به نام خدا

Mini project1 - ML

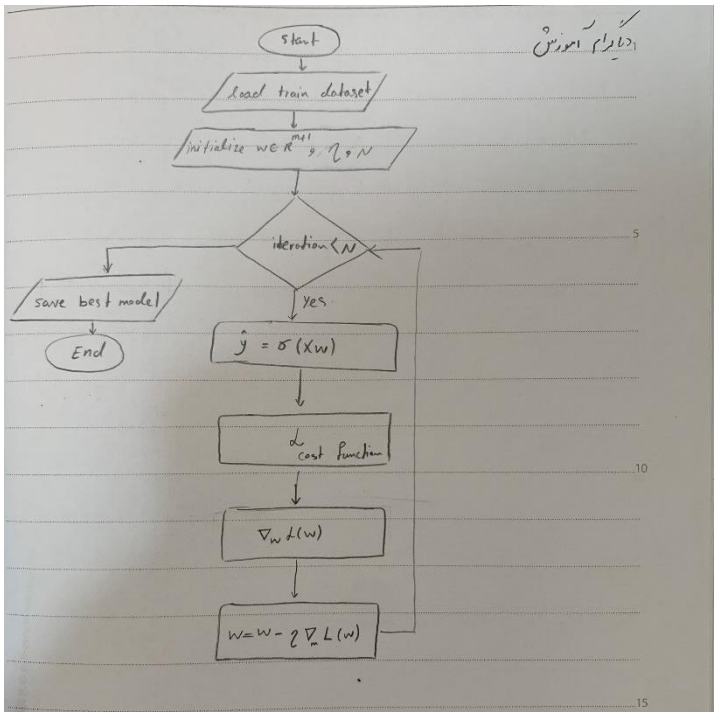
فاطمه صفایی: ۴۰۲۰۷۹۷۴

colab link : https://colab.research.google.com/drive/1P5S5LlqllvKvI4CszPw9XDYFQwjn-cXF?usp=sharing

github: https://github.com/fatemehsafaee/ML-miniPjoject1

سوال ۱:

۱. فرایندآموزش:



در فرایند آموزش، پس از بارگذاری دیتاست آموزشی، ابتدا باید پارامتر های ماشین و نرخ یادگیری و تعداد iteration ها را مشخص نمود. منظور از iteration تعداد دفعاتی است که ماشین، بسته به تابع هزینه، پارامتر ها یا w های خود را اپدیت می کند. این w ها از یک فضای m+1 بعدی هستند که m مربود به تعداد feature های دیتاست و ۱ مربوط به بایاس است.

سپس N بار در هربار، ابتدا مدل تعریف می شود. این مدل می تواند یک تابع sigmoid باشد. ورودی تابع، بردار ویژگی ها در پارامتر های ماشین یا به تعبیری دیگر، یک معادله خط هستند. پس به طور کلی، تمام ورودی ها به مدل داده شده و خروجی مدل یا y^ محاسبه میگردد.

