Università di Pisa

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

Midterm 2
Assignment 1

Hidden Markov Models

Diletta Goglia

Prior analysis
of consumptions

Aim: extract useful and interesting information from data (regularities and repetitions of patterns over the timeseries).

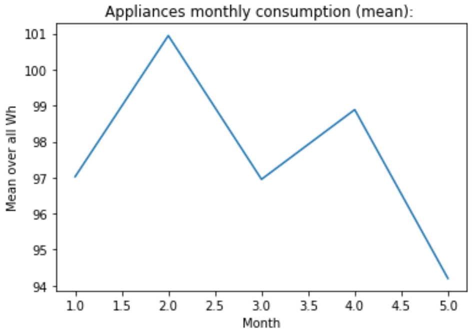
Really interesting: density plots. They are a way to estimate the probability density function of a random variable.

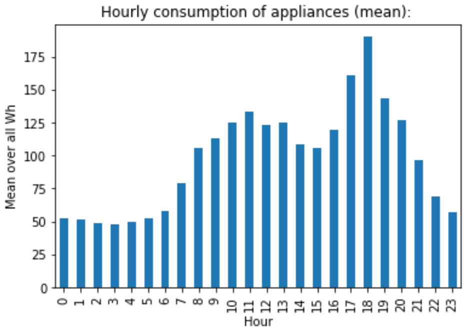
```
#Load CSV with Pandas
ts = pd.read_csv('energydata_complete.csv', header = 0, sep=',', quotechar='"', engine='python',
                 usecols=[0, 1, 2], dtype = {"date" : "datetime64[ns]", "Appliances" : "float", "lights" : "float"})
#selecting the first three columns of the dataset: "data", "Appliances" and "Lights"
#which measure the energy consumption of appliances and lights, respectively, across a period of 4.5 months
#load colums separately
appl_data = pd.read_csv('energydata_complete.csv', header = 0, sep=',', quotechar='"', engine='python',
                        usecols=[0, 1], dtype = {"Appliances" : "float"})
lights_data = pd.read_csv('energydata_complete.csv', header = 0, sep=',', quotechar='"', engine='python',
                          usecols=[0, 2], dtype = {"lights" : "float"})

#datetime format
ts.index = pd.to_datetime(ts['date'], format='%Y-%m-%d %H:%M:%S')
appl_data.index = pd.to_datetime(appl_data['date'], format='%Y-%m-%d %H:%M:%S')
lights_data.index = pd.to_datetime(lights_data['date'], format='%Y-%m-%d %H:%M:%S')

#indexes
ts = ts.set_index('date')
appl_data = appl_data.set_index('date')
lights_data = lights_data.set_index('date')
```

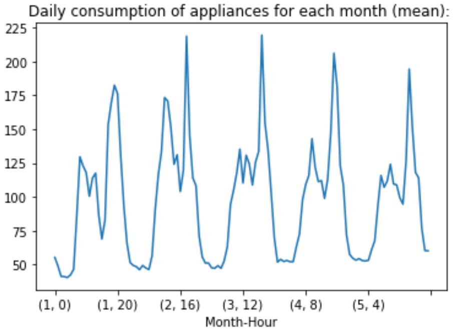
```
ts['Appliances'].groupby(ts.index.month).mean().plot()
ts['Appliances'].groupby(ts.index.hour).mean().plot()
ts['Appliances'].groupby(ts.index.hour).mean().plot(kind='bar')
ts['Appliances'].groupby(ts.index.day).mean().plot(kind='bar')

ts['lights'].groupby(ts.index.month).mean().plot()
ts['lights'].groupby(ts.index.hour).mean().plot()
ts['lights'].groupby(ts.index.hour).mean().plot(kind='bar')
ts['lights'].groupby(ts.index.day).mean().plot(kind='bar')

ts['Appliances'].groupby([ts.index.month, ts.index.hour]).mean().plot()
ts['lights'].groupby([ts.index.month, ts.index.hour]).mean().plot()
ts['Appliances'].plot(kind='hist', label = 'Appliances')
ts['lights'].plot(kind='hist', label = 'lights')

ts['Appliances'].plot(kind='kde', c='r')
ts['lights'].plot(kind='kde', c='r')
```

