# Qui est Andrey Markov?

Markov était un mathématicien russe connu pour ses travaux sur les processus stochastiques. Ses premiers travaux étaient axés sur la théorie des nombres, mais après 1900, il s'est concentré sur la théorie des probabilités, à tel point qu'il a enseigné des cours après sa retraite officielle en 1905 jusqu'à son lit de mort [2]. Au cours de ses recherches, Markov a pu étendre la loi des grands nombres et le théorème central limite à appliquer à certaines séquences de variables aléatoires dépendantes, maintenant connues sous le nom de **chaînes de Markov** [1] [2]. Les chaînes de Markov sont largement applicables à la physique, l'économie, les statistiques, la biologie, etc. Deux des applications les plus connues étaient le mouvement brownien [3] et les marches aléatoires .

# Qu'est-ce que la propriété Markov?

***"... un processus aléatoire où l'avenir est indépendant du passé étant donné le présent." [4]***

Supposons un jeu de lancer de pièces simplifié avec une pièce équitable. Suspendez l'incrédulité et supposons que la propriété Markov n'est pas encore connue et nous aimerions prédire la probabilité de renverser la tête après 10 flips. Dans l'hypothèse d'une dépendance conditionnelle (la pièce a la mémoire des états passés et l'état futur dépend de la séquence des états passés), nous devons enregistrer la séquence spécifique qui mène au 11e flip et les probabilités conjointes de ces flips. Imaginez donc après 10 flips, nous avons une séquence aléatoire de têtes et de queues. La probabilité conjointe de cette séquence est de 0,5 ^ 10 = 0,0009765625. Sous la dépendance conditionnelle, la probabilité de têtes sur le prochain flip est de 0,0009765625 * 0,5 = 0,00048828125.

Est-ce la vraie probabilité de renverser la tête au 11e flip? Sûrement pas!

Nous savons que l'événement de retournement de la pièce ne dépend pas du résultat du retournement avant lui. La pièce n'a pas de mémoire. Le processus de flips successifs n'encode pas les résultats antérieurs. Chaque flip est un événement unique avec une probabilité égale de têtes ou de queues, alias conditionnellement indépendant des états passés. Ceci est la propriété Markov.

# Qu'est-ce qu'un modèle de Markov?

Une chaîne de Markov (modèle) décrit un processus stochastique où la probabilité supposée d'un ou de plusieurs états futurs ne dépend que de l'état actuel du processus et non de tous les états qui l'ont précédé ( *shocker* ).

Entrons dans un exemple simple. Supposons que vous souhaitez modéliser la probabilité future que votre chien se trouve dans l'un des trois états, compte tenu de son état actuel. Pour ce faire, nous devons spécifier l'espace d'état, les probabilités initiales et les probabilités de transition.

Imaginez que vous avez un gros chien très paresseux, nous définissons donc l' **espace d'état** comme dormir, manger ou faire caca. Nous fixerons les probabilités initiales à 35%, 35% et 30% respectivement.

```
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline

# create state space and initial state probabilities

states = ['sleeping', 'eating', 'pooping']
pi = [0.35, 0.35, 0.3]
state_space = pd.Series(pi, index=states, name='states')
print(state_space)
print(state_space.sum())
```

Voir en taille réelle

L'étape suivante consiste à définir les probabilités de transition. Ce sont simplement les probabilités de rester dans le même état ou de passer à un état différent étant donné l'état actuel.

```
# create transition matrix
# equals transition probability matrix of changing states given a state
# matrix is size (M x M) where M is number of states
```

```
q_df = pd.DataFrame(columns=states, index=states)
q_df.loc[states[0]] = [0.4, 0.2, 0.4]
q_df.loc[states[1]] = [0.45, 0.45, 0.1]
q_df.loc[states[2]] = [0.45, 0.25, .3]

print(q_df)

q = q_df.values
print('\n', q, q.shape, '\n')
print(q_df.sum(axis=1))
```

Voir en taille réelle

Maintenant que nous avons configuré les probabilités initiales et de transition, nous pouvons créer un diagramme de Markov à l'aide du package **Networkx** .

