به نام خداوند بخشنده‌ی مهربان

گزارش تمرین عملی دوم هوش مصنوعی

نیم‌سال اول ۱۴۰۰-۱۴۰۱

سید پارسا نشایی - ۹۸۱۰۶۱۳۴

دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

مقدمه

در این تمرین، فرآیند یافتن یک درخت تصمیم و سپس ارزیابی آن روی داده‌های نمونه‌ی ورودی دیابت و نیز مراجعه به رستوران، طی شده و درخت به همراه جزئیات آن به دست آمده است. جزئیات بیش‌تر از چگونگی عملکرد برنامه، در ادامه آمده است.

نحوه‌ی اجرا

برنامه به زبان Python نوشته شده و روی نسخه‌ی ۳ این زبان تست شده است. در صورت اجرا نشدن برنامه، ممکن است مشکل از نصب نبودن ابزار و کتاب‌خانه‌های graphviz و gvgen باشد که برای رسم درخت استفاده شده است. دیگر کتاب‌خانه‌های import شده، کتاب‌خانه‌هایی مانند numpy برای نگه‌داری و محاسبات روی آرایه‌ی اعداد، pandas برای خواندن داده‌ها از CSV و sklearn برای تقسیم تصادفی داده‌ها به بخش‌های یادگیری و تست هستند؛ برای حل خود مسئله، از هیچ ابزار، کتاب‌خانه یا toolkit آماده در راستای حل مسئله استفاده نشده و الگوریتم به

شکل دستی نوشته شده است. برای اجرای برنامه کافیست فایل tree.py در کنار diabetes.csv و restaurant.csv قرار گرفته و اجرا شود. برای تعویض دیتاست ورودی از رستوران به دیابت کافیست مقدار متغیر بولی isRestaurant که در کد تعریف شده، از True به False تغییر یابد.
با توجه به آن‌چه در کلاس گفته شد، به سبب تلاش برای اختصار مطالب گفته شده در این گزارش، از شرح ریزجزئیات کد اجتناب شده است، گرچه اسامی متغیرها به گونه‌ای واضح قرار داده شده‌اند که کد را self-explanatory کرده‌اند و نیاز به توضیح آن نیست.

کارکرد توابع اساسی کد و روند پیاده‌سازی

- تابع findNumberOfMatchings وظیفه شمردن تعداد داده با لیبل true/false برای هر نود را بر اساس ستون آخر داده‌های ورودی دارد.
- تابع findNodeDetails آنتروپی نود را محاسبه کرده تا قابل نمایش در درخت باشد و نیز Yes یا No بودن لیبل لازم برای خروجی آن را تعیین می‌کند.
- تابع addRowsToMap معنای منطقی خاصی ندارد و صرفا یک کد کمک‌کننده برای درج داده در یک مپ (دیکشنری) پایتون است.
- تابع trainChildren تابع اصلی train را روی تمام فرزندان نود کنونی (در صورت وجود) صدا می‌زند که در پیش‌برد روند train شدن درخت لازم است.
- تابع setChildren بر اساس مپ ورودی خود، فرزندان لازم برای نود کنونی را تنظیم می‌کند (به جهت جلوگیری از ایجاد خطا، ابتدا اگر فرزندی برای این نود ثبت شده، پاک می‌شود). فرزند به تازگی تولید شده، دارای عمق یکی بیش‌تر از پدرش است و نیز به لیست featureهای چک شده تا این نود، مقدار value کنونی نیز لازم است افزوده شود، زیرا نود جدید، فرزند نود کنونی است.
- تابع findEntropy فرمول آنتروپی را بر اساس آن‌چه در اسلایدهای درس دیده‌ایم، پیاده‌سازی کرده است.
- تابع trainFromNode وظیفه train اصلی را به عهده دارد. پس از صدا زدن دو تابع کمکی findNumberOfMatchings و findNodeDetails که پیش‌تر نیز شرح داده شده بودند، شرایط خاصی که به پایان جست‌وجو در این شاخه منجر می‌شوند بررسی شده و در صورت تحقق آن‌ها، بازگشت از تابع صورت می‌گیرد (مانند شرایطی که به yes شدن یا no شدن همه‌ی داده‌ها منجر شود و نیازی به تصمیم بعدی نباشد، یا

