بسمه تعالی

دانشگاه شهید بهشتی
دانشکده علوم ریاضی
گروه علوم کامپیوتر

تمرین سری ۲ واحد درسی داده کاوی

سارا چرمچی
۴۰۰۴۲۲۰۶۶

استاد راهنما : جناب آقای دکتر فراهانی

دستیار آموزشی : آقای علی شریفی

اردیبهشت ۱۴۰۱

مقدمه

کلیه تحلیل های این گزارش با استفاده از زبان برنامه نویسی پایتون اجرا شده است و کدها به پیوست موجود است.

تمرین دوم

در این تمرین از دیتاست یک پلتفرم مشهور المانی در زمینه اجاره خانه ها استفاده شده است. اطلاعات این دیتاست بسیار گسترده و تمیزنشده است. ابتدا کتابخانه های مورد استفاده را وارد می کنیم. سپس داده ها را به هر طریقی مناسب است می خوانیم (در اینجا داده های کگل را از طریق کولب فراخوانی می کنیم)

اندازه دیتاست : (۲۶۸۸۵۰, ۴۹) و ۴۹ ستون دارد.

پاک سازی داده ها

پاک سازی داده ها شامل حذف ستون های هجو، حدف داده های تکراری ، بررسی داده های ناموجود و رسیدگی به داده های پرت:

۱. بررسی داده های nan و تصمیم در مورد پر کردن آن ها با روش fillna

ابتدا درصد داده های ناموجود در هر ستون را با دستور df.isna().sum()/len(df) نمایش می دهیم و میبینیم تعداد زیادی از ستون ها داده های ناموجود دارند که ممکن است در آینده باعث خطای مدل شوند پس چنین ستون هایی را حذف می کنیم.

جای گذاری داده های ناموجود در داده های عددی و داده های categorical که درصد بالایی نداشتند که حذف شوند.

داده های عددی: یکی از راه های پرکردن داده های عددی ناموجود، استفاده از میانگین داده های موجود آن ستون است.

داده های categorical: داده های ناموجود در هر ستون را با پرفرکانس ترین دادی آن ستون جای گذاری می کنیم

۲. حذف داده های بی معنی و یا غیر مهم

در ستون livingSpace داده هایی با مقدار ۰ داریم که بی معنی است چرا که آپارتمانی با متراژ صفر نداریم

در ستون totalRent,RentBase داده هایی با مقدار ۰ داریم که بی معنی است چرا که آپارتمانی با اجاره بهای صفر نداریم

در ستون noRooms داده هایی با مقدار بیش از ۱۰۰ داریم که بی معنی است چرا که آپارتمانی با این تعداد اتاق منطقی نیست .

در ستون livingSpace داده هایی با مقدار بیش از ۶۰ متر مربع داریم که اجاره بهای زیر ۳۰ یورو دارند که بی معنی است چرا که آپارتمانی با این متراژ چنین قیمتی ندارد.

برخی داده ها باعث شلوغ شدن دیتاست می شود و در اینجا نیازی نداریم پس خذف می کنیم :

```
df_g.drop(columns=['livingSpaceRange','street','description','facilities','geo_krs','geo_plz','scoutId','telekomUploadSpeed','telekomTvOffer','pricetrend','regio3','noRoomsRange','picturecount','geo_bln','date',\
'houseNumber','streetPlain','firingTypes','yearConstructedRange'],inplace=True)
```

۳. نرمالایز کردن داده ها با روش سنتی میانگین و واریانس : در این مورد، میانگین به اضافه یا منهای ۳ برابر انحراف معیار است که دادههای بزرگتر از میانگین به اضافه ۳ برابر انحراف معیار و کوچکتر از میانگین منهای ۳ برابر انحراف معیار، پرت محسوب می شوند.

۴. برخی داده های categorical را به خاطر تعدد موارد unique می توان کاهش داد، شمارشی از انواع داده ها در فیچر ایالت می کنیم و چند مورد آخر که کمترین فروانی را داشتند در نوع جدید other ذخیره می کنیم.

