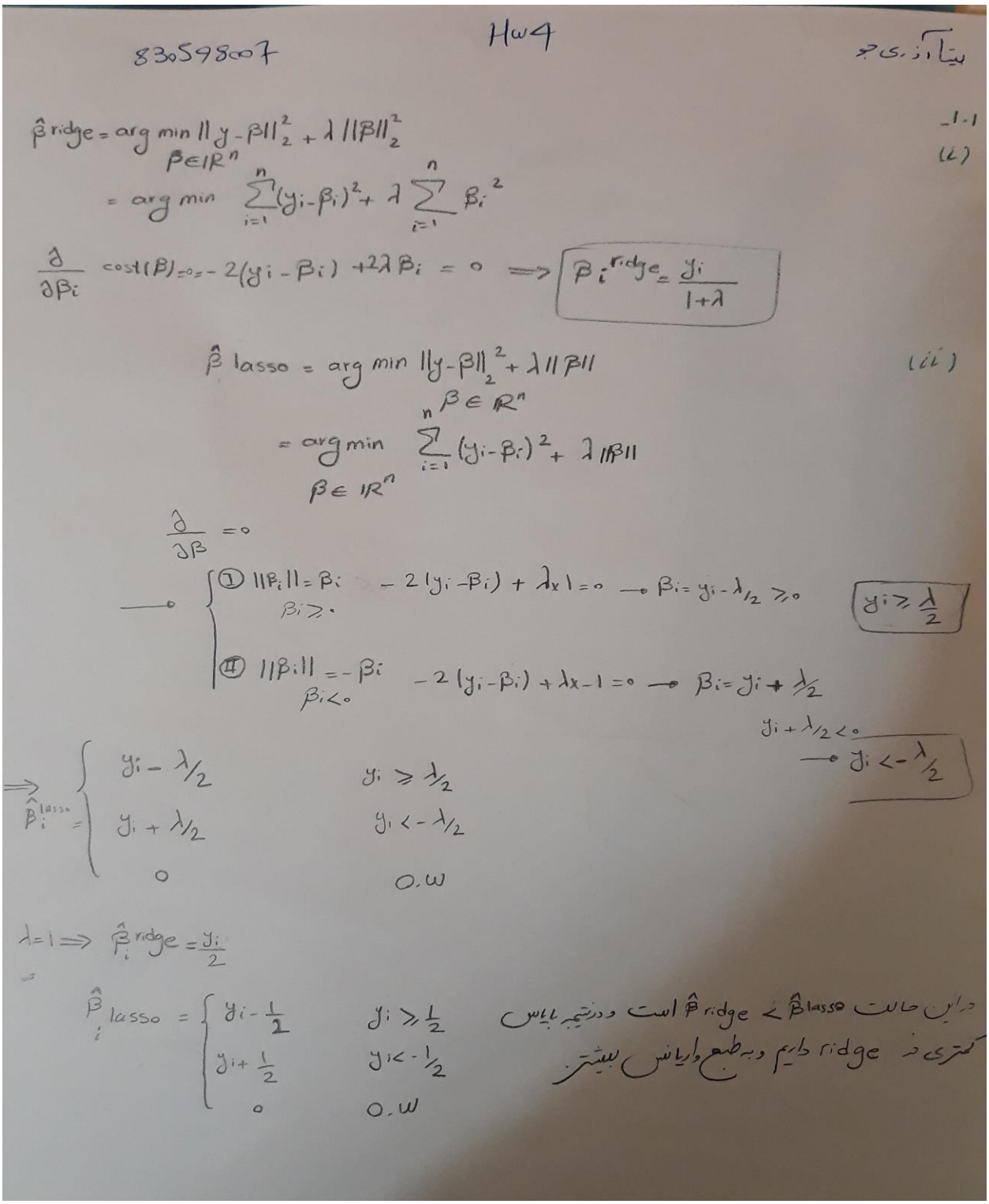
گزارش کار تمرین چهارم

بیتا آذری جو ۸۳۰۵۹۸۰۰۷

بخش اول :



830598007 تمرین چهارم یادگیری ماشین ...یا آذری جو

8.6

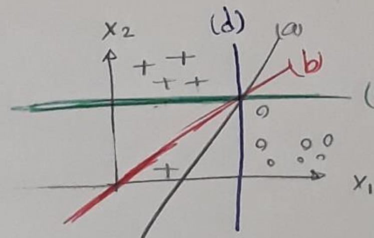
a) F ، این تابع convex است

b) F ، L2 regularization وزن های بزرگتر را بیشتر penalize میکند و این هیچ ربطی به sparsity وزن ها ندارد در واقع هدفش این نیست که مجموعه‌ای از وزن ها را به صفر برساند.

c) T ، هرچه به step function نزدیک تر می‌شویم ، log loss روی داده‌های train کمتر می‌شود.
هدف از regularization جلوگیری از overfitting است معمولاً در این روش $w \to \infty$ نمی‌رود.

d) F ، انتظار ما از افزایش λ کاهش log likelihood روی داده‌های train است. مدل در نهایت بایاس زیادی روی داده‌های train وقت خواهد داشت.
هدف ما ماکزیمم کردن log likelihood و مینیمم کردن log likelihood - است

e) F ، در ابتدا می‌خواهیم از overfitting جلوگیری کنیم انتظار داریم loglikelihood زیاد شود اما در نهایت انتظار داریم کم شود چون flexibility مدل کم می‌شود . مدل در نهایت بایاس زیادی برای داده‌های train , test خواهد داشت چون $w \to \infty$ می‌روند.



