Sejal Jaiswal
31 décembre 2019

doit lire
python
+ 1

Chaînes de Markov en Python: Tutoriel pour débutant

Découvrez les chaînes de Markov, leurs propriétés, les matrices de transition et implémentez-en une vous-même en Python!

Une chaîne de Markov est un système mathématique généralement défini comme un ensemble de variables aléatoires, qui passent d'un état à un autre selon certaines règles probabilistes. Ces transitions satisfont les **Propriété Markov,** qui stipule que la probabilité de transition vers un état particulier dépend uniquement de l'état actuel et du temps écoulé, et non de la séquence d'état qui l'a précédé. Cette caractéristique unique des processus de Markov les rend
sans mémoire.

Dans ce tutoriel, vous découvrirez quand vous pouvez utiliser des chaînes markov, ce que Chaîne de Markov à temps discret est. Vous découvrirez également les composants nécessaires à la création d'un (Discretetime) Modèle de chaîne Markov et certaines de ses propriétés communes. Ensuite, vous allez implémenter un tel modèle simple avec Python en utilisant son engourdi et Aléatoire bibliothèques. Vous apprendrez également certaines des façons de représenter une chaîne de Markov comme un diagramme d'état et matrice de transition .

Vous voulez aborder plus de sujets de statistiques avec Python? Découvrez DataCamp's Pensée statistique en Python cours!

Passons à la transition ...

Pourquoi Markov Chains?

Les chaînes de Markov ont une utilisation prolifique en mathématiques. Ils sont largement employés en économie, en théorie des jeux, en théorie de la communication, en génétique et en finance. Ils se posent largement dans les statistiques statistiques spécialement bayésiennes et les contextes théoriques de l'information. Quand il s'agit de problèmes du monde réel, ils sont utilisés pour postuler des solutions pour étudier les systèmes de régulateur de vitesse dans les véhicules à moteur, les files d'attente ou les files de clients arrivant à un aéroport, les taux de change des devises, etc. pour le moteur de recherche Internet Google, est basé sur un processus de Markov. Simulateur Subreddit de Reddit est un subreddit entièrement automatisé qui génère des soumissions et des commentaires aléatoires en utilisant des chaînes de Markov, donc cool!

Chaîne de Markov

Une chaîne de Markov est un processus aléatoire avec la propriété Markov. Un processus aléatoire ou souvent appelé propriété stochastique est un objet mathématique défini comme une collection de variables aléatoires. Une chaîne de Markov a soit un espace d'état discret (ensemble de valeurs possibles des variables aléatoires) soit un ensemble d'index discret (représentant souvent le temps) - étant donné le fait, il existe de nombreuses variations pour une chaîne de Markov. Habituellement, le terme «chaîne de Markov» est réservé à un processus avec un ensemble discret d'heures, c'est-à-dire une chaîne de Markov à temps discret (DTMC).

Chaîne de Markov à temps discret

Une chaîne de Markov à temps discret implique un système qui se trouve dans un certain état à chaque étape, l'état changeant de façon aléatoire entre les étapes. Les étapes sont souvent considérées comme des moments dans le temps (mais vous pourriez aussi bien faire référence à la distance physique ou à toute autre mesure discrète). Une chaîne de Markov à temps discret est une séquence de variables aléatoires $X_1, X_2, X_3, \ldots$ avec la propriété Markov, de sorte que la probabilité de passer à l'état suivant dépend uniquement de l'état actuel et non des états précédents. Mettre ceci est une formule mathématique probabiliste:

$$Pr(X_{n+1} = x \mid X_1 = X_1, X_2 = X_2, \ldots, X_n = X_n) = Pr(X_{n+1} = x \mid X_n = X_n)$$

Comme vous pouvez le voir, la probabilité de X_{n+1} ne dépend que de la probabilité de X_n qui le précède. Ce qui signifie que la connaissance de l'état précédent est tout ce qui est nécessaire pour déterminer la distribution de probabilité de l'état actuel, satisfaisant la règle de l'indépendance conditionnelle (ou autrement dit: il suffit de connaître l'état actuel pour déterminer l'état suivant).

