UNIVERSITÀ DI PISA

A.Y. 2019/20
« Intelligent systems for pattern recognition »
Master Degree in Computer Science
Artificial Intelligence Curriculum

Midterm 3
Assignment 3

Gated Recurrent Neural Networks (LSTM)

Diletta Goglia

Data loading and preprocessing

Model 1

- Multivariate timeseries (multiple features) with multivariate input data
- 18 sensors in the house, in 9 different rooms, both for temperature and humidity.
- 18 features in input, 1 feature for output (“Appliances” column)
- Measurements from January to April for training
- Month of May for test
- `sklearn.preprocessing` library to scale data

Model 2

- Load only the “appliances” column from dataset
- Different train/test splitting
- Different input/output
- Input shape: passing only one step in the past
- Reshaping the output is not needed (because `return_sequences` in LSTM layer is set to `False`).

```python
''' Model 1) consider temperature and humidity data as input '''
# multivariate input data
#Load CSV with Pandas
dataset = pd.read_csv('energydata_complete.csv', header = 0, sep=',', quotec

#datetime format
dataset.index = pd.to_datetime(dataset['date'], format='%Y-%m-%d %H:%M:%S')
dataset = dataset.set_index('date')
''' DATASET SPLITTING '''

train = dataset["2016-01-11":"2016-04-30"] # jan - apr
test = dataset["2016-05-11":"2016-05-27"] # may

train.index = pd.to_datetime(train.index, format='%Y-%m-%d %H:%M:%S')
test.index = pd.to_datetime(test.index, format='%Y-%m-%d %H:%M:%S')

training_values = train.values
test_values = test.values

''' INPUT - OUTPUT '''
# split into input and outputs
train_X, train_y = training_values[:, 1:], training_values[:, :1]
# temperature e umidità (9+9 tot. 18 feature) sono input
# mentre la prima colonna (appliances) è l'output
test_X, test_y = test_values[:, 1:], test_values[:, :1]


''' PREPROCESSING '''
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit_transform(train_X)
train_X = scaler.transform(train_X)
scaler.fit_transform(train_y)
train_y = scaler.transform(train_y)
scaler.fit_transform(test_X)
test_X = scaler.transform(test_X)
scaler.fit_transform(test_y)
test_y = scaler.transform(test_y)

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape(train_X.shape[0], 1, train_X.shape[1])
test_X = test_X.reshape(test_X.shape[0], 1, test_X.shape[1])
```

Train / Test

```
                     Appliances         T1  ...         T9    RH_9
date                                        ...
2016-01-11 17:00:00        60.0  19.890000  ...  17.033333  45.5300
2016-01-11 17:10:00        60.0  19.890000  ...  17.066667  45.5600
2016-01-11 17:20:00        50.0  19.890000  ...  17.000000  45.5000
2016-01-11 17:30:00        50.0  19.890000  ...  17.000000  45.4000
2016-01-11 17:40:00        60.0  19.890000  ...  17.000000  45.4000
...                         ...        ...  ...        ...      ...
2016-05-27 17:20:00       100.0  25.566667  ...  23.200000  46.7900
2016-05-27 17:30:00        90.0  25.500000  ...  23.200000  46.7900
2016-05-27 17:40:00       270.0  25.500000  ...  23.200000  46.7900
2016-05-27 17:50:00       420.0  25.500000  ...  23.200000  46.8175
2016-05-27 18:00:00       430.0  25.500000  ...  23.200000  46.8450

[19735 rows x 19 columns]
```

Train

Test

Input / Output

Output

Input

Features

```
                     Appliances         T1  ...         T9    RH_9
date                                        ...
2016-01-11 17:00:00        60.0  19.890000  ...  17.033333  45.5300
2016-01-11 17:10:00        60.0  19.890000  ...  17.066667  45.5600
2016-01-11 17:20:00        50.0  19.890000  ...  17.000000  45.5000
2016-01-11 17:30:00        50.0  19.890000  ...  17.000000  45.4000
2016-01-11 17:40:00        60.0  19.890000  ...  17.000000  45.4000
...                         ...        ...  ...        ...      ...
2016-05-27 17:20:00       100.0  25.566667  ...  23.200000  46.7900
2016-05-27 17:30:00        90.0  25.500000  ...  23.200000  46.7900
2016-05-27 17:40:00       270.0  25.500000  ...  23.200000  46.7900
2016-05-27 17:50:00       420.0  25.500000  ...  23.200000  46.8175
2016-05-27 18:00:00       430.0  25.500000  ...  23.200000  46.8450

[19735 rows x 19 columns]
```

