به نام خدا

تمرین سری دهم هوش مصنوعی

سینا علی‌نژاد

۹۹۵۲۱۴۶۹

سوال ۱-

بر اساس کد پیادەسازی و مفهوم و روش الگوریتم likelihood توضیح خواهم داد.

ورودی های تابع likelihood_sample :

Formatted: Heading 2

Evidence: اگر مقدار این آرایه برای هر راس یا گره عدد 1- باشد، آن راس evidence نیست و اگر 0 و 1 باشد، به ترتیب نقیض آن و خود آن مشاهده شده است.

Nodes: شامل رئوسی است که در کوئری آمده اند.

Values: متناظر با nodes میگوید که هر کدام از رئوس کوئری احتمال مقدار true خواسته شده یا false.

Probability: یک آرایه دوبعدی که جدول cpt را شامل میشود و از فایل input.txt استخراج میشود.

N: تعداد رئوس

Graph: این ورودی یا ورودی graph2 معنا پیدا میکند. Graph2 شامل parentهای هر راس متناظر در graph است.

توضیحات: ابتدا لازم است بر اساس bayes net راسها را به صورت topological مرتب کنیم تا بتوانیم روی آن حلقه زده و برای هر راس احتمال آن را با توجه به parent های آن بدست آوریم و نمونه برداری کنیم.

```
for i in range(10000):
    value = [-1] * n
    w = 1
    for vertex in sort_vertex:
        if evidence[vertex] == -1:
            value[vertex] = sample_vertex(vertex, graph2[vertex], value, sort_vertex,
            probability)
        else:
            value[vertex] = evidence[vertex]
            w *= find_row(probability[vertex], value)
    samples.append([value, w])
```

در عکس بالا به تعداد 10000 بار نمونه میگیریم. آرایه value نشان دهنده یک نمونه است. روی رئوس مرتب شده حلقه میزنیم، اگر راس انتخاب شده جزو evidence باشد، از آن نمونه برداری نمیکنیم و فقط احتمال آن را در وزن آن نمونه تاثیر میدهیم. اما اگه راس انتخاب شده جزو evidence نباشد، در آن صورت با استفاده از تابع sample_vertex از آن نمونه برداری میکنیم. تابع sample_vertex با توجه به احتمال parentهای آن راس، احتمال درست بودن خود آن راس را بدست میآورد و یک عدد رندوم بین صفر و یک تولید میکند، اگر این عدد کوچکتر از احتمال درست بودن آن راس باشد، مقدار درست را برای آن در نظر گرفته و در غیر اینصورت مقدار غلط را میگیریم. اگر راس مورد نظر evidence باشد، احتمال آن به شرط parentهایش را با تابع find_row بدست آورده و مقدار w را که در ابتدا مقدار ۱ دارد، در آن ضرب میکند. با این روش وزن آن نمونه بدست آمده حساب میشود، در واقع احتمال مشاهده شدن چنین نمونه ای چقدر است، احتساب این وزن باعث consistent بودن الگوریتم likelihood میشود، یعنی وقتی نمونه ها زیاد شود، به مقدار واقعی احتمال موردنظر همگرا میشود. در واقع به جای اینکه نمونه ای تولید کنیم و

آن را reject کنیم، یک وزن در آن ضرب میکنیم و به نوعی کار آن reject کردن را انجام میدهیم ولی دیگر نمونه نامرتبط تولید نمیکنیم.

نمونه و وزن آن را در آرایه samples اضافه میکنیم.

حالا نوبت آن رسیده که بین این نمونه ها، آن هایی که good_sample هستند، یعنی کوئری‌ها را ارضا میکنند، جدا کنیم.

```
good_sample = 0
sum_sample = 0
for sample in samples:
    flag = 1
    for i in range(len(nodes)):
        if bool(values[i]) != sample[0][nodes[i]]:
            flag = 0
            break
    sum_sample += sample[1]
    if flag == 1:
        good_sample += sample[1]
return good_sample / sum_sample
```

اگر مقدار کوئری‌ها با مقداری که در نمونه است، برابر نباشد، flag را صفر کرده و وزن آن نمونه را فقط به sum_sample اضافه میکنیم ولی اگر کوئری‌ها ارضا شوند، مقدار وزن آن نمونه هم به sum_sample هم به good_sample اضافه میشود.

و در نهایت احتمال خواسته شده حاصل تقسیم good_sample بر sum_sample است.

سوال ۲-

مشکل الگوریتم likelihood sampling این بود که در محاسبه احتمال راس‌های بالاتر، رئوس evidence که پایین تر هستند تاثیر نمیگذارند و ما دنبال نمونه های بهتر و باکیفیت‌تری هستیم.