Pour ce faire, il faut un peu de réflexion flexible. Networkx crée des *graphiques* composés de *nœuds* et d' *arêtes* . Dans notre exemple de jouet, les états possibles du chien sont les nœuds et les bords sont les lignes qui relient les nœuds. Les probabilités de transition sont les *poids.* Ils représentent la probabilité de transition vers un état donné l'état actuel.

Quelque chose à noter est que networkx traite principalement des objets de dictionnaire. Cela dit, nous devons créer un objet dictionnaire qui contient nos bords et leurs poids.

```
from pprint import pprint

# create a function that maps transition probability dataframe
# to markov edges and weights

def _get_markov_edges(Q):
    edges = {}
    for col in Q.columns:
        for idx in Q.index:
            edges[(idx,col)] = Q.loc[idx,col]
    return edges

edges_wts = _get_markov_edges(q_df)
pprint(edges_wts)
```

Voir en taille réelle

Nous pouvons maintenant créer le graphique. Pour visualiser un modèle de Markov, nous devons utiliser *nx.MultiDiGraph ().* Un multidigraphe est simplement un graphe orienté qui peut avoir plusieurs arcs de sorte qu'un seul nœud peut être à la fois l'origine et la destination.

Dans le code suivant, nous créons l'objet graphique, ajoutons nos nœuds, bords et étiquettes, puis dessinons un mauvais tracé networkx lors de la sortie de notre graphique vers un fichier de points.

```
# create graph object
G = nx.MultiDiGraph()

# nodes correspond to states
G.add_nodes_from(states_)
print(f'Nodes:\n{G.nodes()}\n')

# edges represent transition probabilities
for k, v in edges_wts.items():
    tmp_origin, tmp_destination = k[0], k[1]
    G.add_edge(tmp_origin, tmp_destination, weight=v, label=v)
print(f'Edges:')
pprint(G.edges(data=True))

pos = nx.drawing.nx_pydot.graphviz_layout(G, prog='dot')
nx.draw_networkx(G, pos)

# create edge labels for jupyter plot but is not necessary
edge_labels = {(n1,n2):d['label'] for n1,n2,d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G , pos, edge_labels=edge_labels)
nx.drawing.nx_pydot.write_dot(G, 'pet_dog_markov.dot')
```

Voir en taille réelle

Maintenant, regardez le fichier dot.
Voir en taille réelle

Pas mal. Si vous suivez les bords d'un nœud, cela vous indiquera la probabilité que le chien passe à un autre état. Par exemple, si le chien dort, nous pouvons voir qu'il y a 40% de chances que le chien continue de dormir, 40% de chances que le chien se réveille et fasse caca, et 20% de chances que le chien se réveille et mange.

# Qu'est-ce qui rend un modèle de Markov caché?

Considérez une situation où votre chien agit étrangement et vous souhaitez modéliser la probabilité que le comportement de votre chien soit dû à une maladie ou simplement à un comportement excentrique lorsqu'il est en bonne santé.

Dans cette situation, le **véritable** état du chien est *inconnu* , donc **caché** pour vous. Une façon de modéliser cela est de *supposer* que le chien a **des** comportements **observables** qui représentent le véritable état caché. Voyons un exemple.

Nous créons d'abord notre espace d'état - sain ou malade. Nous supposons qu'ils sont équiprobables.

```
# create state space and initial state probabilities

hidden_states = ['healthy', 'sick']
pi = [0.5, 0.5]
state_space = pd.Series(pi, index=hidden_states, name='states')
print(state_space)
print('\n', state_space.sum())
```

Voir en taille réelle

Ensuite, nous créons notre matrice de transition pour les états cachés.

```
# create hidden transition matrix
# a or alpha
#   = transition probability matrix of changing states given a state
# matrix is size (M x M) where M is number of states

a_df = pd.DataFrame(columns=hidden_states, index=hidden_states)
a_df.loc[hidden_states[0]] = [0.7, 0.3]
a_df.loc[hidden_states[1]] = [0.4, 0.6]

print(a_df)

a = a_df.values
print('\n', a, a.shape, '\n')
print(a_df.sum(axis=1))
```

Voir en taille réelle

C'est là que ça devient un peu plus intéressant. Nous créons maintenant la matrice de probabilité d'**émission ou d'observation** . Cette matrice est de taille M x O où M est le nombre d'états cachés et O est le nombre d'états observables possibles.