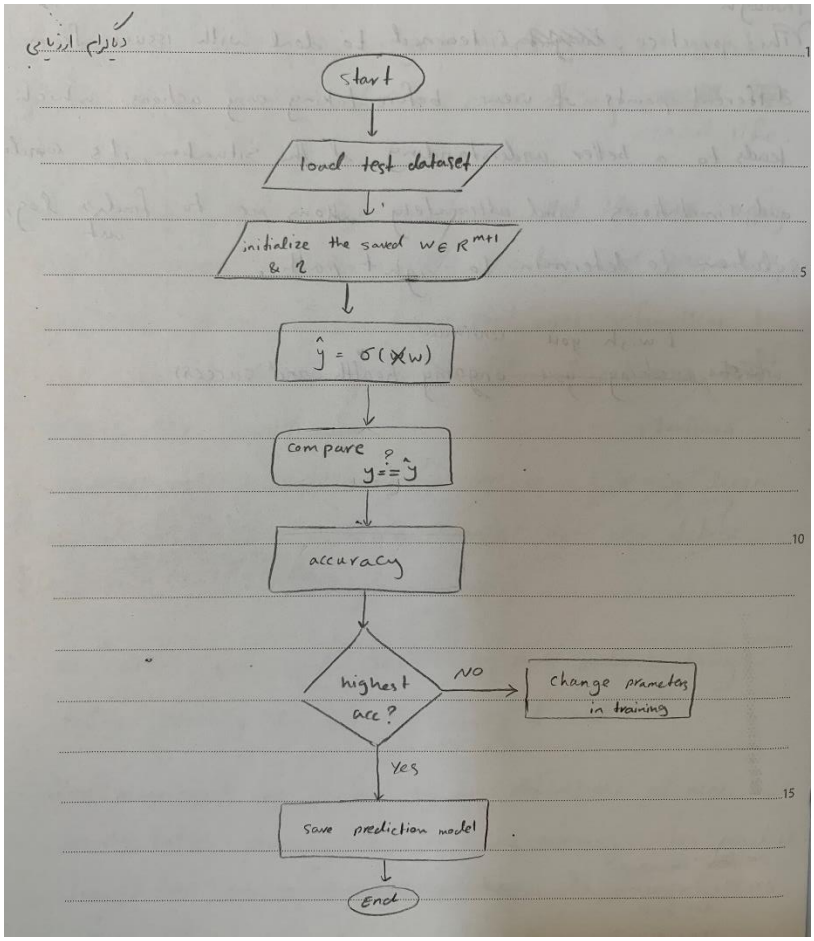
برای ارزیابی آنکه y^ بدست آمده چه میزان با y از پیش تعیین شده برای ورودی ها اختلاف دارد، نیاز به یک تابع هزینه داریم. این تابع هزینه میتواند log-loss یا انواع دیگری باشد.

در بخش بعد باید از تابع هزینه، نسبت به تمام پارامتر های ماشین مشتق بگیریم که به این کار گرادیان گفته میشه. با برابر صفر قرار دادن گرادیان تابع هزینه، میتوانیم w هایی که باعث مینیموم شدن تابع هزینه می شوند را پیدا کنیم. هدف ما به صفر رساندن یا حداقل رساندن تابع هزینه است به این معنی که خروجی مدل بیشترین شباهت را با y ها دارد

در قدم آخر، وزن ها یا پارمتر ها را اپدیت می کنیم سپس با پرارمتر های چدید تا N بار تابع هزینه را محاسبه تا دست آخر به بهترین پارامتر ها درست پیدا کنیم. پس از پایان N، درصورتی که به مرور، تابع هزینه مینیموم شده باشد، یعنی مدل همگرا شده و توانسته بهترین پارامتر را استخراج کند و برای مرحله ارزیابی از همان w ها استفاده کند.

Etta نرخ یادگیری است. به عباتی این اتا است که تعیین میکند پارامتر بروز رسانی شده تا چه اندازه با قبلی تفاوت کند.

فرایند ارزیابی:

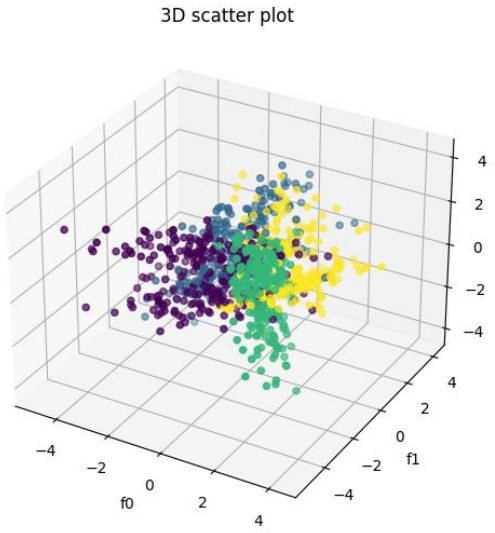


در فرایند ارزیابی، از همان وزن های بدست آمده از مرحله اموزش استفاده میکنیم. دیتاست تست را لود میکنیم. از همان مدل استفاده کرده و با استفاده از وزن های بدست امده، خط تولید شده را بعنوان ورودی به ان میدهیم. پس از محاسبه y^ ها ان ها را با تارگت های اصلی مقایسه میکنیم. در بخش ارزیابی، هدف این است که دقت بالا باشد. به عبارتی، y^ ها بشترین شباهت را با تارگت های اصلی داشته باشند. اگر دقت بدست امده مطلوب نباشد، باید بار دیگر وزن هارا در قسمت اموزش بروز رسانی کرد.