مصورسازی داده ها

سپس با توجه به تمرین سری اول، یک سری مصورسازی جهت درک بهتر دیتاست اجرا می کنیم که در گزارش اول به تفصیل بحث شده است.

مهندسی ویژگی

مهندسی ویژگی فرآیند استخراج ویژگی‌ها از داده‌های خام با استفاده از دانش مسئله است. از این ویژگی ها می توان برای بهبود عملکرد الگوریتم های یادگیری ماشین استفاده کرد و در صورت افزایش عملکرد، بهترین دقت را ارائه می دهد. همچنین می توان گفت که مهندسی ویژگی همان یادگیری ماشین کاربردی است. مهندسی ویژگی مهمترین هنر در یادگیری ماشین است که تفاوت زیادی بین یک مدل خوب و یک مدل بد ایجاد می کند. این سومین مرحله در چرخه عمر هر پروژه علم داده است.

مهندسی ویزگی شامل : داده های پیوسته ، ویژگی های طبقه بندی شده(categorical)، مقادیر از دست رفته، نرمالیزیشن و تاریخ و زمان است. در اینجا روی ویژگی ها categorical متمرکز می شویم :

۱- داده های عددی مورد نیاز را با روش minmax scaler نرمال می کنیم
۲- برخی داده های categorical دارای unique values گسترده هستند که برای تفسیر بهتر مدل تعداد مقادیر یونیک را کاهش می دهیم . در فیچر condition موارد متفوتی وجود دارد پس
column (‘well_kept’, ‘refurbished’, ‘first_time_use’, ‘fully_renovated’, ‘mint_condition’, ‘first_time_use_after_refurbishment’, ‘modernized’, ‘negotiable’, ‘need_of_renovation’, ‘ripe_for_demolition ')
را به سه کلاس جدید و قدیمی و متوسط تقسیم بندی می کنیم.
در فیچر heating_type مجددا به دلیل بالا موارد بیشتر را جدا کرده و مورد های با فرکانس کمتر را در یک کلاس others طبقه بندی می کنیم.
۳- داده های categorical معمولا در مدل سازی های یادگیری ماشین مشکل ساز هستند پس بهتر است به داده های عددی تبدیل شوند و به هر کدام از unique value ها با متد dummy variables عدد نسیت داده می شود. این داده های categorical در دو دسته Boolean و object هستند که همه را به داده های عددی تبدیل می کنیم

در نهایت با feature engineering فیچرهای ما به ۵۰ فیچر مهندسی شده افزایش یافته است.

رگرسیون و محاسبه خطا

۱- محاسبه خطای مدل سازی بدون استفاده از پکیج:

رگرسیون و خطا را از scratch با پایتون و استفاده از pandas اجرا می کنیم. در مدل رگرسیون خطی یک متغیر وابسته داریم و متغیر مستقل. متغیری که برای پیش بینی مقدار متغیر وابسته استفاده می کنیم، متغیر مستقل نامیده می شود. ساده ترین شکل معادله رگرسیون با یک متغیر وابسته و یک متغیر مستقل به فرم زیر است :

y = m * x + b

مقدار وابسته تخمینی : y

ثابت یا بایاس: b

ضریب یا شیب رگرسیون: m

مقدار متغیر مستقل: x

ما از تابع خطای میانگین مربعات به عنوان تابع ضرر استفاده می کنیم : MSE

یک تابع خطای میانگین مربع همانطور که از نام آن پیداست، میانگین مجذور تفاوت بین مقدار واقعی و پیش بینی شده است.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

از آنجایی که مقدار پیش‌بینی‌شده y به slope و constant بستگی دارد، بنابراین هدف ما یافتن مقادیری برای شیب و ثابتconstant است که تابع ضرر را به حداقل می‌رساند یا به عبارت دیگر، تفاوت بین مقادیر پیش‌بینی‌شده و واقعی yرا به حداقل می‌رساند.

