction vs training size')
plt.legend(['Average training','Average validation', 'Average testing'])
#saving the figure2
plt.savefig("figure2.png")
plt.show()
plt.close()
```

```
C:\Users\shyam\anaconda3\envs\tf\lib\site-packages\numpy\core\fromnumeric.py:43: Visi
bleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a l
ist-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depr
ecated. If you meant to do this, you must specify 'dtype=object' when creating the nd
array.
  result = getattr(asarray(obj), method)(*args, **kwds)
```


timestamp vs labels
actual_label
predict_label
labels
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
2850
2875
2900
2925
2950
2975
3000
timestamp


Average prediction vs training size
Average prediction
0.182
0.180
0.178
0.176
0.174
Average training
Average validation
Average testing
2.5
5.0
7.5
10.0
12.5
15.0
17.5
train_size

```
In [ ]:
```

```
In [ ]:
```