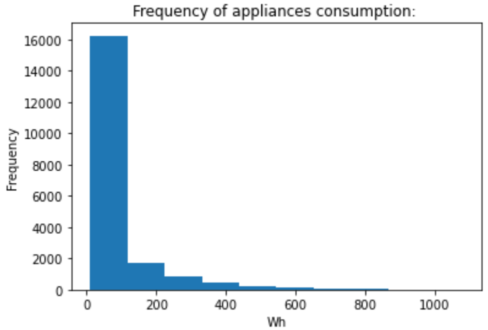
Month | *Hour* | *Month-Hour* | *Spectral domain* | *Density*

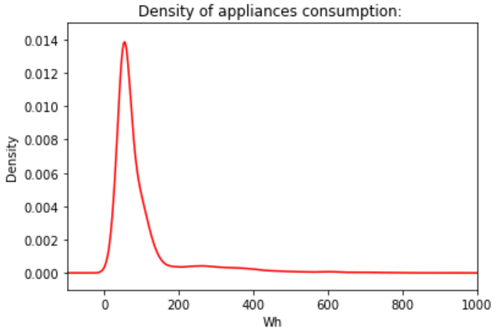
Appliances



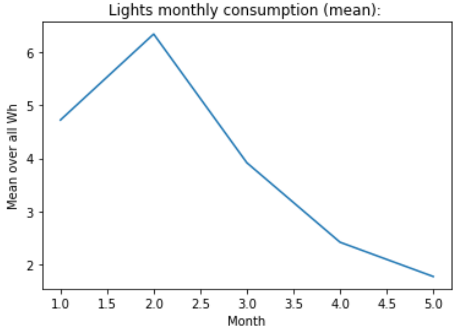


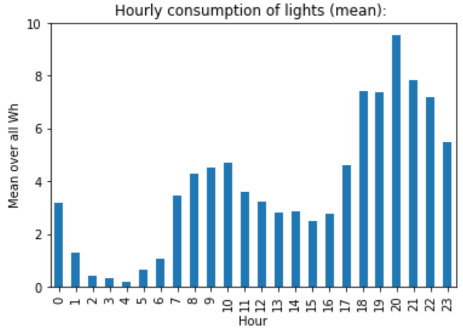


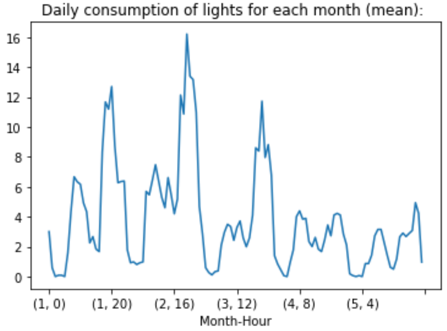


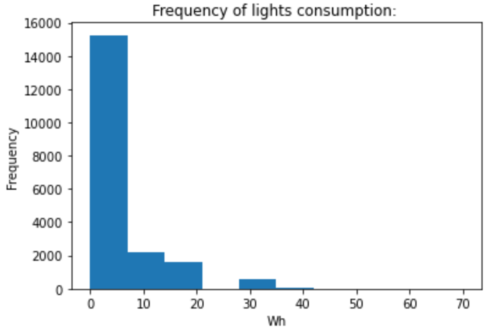


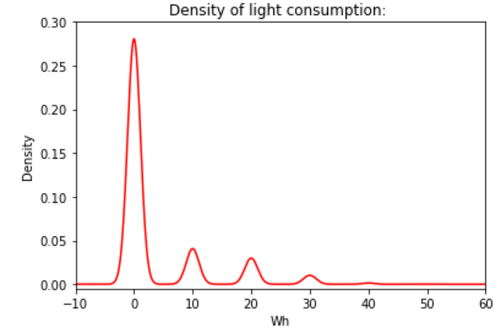
Lights











HMMs

Fit Hidden Markov Models with Gaussian emissions to the data (two separate HMMs, one for appliances and one for light).

```
                        HIDDEN MARKOV MODEL
'''
#number of hidden states
ncomp = 2 # then try also with 3, 5, 7
# Create an HMM with Gaussian emission for 'Appliances'
appl_hmm = GaussianHMM(n_components=ncomp , covariance_type="diag",
                       n_iter=1000, algorithm='viterbi') # model for appliances
# Create an HMM with Gaussian emission for 'light'
lights_hmm = GaussianHMM(n_components=ncomp , covariance_type="diag",
                         n_iter=1000, algorithm='viterbi') # model for lights
# fit models to data
appl_hmm.fit(appl_data)
lights_hmm.fit(lights_data)
```