الگوریتم‌های بهینه‌سازی برای یافتن مجموعه بهینه پارامترها با توجه به مجموعه داده آموزشی استفاده می‌شوند که تابع loss function را به حداقل می‌رساند، در این مورد باید مقدار بهینه شیب (m) و ثابت (b) را پیدا کنیم. یکی از این الگوریتم ها Gradient Descent است.

Gradient Descent تا حد زیادی محبوب ترین الگوریتم بهینه سازی مورد استفاده در یادگیری ماشین است. با استفاده از Gradient Descent ، گرادیان های تابع loss را با توجه به پارامترها به طور مکرر محاسبه می کنیم و تا رسیدن به local minimum ها ، پارامترها را به روز می کنیم.

- مرحله اول : initialize پارامترها

در اینجا، باید مقادیر پارامترهای خود را مقداردهی اولیه کنیم. slope = 0 وconstant = 0 ..
همچنین برای تعیین اندازه گام در هر تکرار و در حین حرکت به سمت حداقل مقدار تابع ضرر، به یک نرخ یادگیری نیاز داریم.

- مرحله دوم : محاسبه مشتقات جزئی را با توجه به پارامترها

برای کاهش یک تابع معمولا ابتدا مشتقات جزبی را نسبت به پارامترها محاسبه کرده و مساوی صفر قرار می دهند

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b))$$

- مرحله سوم : بروزرسانی پارامترها

اکنون، مقادیر پارامترهای خود را با استفاده از معادلات زیر به روز می کنیم مقادیر به روز شده برای پارامترهای ما مقادیری خواهند بود که با هر مرحله تابع ضرر ما را به حداقل می رساند و تفاوت بین مقادیر واقعی و پیش بینی شده را کاهش می دهد.

$$m = m - \mathrm{Lr} \times \frac{\partial L}{\partial \mathrm{m}}$$

$$b = b - \mathrm{Lr} \times \frac{\partial L}{\partial \mathrm{b}}$$

با مراحل شرح داده شده کلاس رگرسیون را با استفاده از پایتون تعریف می کنیم. داده های آموزشی به عنوان input هستند و مقادیر اولیه بایاس و ثابت را برابر صفر قرار میدهیم.

با ()fit در کلاس ، gradient descent پیاده سازی می کنیم به طوریکه با هر تکرار ، مشتقات جزئی تابع را با توجه به پارامترها محاسبه می کنیم و سپس پارامترها را با استفاده از نرخ یادگیری و مقدار گرادیان به روز می کنیم.

با ()predict تابع رگرسیون خطی را با مقادیر بهینه پارامترها ارزیابی می کنیم به عبارتی این متد خط رگرسیون با بهترین fit را تخمین می زند.

با همین پارامترهای بدست آمده خطاها را نیز محاسبه می کنیم.

۲- محاسبه خطای مدل سازی با استفاده از پکیج scikit-learn

پکیج scikit-learn رگرسیون خطی و تمامی متریک های ارزیابی را در قالب توابع آماده ارایه می کند که به راحتی قابل استفاده است.