اگر یک طبقه بندی چند کلاسه از نوع ovr داشته باشیم، باید به اندازه n کلاس، n بار مدل را بسازیم. یعنی یک loop با n بار که مدل مثلا سیگموید، در هر بار y^ یک یا صفر بدهد. یک به منظور اینکه متعلق به کلاس است و صفر یعنی متعلق به آن کلاس نیست. میتوان یک را به تعبیر عدد دیگری نیز نسبت داد چون ما دستمان برای label زدن باز است. پس اگر ۱۰ کلاس داشته باشیم، در N ایتریشن، ده بار مدل را ترین می کنیم و به هر کلاس عدد مثلا ۱ تا ۱۰ نسبت می دهیم. سپس تمامی y^ ها را با y های اصلی مقایسه و تابع هزینه را بدست آورده و بر طبق توضیحات گذشته، وزن هارا بروز رسانی میکنیم. در فرایند ارزیابی نیز باید n بار ورودی ها به مدل داده شوند.

.۲

```
X, y = make_classification(n_samples=1000, n_classes=4,
                           n_features=3, n_informative=3,
                           n_redundant=0, random_state=74,
                           n_clusters_per_class=2)


print(X.shape, y.shape)

fig = plt.figure(figsize = (8, 6))
ax = fig.add_subplot(111, projection = '3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)

ax.set_title("3D scatter plot")
ax.set_xlabel("f0")
ax.set_ylabel("f1")
ax.set_zlabel("f2")

plt.show()
```



در این دیتاست، موضوع چالش برانگیز، درهم رفتگی دیتا هاست. با اتریبیوت class_sep میتوان این موضوع را چالشی تر نیز کرد. مقدار دیفالت آن ۱ است. هرچه نزدیک به صفر باشد دیتاها در هم رفته تر و پردازش سخت تر و هرچه بیشتر باشد، دیتا ها تمیز تر و بیشتر از هم فاصله دارند. به عبارتی، پیدا کردن خطی که بتواند کلاس ها را از هم جدا کند راحت تر می شود.

۳. بطور کلی، انتخاب ’adaptive‘ برای هایپر پارامتر learning rate باعث میشود درصد دقت دیتای تست به اندازه ۱ تا ۳ درصد بیشتر شود.

دو بار از مدل SGD استفاده شده. اولی داده های ورودی اسکیل شده اند و در دومی خیر. با این وجود، حتی اسکیل کردن داده ها نیز باعث نشده درصد دقت از حالت learning_rate=’adaptive’ بیشتر شود.

با random_state های ۱۲ و ۲۷ و ۹۹ نیز امتحان شد اما ۷۴ از همه درصد بهتری میدهد.

لازم به توضیح است که random_state برای ثابت ماندن نتایج و learning_rate گام یادگیری را تعیین می کند.

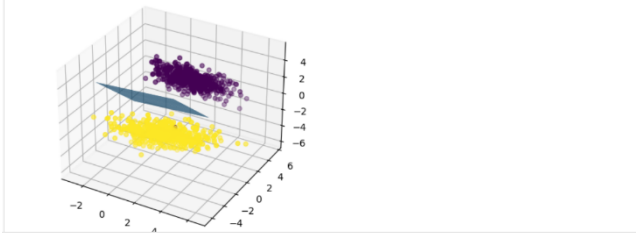
نمودار heatmap برای مدل logistic نیز رسم شده که بهتر میتواند نشان دهد ورودی چه تارگتی داشته و چه تارگتی پیش بینی کرده.

همچنین متوجه شدم که تعداد iteration ها از عددی به بعد، دیگر نمی تواند باعث بهبود مدل شود. بنابراین از عدد ۲۰۰۰ بیشتر نگذاشتم.

علاوه بر همه این ها، مقدار class_sep نیز تاثیر مستقیم روی دقت بالای مدل دارد. هرچه داده ها تمیز تر، مدل بهتر اموزش می بیند.