Train HMMs with different number of hidden states to cluster observations and to capture the fact that there are hidden regimes of consumption:

Hidden states	2	3	5	7
Regimes of consumption	-	-	Very High	Very High
	High	High	High	High
	-	-	-	Medium-High
	-	Medium	Medium	Medium
	-	-	-	Medium-Low
	Low	Low	Low	Low
	-	-	Very Low	Very Low

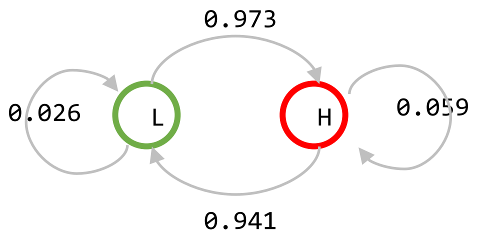
Model Parameters

Extract parameters like the transition matrix and the log likelihood from the trained models, for each one with different number of components (hidden states).

```
# APPL
print('\n\nParameters of appliances model.\n- Transition matrix: \n', appl_hmm.transmat_)
logProb = appl_hmm.score(appl_data)
print('\n Log likelihood: \n', round(logProb,2))
# LIGHTS
print('\n\nParameters of lights model.\n- Transition matrix: \n', lights_hmm.transmat_)
logProb = lights_hmm.score(np.reshape(appl_data,[len(lights_data),1]))
print('\n- Log likelihood: \n', round(logProb,2))
```

Transition matrix

Example of transition matrix for 2 hidden states.



$$\begin{bmatrix} 0.973 & 0.026 \\ 0.059 & 0.941 \end{bmatrix}$$

Log likelihood

Hidden states	Log likelihood
2	- 97986.85
3	- 91386.03
5	- 89268.12
7	- 87121.60

The likelihood of the model improves (the value increase) with more hidden states.

Viterbi plots

Select a subsequence of one month of data on which I performed Viterbi to predict the states of the values (i.e. find the optimal hidden states assignment for the observations).

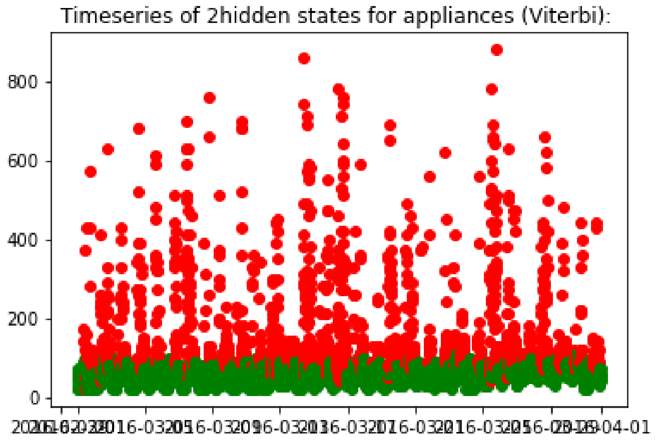
```
# creating a subsequence to perform Viterbi (1 month)
# APPL
subseq = appl_data['2016-03':'2016-04'] # tutto il mese di marzo
subseq.index = pd.to_datetime(subseq.index, format='%Y-%m-%d %H:%M:%S')
# LIGHTS
subseq2 = lights_data['2016-03':'2016-04'] # tutto il mese di marzo
subseq2.index = pd.to_datetime(subseq2.index, format='%Y-%m-%d %H:%M:%S')
#print(subseq)

# Viterbi retrieves the most probable path through a HMM that generates a certain observation
# Decode the optimal sequence of internal hidden state (Viterbi)
hidden_states = appl_hmm.predict(subseq)# Predict the hidden states of HMM
hidden_states2 = lights_hmm.predict(subseq2)
#print('Hidden states:', hidden_states)
#print('Total hidden states assigned:', len(hidden_states))
```

