آتوسا مالمیر چگینی

۹۷۱۰۶۲۵۱

گزارش تمرین چهارم بخش عملی

داده ها را ابتدا پسوندشان را به csv تغییر دادم.

بلاک اول) در ابتدا logistic classifier را پیاده سازی کرده ام. Features همان دیتای ما است بدون ستون آخر و target همان ستون آخر دیتاست است. Num_steps تعداد مراحل آپدیت کردن وزن ها و learning_rate هم سرعت آپدیت شدن وزن ها با استفاده از gradient descent است. اگر add_ones true بود باید ستونی با اعداد ۱ به features اضافه شود. سپس در یک for، features و weights که در ابتدا ۰ بود در هم ضرب میشوند و با گرفتن سیگموئید بر روی حاصل ضربشان و محاسبه ی میزان ارور و سپس استفاده از روش گرادیان، وزن ها آپدیت میشوند.

تابع sigmoid هم با گرفتن لیستی از اعداد سیگموئید آنها را خروجی میدهد.

تابع metric_evaluation دو ورودی real که مقادیر درست هستند و predicted که مقادیری هستند که مدل پیشبینی کرده است، f1score و precision و recall را خروجی میدهد.

تابع predicted_test_data هم وزنها و مقادیر تست را میگیرد و پیشبینی مدل برای این ورودی را محاسبه میکند و خروجی میدهد.

بلاک دوم) در بلاک دوم نوت بوک ابتدا داده ها نرمال سازی شده و سپس داده های تست و آموزش از هم جدا میشوند و تابع logistic_regression روی داده های آموزش محاسبه میشود و سپس داده های تست به تابع predict_test_data داده میشوند و با درنظر گرفتن یک ترشلد به مقادیری که بیشتر از ترشلد باشند ۱ و آنهایی که کمتر هستند ۰ داده میشود. و در آخر متریک ها محاسبه میشوند.

بلاک سوم) دو کلاس Node و DecisionTreeClassifier تعریف کرده ایم و یک سری ویژگیهایی برای هر کدام در نظر گرفته ایم که مهمترینهای آنها در کلاس دوم شامل max_depth است که به گونه ای از overfit شدن داده های آموزش جلوگیری میکند. Min_samples_leaf است که کمترین تعداد داده ای که میتوانند در یک برگ قرار بگیرند را مشخص میکند و در آخر min_samples_split است که حداقل تعداد داده ای را نشان میدهد که همچنان باید split شوند. تعدادی تابع هم در این کلاس تعریف شده اند که عبارتند از:

۱) nodeProbas: که با گرفتن ورودی y درصد داده های کلاس های مختلف آن را محاسبه میکند و خروجی میدهد.
۲) gini: درصدها را گرفته و gini را خروجی میدهد.
۳) calcImpurity: ابتدا nodeProbas ستون target را محاسبه کرده و سپس معیار gini را برای آن محاسبه میکند.
۴) calcBestSplit: به ازای هر ستون و به ازای هر مقدار در آخر ستون با تابع calcImpurity میزان entropy را محاسبه میکند و سپس information gain را به دست میاورد و بررسی میکند که آیا این feature داده ها را بهتر از بقیه feature ها جدا کرده است یا نه و به این ترتیب بهترین ویژگی را برای split کردن داده ها پیدا میکند و داده های split شده را برمیگرداند. به علاوه باید به این نگته اشاره کرد که هر مقدار هر ستون را threshold میگیریم و داده ها را بر اساس آن دو دسته میکنیم و با سنجیدن information gain در نهایت بهترین threshold هم به دست می آید و به همراه بهترین ویژگی برای split کردن داده ها خروجی داده میشود.
۵) buildDT: این تابع اصلی برای ساخت درخت است در ابتدای آن چند شرط پایان الگوریتم بررسی شده اند (مثل max_depth و ...) سپس تابع calcBestSplit داده ها را تقسیم بندی کرده و اگر داده ها بر اساس هیچ ویژگی تقسیم نشده بودند یا این که سمت چپ یا راست آن ها خالی بود الگوریتم تمام میشود. هر بار که داده ها بر اساس یک ویژگی split میشوند عمق درخت یکی زیاد میشود و تابع به صورت بازگشتی برای node های سمت چپ و راست صدا زده میشود.
۶) predictSample: برای یک sample خاص در تست در عمق درخت میرود تا به یک گره برگ برسد و درصد داده های دو کلاس را برای داده های موجود در آن نود برمیگرداند.
۷) predict: داده های تست را یکی یکی به predictSample داده و در خروجی predictSample کلاسی را که درصد بیشتری از داده ها عضو آن هستند به عنوان پیشبینی این داده انتخاب میکند و تمام این پیشبینی ها را خروجی میدهد.