La matrice d'émission nous indique la probabilité que le chien se trouve dans l'un des états cachés, étant donné l'état actuel observable.

Gardons les mêmes états observables de l'exemple précédent. Le chien peut dormir, manger ou faire caca. Pour l'instant, nous faisons de notre mieux pour remplir les probabilités.

```
# create matrix of observation (emission) probabilities
# b or beta = observation probabilities given state
```

```
# matrix is size (M x O) where M is number of states
# and O is number of different possible observations

observable_states = states

b_df = pd.DataFrame(columns=observable_states, index=hidden_states)
b_df.loc[hidden_states[0]] = [0.2, 0.6, 0.2]
b_df.loc[hidden_states[1]] = [0.4, 0.1, 0.5]

print(b_df)

b = b_df.values
print('\n', b, b.shape, '\n')
print(b_df.sum(axis=1))
```

Voir en taille réelle

Maintenant, nous créons les bords du graphique et l'objet graphique.

```
# create graph edges and weights

hide_edges_wts = _get_markov_edges(a_df)
pprint(hide_edges_wts)

emit_edges_wts = _get_markov_edges(b_df)
pprint(emit_edges_wts)
```

Voir en taille réelle

```
# create graph object
G = nx.MultiDiGraph()

# nodes correspond to states
G.add_nodes_from(hidden_states)
print(f'Nodes:\n{G.nodes()}\n')

# edges represent hidden probabilities
for k, v in hide_edges_wts.items():
    tmp_origin, tmp_destination = k[0], k[1]
    G.add_edge(tmp_origin, tmp_destination, weight=v, label=v)

# edges represent emission probabilities
for k, v in emit_edges_wts.items():
    tmp_origin, tmp_destination = k[0], k[1]
    G.add_edge(tmp_origin, tmp_destination, weight=v, label=v)

print(f'Edges:')
pprint(G.edges(data=True))
```

```
pos = nx.drawing.nx_pydot.graphviz_layout(G, prog='neato')
nx.draw_networkx(G, pos)

# create edge labels for jupyter plot but is not necessary
emit_edge_labels = {(n1,n2):d['label'] for n1,n2,d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G , pos, edge_labels=emit_edge_labels)
nx.drawing.nx_pydot.write_dot(G, 'pet_dog_hidden_markov.dot')
```

Voir en taille réelle

Voir en taille réelle

The hidden Markov graph is a little more complex but the principles are the same. For example, you would expect that if your dog is eating there is a high probability that it is healthy (60%) and a very low probability that the dog is sick (10%).

Now, what if you needed to discern the health of your dog over time given a sequence of observations?

```
# observation sequence of dog's behaviors
# observations are encoded numerically

obs_map = {'sleeping':0, 'eating':1, 'pooping':2}
obs = np.array([1,1,2,1,0,1,2,1,0,2,2,0,1,0,1])

inv_obs_map = dict((v,k) for k, v in obs_map.items())
obs_seq = [inv_obs_map[v] for v in list(obs)]

print( pd.DataFrame(np.column_stack([obs, obs_seq]),
                columns=['Obs_code', 'Obs_seq']) )
```

View fullsize

Using the **Viterbi** algorithm we can identify the most likely sequence of hidden states given the sequence of observations.

De haut niveau, l'algorithme de Viterbi s'incrémente sur chaque pas de temps, trouvant la probabilité **maximale** de tout chemin qui arrive à l'état **i** au temps **t** , qui a ***également*** les observations correctes pour la séquence jusqu'au temps **t** .