3D tensors reshape

```
Train_input (temperature and humidity)
 (15882, 18)
Train_output (energy)
 (15882, 1)
Test_input (temperature and humidity)
 (2413, 18)
Test_output (energy)
 (2413, 1)


After reshaping:
(15882, 1, 18) (15882, 1) (2413, 1, 18) (2413, 1)
```

Code snippets

- Keras Sequential API
- Build LSTM
- Add layers
- Model fit
- Check loss
- Predict

Model 1

```python
''' HYPERPARAM '''
learning_rate=1e-4
batch_size=70
epochs=20

''' BUILDING THE FIRST MODEL '''
# using Sequential API
model = tf.keras.Sequential()   # instaciate a model using Sequential class
                                # --> will contruct a pipeline of layers
# building add one layer at time
model.add(layers.LSTM(18, activation='tanh', return_sequences=True,
                      input_dim=(train_X.shape[2])))
# set the return_sequences to True, the output shape becomes a 3D array
model.add(layers.Dropout(0.5))
# Dropout regularize the model by ramdomly tuting off some neurons
# --> prevent overfitting
model.add(layers.Dense(1))
# then you don't need to specify the input shape again because
# it is automatically inferred by sequential layer

model.compile(optimizer='adam',
              loss='mae',
              metrics=['accuracy'])

print('\n', model.summary())
''' MODEL FIT '''
# fit model
history = model.fit(train_X, train_y, validation_split=0.2, epochs=epochs,
                    batch_size=batch_size, verbose=1, shuffle=False)

# Plot Model Loss
# list all data in history
print(history.history.keys())
# summarize history for loss
plt.figure(3)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss']) #RAISE ERROR
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
''' MODEL PREDICTION '''
# make a prediction
test_predict0 = model.predict(test_X)
#reshape (because return_sequences was set to True)
test_predict = test_predict0.reshape((test_predict0.shape[0],
                                      test_predict0.shape[2]))
# invert predictions
test_predict = scaler.inverse_transform(test_predict)
```

18 input features

Model 2

pretty much the same, but…

```python
''' PREPROCESSING'''

scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
train_size = int(len(dataset) * 0.80)
test_size = len(dataset) - train_size
train1, test1 = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

''' DATASET SPLITTING '''

look_back = 1
X_train, Y_train = create_dataset(train1, look_back)
X_test, Y_test = create_dataset(test1, look_back)
# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
''' BUILDING THE SECOND MODEL

# using Sequential API
model = tf.keras.Sequential()   # instaciate a model using Sequential class
                                # --> will contruct a pipeline of layers
model.add(layers.LSTM(10, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1))
#model.compile(loss='mean_squared_error', optimizer='adam')
model.compile(optimizer='adam',   # try also adam
              loss='mae')

''' MODEL FIT '''
# fit model
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, v
                    verbose=1, shuffle=False)
model.summary()
```

1 timestep

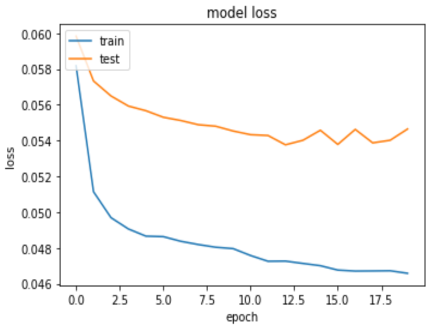
1 input feature

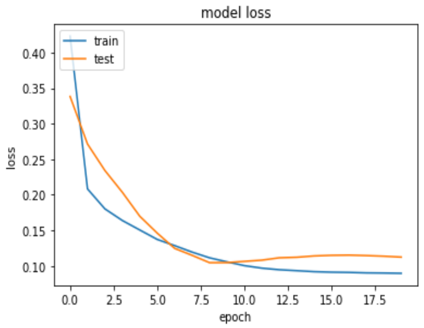
Model 1:

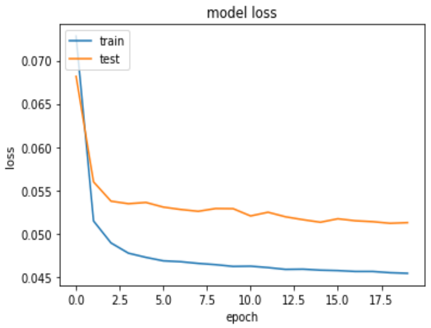
Predict the current energy expenditure given as input information the temperature (T_i) and humidity (RH_i) information from all the i sensors in the house.

