طراحی سیستم‌های یادگیری ماشین

نیم‌سال اول ۱۴۰۴-۱۴۰۵

مدرس: دکترسید صالحی

دانشگاه صنعتی شریف

تمرین تئوری اول

نام و نام‌خانوادگی: اردلان سیاوش پور

شماره دانشجویی: ۹۹۱۰۹۸۹۶

پرسش ۱

با توجه به جدول و محدودیت‌های داده‌شده و مقایسه‌ی معیارها، مدل B گزینه‌ی مناسب‌تری برای استقرار در محیط تولید است؛ زیرا با این‌که دقت آن کمی کمتر از مدل A است (۹۳% به‌جای ۹۶%)، ولی در عوض در تأخیر، انصاف و تفسیرپذیری بهبود چشم‌گیری دارد و در مجموع تعادل معقول‌تری میان نیازهای تیم‌های فنی، محصولی و حقوقی ایجاد می‌کند.

آ) تأثیر تأخیر روی تجربه‌ی کاربر و ظرفیت سیستم: در یک سامانه‌ی استخدام برخط، تأخیر ۲ ثانیه‌ای مدل A برای هر درخواست، از دید کاربر کند محسوب می‌شود؛ رابط کاربری را سنگین و پاسخ‌گویی سیستم را محدود می‌کند و در نهایت تعداد درخواست‌هایی را که در واحد زمان می‌توان پاسخ داد کاهش می‌دهد. مدل C از نظر تأخیر و با زمان ۱ ثانیه‌ای ایده‌آل است، اما مدل B با تأخیر حدود ۱٫۱ ثانیه، عملاً رفتاری بسیار نزدیک به مدل C دارد و این اختلاف کم برای اغلب کاربران قابل تشخیص نیست، در حالی که نسبت به مدل A جهش قابل توجهی در سرعت پاسخ و ظرفیت پردازش به‌وجود می‌آورد.

ب) نقش انصاف (**Fairness**) در اعتماد و ریسک حقوقی: مدل A از نظر معیارهای انصاف ضعیف است و چون بر اساس داده‌های تاریخی آموزش دیده، احتمال بازتولید تبعیض‌ها و سوگیری‌های گذشته زیاد است. از طرف دیگر، تفسیرپذیری پایین آن باعث می‌شود در صورت شکایت یا پرسش حقوقی، توضیح و دفاع از رفتار مدل دشوار شود. در مقابل، در مدل B «سوگیری قابل تنظیم و نظارت» تعریف شده است؛ یعنی می‌توان خروجی آن را از منظر گروه‌های مختلف پایش کرد و در صورت مشاهده‌ی رفتار تبعیض‌آمیز، تنظیمات را اصلاح نمود. این ویژگی برای جلب اعتماد متقاضیان و همین‌طور کاهش ریسک حقوقی و ضربه‌ی اعتباری به شرکت حیاتی است.

پ) اهمیت تفسیرپذیری در تصمیم‌های حساس انسانی: در فرآیند استخدام، افراد ممکن است توضیح رد شدن خود را بخواهند و نهادهای نظارتی نیز می‌توانند شفافیت در تصمیم‌های الگوریتمی را مطالبه کنند. مدل A به دلیل تفسیرپذیری بسیار کم، عملاً یک «جعبه‌ی سیاه» است و امکان ارائه‌ی توجیه مشخص برای تصمیم ندارد. مدل C از نظر تفسیرپذیری بسیار خوب است اما بهای آن، افت قابل‌توجه در دقت است. مدل B با سطح تفسیرپذیری متوسط این امکان را فراهم می‌کند که عوامل اثرگذار بر تصمیم تا حدود مناسبی تشریح شوند، بدون آن‌که قربانی دقت بسیار زیادی شویم.

ت) چگونگی موازنه‌ی معیارها و نتیجه‌گیری: اگر تنها معیار تصمیم‌گیری دقت بود، انتخاب طبیعی مدل A بود و اگر فقط تفسیرپذیری و کمی بهبود در تأخیر مهم بود، مدل C برتر می‌شد. اما در مسئله‌ی حاضر باید هم‌زمان به چهار جنبه توجه کنیم: دقت قابل قبول، سرعت پاسخ، انصاف و قابلیت دفاع در برابر کاربران و نهادهای نظارتی. مدل B با داشتن دقت بالا، تأخیر نزدیک به مدل C، انصاف قابل کنترل و تفسیرپذیری متوسط، بهترین مصالحه را میان این معیارها برقرار می‌کند و در نتیجه منطقی‌ترین گزینه برای استقرار در محیط تولید است.