… and plot the timeseries data highlighting (with different colours) the hidden state assigned to each timepoint by the Viterbi algorithm.

```
                                3 HS
'''

# APPL
plt.figure(0)
plt.title(str(ncomp) + 'hidden states for appliances (Viterbi): ')
for i in range(len(hidden_states)):
    if hidden_states[i] == 1:
        plt.scatter(subseq.index[i], subseq['Appliances'][i], c='r', label='High')
    if hidden_states[i] == 2:
        plt.scatter(subseq.index[i], subseq['Appliances'][i], c='y', label='Medium')
    if hidden_states[i] == 0:
        plt.scatter(subseq.index[i], subseq['Appliances'][i], c='g', label='Low')
# LIGHTS
plt.figure(1)
plt.title(str(ncomp) + 'hidden states for lights (Viterbi): ')
for i in range(len(hidden_states2)):
    if hidden_states2[i] == 1:
        plt.scatter(subseq2.index[i], subseq2['lights'][i], c='r', label='High')
    if hidden_states2[i] == 2:
        plt.scatter(subseq2.index[i], subseq2['lights'][i], c='y', label='Medium')
    if hidden_states2[i] == 0:
        plt.scatter(subseq2.index[i], subseq2['lights'][i], c='g', label='Low')
```

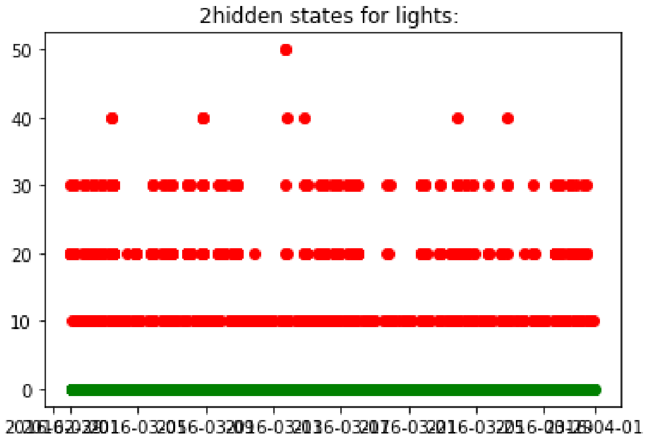
Appliances *Lights*

2 hidden states

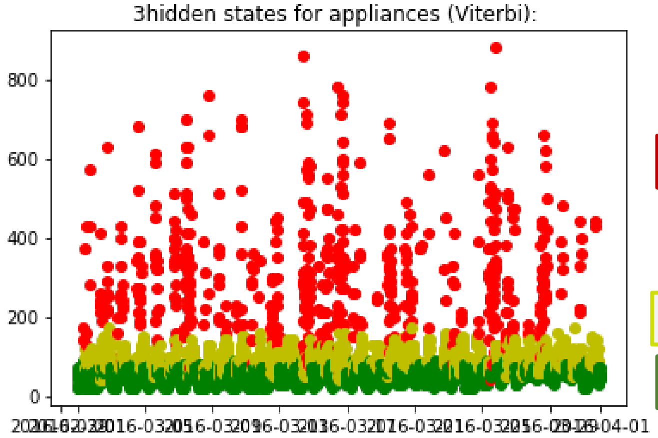


High

Low



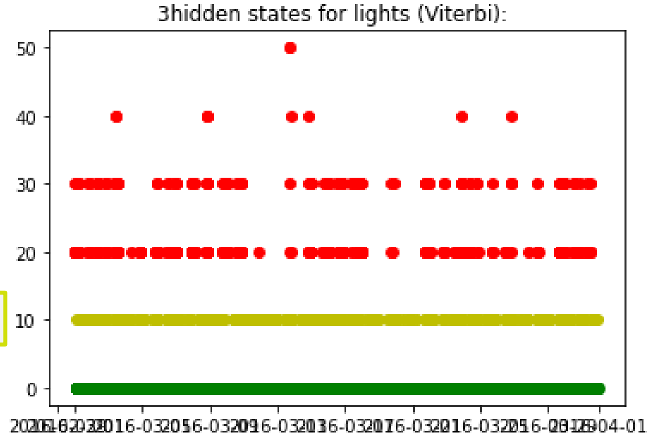
3 hidden states



High

Medium

Low



Viterbi plots (II)