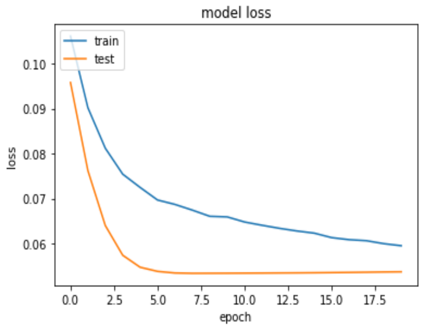
Tuning hyperparameters

- Dropout
- Number of hidden layers
- Number of neurons for LSTM layer
- Optimizers
- Learning rate
- Loss functions
- Activation functions
- Batch size
- Range of feature scaling





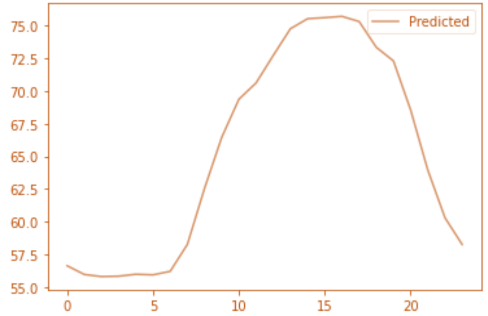


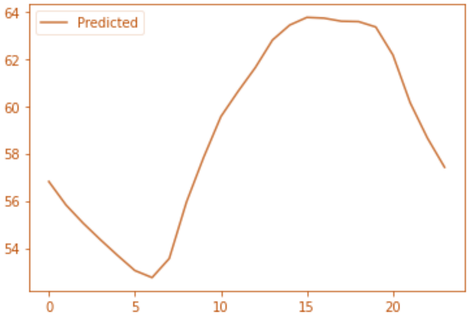


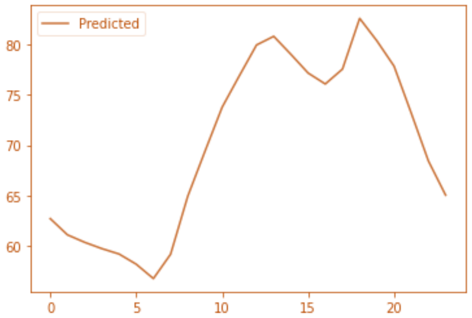
Results

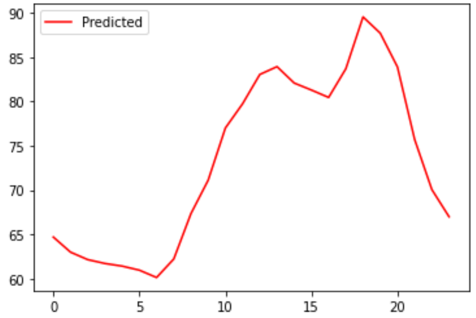
* Plotted grouping by mean value of energy consumption