رسیدن به حداکثر عمق مشخص شده که در این مسئله برابر ۸ مشخص شده که عملا به دلیل تعداد ویژگی‌ها تاثیری ندارد، اما در عمل وجود این کران بالای عمق لازم است). سپس محاسبات مربوط به محاسبه‌ی gain ها و تنظیم فرزندان انجام شده که مطابق عملکرد معقول و مرسوم یادگیری درخت تصمیم است و سپس یادگیری به شکل بازگشتی روی فرزندان انجام می‌شود. در نهایت و در صورت لزوم (اگر به انتها رسیده باشیم و فرزندی نمانده باشد)، لیبل Yes/No روی این نود زده خواهد شد.

- تابع representation: این تابع text ای که لازم است به ازای هر نود در درخت نمایش داده شود را برمی‌گرداند. اگر این نود فرزندی نداشته باشد، متن آن شامل همان Yes/No است پس همان نمایش داده می‌شود، وگرنه فیچر کنونی به همراه آنتروپی و gain نمایش داده می‌شوند.
- تابع categorize: این تابع وظیفه‌ی گسسته‌سازی (چند دسته کردن) ورودی‌های با مقدار پیوسته و یا زیاد را به عهده دارد. پس از چند آزمون و خطا، تصمیم گرفتم که مقدار ۶ را به عنوان تعداد کل دسته‌ها قرار دهم. دسته‌بندی به کمک توابع pandas و numpy انجام گرفته است.
- تابع gvGen وظیفه‌ی تولید درخت نمایش داده شده (اگر رستوران باشد، اعداد مستقیم وگرنه بازه‌ای نمایش داده می‌شوند) و saveGraph نیز وظیفه‌ی رندر کردن آن در PDF را به عهده دارد که به این دو منظور، از کتاب‌خانه‌ی آماده استفاده شده است.
- تابع findAccuracy به ازای هر سطر داده، تابع findAccuracyOfEntry را فراخوانی کرده و درصد درستی درخت روی داده‌ی داده شده را بر می‌گرداند.
- تابع findAccuracyOfEntry، در صورت رسیدن به برگ تابع findAccuracyOfLeaf را فراخوانی کرده وگرنه به ازای فرزندان، به شکل بازگشتی خود را فراخوانی می‌کند (اگر داده موجود نبوده، به رندوم صحیح یا غلط گرفته می‌شود).
- تابع findAccuraryOfLeaf در صورتی که Yes/No بودن نود برگ کنونی با مقداری که در ستون آخر CSV آمده یکسان باشد، یکی به تعداد جواب‌های صحیح اضافه می‌کند.
- کلاس Node شامل سازنده، تابع isChild که وقتی درست است که نود برگ بوده و فرزندی نداشته باشد (منظور از isChild آن است که این نود صرفا فرزند است و پدر نود دیگری نیست – به عبارت دیگر، یک برگ است)، تابع removeChildren که آرایه‌ی فرزندان را خالی می‌کند، تابع setGain که gain را از روی آنتروپی و

remainder محاسبه می‌کند و تابع calculateStatsForFeature که تعداد مچ‌های صحیح و غلط و نیز remainder را حساب کرده و سپس setGain را فراخوانی می‌کند، است.