پرسش ۲

تابع مطلوبیت به صورت زیر تعریف شده است:

$$U(n) = \alpha n - \lambda L, \qquad L = L_0(1.3)^n$$

که در آن α نشان‌دهنده‌ی سود افزایش دقت با اضافه کردن هر زیرمدل و λ ضریب جریمه‌ی تأخیر است.
برای بیشینه کردن U نسبت به n، مشتق می‌گیریم:

$$\frac{dU}{dn} = \alpha - \lambda \frac{dL}{dn} = \alpha - \lambda L_0 (1.3)^n \ln(1.3).$$

شرط بهینگی (نقطه‌ی بحرانی) زمانی است که:

$$\frac{dU}{dn} = 0 \;\Rightarrow\; \alpha = \lambda L_0 (1.3)^n \ln(1.3).$$

پس:

$$(1.3)^n = \frac{\alpha}{\lambda L_0 \ln(1.3)}$$

و در نتیجه:

$$n^* = \frac{\ln\!\left(\dfrac{\alpha}{\lambda L_0 \ln(1.3)}\right)}{\ln(1.3)}.$$

از آنجا که:

$$\frac{d^2U}{dn^2} = -\lambda L_0 (1.3)^n \big(\ln(1.3)\big)^2 < 0,$$

تابع U در این نقطه مقعر است و بنابراین n^* یک ماکزیمم موضعی (و در اینجا سراسری) است. در پیاده‌سازی واقعی، چون n تعداد زیرمدل‌ها و باید عدد صحیح نامنفی باشد، مقدار n^* را به نزدیک‌ترین عدد صحیح معتبر گرد می‌کنیم.

تفسیر در طراحی سیستم: در مقدار بهینه‌ی n^*، «سود حاشیه‌ای» افزودن یک زیرمدل جدید (افزایش دقت به اندازه‌ی α) دقیقاً برابر می‌شود با «هزینه‌ی حاشیه‌ای» آن از نظر تأخیر ($\lambda \frac{dL}{dn}$). هر چه λ بزرگ‌تر باشد (اهمیت بالاتر تأخیر)، مقدار بهینه‌ی n^* کوچک‌تر می‌شود و زیرمدل‌های کمتری در ensemble قرار می‌گیرند. در مقابل، اگر α بزرگ‌تر باشد (ارزش زیاد افزایش دقت)، سیستم به سمت استفاده از زیرمدل‌های بیشتر (افزایش n^*) میل می‌کند. در محیط تولید باید تعداد زیرمدل‌ها را نزدیک به n^* انتخاب کنیم؛ یعنی تا جایی مدل اضافه می‌کنیم که از آن به بعد، نفع دقت دیگر جبران‌کننده‌ی جریمه‌ی تأخیر نباشد.

پرسش ۳

تابع زیان کلی ناشی از تازگی ویژگی‌ها طبق صورت سؤال:

$$L = \sum_i e^{-\lambda_i \Delta t_i},$$

که در آن Δt_i فاصله‌ی زمانی بین دو به‌روزرسانی متوالی ویژگی iاُم است. هزینه‌ی کل نیز به شکل

$$J = \sum_i \left(\frac{C_i}{\Delta t_i} + \beta e^{-\lambda_i \Delta t_i} \right)$$

تعریف شده است. بنابراین برای هر ویژگی، مسئله‌ی بهینه‌سازی مستقل زیر را داریم:

$$\min_{\Delta t_i > 0} J_i(\Delta t_i), \qquad J_i(\Delta t_i) = \frac{C_i}{\Delta t_i} + \beta e^{-\lambda_i \Delta t_i}.$$

مشتق J_i نسبت به Δt_i:

$$\frac{dJ_i}{d\Delta t_i} = -\frac{C_i}{\Delta t_i^2} - \beta \lambda_i e^{-\lambda_i \Delta t_i}.$$

برای تمام $\Delta t_i > 0$، هر دو جمله‌ی سمت راست منفی‌اند، در نتیجه:

$$\frac{dJ_i}{d\Delta t_i} < 0 \quad \Rightarrow \quad J_i(\Delta t_i) \text{ یک تابع کاملاً نزولی نسبت به } \Delta t_i \text{ است.}$$

پس کمینه‌ی J_i در بزرگ‌ترین فاصله‌ی زمانی مجاز حاصل می‌شود؛ اگر سقفی برای Δt_i تعیین نشده باشد، مقدار بهینه به سمت

$$\Delta t_i^* \to \infty$$

می‌رود؛ یعنی طبق این تابع هزینه، بهترین تصمیم آن است که ویژگی تا حد امکان با فاصله‌های زمانی طولانی و به صورت batch بازمحاسبه شود.