Predictions

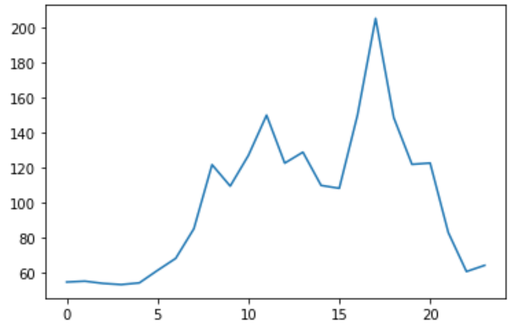








Real data

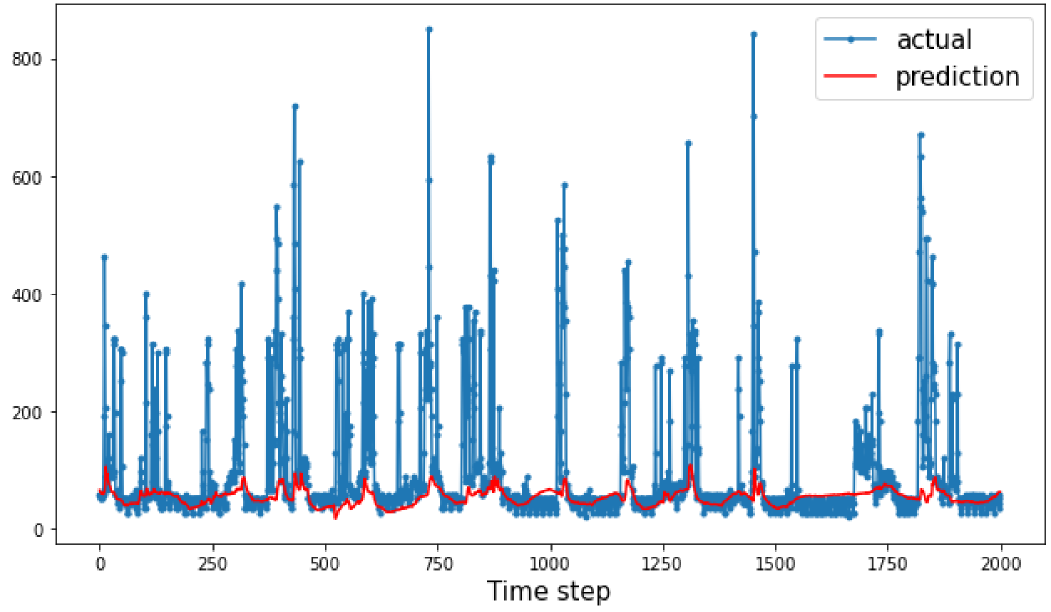


The model is able to capture the overall trend.

Model 1:

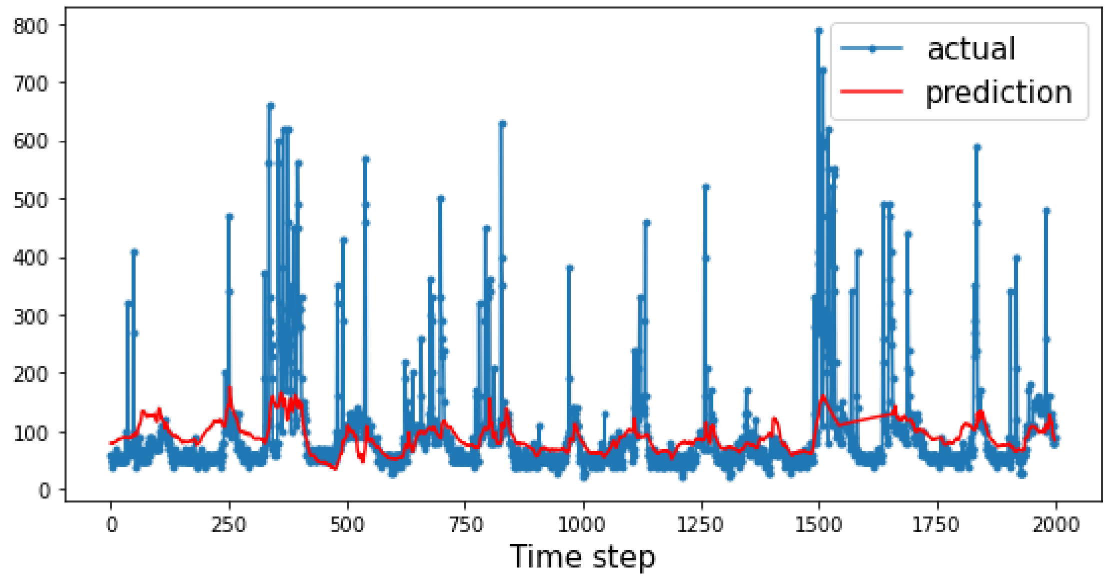
Predicted values VS Real Data

Validation



Predicted values are improved by tuning the model parameters and, at the end, they are almost exactly like ground truth data.