بلاک چهارم) یک instance از DecisionTreeClassifier میسازیم و داده های train را fit کرده و داده های تست را پیشبینی میکنیم و سپس متریک های مختلف را محاسبه میکنیم.

بلاک پنجم) جواب سوال بخش دوم است.

بلاک ششم) در این بخش قصد داریم که راه حل بیان شده در بخش دوم را بررسی کنیم و ببینیم که چقدر کارا است. از آنجایی که قرار است دیتاست را بالانس کنیم ابتدا index دادهایی با ستون آخر ۱ را در one_indexes

نگه میداریم. سپس proportion را محاسبه میکنیم که درصد داده های کلاس 0 به 1 است. datasett هم تنها سطرهایی با مقدار ستون آخر ۱ هستند. میخواهیم به تعداد int(propotion) بار داده های کلاس ۱ در دیتاست بیایند تا به این ترتیب دیتاست بالانس شود. پس باید int(proportion) – 1 بار datasett را به دیتاست اصلی بچسبانیم. حالا با درصدی مشابه با آزمایش های قبلی test و train میسازیم و درخت تصمیم را برای آن اجرا میکنیم و میبینیم که recall و f1score به طور قابل توجهی بهبود یافته اند. (توجه کنید که در این بلاک داده های تست جدید بررسی شده اند و نتایج گزارش شده مربوط به آنها (داده های بالانس شده) میباشد.)

بلاک هفتم) نتایج درخت تصمیم بلاک قبلی برای test_set قبلی (داده های غیر بالانس). که باز هم همانطور که میبینید، راه حل پیشنهادی بخش دوم تاثیر به سزایی روی recall و f1score داشته است.

بلاک هشتم) میخواهیم میزاین بهتر شدن logistic را هم بسنجیم برای این کار ابتدا داده ها را به صورت آرایه های numpy در آورده ایم.

بلاک نهم) در این جا روی dataset جدید (داده های بالانس) مدل را آموزش داده ایم و سپس روس همی داده ها آن را تست کرده ایم. و نتیجه را با threshold scanning بهبود داده ایم.

بلاک دهم) این بلاک مربوط به بخش چهارم تمرین است. کلاس DecisionStump را داریم که یک سری ویژگی را نگه میدارد مثلا این که بر اساس چه feature و با چه ترشلدی داده ها را به دو دسته تقسیم کرده ایم. کلاس Adaboost را داریم که n_clf تعداد classifier های آن است. تابع fit را دارد که در آن مانند اسلایدهای درس ابتدا وزن همه ی داده ها را 1/n (n تعداد داده ها است.) در نظر میگیرد و به اندازه ی n_clf نمونه از کلاس DecisionStump میسازد و min_error تعریف میکند که کمترین میزان ارور به ازای حالت های مختلف تقسیم دیتاست با استفاده از feature های مختلف را نگه میدارد. حالا به ازای تمام مقادیر یکتا در هر feature بررسی میکند که اگر آن مقدار threshold باشد min_error بهتر میشود یا نه؟. به ازای آن مقدار حالا تمام آنهایی که کمتر از آن هستند را جزو کلاس 1- و آن هایی که بیشتر از این ترشلد هستند را جزو کلاس 1 در نظر میگیرد. حالا ارور را محاسبه میکند که جمع وزنهای داده هایی هستند که اشتباه کلاس بندی شده اند. حالا اگر ارور بیشتر از نیم بود یعنی بیشتر داده ها اشتباه کلاس بندی شده اند که در این حالت اگر لیبلی که به هر داده میدهیم را برعکس کنیم error کمتر از نیم میشود. حالا error را با min_error مقایسه میکنیم و در صورتی که کمتر بود یعنی این ویژگی و این مقدار از آن بهترین feature و بهترین threshold تا این جا هستند. حالا باید آلفاها را محاسبه کنیم. مانند آپدیت کردن weight ها برای این هم ترشلد داریم و کلاس بندی میکنیم و در آخر هم نرمالسازی انجام میدهیم.

تابع predict هم تعدادی داده تست میگیرد و برای هر classifier به هر داده تست لیبل میدهد و در آخر sign جمع prediction های classifier های مختلف را خروجی میدهد.

بلاک یازدهم) در این بلاک ابتدا لیبل های صفر داده های train را 1- میکنیم و بعد adaboost را آموزش میدهیم. سپس داده های تست غیر بالانس را (داده های اصلی) به تابع predict میدهیم و متریک های مختلف را بررسی میکنیم. همانطور که مشخص است مدل اصلا نتوانسته است حتی یک داده را به کلاس ۱ نسبت بدهد.

بلاک دوازدهم) از راه حل بخش دو استفاده میکنیم و adaboost را روی داده های بالانس شده آموزش میدهیم و با این کار میبینیم که fscore ۰,۳۵ زیاد میشود.