تعبیر سامانه‌ای: هم بخش هزینه‌ی محاسباتی $\frac{C_i}{\Delta t_i}$ و هم ترم جریمه‌ی تازگی $\beta e^{-\lambda_i \Delta t_i}$ با افزایش Δt_i کاهش می‌یابند؛ بنابراین خود این فرمول ما را تشویق به انتخاب بازه‌های کوچک (حالت streaming) نمی‌کند. تنها زمانی باید یک ویژگی را از حالت batch به سمت streaming ببریم که قید بیرونی دیگری وجود داشته باشد؛ مثلاً محدودیت سختی روی حداکثر تاخیر مجاز مقدار ویژگی، یا این‌که λ_i و ضریب β به‌گونه‌ای بزرگ انتخاب شوند که سیستم ناچار شود Δt_i را خیلی کوچک نگه دارد. در چنین شرایطی، پیاده‌سازی حالت streaming برای برآورده کردن الزامات تازگی توجیه‌پذیر می‌شود.

پرسش ۴

آ) در سامانه‌های بلادرنگ، حتی چند میلی‌ثانیه می‌تواند روی تجربه‌ی کاربر و هزینه‌ی زیرساخت اثر بگذارد. قالب‌های دودویی مانند Protobuf نسبت به قالب متنی JSON چند مزیت مهم دارند: اندازه‌ی پیام‌ها معمولاً کوچک‌تر است، در نتیجه حجم ترافیک شبکه و زمان انتقال داده کمتر می‌شود؛ عملیات سریال‌سازی و دسریال‌سازی به دلیل نمایش باینری و داشتن schema مشخص، سریع‌تر انجام می‌شود و بار کمتری روی پردازنده ایجاد می‌کند؛ علاوه بر این، به دلیل نوع‌دار بودن فیلدها و قرارداد ثابت، خطاهای ناشی از تغییر نام یا نوع فیلدها زودتر آشکار می‌شود. مجموع این عوامل موجب کاهش زمان پاسخ و کاهش هزینه‌ی زیرساخت برای یک سیستم توصیه‌گر بلادرنگ می‌شود.

ب) بخش تراکنشی سیستم متکی بر پایگاه داده است. در این لایه به ویژگی‌هایی مانند تراکنش‌های ACID، یکپارچگی مالی و امکان بازیابی مطمئن نیاز داریم؛ بنابراین استفاده از یک پایگاه داده‌ی رابطه‌ای مانند MySQL و الگوی جریان داده‌ی مبتنی بر دیتابیس (خواندن و نوشتن رکورد به صورت پایدار و با تأخیر کم) مناسب است. جریان رویدادهای کاربر روی Kafka نمونه‌ی کلاسیک جریان پیام‌محور است؛ رویدادها به صورت پیام در topicها ثبت می‌شوند، مصرف‌کننده‌های متعدد می‌توانند آنها را با تأخیر کم و به شکل افقی مقیاس‌پذیر مصرف کنند و ویژگی‌هایی مانند بازپخش رویداد و تحمل خطا فراهم می‌شود. سرویس استنتاج بلادرنگ از طریق RPC (مثلاً gRPC) در دسترس قرار می‌گیرد؛ زیرا درخواست پیش‌بینی باید synchronous، با قرارداد مشخص برای ورودی و خروجی و با تأخیر بسیار کم انجام شود و RPC برای این نوع تعامل service-to-service، کنترل خوبی روی timeout، نسخه‌بندی و کارایی فراهم می‌کند.

پ) در عمل، رویدادهای کاربران و خروجی‌های مدل به صورت پیام‌های Protobuf در Kafka لاگ می‌شوند. در بازه‌های زمانی مشخص (مثلاً شب‌ها)، یک job آفلاین به عنوان consumer به topicهای مربوط متصل می‌شود، پیام‌های پروتوباف را می‌خواند، آنها را به قالب مناسب برای data lake (مثلاً فایل‌های Parquet) تبدیل می‌کند و مجموعه‌ی داده‌ی برچسب‌دار و ویژگی‌ها را برای آموزش مجدد مدل آماده می‌سازد. چون در هر دو سمت (آنلاین و آفلاین) از همان schema پروتوباف استفاده می‌شود، تعریف و محاسبه‌ی فیچرها در دو مسیر یکسان می‌ماند و خطر ناهماهنگی train/serve کاهش می‌یابد. هماهنگ بودن این دو پایپ‌لاین بسیار مهم است؛ اگر محاسبه‌ی ویژگی‌ها، نرمال‌سازی‌ها یا منطق ساخت داده در بخش بلادرنگ و بخش آموزشی متفاوت باشد، مدلی که روی داده‌ی تاریخی خوب کار می‌کند در زمان serve آنلاین عملکرد ضعیف و غیرقابل پیش‌بینی خواهد داشت. این موضوع debug را سخت می‌کند و مقایسه‌ی نسخه‌های مختلف مدل را مختل می‌سازد. با یکپارچه نگه داشتن schema و کد استخراج ویژگی در مسیرهای online و offline، می‌توان پایداری و قابلیت اعتماد سامانه‌ی توصیه‌گر را حفظ کرد.