Les valeurs possibles de X_i former un ensemble dénombrable S appelé **territoire de l'État** de la chaîne. L'espace d'état peut être n'importe quoi: lettres, chiffres, scores de basket-ball ou conditions météorologiques. Alors que le paramètre temporel est généralement discret, l'espace d'état d'une chaîne de Markov temporelle discrète n'a pas de restrictions largement acceptées et fait plutôt référence à un processus sur un espace d'état arbitraire. Cependant, de nombreuses applications des chaînes de Markov utilisent des espaces d'états finis ou comptables infinis, car ils ont une analyse statistique plus simple.

Modèle

Une chaîne de Markov est représentée à l'aide d'un automate probabiliste (cela ne semble compliqué!). Les changements d'état du système sont appelés transitions. Les probabilités associées à divers changements d'état sont appelées probabilités de transition. Un automate probabiliste inclut la probabilité d'une transition donnée dans la fonction de transition, la transformant en une matrice de transition.

Vous pouvez le considérer comme une séquence de graphes dirigés, où les bords du graphe n sont étiquetés par les probabilités de passer d'un état au temps n aux autres états au temps n + 1, $Pr(X_{n+1} = x \mid X_n = X_n)$. Vous pouvez lire ceci comme la probabilité d'aller à l'état X_{n+1} valeur donnée de l'état X_n.
Les mêmes informations sont représentées par **matrice de transition** du temps n au temps n + 1. Chaque état de l'espace d'état est inclus une fois sous forme de ligne et de nouveau sous forme de colonne, et chaque cellule de la matrice vous indique la probabilité de passer de l'état de sa ligne à l'état de sa colonne.

Si la chaîne de Markov a N états possibles, la matrice sera une matrice N x N, de sorte que l'entrée (I, J) est la probabilité de transition de l'état I à l'état J.En outre, la matrice de transition doit être une matrice stochastique, une matrice dont les entrées dans chaque ligne doivent correspondre exactement

1. Pourquoi? Puisque chaque ligne représente sa propre distribution de probabilité.

Ainsi, le modèle est caractérisé par un espace d'états, une matrice de transition décrivant les probabilités de transitions particulières et un état initial à travers l'espace d'états, donné dans la distribution initiale.

Beaucoup de mots hein?

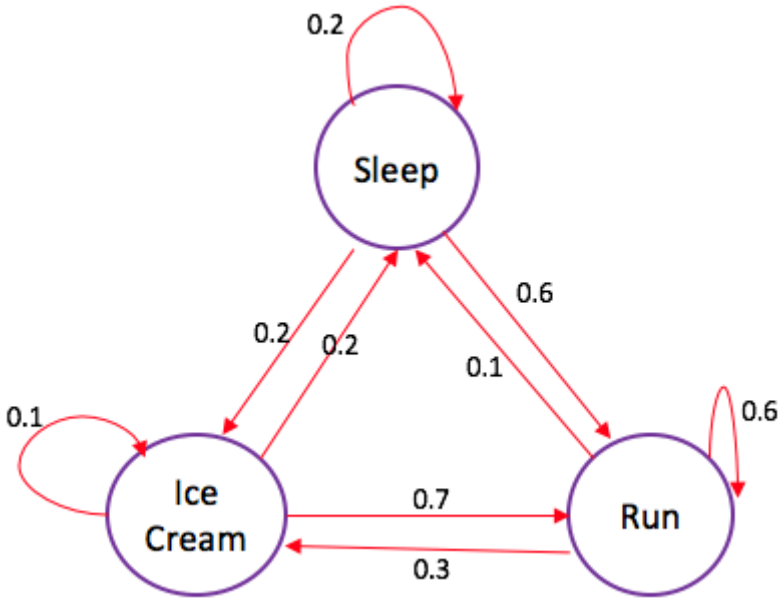
Voyons un exemple simple pour comprendre les concepts:

Quand Cj est triste, ce qui n'est pas très courant: elle va courir, goûte la glace ou fait une sieste.

D'après les données historiques, si elle a passé une triste journée à dormir. Le lendemain, il est probable à 60% qu'elle ira courir, à 20% elle restera au lit le lendemain et à 20% de chances qu'elle se raccroche à la crème glacée.

Quand elle est triste et va courir, il y a 60% de chances qu'elle aille courir le lendemain, 30% qu'elle se gorge de glace et seulement 10% de chances qu'elle passe son sommeil le lendemain.