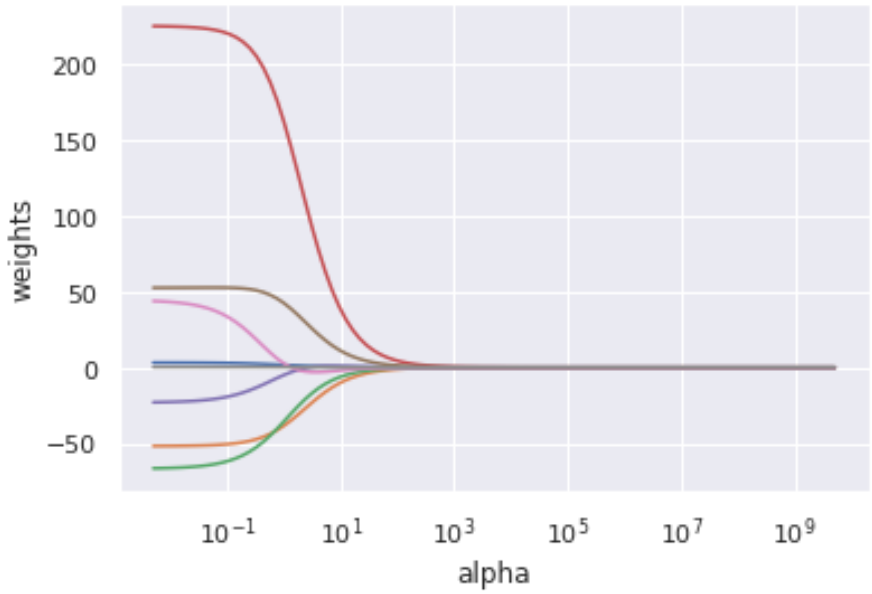
رگرسون Ridge

پکیج scikit-learn رگرسون Ridge و Lassoو تمامی متریک های ارزیابی را در قالب توابع آماده ارایه می کند. توابع اصلی عبارتند از Ridge() که می‌تواند برای برازش مدل‌های رگرسیون Ridge استفاده شود، و Lasso() که با مدل‌های lasso مطابقت دارد. آنها همچنین دارای مدل های متقابل هستند: RidgeCV() و LassoCV().

تابع Ridge () دارای یک آرگومان آلفا (λ ، همان آلفا است که برای tunning استفاده می کنیم) است که برای تنظیم مدل استفاده می شود. آرایه‌ای از مقادیر آلفا از خیلی بزرگ تا خیلی کوچک را تولید می‌کنیم، برای مدل های تهی تا least squares fit

مرتبط با هر مقدار آلفا، بردار ضرایب رگرسیون Ridge است که آن را در ضرایب ماتریسی ذخیره می کنیم. در این مورد، یک ماتریس ۸×۱۰۰، با ۸ سطر (یکی برای هر فیچر) و ۱۰۰ ستون (یکی برای هر مقدار آلفا) است.چون می خواهیم متغیرها را به گونه ای استاندارد کنیم که در یک scale باشند. برای این کار می توانیم از پارامتر normalize = True استفاده کنیم.

انتظار داریم پس نرمال سازی و استفاده از الفاهای زیاد ضرایب coefficient estimates به مراتب کوچک تر باشند.



سپس داده های اموزش و تست را جداسازی می کنیم.

مدل را روی داده های اموزش fit می کنیم و روی داده های تست با استفاده از alpha=4 مقدار mse را بدست می اوریم

MSE: 142306.4742627567

دوباره با الفای خیلی بزرگ مدل را fit کرده و تخمین می زنیم:

MSE:166667.595434925

این پنالتی بزرگ ضرایب را به درجات بزرگ shrink می کند و عملا به مدل محدود به intercept تبدیل می کند. این over-shrinking مدل را بایاس می کند در نتیجه MSE بیشتر می شود.

بنابراین برازش یک مدل رگرسیون پشته با آلفا = ۴ منجر به آزمون MSE بسیار پایین‌تری نسبت به برازش یک مدل با intercept می‌شود. اکنون بررسی می کنیم که آیا به جای انجام رگرسیون least squares، انجام رگرسیون ridge با آلفا = ۴ فایده ای دارد یا خیر. رگرسیون least squares، ridge با آلفا = ۰ است.

MSE: 106012.5911036244

به نظر می رسد که ما واقعاً نسبت به least squares، وضعیت بهتری است. پس به جای انتخاب دلخواه آلفا =۴، بهتر است cross validation برای انتخاب پارامتر tunning آلفا استفاده کنید. با استفاده از cross-validated ridge regression function ، RidgeCV() می توان این کار را انجام داد. به‌طور پیش‌فرض، این تابع cross validation تعمیم‌یافته (شکل کارآمد LOOCV) را انجام می‌دهد، اگرچه این می‌تواند با استفاده از آرگومان cv تغییر داد.