پرسش ۵

فرض کنید حدود ۱۵% مقادیر جدول مفقود است و این مفقودی‌ها به طور یکنواخت بین ویژگی‌ها پخش نشده‌اند. چهار روش رایج برای برخورد با مقادیر مفقود و ارزیابی آنها از نظر سوگیری، تفسیرپذیری و امکان پیاده‌سازی در تولید به شکل زیر قابل جمع‌بندی است.

روش اول ـ حذف رکوردهای دارای مقدار مفقود

در این روش هر سطری که دست‌کم یک مقدار گمشده دارد، کنار گذاشته می‌شود. اگر رخداد مفقودی کاملاً تصادفی نباشد و در برخی گروه‌ها (مثلاً یک نوع کاربر یا یک منطقه) بیشتر اتفاق بیفتد، توزیع داده‌ها به طور جدی تغییر می‌کند و سوگیری بزرگی ایجاد می‌شود. از نظر فهم و توضیح، این روش بسیار ساده و روشن است. در محیط تولید، پیاده‌سازی آن آسان و ارزان است، اما به دلیل حذف بخش قابل توجهی از داده‌ها، برای وضعیتی که ۱۵% مفقودی ناهمگن داریم، معمولاً انتخاب مطلوبی نیست.

روش دوم ـ جایگزینی با آمارهای ساده (میانگین، میانه)

برای هر ویژگی عددی می‌توان مقادیر مفقود را با میانگین یا میانه‌ی همان ویژگی جایگزین کرد و برای ویژگی‌های اسمی از مد (رایج‌ترین مقدار) استفاده نمود. این رویکرد حجم داده را حفظ می‌کند، اما واریانس را کاهش داده و مقادیر را به سمت مرکز توزیع می‌کشد؛ بنابراین یک سوگیری متوسط ایجاد کرده و می‌تواند ساختار همبستگی میان ویژگی‌ها را مختل کند. از نظر تفسیرپذیری، به‌خوبی قابل توضیح است که چه مقداری و بر چه مبنایی جایگزین شده است. در محیط تولید نیز اجراکردن آن بسیار ساده و کم‌هزینه است و غالباً به عنوان baseline مورد استفاده قرار می‌گیرد.

روش سوم ـ جایگزینی مبتنی بر مدل (مثل رگرسیون یا **KNN Imputation**)

در این روش برای هر ویژگی که مقادیر مفقود دارد، مدلی بر پایه‌ی سایر ویژگی‌ها آموزش داده می‌شود تا مقدار گمشده را تخمین بزند. اگر مدل‌ها به‌خوبی یاد گرفته باشند، سوگیری آماری کمتر شده و ساختار همبستگی داده بهتر حفظ می‌شود؛ اما در صورت کمبود داده یا نامتوازن بودن آن، ممکن است سوگیری‌های پیچیده و پنهان به وجود آید. تفسیرپذیری این روش پایین‌تر است، چون مشخص نیست برای هر سطر دقیقاً بر چه اساس عدد پیش‌بینی شده است، به‌خصوص اگر مدل پیچیده باشد. از منظر تولید، نیاز به محاسبه‌ی مدل‌های اضافی در زمان serve دارد و معماری سیستم را پیچیده‌تر می‌کند؛ در نتیجه هزینه و احتمال بروز خطا یا خرابی افزایش می‌یابد.

روش چهارم ـ استفاده از مقدار ثابت همراه با پرچم مفقود بودن

برای ویژگی‌های عددی می‌توان مقدار مفقود را با یک مقدار ثابت (مثلاً صفر یا میانه) جایگزین کرد و در کنار آن یک ویژگی دودویی جدید تعریف نمود که نشان بدهد مقدار اصلی مفقود بوده است. در ویژگی‌های دسته‌ای نیز می‌توان یک دسته‌ی جدید با عنوان missing اضافه کرد. این روش به مدل اجازه می‌دهد متوجه شود که خود «مفقود بودن» یک سیگنال اطلاعاتی است؛ بنابراین نسبت به جایگزینی ساده با میانگین، سوگیری کمتری دارد و اگر الگوریتم مناسب استفاده شود، می‌تواند الگوهای مربوط به مفقودی را بیاموزد. از نظر تفسیرپذیری وضعیت روشن است، چون دقیقاً می‌دانیم چه زمانی مقدار واقعی و چه زمانی مقدار جایگزین به‌کار رفته است. برای محیط تولید، اجرای آن نسبتاً ساده (افزودن یک ستون پرچم و یک مقدار پیش‌فرض) است و نیازی به مدل‌های جداگانه برای برآورد مقادیر گمشده ندارد؛ به همین دلیل در بسیاری از سامانه‌های عملی، تعادل خوبی بین سوگیری، تفسیرپذیری و هزینه‌ی پیاده‌سازی فراهم می‌کند.