۴. در حالت ۲ کلاسه، بخوبی میتوان صفحه را ترسیم کرد:

```
import numpy as np
tmp = np.linspace(-3, 3, 51)
X1, X2 = np.meshgrid(tmp, tmp)

z = lambda X1, X2: (-model.intercept_[0] - model.coef_[0][0] * X1 - model.coef_[0][1] * X2) / model.coef_[0][2]

# Plot the decision boundary and data points
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X1, X2, z(X1, X2), alpha=0.9)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)
plt.show()
```



در حالتی که ۴ کلاس داریم، صفحه رسم شده چون دوبعدی است نمی تواند تمام ۴ کلاس را از هم تفکیک کند. توضیح قابل توجه درباره کدی که برای رسم مرز تصمیم در فضای سه بعدی نوشته شده است:

```
tmp = np.linspace(-3, 3, 51)
```

```
X1, X2 = np.meshgrid(tmp, tmp)

z = lambda X1, X2: (-model.intercept_[0] - model.coef_[0][0] * X1 -
model.coef_[0][1] * X2) / model.coef_[0][2]
```

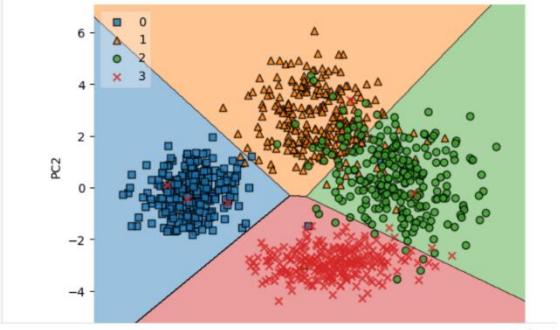
مرز تصمیم ما در یک فضای سه بعدی باید یک صفحه باشد بنابراین نیاز به یک ماتریس دو بعدی داریم. ابتدا یک ارایه از نقاط بین ۳ و -۳ درست میکنیم. پیس با استفاده از آن دو ماتریس دو بعدی ۵۱ در ۵۱ می سازیم.

در تابع lamda، coef_[0][0], coef_[0][1], coef_[0][2] سه ستون X هستند یا سه فیچر X. در معادله صفحه، جمله اول بعنوان عرض از مبدا و آنکه تقسیم شده، به منظور تعیین شیب صفحه می باشد.

برای درک بهتر، یک مدل دو بعدی از دیتاست نیز رسم شده. ایتدا با استفاده از PCA، تعداد ویژگی ها به دو بعد کاهش می باید و سپس دوباره مدل آموزش می بیند و با استفاده از plot_decision_regions مرز تصمیم کشیده می شود:

```
from sklearn.decomposition import PCA
from mlxtend.plotting import plot_decision_regions

pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)
model.fit(X_2d, y)

plot_decision_regions(X_2d, y, clf=model, legend=2)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:
  ax.scatter(
```



۵. با استفاده از لینک مربوطه، از محیط گرافیکی drawdata استفاده کردم و یک دیتاست با دو ویژگی و ۴ کلاس رسم کردم. سپس بصورت یک دیتافریم دراورده و x و y آن را بعنوان دو ویژگی دیتاست و ستون label بعنوان تارگت.
مدل های logestic regression و SGD را همانند قبل برای دیتا ست جدید تعریف کردم. در مدل logistic resression، از solver = lbfgs به جای sag استفاده شد که افزایش ۳۰ درصدی در دقت مدل داشت.
در مدل SGD نیز نرخ یادگیری adaptive بهترین دقت را به مدل می دهد. همچنین با افزایش eta از ۰.۰۱ به ۰.۰۹ نیز بهبود یک درصدر در دقت مدل مشاهده می شود.
برای استفاده از ستون لیبل ها ابتدا با map. و یک دیکشنری، لیبل های a تا d را به ۰ و ۱ و ۲ و ۳ تبدیل کرده و سپس کل مدل را با ورودی ها و لیبل ها فیت و بعد مرز تصمیم را رسم می کنیم.

مدل logistic regression بسیار بهتر از SGD عمل کرده.