L'algorithme garde également une trace de l'état avec la probabilité la plus élevée à chaque étape. À la fin de la séquence, l'algorithme effectuera une itération en arrière en sélectionnant l'état qui "a gagné" à chaque pas de temps, et en créant ainsi le chemin le plus probable, ou la séquence probable d'états cachés qui ont conduit à la séquence d'observations.

```
# define Viterbi algorithm for shortest path
# code adapted from Stephen Marsland's, Machine Learning An Algorithmic Perspective, Vol. 2
# https://github.com/alexsosn/MarslandMLAlgo/blob/master/Ch16/HMM.py
```

```
def viterbi(pi, a, b, obs):

    nStates = np.shape(b)[0]
    T = np.shape(obs)[0]

    # init blank path
    path = np.zeros(T)
    # delta --> highest probability of any path that reaches state i
    delta = np.zeros((nStates, T))
    # phi --> argmax by time step for each state
    phi = np.zeros((nStates, T))

    # init delta and phi
    delta[:, 0] = pi * b[:, obs[0]]
    phi[:, 0] = 0

    print('\nStart Walk Forward\n')
    # the forward algorithm extension
    for t in range(1, T):
        for s in range(nStates):
            delta[s, t] = np.max(delta[:, t-1] * a[:, s]) * b[s, obs[t]]
            phi[s, t] = np.argmax(delta[:, t-1] * a[:, s])
            print('s={s} and t={t}: phi[{s}, {t}] = {phi}'.format(s=s, t=t, phi=phi[s, t]))

    # find optimal path
    print('-'*50)
    print('Start Backtrace\n')
    path[T-1] = np.argmax(delta[:, T-1])
    #p('init path\n    t={} path[{}-1]={}\n'.format(T-1, T, path[T-1]))
    for t in range(T-2, -1, -1):
        path[t] = phi[path[t+1], [t+1]]
        #p(' '*4 + 't={t}, path[{t}+1]={path}, [{t}+1]={i}'.format(t=t, path=path[t+1],
i=[t+1]))
        print('path[{}] = {}'.format(t, path[t]))

    return path, delta, phi

path, delta, phi = viterbi(pi, a, b, obs)
print('\nsingle best state path: \n', path)
print('delta:\n', delta)
print('phi:\n', phi)
```

Voir en taille réelle

Jetons un coup d'oeil au résultat.

```
state_map = {0:'healthy', 1:'sick'}
```

```
state_path = [state_map[v] for v in path]

(pd.DataFrame()
 .assign(Observation=obs_seq)
 .assign(Best_Path=state_path))
```

Voir en taille réelle

# Un modèle de Markov caché pour la détection de régime

Vous vous demandez probablement maintenant comment appliquer ce que nous avons appris sur les modèles de Markov cachés à la finance quantitative.

Considérez que le plus grand obstacle auquel nous sommes confrontés lorsque nous essayons d'appliquer des techniques prédictives aux rendements des actifs est les séries chronologiques non stationnaires. En bref, cela signifie que la moyenne et la volatilité attendues des rendements des actifs évoluent avec le temps.

La plupart des modèles de séries chronologiques supposent que les données sont stationnaires. Il s'agit d'une faiblesse majeure de ces modèles.

Au lieu de cela, formulons le problème différemment. Nous savons que les séries chronologiques présentent des périodes temporaires où les moyennes et les variances attendues sont stables dans le temps. Ces périodes ou *régimes* peuvent être assimilés à *des états cachés* .

If that's the case, then all we need are observable variables whose behavior allows us to infer the true hidden state(s). If we can better estimate an asset's most likely regime, including the associated means and variances, then our predictive models become more adaptable and will likely improve. We can also become better risk managers as the estimated regime parameters gives us a great framework for better scenario analysis.

In this example, the observable variables I use are: the underlying asset returns, the Ted Spread, the 10 year - 2 year constant maturity spread, and the 10 year - 3 month constant maturity spread.

```
import pandas as pd
import pandas_datareader.data as web
import sklearn.mixture as mix

import numpy as np
import scipy.stats as scs
```

```
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from matplotlib.dates import YearLocator, MonthLocator
%matplotlib inline

import seaborn as sns
import missingno as msno
from tqdm import tqdm
p=print
```