پرسش ۶

پس از اضافه کردن تعداد زیادی ویژگی جدید و مشاهده‌ی افت عملکرد، می‌توان سه نوع تحلیل تشخیصی انجام داد تا علت اصلی مشکل مشخص شود.

تحلیل اول ـ بررسی تکرار و همبستگی شدید بین فیچرها در این گام، ماتریس همبستگی یا ماتریس اطلاعات متقابل بین ویژگی‌ها را محاسبه می‌کنیم. اگر بین دو یا چند فیچر، ضریب همبستگی بسیار بالا یا اطلاعات متقابل زیاد مشاهده شود، می‌توان نتیجه گرفت که ویژگی‌های زائد و تکراری وارد مدل شده‌اند و این موضوع می‌تواند باعث ناپایداری مدل، مشکلات عددی یا overfitting شود. متریک اصلی مورد استفاده در این تحلیل ضریب همبستگی و Mutual Information است.

تحلیل دوم ـ کشف نشت داده (**Data Leakage**) در این مرحله، برای هر ویژگی مقدار اطلاعات متقابل آن با برچسب را روی داده‌ی آموزش و اعتبارسنجی محاسبه می‌کنیم و فیچرهایی را که MI بسیار بالا دارند یا در صورت استفاده‌ی تنها از چند فیچر، عملکرد مدل به‌طرز غیرواقعی بالا می‌رود، زیر ذره‌بین قرار می‌دهیم. همچنین با اعتبارسنجی مبتنی بر زمان (time-based validation) بررسی می‌کنیم که آیا اطلاعات از آینده به گذشته نشت کرده است یا نه. متریک‌های کلیدی این تحلیل همان Mutual Information بین فیچر و برچسب و نیز عملکرد مدل‌های ساده (مثلاً AUC) هنگام استفاده از فیچرهای مشکوک است.

تحلیل سوم ـ بررسی رانش ویژگی‌ها در طول زمان (**Feature Drift**) در این تحلیل، توزیع هر فیچر را بین دوره‌ی آموزش و دوره‌های جدید مقایسه کرده و شاخص پایداری جمعیت (Population Stability Index یا PSI) را برای هر ویژگی محاسبه می‌کنیم. فیچرهایی که PSI بالایی دارند دچار drift شده‌اند. علاوه بر این، می‌توان مدل را در پنجره‌های زمانی مختلف دوباره آموزش داد و روند تغییر اهمیت هر ویژگی را در طول زمان بررسی کرد؛ اگر اهمیت یک فیچر به‌طور سیستماتیک کاهش یابد (Feature Importance Decay)، نشان می‌دهد که نقش آن فیچر در پیش‌بینی کم‌رنگ شده و احتمالاً توزیع یا رابطه‌اش با هدف تغییر کرده است.

پرسش ۷

آ) در رویکرد طبقه‌بندی چندکلاسه، هر ویدیو یک کلاس مجزا محسوب می‌شود و مدل مستقیماً احتمال انتخاب هر ویدیو را خروجی می‌دهد؛ در نتیجه اندازه‌ی لایه‌ی خروجی برابر تعداد کل ویدیوهاست. اگر هزاران ویدیو در سیستم داشته باشیم، لایه‌ی خروجی بسیار بزرگ می‌شود، آموزش شبکه دشوار و پرهزینه می‌گردد و با اضافه شدن هر ویدیوی جدید، باید ابعاد خروجی را تغییر داد و عملاً مدل را دوباره آموزش داد؛ بنابراین این روش از نظر مقیاس‌پذیری برای کاتالوگ‌های بزرگ مناسب نیست. در مقابل، در روش رگرسیونِ امتیازدهی، مدل یک تابع نمره‌دهی روی زوج کاربر—ویدیو می‌آموزد؛ یعنی ورودی مدل ترکیبی از ویژگی‌های کاربر و ویژگی‌های یک ویدیو است و خروجی یک نمره‌ی علاقه‌مندی. برای هر کاربر، ابتدا مجموعه‌ای از ویدیوهای کاندید انتخاب می‌شود، سپس مدل برای هر کاندید یک امتیاز محاسبه می‌کند و ویدیوها بر حسب این امتیاز رتبه‌بندی می‌شوند. در این چارچوب، اضافه شدن ویدیوی جدید صرفاً به معنای ورود نمونه‌های جدید با فیچرهای همان ویدیو است و نیازی به تغییر ساختار خروجی یا آموزش از صفر نیست؛ بنابراین برای پلتفرمی با هزاران ویدیو و کاتالوگ دائماً در حال تغییر، روش امتیازدهی رگرسیونی همراه با مرحله‌ی candidate generation معمولاً مناسب‌تر و مقیاس‌پذیرتر از طبقه‌بندی چندکلاسه‌ی مستقیم است.