سوال ۲:

۱. این دیتا ست بعنوان یک دیتاست مرجع برای یادگیری ماشین و استفاده از الگوریتم های آن استفاده می شود. شامل داده های جمع آوری شده سالم و معیوب از بلبرینگ را ارائه می دهد. داده های معیوب، شامل ده نوع هستند که هر کدام مربوط به بخشی از بلبرینگ هستند. در این تمرین از داده های معیوب مربوط به بخش Inner race fault (0.007 inch) استفاده می شود. همچنین از بخش DE داده ها برداشته شده اند که DE به بخش drive-end اشاره دارند[1]. بطور کلی در زمینه های صنعتی نیز از این دیتاست استفاده می شود. در تشخیص خرابی، پیش بینی نقص و خرابی و در تعمیرات از این داده ها استفاده می شود. دیتاست شتمل ۴ کلاس داده های مربوط به بلبرینگ های سالم، بلبرینگ های دارای نقص در انتهای درایو و نقس در انتهای فن هستند.
برای خواندن دیتاست، نام فایل‌های داده اولین حرف نشان‌دهنده موقعیت خرابی، سه عدد بعدی قطر خرابی و آخرین عدد بار بلبرینگ را نشان می‌دهد. به عنوان مثال، فایل داده 'B007_0' شامل داده‌های خرابی بلبرینگ با قطر خرابی ۰.۰۰۷ اینچ است که تحت بار موتور ۰ اسب بخار کار می‌کند.
با دستور !wget مستقیما دیتاست در کولب اپلود شد. به این دلیل که امکان دانلود مستقیم فایل وجود نداشت.
https://medium.com/@500087551/all-you-need-to-know-about-cwru-dataset-8d391577d8f2

۲. کد این بخش در کولب قابل دسترسی است

الف: استفاده از دو حلقه for تو در تو تا در ۱۰۰ ردیف ۲۰۰ سلول ایجاد کند. الگوریتم به طوری است که هر سلول دیتا ست تنها یکبار استفاده شده و شامل داده های تکراری نیست.

ب: استخراج ویژگی کمک میکند تابتوان از داده های پیچیده و بزرگ، اطلاعات مفید و دلخواه و مرتبط با هدف را استخراج کرد. همین باعث می شود تا الوریتم یادگیری ماشین بتواند با دقت بالاتری فرایند آموزش را انجام دهد. این کار به کاهش ابعاد دیتاست نیز کمک می کند چرا که تنها داده های مرتبط استفاده می شوند.

برای ارزیابی، یک دیتاست نمونه با اعداد صحیح بین ۱ تا ۱۰ در ۳ ردیف و ۵ ستون ایجاد شد و تابع مورد نظر روی آن تست شد.

یک ستون تارگت محتوی ۱ برای داده های نرمال و ۰ برای داده های ناقص به انتهای دیتاست ها اضافه شد.

ج: شافل کردن دیتا از اینکه دیتا ها با نظم خاصی قرار بگیرند جلوگیری میکند. در غیر این صورت ممکن است باعث اور فیتتیگ یا یادگیری بد مدل شود. در صورت شافل کردن، میتوان احتمال داد که از هر کلاس، تعداد داده خوبی در فرایند یادگیری شرکت دارد. در کراس ولیدیشن، به ارزیابی دقیقتر داده ها کمک میکند.

د: روش min-max: در این نوع از نرمالایز کردن، مقادیر ویژگی ها در یک محدوده مشخص بین ۰ و ۱ مقیاس بندی می شود.

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

و روش استاندارد سازی. در این روش مقادیر ویژگی ها بگونه ای مقیاسس بندی می شوند که میانگین آن ها صفر و انحراف مغیار آن ها ۱ شود:

