دانشکده: مهندسی کامپیوتر

موضوع: داک تمرین عملی چهارم AI

علی شکوهی

شماره دانشجویی: 400521477

تابع load_model

در این تابع ابتدا متغیرهای main_graph، V، names و cpts را می‌سازیم. بعد از باز کردن فایل model.txt باید خوانده شود و متغیرها مقداردهی شوند. همانگونه که در داک سوال گفته شده، برای هر سطر از ورودی‌ها باید احتمال False و True بودن هر متغیر در نظر گرفته شود. همچنین متغیر main_graph که دارای دو children_nodes key و parents_nodes می‌باشد، به وسیله temp_graph مقداردهی می‌شود. همچنین به یک متغیر new_cpts نیاز داریم که در آن با استفاده از names مقداردهی می‌شود. یعنی به جای استفاده از اسم متغیر، از شماره آن استفاده شده است. در نهایت هم متغیرهای main_graph, V, names, cpts, new_cpts را به عنوان خروجی برمی‌گرداند.

تابع prior_sample

ابتدا متغیرهای nodes و evidence از روی کوئری داده‌شده ساخته می‌شوند. سپس راس‌های گراف با استفاده از تابع topological_sort، سورت می‌شوند. سپس همانگونه که گفته شده، باید 10000 بار نمونه برداری کنیم. ابتدا برای هر راس با استفاده از تابع sample_vertex نمونه برداری انجام می‌شود و مقدار آن داخل لیست values ذخیره می‌شود. در مرحله بعدی مقدار هر راس داخل value با evidence مربوط به آن چک می‌شود و اگر مقدار flag 1 باقی بماند، آن نمونه را به لیست اضافه می‌کنیم. پس از به دست آوردن نمونه‌ها، باید نمونه‌هایی که با شرط query سازگار بودند (مقدارشان یکی بود) را به عنوان نمونه‌های قابل قبول اضافه کنیم. در آخر هم مقدار نمونه‌های قابل قبول را بر کل تعداد نمونه‌ها تقسیم می‌کنیم و به عنوان خروجی برمی‌گردانیم.

تابع rejection_sample

این روش نیز همانند روش prior می‌باشد; با این تفاوت که در این‌جا نمونه‌هایی که با شرط‌های ما سازگاری ندارند، از ابتدا در نظر گرفته نمی‌شوند.در این روش نوعی validation وجود دارد که نمونه‌های اضافی را تولید نکند.

تابع likelihood_sample

این روش مشابه روش قبل می‌باشد. در این‌جا متغیر weight که نشان‌دهنده وزن هر متغیر می‌باشد نیز وجود دارد. در این روش پس از بررسی همه راس های گراف، اگر evidence نبودند، باید نمونه برداری ساده روی آن‌ها انجام شود که نیاز به استفاده از وزن نیست. ولی اگر آن راس جزو evidence ها بود، در این حالت این متغیر یک بار نمونه برداری می‌شود و پس از آن در وزن آپدیت شده ضرب می‌شود. در مرحله بعدی نیز validation انجام می‌شود و نمونه‌هایی که با شرط query سازگار هستند (مقدارشان یکی بود) را به عنوان نمونه‌های قابل قبول اضافه می‌کنیم. مقدار samples_sum نیز در هر مرحله یکی اضافه می‌شود. در آخر مقدار accepted_sample / samples_sum به عنوان خروجی برگردانده می‌شود.

تابع gibbs_sample

در این روش مانند سایر روش‌ها ابتدا گراف را با استفاده از topological_sort، سورت می‌کنیم. در مرحله بعد در راس‌های گراف چک می‌کنیم که اگر evidence بودند، مقدارشان را True قرار می‌دهیم و در غیر اینصورت، به صورت رندوم یک احتمال ایجاد می‌کنیم که اگر از 0.5 کمتر بود، مقدارش را True و در غیر اینصورت، مقدارش را False می‌گذاریم. سپس در مرحله بعد باید نمونه تولید شود. در این مرحله اگر راس evidence باشد، مقدارش را داخل next_value می‌ریزیم و در غیر اینصورت با استفاده از sample_vertex آن را نمونه برداری می‌کنیم. در مرحله آخر مانند سایر روش‌های دیگر، validation انجام می‌شود یعنی پس از به دست آوردن نمونه‌ها، باید نمونه‌هایی که با شرط query سازگار بودند (مقدارشان یکی بود) را به عنوان نمونه‌های قابل قبول اضافه کنیم. در آخر هم مقدار نمونه‌های قابل قبول را بر کل تعداد نمونه‌ها تقسیم می‌کنیم و به عنوان خروجی برمی‌گردانیم.

تابع read_queries

در این تابع باید کوئری‌های داده شده خوانده شوند. برای این کار ابتدا path داده شده باز می‌شود و با استفاده از json.loads خوانده می‌شود. در مرحله بعد قسمت اول(ایندکس 0) فایل به عنوان queries خوانده می‌شود و قسمت دوم (ایندکس 1) فایل به عنوان evidence خوانده می‌شود و این دو مقدار به عنوان خروجی برگردانده می‌شود.

موارد تغییر داده شده:

در تابع exact_inference دو خط اضافه شده که با کامنت مشخص شده است.

در تابع draw_plot مقدار X برابر با [1,2,3] قرار داده شده است چون در حالت قبل به ارور می‌خورد.