Using pandas we can grab data from Yahoo Finance and FRED.

```
# get fed data

f1 = 'TEDRATE' # ted spread
f2 = 'T10Y2Y' # constant maturity ten yer - 2 year
f3 = 'T10Y3M' # constant maturity 10yr - 3m

start = pd.to_datetime('2002-01-01')
end = pd.datetime.today()

mkt = 'SPY'
MKT = (web.DataReader([mkt], 'yahoo', start, end)['Adj Close']
       .rename(columns={mkt:mkt})
       .assign(sret=lambda x: np.log(x[mkt]/x[mkt].shift(1)))
       .dropna())

data = (web.DataReader([f1, f2, f3], 'fred', start, end)
        .join(MKT, how='inner')
        .dropna()
       )

p(data.head())

# gives us a quick visual inspection of the data
msno.matrix(data)
```

View fullsize

Next we will use the **sklearn's GaussianMixture** to fit a model that estimates these regimes. We will explore *mixture models* in more depth in part 2 of this series. The important takeaway is that mixture models implement a closely related unsupervised form of density estimation. It makes use of the expectation-maximization algorithm to estimate the means and covariances of the hidden states (regimes). For now, it is ok to think of it as a magic button for guessing the transition and emission probabilities, and most likely path.

We have to specify the number of components for the mixture model to fit to the time series. In this example the components can be thought of as regimes. We will arbitrarily classify the regimes as High, Neutral and Low Volatility and set the number of components to three.

```
# code adapted from http://hmmlearn.readthedocs.io
# for sklearn 18.1

col = 'sret'
select = data.ix[:].dropna()

ft_cols = [f1, f2, f3, 'sret']
X = select[ft_cols].values

model = mix.GaussianMixture(n_components=3,
                            covariance_type="full",
                            n_init=100,
                            random_state=7).fit(X)

# Predict the optimal sequence of internal hidden state
hidden_states = model.predict(X)

print("Means and vars of each hidden state")
for i in range(model.n_components):
    print("{0}th hidden state".format(i))
    print("mean = ", model.means_[i])
    print("var = ", np.diag(model.covariances_[i]))
    print()

sns.set(font_scale=1.25)
style_kwds = {'xtick.major.size': 3, 'ytick.major.size': 3,
              'font.family':u'courier prime code', 'legend.frameon': True}
sns.set_style('white', style_kwds)

fig, axs = plt.subplots(model.n_components, sharex=True, sharey=True, figsize=(12,9))
colors = cm.rainbow(np.linspace(0, 1, model.n_components))

for i, (ax, color) in enumerate(zip(axs, colors)):
    # Use fancy indexing to plot data in each state.
    mask = hidden_states == i
    ax.plot_date(select.index.values[mask],
                 select[col].values[mask],
                 ".-", c=color)
    ax.set_title("{0}th hidden state".format(i), fontsize=16, fontweight='demi')

    # Format the ticks.
```

```
    ax.xaxis.set_major_locator(YearLocator())
    ax.xaxis.set_minor_locator(MonthLocator())
    sns.despine(offset=10)

plt.tight_layout()
fig.savefig('Hidden Markov (Mixture) Model_Regime Subplots.png')
```

View fullsize

In the above image, I've highlighted each regime's daily expected mean and variance of SPY returns. It appears the 1th hidden state is our low volatility regime. Note that the 1th hidden state has the largest expected return and the smallest variance.The 0th hidden state is the neutral volatility regime with the second largest return and variance. Lastly the 2th hidden state is high volatility regime. We can see the expected return is negative and the variance is the largest of the group.
View fullsize

```
sns.set(font_scale=1.5)
states = (pd.DataFrame(hidden_states, columns=['states'], index=select.index)
          .join(select, how='inner')
          .assign(mkt_cret=select.sret.cumsum())
          .reset_index(drop=False)
          .rename(columns={'index':'Date'}))
p(states.head())

sns.set_style('white', style_kwds)
order = [0, 1, 2]
fg = sns.FacetGrid(data=states, hue='states', hue_order=order,
                   palette=scolor, aspect=1.31, size=12)
fg.map(plt.scatter, 'Date', mkt, alpha=0.8).add_legend()
sns.despine(offset=10)
fg.fig.suptitle('Historical SPY Regimes', fontsize=24, fontweight='demi')
fg.savefig('Hidden Markov (Mixture) Model_SPY Regimes.png')
```