Enfin, quand elle s'adonne à la crème glacée un jour triste, il y a à peine 10% de chances qu'elle continue d'avoir de la glace le lendemain également, 70% elle est susceptible d'aller courir et 20% de chance qu'elle passe à dormir le lendemain journée.



La chaîne de Markov représentée dans le diagramme d'état a 3 états possibles: sommeil, course, glace. Ainsi, la matrice de transition sera une matrice 3 x 3. Remarquez que les flèches sortant d'un état totalisent toujours exactement 1, de même les entrées de chaque ligne de la matrice de transition doivent totaliser exactement 1 - représentant la distribution de probabilité. Dans la matrice de transition, les cellules font le même travail que les flèches dans le diagramme d'état.

CURRENT STATE

NEXT STATE

	SLEEP	RUN	ICE CREAM
SLEEP	0.2	0.6	0.2
RUN	0.1	0.6	0.3
ICE CREAM	0.2	0.7	0.1

Maintenant que vous avez vu l'exemple, cela devrait vous donner une idée des différents concepts liés à une chaîne de Markov. Mais, comment et où pouvez-vous utiliser ces théories dans la vraie vie?

Avec l'exemple que vous avez vu, vous pouvez maintenant répondre à des questions telles que: "A partir de l'état: sommeil, quelle est la probabilité que Cj s'exécute (état: exécuter) à la fin d'une triste durée de 2 jours?"

Travaillons celui-ci: Pour passer de l'état: sommeil à état: exécuter, Cj doit soit rester à l'état: dormir le premier mouvement (ou jour), puis passer à l'état: exécuter le prochain (deuxième) mouvement (0,2 ·

0,6); ou passer à l'état: courir le premier jour puis y rester le deuxième (0,6 · 0,6) ou elle pourrait passer à l'état: crème glacée au premier mouvement puis à l'état: exécuter dans le second (0,2 · 0,7). Donc, la probabilité: ((0,2 · 0,6) + (0,6 · 0,6) + (0,2 ·

0,7)) = 0,62. Donc, nous pouvons maintenant dire qu'il y a 62% de chances que Cj passe à l'état: courir après deux jours de tristesse, si elle a commencé dans l'état: dormir.

J'espère que cela vous a donné une idée des différentes questions auxquelles vous pouvez répondre en utilisant un réseau Markov Chain.

De plus, avec cela à l'esprit, il devient plus facile de comprendre certaines propriétés importantes des chaînes de Markov:

- Réductibilité: une chaîne de Markov est dite irréductible s'il est possible d'accéder à n'importe quel état à partir de n'importe quel état. En d'autres termes, une chaîne de Markov est irréductible s'il existe une chaîne d'étapes entre deux états quelconques qui a une probabilité positive.
- Périodicité: un état dans une chaîne de Markov est périodique si la chaîne ne peut revenir à l'état qu'à des multiples d'un entier supérieur à 1. Ainsi, à partir de l'état «i», la chaîne ne peut revenir à «i» qu'à des multiples de la période 'k', et k est le plus grand entier de ce type. L'état 'i' est apériodique si k = 1 et périodique si k> 1.

- Transitoire et récurrence: un état «i» est dit transitoire si, étant donné que nous commençons dans l'état «i», il y a une probabilité non nulle que nous ne revenions jamais à «i». L'état i est récurrent (ou persistant) s'il n'est pas transitoire. Un état récurrent est connu sous le nom de récurrent positif s'il est censé revenir dans un nombre fini d'étapes et récurrent nul dans le cas contraire.

- Ergodicité: un état «i» est dit ergodique s'il est apériodique et positif récurrent. Si tous les états d'une chaîne de Markov irréductible sont ergodiques, alors la chaîne est dite ergodique.

- État absorbant: un état i est appelé absorbant s'il est impossible de quitter cet état. Par conséquent, l'état «i» **absorbe si $p_{ii} = 1$ et $p_{ij} = 0$ pour $i \neq j$. Si chaque état peut atteindre un état absorbant, alors la chaîne de Markov est une** chaîne de Markov absorbante.

Pointe: si vous voulez également voir une explication visuelle des chaînes de Markov, assurez-vous de visiter cette page .