Test

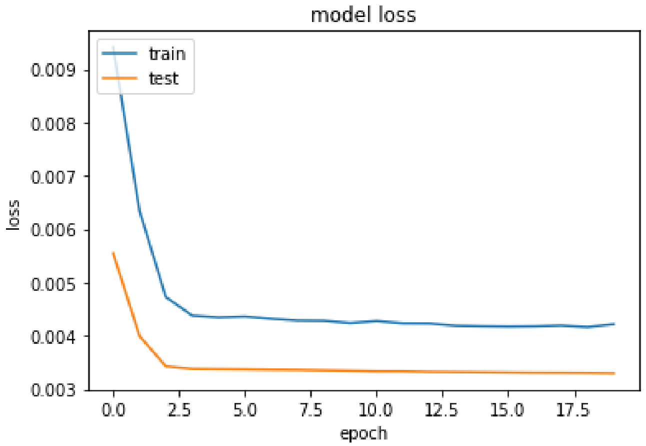


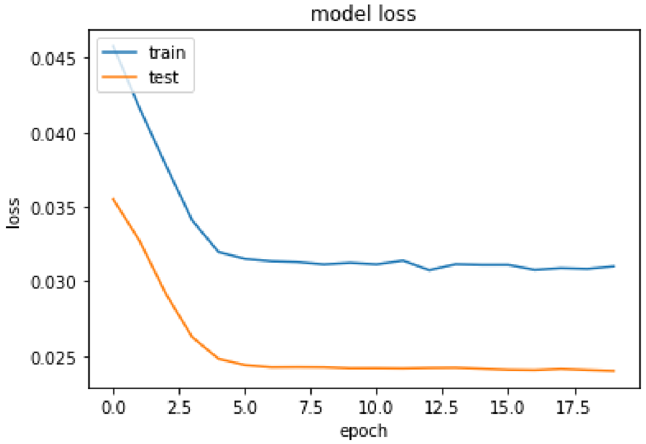
Not perfect… but really good prediction!

Model 2:

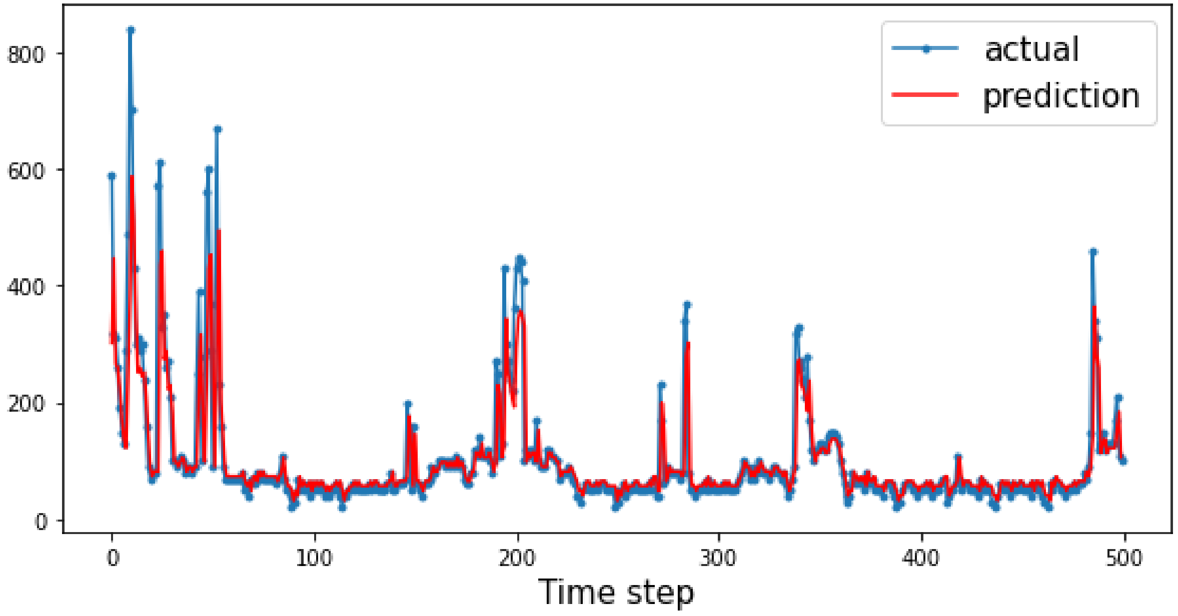
Setup a one step-ahead predictor for energy expenditure, i.e. given the current energy consumption, predict its next value.
Predict the value at the current time step by using the history (*n* time steps from it, in this case, with n=1)

Tuning hyperparameters





Results



Final considerations

Pros and cons of Teacher Forcing (used in the first model)

Pros:

- Training with *Teacher Forcing* converges faster. If we do not use *Teacher Forcing*, the hidden states of the model will be updated by a sequence of wrong predictions, errors will accumulate, and it is difficult for the model to learn from that.