View fullsize

Voici le tableau des prix SPY avec les régimes codés par couleur superposés.
Voir en taille réelle

# Conclusion

Dans cet article, nous avons discuté des concepts de la propriété Markov, des modèles Markov et des modèles Markov cachés. Nous avons utilisé le package networkx pour créer des diagrammes de chaîne de Markov et GaussianMixture de sklearn pour estimer les régimes historiques. Dans la partie 2, nous discuterons plus en détail des modèles de mélange. Pour des informations plus détaillées, je recommanderais de consulter les références. Setosa.io est particulièrement utile pour combler les lacunes dues aux visualisations hautement interactives.

# Références


1. https://en.wikipedia.org/wiki/Andrey_Markov
2. https://www.britannica.com/biography/Andrey-Andreyevich-Markov
3. https://www.reddit.com/r/explainlikeimfive/comments/vbxfk/eli5_brownian_motion_and_what_it_has_to_do_with/
4. http://www.math.uah.edu/stat/markov/Introduction.html
5. http://setosa.io/ev/markov-chains/
6. http://www.cs.jhu.edu/~langmea/resources/lecture_notes/hidden_markov_models.pdf
7. https://github.com/alexsosn/MarslandMLAlgo/blob/master/Ch16/HMM.py
8. http://hmmlearn.readthedocs.io





- INTRO TO EXPECTATION-MAXIMIZATION, ...
- COMPRENDRE LES VARIABLES CACHÉES AVEC ...

**Adrian** il y a 10 mois · 0 J'aime

Quelqu'un d'autre a cette erreur? NameError

Traceback (dernier appel le plus récent)
<ipython-input-7-0d427f23a0c7> dans <module>
----> 1 states_

NameError: le nom 'states_' n'est pas défini

J'utilise python 3.7.3


**Gary Bake** il y a 2 mois · 0 J'aime

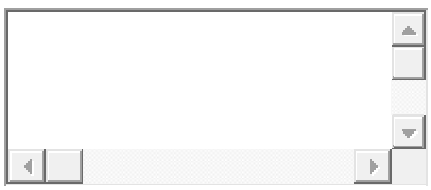
On dirait une erreur. J'ai utilisé des "états" et cela fonctionne très bien.

```
G.add_nodes_from(states)
```

**Abarni** Il y a un an · 0 J'aime

Excellent article. Voici le lien vers une autre introduction:

http://www.adeveloperdiary.com/data-science/machine-learning/introduction-to-hidden-markov-model/



**Bravid** Il y a un an · 0 J'aime

Salut

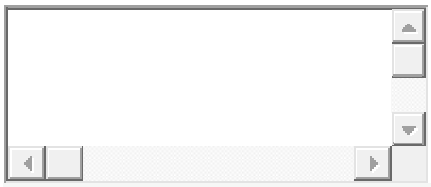
Très petite correction de l'implémentation de viterbi - potentiellement comme un changement de comportement à travers les versions de python (je suis sur 3.6.7 avec numpy 1.15.3), l'initialisation de path doit spécifier le type de données comme int ou la trace arrière échoue avec un tableau erreur de découpage.

# init blank path

```
path = np.zeros(T,dtype=int)
```

Grand tutoriel - m'aide énormément. Je vous remercie

David



**mouna** il y a 2 ans · 0 J'aime

Bonjour

s'il vous plaît pourriez-vous dire comment avez-vous estimé les probabilités pi, a et b dans le modèle de Markov caché avec l'algorithme de Viterbi ??? et merci

**Alex** il y a 2 ans · 0 J'aime

Très beau post! J'ai beaucoup appris de, merci beaucoup!

Quelle est exactement la relation entre le modèle de mélange gaussien et HMM?

Je ne sais pas comment les probabilités de transition entre les états sont estimées dans cet exemple.

D'après ma compréhension, le GMM lui-même n'a pas de notion de temps ou de séquence (?)

Je vous remercie!