ب) اگر مدل A باعث افزایش CTR شود ولی میانگین زمان تماشا کاهش یابد، یعنی مدل کاربر را به کلیک کردن روی ویدیوهایی ترغیب می‌کند که خیلی زود رها می‌شوند؛ به عبارت دیگر، متریکی که مدل بر اساس آن آموزش دیده (نرخ کلیک) با هدف تجاری اصلی ـ که احتمالاً افزایش زمان تماشای باکیفیت و نگه‌داشتن کاربر در پلتفرم است ـ هم‌راستا نیست. این وضعیت شبیه تولید عنوان‌های clickbait است: کلیک زیاد، ولی رضایت و درگیری واقعی پایین. برای تعریف متریک مناسب‌تر، باید زمان تماشا را وارد تابع هدف کنیم. یکی از گزینه‌ها تعریف متریک به صورت امید ریاضی زمان تماشا به ازای هر نمایش است:

$$\mathrm{EWT} = \mathrm{CTR} \times \mathrm{AvgWatchTimeGivenClick}$$

یا استفاده از تابع هدف ترکیبی مانند:

$$\mathrm{Objective} = \alpha \times \mathrm{CTR} + \beta \times \mathrm{WatchTime}$$

که ضرایب α و β متناسب با اهداف تجاری تنظیم می‌شوند. می‌توان از معیارهایی مانند زمان تماشا در هر جلسه، زمان تماشا در بازه‌ی زمانی مشخص پس از توصیه، یا نرخ تکمیل ویدیو نیز کمک گرفت تا مدل به جای حداکثر کردن صرفِ کلیک، تعامل پایدار و ارزشمند کاربر را بهینه کند.

پرسش ۸

رویکرد اول: یک مدل واحد با تابع زیان ترکیبی فرض کنید خروجی مدل $f(x)$ است و برای دو هدف مختلف دو تابع زیان داریم، مثلاً:

$$L_{\text{engage}} = \mathrm{CrossEntropy}(y_{\text{click}}, \hat{y}_{\text{click}}), \qquad L_{\text{quality}} = \mathrm{CrossEntropy}(y_{\text{quality}}, \hat{y}_{\text{quality}}).$$

می‌توان تابع زیان کلی را به صورت

$$L_{\text{total}} = \alpha L_{\text{engage}} + \beta L_{\text{quality}}$$

تعریف کرد که در آن α و β نشان‌دهنده‌ی اهمیت نسبی هر هدف‌اند. گرادیان نسبت به پارامترهای مدل برابر است با:

$$\nabla_\theta L_{\text{total}} = \alpha \nabla_\theta L_{\text{engage}} + \beta \nabla_\theta L_{\text{quality}},$$

بنابراین به‌روزرسانی پارامترها هم‌زمان تحت تأثیر هر دو هدف قرار می‌گیرد. مزیت این رویکرد آن است که تنها یک مدل و یک پایپ‌لاین serve داریم، پیاده‌سازی و نگه‌داری در محیط تولید ساده‌تر است و خود فرآیند بهینه‌سازی، trade-off بین اهداف را یاد می‌گیرد. از سوی دیگر، تنظیم ضرایب α و β دشوار است و ممکن است جهت گرادیان‌ها برای دو زیان در تضاد باشد، در نتیجه مدل بیشتر به یک هدف توجه کند و هدف دیگر قربانی شود. اگر هدف جدیدی مثلاً مربوط به انصاف یا ریسک حقوقی اضافه شود:

$$L_{\text{total}} = \alpha L_{\text{engage}} + \beta L_{\text{quality}} + \gamma L_{\text{fairness}},$$

باید مدل با ترکیب جدید زیان‌ها مجدداً آموزش داده شود و تشخیص سهم هر هدف در شکل‌گیری رفتار مدل سخت‌تر می‌شود. انعطاف‌پذیری در برابر تغییر سیاست‌ها محدود است، چون هر تغییر در وزن‌ها مستلزم بازآموزی مدل بزرگ است.

رویکرد دوم: چند مدل مستقل برای اهداف مختلف در این رویکرد برای هر هدف یک مدل جداگانه با زیان ویژه‌ی خود آموزش داده می‌شود:

$$L^{(1)} = L_{\text{engage}}(\theta_1), \qquad L^{(2)} = L_{\text{quality}}(\theta_2),$$