$$X_{\text{std}} = \frac{X - \mu}{\sigma}$$

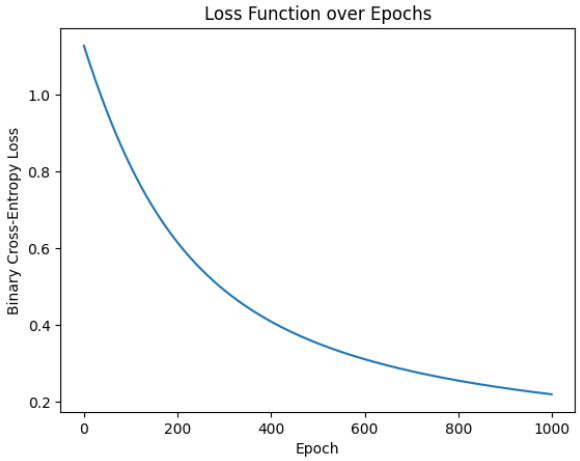
بطور کلی، نرمال سازی داده ها باعث کاهش زمان مورد نیاز برای فرایند یادگیری می شود. به این دلیل که در مقیاس های بزرگ، مدل ممکن است دچار محاسبات عددی دشوار در حین فرایند آموزش شود و با نرمال سازی، الگوریتم ها سریع تر همگرا می شوند چون تمام ویژگی ها در یک مقیاس مشابه قرار گرفته اند. این کار باعث می شود تا مدل قابلیت تعمیم پذیری نیز به داده های جدید پیدا کند. برای تضمین عملکرد صحیح مدل، داده های تست نیز باید نرمالایز شوند. این بخاطر این است که از ریسک اور فیت شدن مدل جلوگیری بشود.

در کد از روش دوم استفاده شده است. با استفاده از standard scaler.

۳. یکی از مشکلاتی که در حین اجرای فرایند یادگیری با آن موجه شدم، تغییر ابعاد y_train بود. Y_train بعد از تقسیم بندی دارای ابعاد (,134) بود و این مشکل ایجاد میکرد. بنابراین پیش از بازخوانی توابع از دستور reshape برای اصلاح ابعاد آن استفاده شد. نرخ یادگیری نیز کم انتخاب شد چون با نرخ بالا بعد از گذشت ۲۰۰ ایپاک، ارور بیشتر می شد.
نمودار تابع اتلاف در هر ایپاک رسم شده است. پس از اموزش، از همان w های بدست امده در اخرین ایپاک استفاده میکنیم و به ورودی مدل اعمال می کنیم.

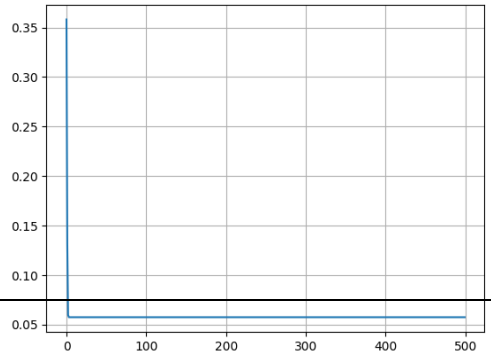
نمودار تابع اتلاف نشان می دهد که پس از گذشت هر ایپاک، ارور و خطای حاصله از استفاده از کراس انتروپی بین خروجی مدل و تارگت های اصلی در حال کاهش و همگرا شدن به صفر است. اگر نمودار به طور مدارم کاهش یابد و به یک مقدار پایدار برسد، نشان دهنده عملکرد خوب مدل است. اما ممکن است برخی اوقات بدلیل زیاد بود ایتریشن ها، مدل بیش از حد آموزش ببیند. این سبب می شود با وجود اینکه نمودار نزولی باشد اما اگر ورودی های جدید به مدل اعمال شود نتواند درست پیش بینی کند.

یکی از راه حل های موثر می تواند کاهش تعداد ایتریشن ها باشد.