Mean and variance for each hidden state:

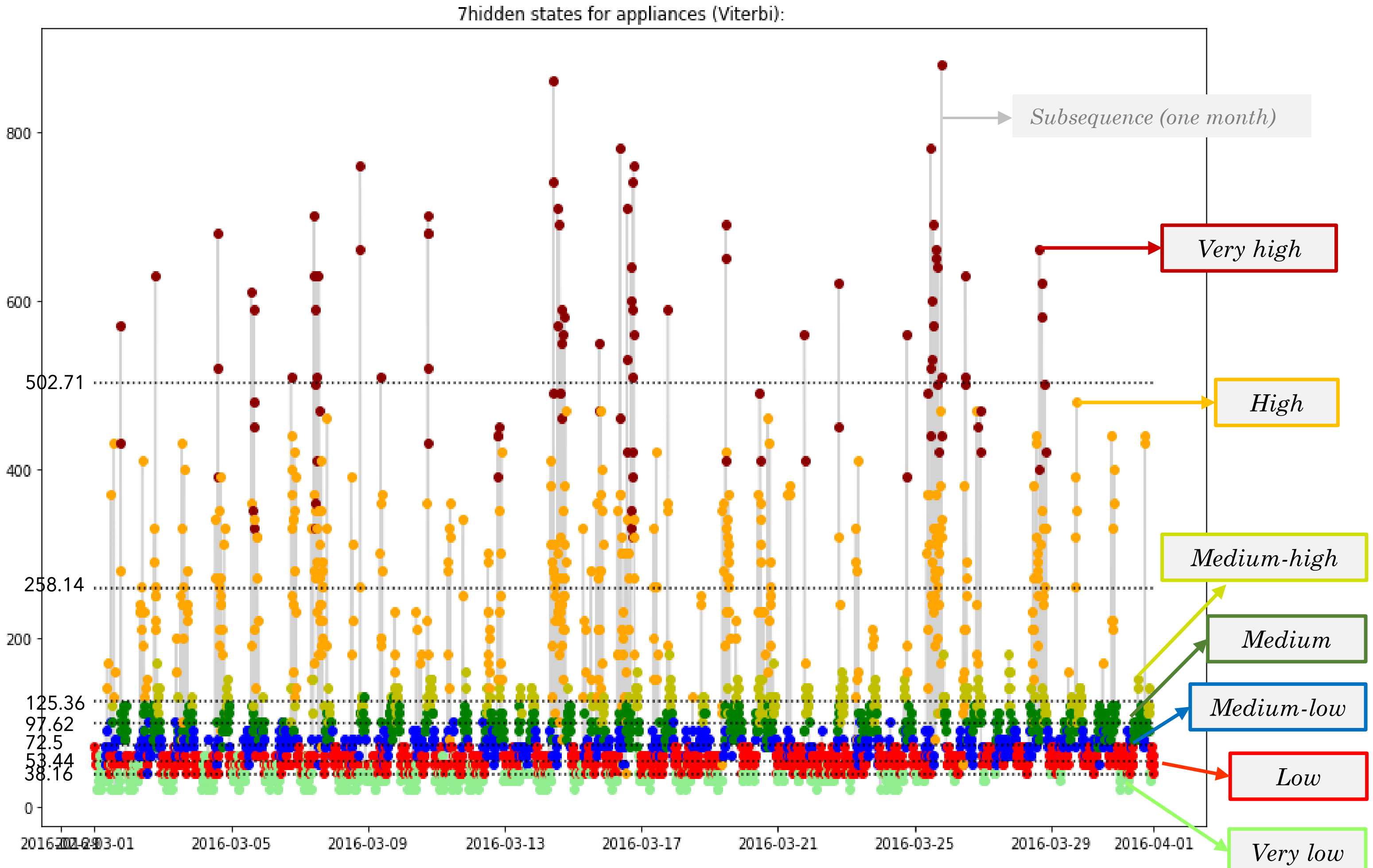
```
                                STATISTICS
'''
# Indicate the component numbers and mean and var of each component
# APPL
print("\n\nStatistics of appliances model (means and vars of each hidden state
for i in range(appl_hmm.n_components):
    print("\nHidden state: ", i+1)
    print("mean = ", round(appl_hmm.means_[i][0], 2))
    print("var = ", round(np.diag(appl_hmm.covars_[i])[0], 2))
# LIGHTS
print("\n\nStatistics of lights model (means and vars of each hidden state) :
for i in range(lights_hmm.n_components):
    print("\nHidden state: ", i+1)
    print("mean = ", round(lights_hmm.means_[i][0], 2))
    print("var = ", round(np.diag(lights_hmm.covars_[i])[0], 2))
```