در روش gibbs مانند بقیه روش ها ابتدا topological sort را صدا میزنیم.

```
    for i in range(n):
        if evidence[i] != -1:
            value[i] = evidence[i]
        else:
            if np.random.random() < 0.5:
                value[i] = True
            else:
                value[i] = False
```

مرحله اول و دوم این الگوریتم در تکه کد بالا خلاصه میشود، یعنی fix کردن evidence ها و سپس مقدار دهی اولیه دیگر رئوس به صورت کاملا رندوم و با احتمال 0.5، این نمونه اولیه ما است که باید پس از قدم سوم به نمونه‌ای با کیفیت که evidence ها در آن موثر هستند، تبدیل شود.

```
for i in range(10000):
    new_value = [-1] * n
    for vertex in sort_vertex:
        if evidence[vertex] == -1:
            value[vertex] = -1
            value[vertex] = sample_vertex(vertex, graph2[vertex], value, sort_vertex,
            probability)
            new_value[vertex] = value[vertex]
        else:
            new_value[vertex] = value[vertex]
    samples.append(new_value)
```

در اینجا اگر راس انتخاب شده evidence نباشد، آن را خالی میکنیم و با توجه به احتمال بقیه رئوس، احتمال درست بودن این راس را بدست آورده و دوباره آن را نمونه برداری میکنیم و نمونه اولیه را آپدیت میکنیم، این فرایند را برای همه non-evidence ها انجام میدهیم ولی به evidenceها کاری نداریم. البته حالت درست‌تر این است که این روند را دوباره از اول روی non-evidence ها اعمال کنیم تا زمانی که احتمال هر کدام از این non-evidence ها به عددی همگرا شود ولی در اینجا فقط یک سری این کار را کردیم. در نهایت این نمونه با کیفیت که همان new_value است را در samples اضافه میکنیم.

```
good_sample = 0
for sample in samples:
    flag = 1
    for i in range(len(nodes)):
        if bool(values[i]) != sample[nodes[i]]:
            flag = 0
            break
    if flag == 1:
        good_sample += 1
return good_sample / len(samples)
```

در مرحله آخر همانند prior sampling نمونه هایی که کوئری را ارضا میکنند جدا میکنیم و تعداد این نمونه های خوب را بر تعداد کل نمونه های گرفته شده برابر با احتمال مورد نظر ماست.

سوال ۳-

:Graph1

ورودی ششم: در این کوئری فقط روش gibbs با جواب اصلی فاصله زیادی دارد و دلیل آن این است که در روش gibbs میخواهیم پس از چندبار آپدیت کردن نمونه اولیه به نمونه ای خوب برسیم ولی در گراف مورد نظر رئوس non-evidence یا مثل A و E به رئوس دیگر وابسته نیستند یا مثل D تاثیر کمی از رئوس non-evidence میگیرند.

ورودی دوم: در این کوئری نیز همانند کوئری ششم به دلیل اینکه راس A,E وابستگی به non-evidenceهای دیگر ندارند ولی از آنجا که B از A تاثیر میگیرد برای همین به اندازه کوئری ششم خطا ندارد.

:Graph2

ورودی ششم: خروجی این ورودی جالب است. در اینجا یکی از evidenceهای ما احتمال بسیار کمی دارد، برای همین در روش prior از ده هزار نمونه ای که میسازیم، خیلی‌شان بدرد ما نخواهند خورد و در واقع احتمال بدست آمده گویی با آزمایش های کمی بدست آمده و اصلا نزدیک نخواهد بود. روش rejection هم تفاوت زیادی با prior ندارد و فقط نمونه های بدردنخور همانجا پس از تولید حذف میشوند.

اما روش likelihood این مشکل را حل میکند به این شکل که فقط نمونه های بدردبخور تولید میکند و به آن ضریب میدهد.

روش gibbs هم تقریبا جواب خوبی داده ولی از آنجا که راس A فقط به evidence ها وابسته است، از روش likelihood مقداری بیشتر خطا داشته، در واقع تلاش ما برای تولید نمونه های باکیفیت تر خیلی موثر نبود.

Graph3:

ورودی دوم: همانند ورودی های دوم و ششم گراف اول، در اینجا همه non-evidence ها وابستگی به رئوس دیگر ندارند، پس روش gibbs موثر نخواهد بود.

ورودی سوم: در اینجا کوئری خواسته شده به شرط evidence به طور مستقیم در cpt آورده شده است و از اول مقدار آن را داریم ، برای همین همه الگوریتم ها تقریبا خطای صفر دارند.

به طور کلی در این نمودار ها هرچه مقدار AE بیشتر باشد، خطای آن روش نمونه برداری بیشتر است.

در روش Prior و rejection اگر احتمال evidence ها کم باشد، نتیجه مناسبی حاصل نمیشود زیرا اکثرشان بدرد نمیخورند.

برای حل مشکل بالا از likelihood استفاده میکنیم تا با تعداد نمونه کمتری بتوانیم به نتیجه واقعی همگرا شویم.

در روش likelihood ممکن است نمونه های دیده شده وزن کمی داشته باشند زیرا در ساخت این نمونه ها از evidenceهایی که پایین تر هستند در انتخاب رئوس بالاتر تاثیر گرفته نمیشود. برای حل این مشکل از روش gibbs استفاده میشود تا نمونه های با کیفیت تر با وزن بیشتر تولید شوند.