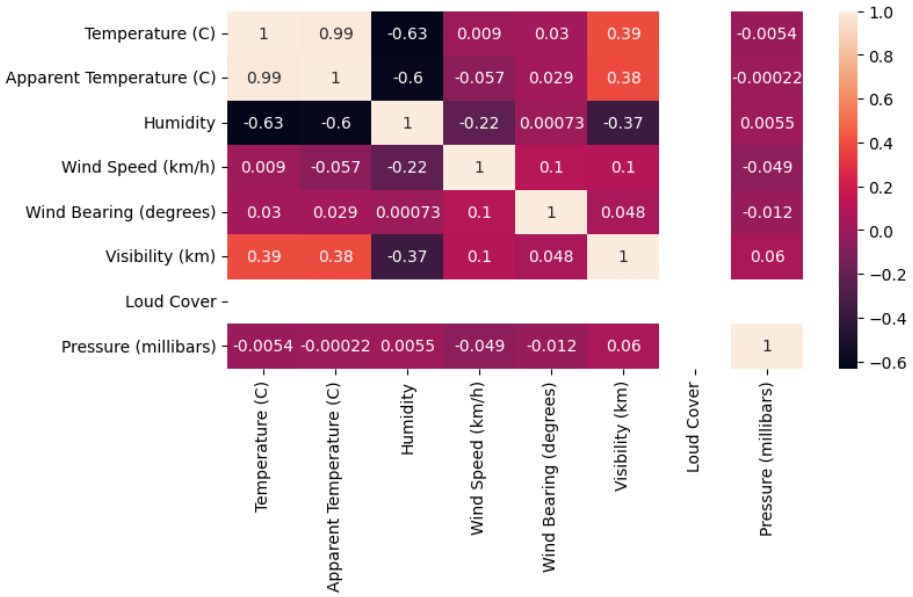
۴. دقت های بدست آمده پس از استفاده از کتابخانه های پایتون، بهتر هستند. یک دلیل آن می تواند وجود پارامتر های آن ها باشند که در کد دست نویس وجود ندارند. البته کد ما از گرادیان نزولی برای بهینه سازی و بروز رسانی ضرایب استفاده کرد در حالی که کتابخانه های پایتون از بهینه سازی های پیشرفته تری استفاده می کنند.

در حالت کلی امکان پذیر نیست که بتوان مانند سوال قبل، خطا را در هر ایپاک محاسبه و در آخر بصورت گراف نمایش داد. چون مدل پایتونی، در n ایتریشن، به یکباره w بهینه را محاسبه میکند و خطا را ذخیره نمی کند. تابع logistic regression در کتابخانه sklearn متدی به نام predict_proba دارد. خروجی این متد، احتمالات کلاس ۱ و ۰ است. در فلوچارت فرایند یادگیری، خروجی سیگموید نیز بصورت احتمال نمایش داده می شد. می توان از این متد استفاده کرد و خروجی آن را بعنوان y_hat در نظر گرفت و با استفاده از تابع log_loss که همان کراس انتروپی است، در هر بار اموزش، خطا را محاسبه و روی نمودار نمایش داد. این کد در هر بار ایپاک، یادگیری را از همان ایپاکی که رها کرده بود شروع می کند. این نمودار نشان می دهد که در همان تعداد اموزش اولیه، مدل توانسته همگرا شود.



سوال ۳:

۱. هیت مپ دیتاست بصورت زیر است:



این نمودار با استفاده از ماتریس همبستگی دیتاست بدیت امده. ماتریس همبستگی تنها روی ستون هایی اعمال می شود که دیتای ان ها بصورت عددی هستند. این ماترین عددی بین ۱ تا -۱ برای هر دو فیچر بدست می دهد. عدد یک به آن معنی است که این دو فیچر یکدیگر را تقویت یا تضعیف میکند. با افزایش یکی دیگری نیز افزایش می یابد. -۱ نیز همان ولی در خلاف جهت و ۰ نیز یعنی فیچر ها از هم مستقل اند. بعنوان مثال رطوبت رابطه در خلاف جهت با دما دارد. یعنی با بالا رفتن دما، رطوبت کمی کاهش می یابد. این رابطه برای دمای ظاهری و رطوبت نیز صادف است با این تفاوت که رابطه بین رطوبت و دما به اندازه ۰.۳ وابسته تر از رابطه بین دمای ظاهری و رطوبت است. ستون loud cover نیز کلا شامل دیتای صفر است.

نمودار هیستوگرام فراوانی هر فیچر نمایش داده شده است. این نمودار های نشان میدهند که بطور مثال، در این دیتاست ۱۰ ساله، حدود ۳۰۰۰۰ ساعت دما بین ۵ تا ۱۰ درجه بوده. شکل نمودار یک نمودار گوسی است که میانگین آن محدودا بین ۱۰ درجه سانتی گراد است. پس این نمودار نشان می دهد که میانگین دما محدوده ۱۰ درجه سانتی گراد بیشترین فراوانی را داشته. در نمودار رطوبت نیز مشاهده می شود که حدود ۱۴۰۰۰ ساعت، رطوبت به میزان ۰.۹ بوده. به این معنی که در طول این ده سال، اکثر اوقات، رطوبت هوا زیاد بوده. نمودار جهت باد دارای سه قله است. یعنی سه جهت باد به ترتیب دارای بیشترین فراوانی در دیتاست بوده اند. نمودار وضوح دید دارای تقارن نیست و تقریبا می توان گفت که در اکثر اوقات سال، وضوح دید یکسان بوده و نوسان نداشته.