که θ_1 و θ_2 پارامترهای دو مدل هستند. هنگام serving، برای هر خبر و کاربر، دو نمره تولید می‌شود:

$$s_{\text{engage}} = f_1(x), \qquad s_{\text{quality}} = f_2(x),$$

و سپس یک ترکیب‌کننده (مثلاً یک تابع خطی یا مدل کوچکی دیگر) امتیاز نهایی را می‌سازد:

$$s_{\text{final}} = w_1 s_{\text{engage}} + w_2 s_{\text{quality}}$$

یا یک تابع غیرخطی مشابه. مزیت اصلی این راهکار، انعطاف‌پذیری بالاست: می‌توان مدل تعامل را بدون دست زدن به مدل کیفیت جداگانه بهینه و بازآموزی کرد، یا برای یک بازار جدید فقط یکی از مدل‌ها را تغییر داد. از نظر تفسیرپذیری نیز بهتر است، چون می‌توان جداگانه دید که هر خبر از نظر تعامل چه نمره‌ای دارد و از نظر کیفیت چه نمره‌ای؛ و سیاست‌های تجاری را در لایه‌ی ترکیب اعمال کرد (مثلاً محدود کردن سهم اخبار با کیفیت پایین). در مقابل، هزینه‌ی بازآموزی و نگه‌داری بیشتر است، چون چند مدل باید monitor، deploy و به‌روزرسانی شوند؛ همچنین در زمان serve، محاسبه‌ی چند پیش‌بینی می‌تواند روی تأخیر و هزینه‌ی زیرساخت اثر بگذارد. علاوه بر این، بهینه‌سازی چندهدفه در سطح گرادیان انجام نمی‌شود و تضمینی نیست که ترکیب ساده‌ی

$$s_{\text{final}}$$

به شکل خطی، بهترین trade-off میان L_{engage} و L_{quality} را ایجاد کند؛ وزن‌های w_1 و w_2 معمولاً باید با آزمایش‌های A/B و با توجه به متریک‌های کسب‌وکار تنظیم شوند.

به طور خلاصه، مدل واحد با زیان ترکیبی

$$L_{\text{total}} = \sum_k \lambda_k L_k$$

از نظر سادگی معماری و یکپارچگی گرادیان‌ها جذاب است، اما تنظیم ضرایب λ_k و تفسیر رفتار مدل دشوار بوده و هر تغییر در اهداف نیازمند بازآموزی است. در مقابل، مجموعه‌ای از مدل‌های مستقل با زیان‌های

$$L_1, L_2, \ldots, L_K$$

و یک لایه‌ی ترکیب نمره، کنترل و شفافیت سیاست را افزایش می‌دهد و به واحدهای تجاری اجازه می‌دهد trade-off ها را بدون دستکاری مدل‌های زیری تغییر دهند، اما هزینه‌ی محاسباتی، پیچیدگی عملیاتی و ریسک ناهماهنگی بین مدل‌ها را بالا می‌برد. انتخاب بین این دو رویکرد در طراحی سیستم رتبه‌بندی خبر به اولویت‌دهی میان سادگی عملیاتی و میزان نیاز به کنترل و شفافیت برای ذی‌نفعان بستگی دارد.

پرسش ۹

فرض کنید تنها ۰٫۱% تراکنش‌ها متقلبانه‌اند و بیشتر داده‌ها از یک منطقه‌ی جغرافیایی خاص جمع‌آوری شده‌اند.

آ) نمونه‌گیری تصادفی ساده: در نمونه‌گیری تصادفی ساده، از میان ۱۰۰ میلیون رکورد، یک زیرمجموعه را به طور یکنواخت و تصادفی انتخاب می‌کنیم. در این صورت نسبت کلاس‌ها همان ۰٫۱% باقی می‌ماند؛ بنابراین در مجموعه‌ی آموزش، تعداد مثال‌های تقلب بسیار اندک است و مدل عمدتاً الگوهای کلاس سالم را یاد می‌گیرد و برای کلاس تقلب وزن کمی قائل می‌شود. ترکیب جغرافیایی داده‌ها نیز همان توزیع اولیه را حفظ می‌کند؛ اگر در ابتدا داده‌ها بیش از حد از یک منطقه‌ی خاص آمده باشند، این سوگیری به همان شکل منتقل می‌شود.

نمونه‌گیری طبقه‌ای: در نمونه‌گیری طبقه‌ای، داده‌ها را بر اساس کلاس (و در صورت لزوم متغیرهای مهم دیگری مانند منطقه) به طبقه‌هایی تقسیم کرده و از هر طبقه، به‌طور کنترل‌شده نمونه می‌گیریم. مثلاً می‌توان همه‌ی تراکنش‌های تقلب و تنها بخشی از تراکنش‌های سالم را انتخاب کرد تا نسبت تقلب در آموزش افزایش یابد. در این حالت مدل نمونه‌های بیشتری از کلاس تقلب مشاهده می‌کند و آن را بهتر می‌آموزد؛ اما توزیع حاشیه‌ای کلاس‌ها دیگر معادل واقعیت نیست و اگر در زمان ارزیابی یا تنظیم آستانه‌ها این موضوع را لحاظ نکنیم، برآورد احتمال خروجی مدل دچار سوگیری خواهد شد. اگر طبقه‌بندی را بر اساس منطقه نیز انجام دهیم، تا حدی می‌توانیم سوگیری جغرافیایی را کنترل کنیم.