Chaînes de Markov en Python

Essayons de coder l'exemple ci-dessus en Python. Et bien que dans la vraie vie, vous utiliseriez probablement une bibliothèque qui code les chaînes de Markov de manière beaucoup plus efficace, le code devrait vous aider à démarrer ...

Importons d'abord certaines des bibliothèques que vous utiliserez.

```
import numpy as np import random as
rm
```

Définissons maintenant les états et leur probabilité: la matrice de transition. Rappelez-vous, la matrice va être une matrice 3 X 3 puisque vous avez trois états. De plus, vous devrez définir les chemins de transition, vous pouvez également le faire en utilisant des matrices.

```
#   L'espace d'états
states = ["Sleep", "Icecream", "Run"]

#   Séquences d'événements possibles
transitionName = [["SS", "SR", "SI"], ["RS", "RR", "RI"], ["IS", "IR", "II"]]

#   Matrice de probabilités (matrice de transition)
transitionMatrix = [[0.2,0.6,0.2], [0.1,0.6,0.3], [0.2,0.7,0.1]]
```

Oh, assurez-vous toujours que les probabilités se résument à 1. Et cela ne fait pas de mal de laisser des messages d'erreur, au moins lors du codage!

```
si
sum (transitionMatrix [0]) + sum (transitionMatrix [1]) + sum (transitionMatrix [1])! = 3:

        print ("Quelque part, quelque chose s'est mal passé. Matrice de transition, peut-être?") else: print ("Tout va bien se passer, vous devez continuer !!;)")) Tout va bien se passer, vous devez continuer !! ;)
```

Maintenant, codons la vraie chose. Vous utiliserez le `numpy.random.choice` pour générer un échantillon aléatoire à partir de l'ensemble des transitions possibles. Bien que la plupart de ses arguments soient explicites, le `p` Pourrait ne pas être. Il s'agit d'un argument facultatif qui vous permet de saisir la distribution de probabilité de l'ensemble d'échantillonnage, qui est la matrice de transition dans ce cas.

```
#   Une fonction qui implémente le modèle de Markov pour prévoir l'état / l'humeur. def activity_forecast (jours):

        #   Choisissez l'état de départ activityToday =
        "Sleep"
        print ("État de départ:" + activitéAujourd'hui)
        #   Doit stocker la séquence des états pris. Donc, cela n'a que l'état de départ pour l'instant.

        activityList = [activityToday]
```

```
        i = 0
        #    Pour calculer la probabilité de ActivityList prob = 1

        pendant que i! = jours:
                si activityToday == "Sleep":
                        changement =
np.random.choice (transitionName [0], replace = True, p = transitionMatrix [0])
                        si change == "SS":
                                prob = prob * 0.2 activityList.append ("Sleep") passe


                        elif change == "SR":
                                prob = prob * 0.6 activityToday = "Run"
                            activityList.append ("Run") sinon:


                                prob = prob * 0.2 activityToday = "Icecream"
                                activityList.append ("Icecream")

                elif activityToday == "Exécuter":
                        changement =
np.random.choice (transitionName [1], replace = True, p = transitionMatrix [1])
                        si changement == "RR":
                                prob = prob * 0.5 activitéList.append ("Exécuter")
                                passe

                        changement elif == "RS":
                                prob = prob * 0.2 activityToday = "Sleep"
                            activityList.append ("Sleep") sinon:


                                prob = prob * 0.3 activityToday = "Icecream"
                                activityList.append ("Icecream")

                elif activityToday == "Icecream":
                        changement =
np.random.choice (transitionName [2], replace = True, p = transitionMatrix [2])
                        si changement == "II":
                                prob = prob * 0,1
                                pass activityList.append ("Icecream")

                        changement elif == "EST":
                                prob = prob * 0.2 activityToday = "Sleep"
                            activityList.append ("Sleep") sinon:


                                prob = prob * 0.7 activityToday = "Run"
                                activityList.append ("Run")

                i + = 1
        print ("États possibles:" + str (ActivityList))
        print ("État final après" + str (jours) + "jours:" + activityToday) print ("Probabilité de la séquence d'états possible:" + str (prob))


#    Fonction qui prévoit l'état possible pour les 2 prochains jours activity_forecast (2) État de démarrage: veille


États possibles: ['Sleep', 'Sleep', 'Run'] État final après 2 jours: Run

Probabilité de la séquence d'états possible: 0,12
```