۲. از آنجایی که داده های temperature و apparent temperature ۹۹ درصد شبیه به هم اند تنها تخمین را روی temperatureانجام می دهیم.

با اعمال LS بار اول میانگین مربعات خطا برابر با ۵۴ و با اضافه کردن فیچر Visibility، میانگین مربعات خطا به ۵۱ رسید. هدف از انتخاب Visibility این بود که بیشترین همبستگی را با دما دارد. همچنین با انتخاب ۰.۳۳ درصد از دیتاست بعنوان داده ارزیابی، mse یک واحد بهبود پیدا کرد. نمودار رسم شده روی تعدادی از نقاط دیتاست نشان می دهد تا حدودی توانسته خط مورد نظر را پیدا کند. انتخاب هر کدام از معیار های رطوبت و یا وضوح دید بعنوان فیچر، میانگین مربعات خطا را زیاد می کند. با اضافه کردن سرعت باد به فیچر ها، این عدد به ۴۹ کاهش پیدا می کند. این عدد بیانگر میانگین مربع اختلاف بین دیتای پیش بینی شده و دیتای واقعی است.

الگوریتم های RLS نیز مشابه LS رفتار می کنند. با این تفاوت که در هر تکرار میانگین مربعات خطا محاسبه و صرایب مدل بروز می شوند. این بروز رسانی در یک حلقه for انجام می شود و مدام ارور ها محاسبه می شوند روی دیتای موزشی و سپس ضرایب بروز می شوند.

در ارزیابی صورت گرفته با اعمال RLS، انتظار این بود که مقدار mse کمتر باشد ولی تقریبا ۲.۵ برابر قبل بود. البته با اضافه کردن فیچر های وضوح دید و سرعت باد، به عدد ۷۵ کاهش یافت. مقدار forgetting facor نیز تغییر یافت اما بهترین مقدار همان ۰.۹۹ است که نشان می دهد مدل وزن زیادی به دیتاهای جدید می دهد.

۳. گاهی لازم است تا دیتاست وزن گذاری شود. به این معنا که مدل تنها روی بخشی از دیتاست که وزن بیشتری دارد تمرکز بیشتری قرار دهد. کمک میکند تا فرایند یادگیری به سمتی که ما میخواهیم هدایت شود.

ضریب R-squared معیاری است که نشان می‌دهد چه قدر متغیرهای مستقل در مدل می‌توانند تغییرات در متغیر وابسته را توضیح دهند. این معیار از ۰ تا ۱ است به عبارت دیگر، هر چقدر مقدار R-squared نزدیکتر به ۱ باشد، مدل بهتری را نشان می‌دهد و توانایی پیش‌بینی بهتری دارد. در نتایج مدل، مقدار آن ۳۹ درصد است یعنی فیچر ها ۳۹ درصد میتوانند y را توضیح یا پیش بینی کنند.

۴. الگوریتم مبتنی بر تجزیه QR-Decomposition-Based RLS یک روش پیشرفته برای اجرای الگوریتم RLS است. این الگوریتم از تجزیه QR برای مثلثی‌سازی ماتریس داده‌های ورودی استفاده می‌کند و به عنوان یک روش جایگزین برای پیاده‌سازی RLS مطرح شده است. یکی از مزایای اصلی ین الگوریتم، امکان پیاده‌سازی در آرایه‌های سیستولیک و رفتار عددی بهبود یافته هنگام در نظر گرفتن اثرات کوانتیزاسیون است .

الگوریتم‌های RLS مبتنی بر تجزیه QR به دلیل کاهش پیچیدگی محاسباتی و بهبود پایداری عددی، برای پردازش سیگنال‌های واقعی در زمان واقعی مورد توجه قرار گرفته‌اند.

https://link.springer.com/chapter/10.1007/978-3-030-29057-3_9