Cons:

- During inference, since there is usually no ground truth available, the model will need to feed its own previous prediction back to itself for the next prediction. Therefore there is a discrepancy between training and inference, and this might lead to poor model performance and instability. This is known as *Exposure Bias* in literature.

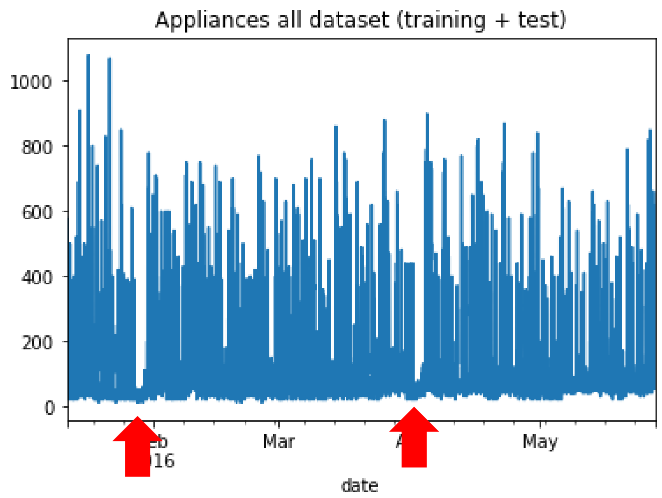
"… [teacher forcing] can result in problems in generation as small prediction error compound in the conditioning context. This can lead to poor prediction performance as the RNN's conditioning context (the sequence of previously generated samples) diverge from sequences seen during training."

"The disadvantage of strict teacher forcing arises […] the fed-back inputs that the network sees during training could be quite different from the kind of inputs that it will see at test time."

Deep Learning, ch. 10

Why predictions in the second model are better?

- Second model has much less parameters to train
- Lag in appliances dataset
- Not all the temperatures and humidity measurement must be related to appliances consumption!



Time series correlations

	Appliances	T1	RH_1	...	RH_8	T9	RH_9
Appliances	1.000000	0.055447	0.086031	...	-0.094039	0.010010	-0.051462
T1	0.055447	1.000000	0.164006	...	-0.006441	0.844777	0.071756
RH_1	0.086031	0.164006	1.000000	...	0.736196	0.115263	0.764001
T2	0.120073	0.836834	0.269839	...	0.068534	0.675535	0.157346
RH_2	-0.060465	-0.002509	0.797535	...	0.679777	0.054544	0.676467
T3	0.085060	0.892402	0.253230	...	0.044427	0.901324	0.134602
RH_3	0.036292	-0.028550	0.844677	...	0.828822	-0.195270	0.833538
T4	0.040281	0.877001	0.106180	...	-0.095192	0.889439	-0.025549
RH_4	0.016965	0.097861	0.880359	...	0.847259	-0.044518	0.856591
T5	0.019760	0.885247	0.205797	...	0.016388	0.911055	0.072308
RH_5	0.006955	-0.014782	0.303258	...	0.359840	-0.138509	0.272197
T6	0.117638	0.654769	0.316141	...	0.073721	0.667177	0.184424
RH_6	-0.083178	-0.615045	0.245126	...	0.489580	-0.738940	0.391943
T7	0.025801	0.838705	0.021397	...	-0.209961	0.944776	-0.077690
RH_7	-0.055642	0.135182	0.801122	...	0.883984	0.028055	0.858686
T8	0.039572	0.825413	-0.030053	...	-0.209532	0.869338	-0.156820
RH_8	-0.094039	-0.006441	0.736196	...	1.000000	-0.113014	0.855812
T9	0.010010	0.844777	0.115263	...	-0.113014	1.000000	-0.008683
RH_9	-0.051462	0.071756	0.764001	...	0.855812	-0.008683	1.000000

[19 rows x 19 columns]

Almost NOT related!

* the correlation does not depend on a cause-effect relationship but on the tendency of one variable to change according to another

UNIVERSITÀ DI PISA

A.Y. 2019/20
« Intelligent systems for pattern recognition »
Master Degree in Computer Science
Artificial Intelligence Curriculum

Thanks for your attention