**ann** il y a 2 ans · 0 J'aime

MKT = (web.DataReader ([mkt], 'yahoo', début, fin) ['Adj Close']

.rename (colonnes = {mkt: mkt})

.assign (sret = lambda x: np.log (x [ mkt] / x [mkt] .shift (1)))

.dropna ())

Je pense que le .shift (1) a de mauvais arguments, il devrait être -1

**Ludovic** il y a 2 ans · 0 J'aime

Très gentil merci. Dans l'attente de la partie 3

Petite faute de frappe possible: "Par exemple, vous vous attendez à ce que si votre chien mange, il y a une forte probabilité qu'il soit en bonne santé (60%) et une très faible probabilité que le chien soit malade (10%)."
Cela ne devrait-il pas se lire à 86% / 14%?

il y a 2 ans · 0 J'aime

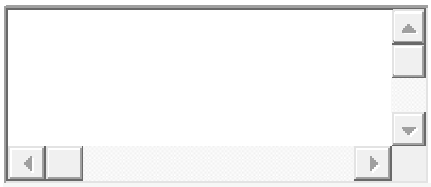
Très beau message !!

**Simone** il y a 3 ans · 0 J'aime

Quelques questions techniques sur le résultat.

Pourquoi prendre la dernière colonne du résultat du tableau "Moyennes et variables de chaque état caché". Que représente la 3 première colonne

Graphes - l'ordre des états n, n + 1, n + 2 sont dans des ordres différents de l'exemple donné lorsque j'ai implémenté le code



**Shubham Jain** il y a 3 ans · 0 J'aime

Très bien expliqué.

**Stone Richnau** il y a 3 ans · 0 J'aime

Excellent article! Une petite faute de frappe dans

fg = sns.FacetGrid (data = states, hue = 'states', hue_order = order,

palette = scolor, aspect = 1.31, size = 12)

écolier devrait probablement être des couleurs

Merci

Pierre

**Ming** il y a 3 ans · 0 J'aime

Vous avez utilisé des informations futures lorsque vous lancez toute la série de retours dans HMM.

**Brian Christopher** il y a 3 ans · 0 J'aime

Salut Ming,

Il s'agissait d'un exemple jouet de classification des régimes historiques. Aucune prédiction n'a été faite, il est donc correct que le modèle ait été formé sur toute la série chronologique. Dans la partie 3 de cette série, je montrerai une méthode que j'ai recherchée qui utilise des régimes historiques pour la prédiction sans biais d'anticipation.

**Dmitry** il y a 3 ans · 0 J'aime

Il ne semble pas y avoir de différence entre les régimes de volatilité faible et moyenne (à en juger par le graphique S&P). Pourquoi ne pas utiliser seulement deux états?

**Brian Christopher** il y a 3 ans · 0 J'aime

Dmitry,

Vous pouvez utiliser 2 états si vous le souhaitez. Parce que les modèles de mélange sont une forme d'apprentissage non supervisé, il n'y a pas de nombre unique de composants qui soit toujours le meilleur. Des recherches sont en cours sur les meilleures pratiques en matière de sélection des composants, dont j'aborderai certaines dans les deux prochains articles de la série. Si vous voulez prendre une longueur d'avance, je vous recommande de consulter la documentation sklearn pour les modèles de mélange.

OBTENIR LES MISES À JOUR!

## Souscrire

Inscrivez-vous avec votre adresse e-mail pour recevoir des nouvelles et des mises à jour.

S'INSCRIRE

Nous respectons votre vie privée.

LIEN AFFILIÉ

Blog RSS

@BLACKARBSCEO

**BCR**

Génération de données ETF synthétiques (partie 2) - Modèles de mélanges gaussiens https://t.co/iKCppy28rX

14 août 2019 08:28

**BCR**

Un portefeuille Dead Simple à 2 actifs qui écrase le S & amp; P500 (Partie 3) https://t.co/S4fClql8Mq

3 mai 2019, 12:49 PM

**BCR**

RT @ jakevdp : Une fonctionnalité peu connue dans IPython et Jupyter que je trouve utile à l'occasion: achèvement de caractères génériques d'attribut nam… https://t.co/tIvLtJXsTx

13 mars 2019 à 10h49

## ARCHIVER