Appliances				
	Hidden state	Regime	Mean	Var
5th state	6	Very low	38.16	96.12
3rd state	1	Low	53.44	103.6
0th state	4	Medium-low	72.5	50.23
2nd state	3	Medium	97.62	114.53
6th state	7	Medium-high	125.36	239.86
4th state	5	High	258.14	7731.38
1st state	2	Very-high	502.71	22468.09

Lights				
	Hidden state	Regime	Mean	Var
0th state	1	Very low	0.0	0.0
2nd state	3	Low	10.0	0.0
1st state	2	Medium-low	20.0	0.0
4th state	5	Medium	30.0	0.0
6th state	7	Medium-high	40.0	0.0
3rd state	4	High	49.97	1000.0
5th state	6	Very-high	52.72	38.05

Plotting a timeseries with seven hidden states for appliances, with means lines for each state and the real subsequence (1 month of data) in the background.

For each state, depending on the relative average of consumption, Viterbi assign a different regime. To understand and associate what hidden state is assigned for each cluster, I computed the mean value corresponding to each state and ordered the couples state-mean by mean value.



Sampling

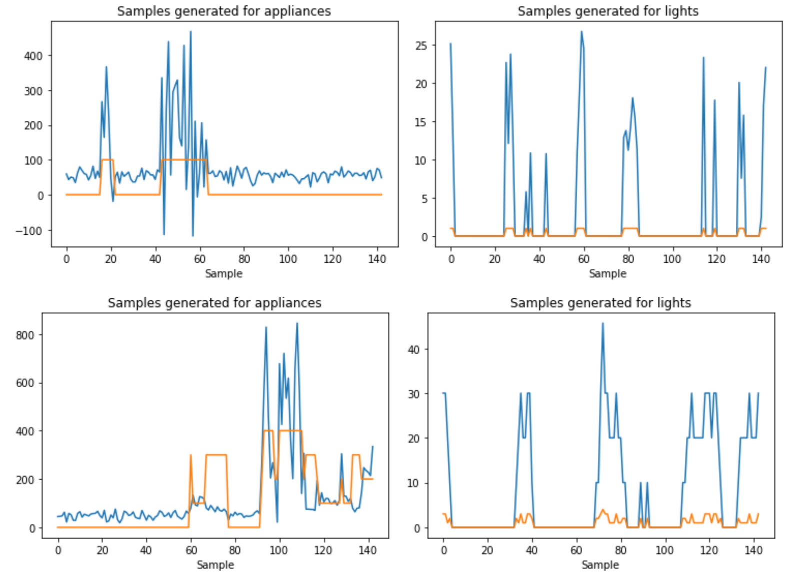
Sampling a sequence representing one day (i.e. 143 measurements) from the trained HMMs.

```
                    SAMPLING
'''
X1, Z1 = appl_hmm.sample(143) #number of measurement per day
X2, Z2 = lights_hmm.sample(143)
'''
                PROB DISTRIB COMPARISON
'''
# APPL
X1 = pd.DataFrame(X1, columns=['appl_generated'])
X1['appl_generated'].plot(kind='kde', label='Generated sample')
plt.axis([-100,1000,-0.001,0.015]) #rescaling x axes
# LIGHTS
X2 = pd.DataFrame(X2, columns=['lights_generated'])
X2['lights_generated'].plot(kind='kde', label='Generated sample')
plt.axis([-10,60,-0.005,0.3]) #rescaling x axes
```

Example of samples generated with 2 and 5 hidden state: the more are the states, the more they capture high variances of consumption.