در کد اصلی، متغیر isRestaurant که پیش‌تر گفته شد و نیز آرایه‌ها و متغیرهای گلوبال استفاده شده در تابع تعریف شده‌اند و سپس CSV مربوطه به کمک pandas خوانده شده است. ستون‌ها (هدر) و نیز بدنه جدا شده‌اند (طبق بررسی خروجی‌ها، مشاهده شد که لازم است نسخه‌ی transpose شده‌ی بدنه‌ی خوانده شده توسط pandas پردازش شود). سپس اگر داده رستوران نبود، گسسته‌سازی و دسته‌بندی لازم است و انجام می‌شود. هم‌چنین، در حالت رستوران داده‌ی تست نداریم، اما در حالت دیابت، ۲۰ درصد داده توسط تابع کتاب‌خانه‌ای train_test_split به عنوان داده‌ی آزمایشی در نظر گرفته می‌شود. سپس node ریشه ساخته شده و عملیات train آغاز می‌شود. دقت train – و اگر داده دیابت باشد، دقت test – چاپ شده و درخت به دست آمده در فایل PDF ای در کنار فایل‌های تمرین ذخیره می‌شود.

داده‌ی CSV مربوط به دیابت داده شده، اما داده‌ی رستوران باید بر اساس اسلایدها به دست آید و در نتیجه یک CSV به شکل دستی طبق اسلاید ۲۷ ساخته شده و در کنار فایل‌های تمرین آپلود شده است. در متغیرهای کدگذاری‌شده‌ای مانند Patrons یا Type، نحوه‌ی تخصیص کدها به رشته‌ها به ترتیب با شروع از صفر و طبق ترتیب اسلاید ۲۲ بوده است.

خروجی به دست آمده و نتیجه‌گیری

- در مثال رستوران، دقت training صد درصد بوده (که با توجه به حجم کم داده‌ها طبیعی است) و درخت به دست آمده نیز منطبق بر درخت اسلاید ۳۲ است. خروجی حاصل در فایل restaurant.pdf آمده است.
- در مثال دیابت، در هر بار اجرا دقت‌ها طبیعتا به دلیل split نه لزوما یکسان داده‌های یادگیری و تست، تفاوت داشته‌اند، اما به عنوان مثال در یکی از دفعات اجرا که خروجی آن ضمیمه شده، دقت train برابر حدودا ۹۱.۶۹۳ درصد و دقت test برابر حدودا ۷۲.۷۲۷ درصد بوده است. خروجی حاصل در فایل diabetes.pdf آمده است.

با بررسی دقت به دست آمده، نتیجه می‌شود که الگوریتم از مقداری overfit برخوردار است. و دقت الگوریتم آن‌چنان بالا نیست، اما هم‌چنان تا حد مناسبی قابل‌قبول است. البته، ممکن است مشکل برنامه از الگوریتم نباشد و مشکل از داده‌های بعضا نامعقول موجود در دیتاست باشد – مانند تعداد زیادی از افراد با skin thickness برابر صفر (پوستی از برگ گل هم نازک‌تر!) و یا فشار خون برابر صفر (که قطعا نمی‌توانسته زنده باشد!) و یا موارد دیگر – که احتمالا به دلیل عدم اندازه‌گیری مقدار این کمیت برای آن افراد مشخص رخ داده است و احتمالا در صورت حذف این داده‌ها و یا اندازه‌گیری مقدار واقعی این کمیت‌ها توسط سازندگان دیتاست، احتمال بهبود الگوریتم به میزان قابل توجهی وجود دارد و یک تغییر که می‌توان برای بهبود عملکرد الگوریتم در داده‌های تست اعمال کرد، حذف داده‌هایی است که یکی از فیلدهای آن‌ها که نباید صفر باشد، صفر است.

چالش‌های اصلی‌ام در فرآیند انجام تمرین، یافتن روش تولید گرافیکی گراف و نیز باگ‌های رخ داده در الگوریتم تولید درخت تصمیم بوده‌اند؛ در کدی که در ابتدا نوشتم، به دلیل خطای ساده‌ی عملگری در محاسبه‌ی remainder، مقادیر غلطی برای gain به دست می‌آمد که پیدا کردن منشا باگ که یک خطای تایپی در استفاده از عملگر تفریق به جای جمع بود، متاسفانه مدت قابل توجهی به طول انجامید.