می بینیم که مقدار آلفای که منجر به کوچکترین خطای cross validation می شود ۰.۰۰۵ است. پس MSE مربوط به این مقدار آلفا را محاسبه می کنیم : 106011.92540581664

این نشان دهنده یک پیشرفت بیشتر نسبت به MSE است که با استفاده از آلفا =۴ دریافت کردیم. در نهایت، مدل رگرسیون ridge را بر روی مجموعه داده کامل، با استفاده از مقدار آلفا انتخاب شده توسط cross validation ، دوباره تنظیم می کنیم و تخمین های ضرایب را بررسی می کنیم. البته همانطور که انتظار داریم هیچیک از ضرایب صفر مطلق نشده است چراکه مدل رگرسیون ridge مقادیر را انتخاب نمی کند.

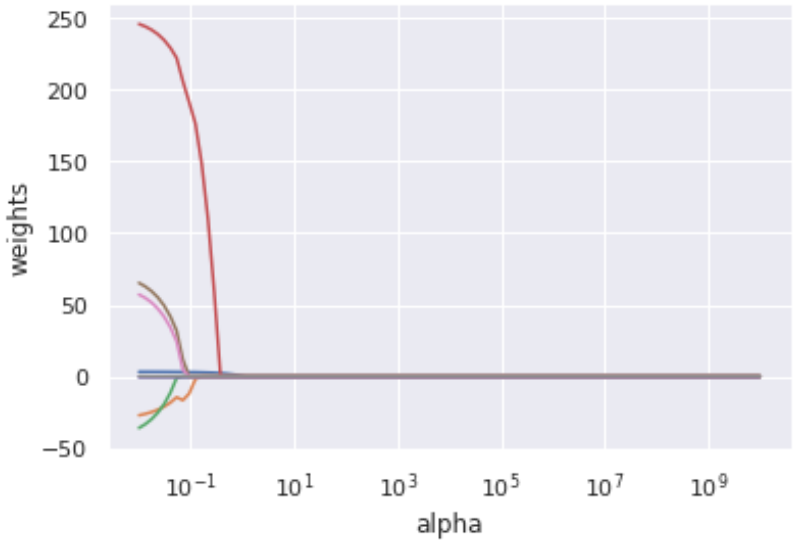
Lasso

دیدیم که رگرسیون ridge با انتخاب آلفای مناسب می‌توان خطایی بهتر از از least squares بدست آورد. اکنون بررسی می کنیم آیا Lasso می تواند مدلی دقیق‌تر یا قابل تفسیرتر از رگرسیون ridge ارائه دهد. از تابع Lasso() استفاده می کنیم. آرگومان max_iter = 10000 قرار داده و سایر پارامترها را مانند رگرسیون ridge قرار می دهیم.

از کتابخانه scikit-learn داریم :

max_iter : int, optional

The maximum number of iterations



توجه داریم که که در نمودار ضرایب که وابسته به انتخاب پارامتر tunning است ، برخی از ضرایب دقیقا برابر با صفر هستند.
اکنون cross validation ۱۰ فولد را برای انتخاب بهترین آلفا استفاده می کنیم، مدل را مجددا fit کرده و محاسبه خطای تست را انجام می دهیم.

MSE: 106014.0525569379

این به طور قابل‌توجهی کمتر از مجموعه آزمایشی MSE مدل صفر و least squares است و فقط کمی بدتر از MSE رگرسیون ridge با آلفای انتخاب شده توسط cross validation است.

با این حال، Lasso دارای مزیت قابل توجهی نسبت به رگرسیون ridge است، زیرا تخمین‌های ضرایب حاصل کم هستند. در اینجا می بینیم که برخی موارد دقیقاً صفر هستند.