نمونه‌گیری وزنی (اهمیتی): در این روش به جای حذف یا تکرار رکوردها، در تابع زیان به نمونه‌ها وزن‌های متفاوت اختصاص می‌دهیم (مثلاً وزن بیشتر برای تراکنش‌های تقلب یا مناطق کم‌نماینده). بدین ترتیب، همه‌ی داده‌ها در آموزش باقی می‌مانند، اما بهینه‌ساز به نمونه‌های با وزن بالاتر توجه بیشتری دارد. این روش بدون دستکاری مستقیم توزیع داده، عدم‌توازن کلاس را تا حد خوبی جبران می‌کند؛ البته اگر وزن‌ها بیش از حد بزرگ انتخاب شوند، خطر افزایش واریانس و overfitting روی تعداد کمی نمونه‌ی پُروزن وجود دارد.

ب) در این مسئله، استفاده از accuracy به‌عنوان معیار اصلی می‌تواند کاملاً گمراه‌کننده باشد؛ چرا که اگر مدل همیشه «سالم بودن» تراکنش را پیش‌بینی کند، در حدود ۹۹٫۹% مواقع درست خواهد بود و accuracy نزدیک به ۰٫۹۹۹ به دست می‌آید، در حالی که مدل هیچ تقلبی را شناسایی نمی‌کند. برای مسئله‌ی کشف تقلب، باید از متریک‌هایی استفاده کرد که نسبت به کلاس نادر حساس‌تر هستند؛ مانند precision، recall، F1، AUC منحنی PR یا متریک‌های مبتنی بر هزینه که سود یا زیان مالی کشف یا عدم کشف هر تراکنش را در نظر می‌گیرند (مثلاً حداکثر کردن سود مورد انتظار یا کمینه کردن هزینه‌ی مورد انتظار بر اساس ماتریس هزینه).

پ) چند رویکرد معروف برای مقابله با عدم توازن در این مسئله عبارت‌اند از:

Undersampling: در این روش بخشی از تراکنش‌های سالم را حذف می‌کنیم تا نسبت تقلب به سالم افزایش یابد؛ برای مثال، همه‌ی نمونه‌های تقلب و تنها زیرمجموعه‌ای از نمونه‌های سالم را نگه می‌داریم. مزیت این روش، کوچک‌تر و متعادل‌تر شدن مجموعه‌ی داده است، آموزش سریع‌تر انجام می‌شود و مدل روی مثال‌های تقلب تمرکز بیشتری دارد. ریسک اصلی، از دست رفتن اطلاعات و الگوهای مهم در کلاس سالم یا در برخی مناطق جغرافیایی است که می‌تواند باعث عملکرد ضعیف مدل در شرایط واقعی شود.

Oversampling و **SMOTE**: در oversampling نمونه‌های تقلب را تکرار می‌کنیم و در SMOTE، نمونه‌های مصنوعی جدیدی در همسایگی نمونه‌های تقلب تولید می‌کنیم تا اندازه‌ی کلاس نادر افزایش یابد، بدون آنکه چیزی از کلاس غالب حذف شود. مزیت این روش‌ها آن است که تمام اطلاعات موجود در کلاس سالم حفظ می‌شود و در عین حال، تعداد نمونه‌های کلاس تقلب بالا می‌رود؛ در نتیجه مدل تعادل بهتری بین دو کلاس پیدا می‌کند. خطر oversampling ساده، overfitting روی نمونه‌های تکراری است و در SMOTE امکان تولید نقاط غیرواقعی یا به‌هم‌زدن ساختار محلی داده وجود دارد؛ به‌خصوص هنگامی که تقلب‌ها تنها از یک منطقه‌ی جغرافیایی محدود هستند و فضا به‌خوبی پوشش داده نشده است؛ در این صورت نقاط مصنوعی ممکن است توزیع واقعی را به‌درستی منعکس نکنند.

Focal Loss: در این رویکرد، تابع زیان به گونه‌ای تغییر می‌کند که سهم نمونه‌های آسان و پرشمار (معمولاً کلاس سالم) در زیان کاهش یابد و تمرکز به سمت نمونه‌های سخت و کم‌شمار (تقلب) منتقل شود. به طور معمول یک ضریب مانند $(1-p_t)^{\gamma}$ به زیان cross-entropy ضرب می‌شود تا نمونه‌هایی که مدل آنها را با اطمینان بالا درست پیش‌بینی می‌کند، نقش کمتری در گرادیان داشته باشند. مزیت این روش آن است که بدون دستکاری توزیع داده، عدم‌توازن کلاس را در سطح بهینه‌سازی جبران می‌کند و نیازی به resampling ندارد. ریسک آن، حساسیت نسبت به پارامترهای تنظیم (مانند γ و وزن کلاس‌ها) و احتمال تمرکز بیش از حد روی نمونه‌های نویزی یا دارای برچسب اشتباه است که می‌تواند منجر به overfitting روی outlierها شود.