Vous obtenez un ensemble aléatoire de transitions possibles avec la probabilité que cela se produise, à partir de l'état: Sleep. Étendez le programme plus loin pour peut-être l'itérer quelques centaines de fois avec le même état de départ, vous pouvez alors voir la probabilité attendue de se terminer à un état particulier avec sa probabilité. Réécrivons la fonction

Activity_forecast et ajoutez un nouveau jeu de boucles pour ce faire ...

```
def activity_forecast (jours):
        #    Choisissez l'état de départ activityToday = "Sleep"
        activityList = [activityToday] i = 0 prob = 1



        pendant que i! = jours:
                si activityToday == "Sleep":
                        changement =
np.random.choice (transitionName [0], replace = True, p = transitionMatrix [0])
                        si change == "SS":
                                prob = prob * 0.2 activityList.append ("Sleep") passe


                        elif change == "SR":
                                prob = prob * 0.6 activityToday = "Run"
                            activityList.append ("Run") sinon:


                                prob = prob * 0.2 activityToday = "Icecream"
                                activityList.append ("Icecream")

                elif activityToday == "Exécuter":
                        changement =
np.random.choice (transitionName [1], replace = True, p = transitionMatrix [1])
                        si changement == "RR":
                                prob = prob * 0.5 activitéList.append ("Exécuter")
                                passe

                        changement elif == "RS":
                                prob = prob * 0.2 activityToday = "Sleep"
                            activityList.append ("Sleep") sinon:


                                prob = prob * 0.3 activityToday = "Icecream"
                                activityList.append ("Icecream")

                elif activityToday == "Icecream":
                        changement =
np.random.choice (transitionName [2], replace = True, p = transitionMatrix [2])
                        si changement == "II":
                                prob = prob * 0,1
                                pass activityList.append ("Icecream")

                        changement elif == "EST":
                                prob = prob * 0.2 activityToday = "Sleep"
                            activityList.append ("Sleep") sinon:


                                prob = prob * 0.7 activityToday = "Run"
                                activityList.append ("Run")

                i + = 1 activité de retour Liste
```

```
#   Pour enregistrer chaque ActivityList list_activity = []
count = 0


#   La «plage» commence du premier comptage jusqu'à, mais à l'exclusion du dernier comptage pour les itérations de la plage (1 10000):

                list_activity.append (activity_forecast (2))

#   Consultez toutes les `ActivityList` que nous avons collectées
# print (list_activity)

#   Parcourez la liste pour obtenir un décompte de toutes les activités se terminant par l'état: «Exécuter»

pour small_list dans list_activity:
        if (small_list [2] == "Run"):
                compter + = 1

#   Calculez la probabilité de partir de l'état: «Sleep» et de terminer à l'état: «Run»

pourcentage = (nombre / 10000) * 100
print ("La probabilité de commencer à l'état: 'Sleep' et de se terminer à l'état: 'Run' =" + str (pourcentage) + "%")

La probabilité de commencer à l'état: 'Sleep' et de finir à l'état: 'Run' =
62,419999999999995%
```

Comment nous sommes-nous approchés des 62% souhaités?

Remarque Il s'agit en fait de la "loi des grands nombres", qui est un principe de probabilité qui stipule que les fréquences des événements avec la même probabilité d'occurrence sont égales, mais seulement s'il y a suffisamment d'essais ou d'instances. En d'autres termes, à mesure que le nombre d'expériences augmente, le rapport réel des résultats convergera vers un rapport théorique ou attendu des résultats.

État d'esprit de Markov

Ceci conclut le didacticiel sur les chaînes de Markov. Vous avez été initié aux chaînes de Markov et vu certaines de ses propriétés. Les chaînes de Markov simples sont l'un des sujets fondamentaux requis pour commencer avec la science des données en Python. Si vous souhaitez plus de ressources pour commencer avec les statistiques en Python, assurez-vous de vérifier cette page .

Êtes-vous intéressé à explorer des études de cas plus pratiques avec des statistiques en Python? Découvrez DataCamp's Études de cas en pensée statistique ou Analyse de réseau en Python cours.