2HS

5HS



Comparison between generated samples ad truth data

Comparison between new generated samples (blue) and real data (red).

The comparison is performed using the corresponding probability distributions (i.e. density plots).

It is clear that the new samples generated are representative of the probability distribution of the real observations.

In particular, samples distributions get closer and closer to real distributions when hidden states in the model increases, but till the 5-th.

In fact, it emerges that 5 hidden states is the number that fit the best the real probability distribution: with 7 hidden states, density plots of the new generated samples departed too much from the real data distribution.

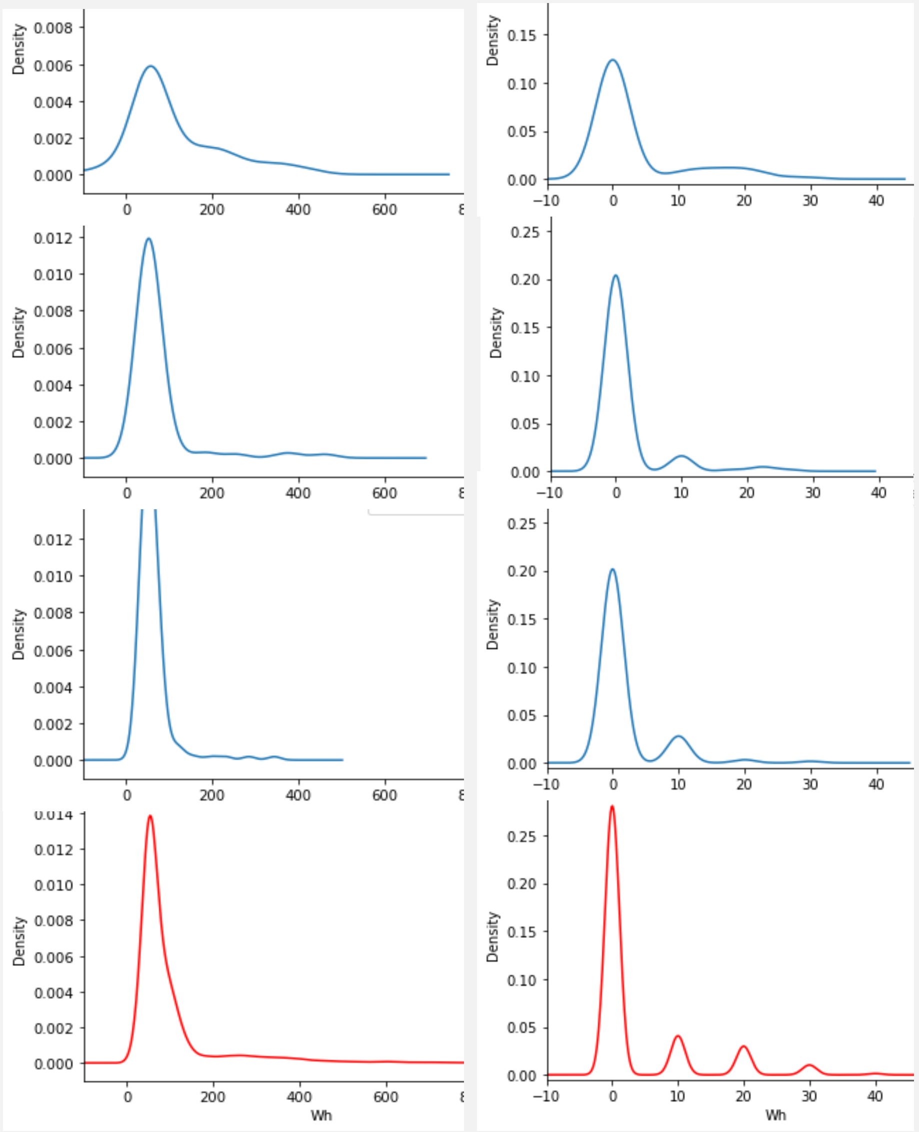
Appliances | *Lights*

2HS

3HS

5HS

Real data



UNIVERSITÀ DI PISA

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

Thanks for your attention