8.7

a) خیر تابع لزوماً یکتا نیست . هر روش gradient descent به تعداد iteration ها و learning rate بستگی دارد هرچه iteration بیشتر باشد ، و هم عدد (ΣW?) خوبی انتخاب شود نتیجه‌ی بهتری می‌گیریم.
چون داده‌ها به صورت خطی جداپذیرند می‌توان از logistic regression برای یافتن خط استفاده کرد و خطای classification نخواهیم داشت.

b) در صورتی که $w_0 \to \infty$ خط جداساز (0,0) را رد خواهد کرد و طبق شکل که با خط قرمز کشیده شده ۲ یک نمونه + را در دسته‌ی o تشخیص خواهد داد . در واقع logistic regression بهترین خطی را پیدا خواهد کرد که از (0,0) می‌گذرد.

c) در صورتی که $w_1 \to \infty$ ، logistic regression بهترین خط افقی را خواهد یافت مانند خط سبز در شکل بالا .
در این صورت هم یک نمونه را نادرست classify می‌کند.

d) در این صورت خط جداساز یک خط عمودی خواهد بود مانند خط آبی در شکل بالا . خطا نخواهیم داشت.

بخش دوم :

کد الگوریتم logistic regression با روش گرادیان کاهشی در شکل زیر آمده است:

```
class logistic_regression:

    predicted_reults = []
    losses=[]
    def __init__(self,X_train, X_test, y_train, y_test,alpha=0.3,epochs=20000):
        self.X_train=X_train
        self.y_train=y_train
        self.X_test=X_test
        self.y_test=y_test
        self.alpha = alpha
        self.epochs = epochs


    def sigmoid(self,z):
        return 1.0 / (1 + np.exp(-z))

    def loss(self,h):
        return np.sum(-self.y_train * np.log(h) - (1 - self.y_train) * np.log(1 - h))/self.X_train.shape[0]

    def graient_descent(self,h): # 1/m *(h-y)*x for every element of matrix
        return np.dot(h-self.y_train,self.X_train)/self.X_train.shape[0]

    def train(self):
        self.theta=np.zeros(X_train.shape[1],dtype=np.float128)

        for i in range(self.epochs):
            z = np.dot(X_train,self.theta)
            h=self.sigmoid(z)
            self.theta=self.theta-(self.alpha*self.graient_descent(h))

            self.losses.append(self.loss(h))
```

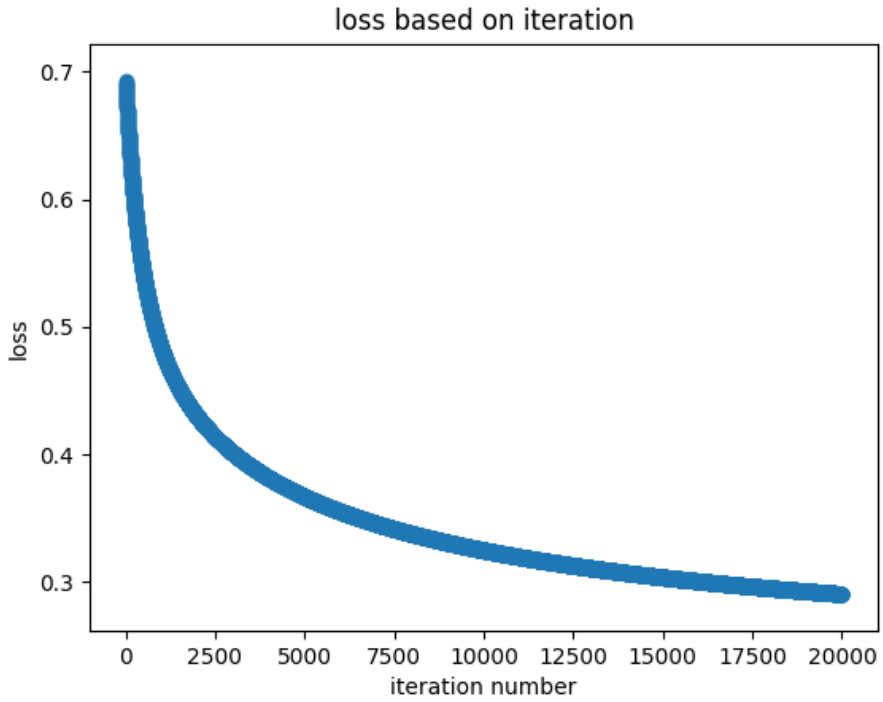
داده ها ابتدا normalize شدند طبق فرمول زیر :

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

دقت حاصل از این الگوریتم بالا بوده و خطای آن ۹ درصد بود. تعداد پیمایش ها به طور پیش فرض ۲۰۰۰۰ تا و نرخ یادگیری (alpha) هم ۰,۳ در نظر گرفته شد.

```
bita@bita-K401UQK:~/programming/Logistic Regression$ python logistic_regression.py
error rate :  9.338%
```

در شکل زیر هم مشاهده می شود هرچه تعداد پیمایش ها بیشتر شود مقدار loss function کمتر خواهد شد و ما به هدفمان که کمینه کردن loss function بود نزدیک می شویم.



می بینیم که در شکل بالا مقدار loss پس از عبور از ۰,۳ با سرعت بسیار کمتری کاهش می یابد. در این مسئله با پارامترهایی که ست کردیم در نهایت ۰,۲۹ به دست آمد. حتی زمانی که تعداد پیمایش ها هم ۱۰۰۰۰۰ گذاشتم به ۰,۲۶ رسیدم که حاکی از کاهش سرعت کم شدن loss function بود یعنی ما تقریبا در minimum قرار گرفتیم.

مقایسه با Naïve Bayes :

در این بخش از Naïve Bayes برای بررسی دقت مجدد استفاده شد. در این جا مجبور شدیم ۱۰ ویژگی آخر را حذف کنیم که شامل فرکانس تعدادی کاراکتر غیر حرف بود که خود الگوریتم موقع tokenize کردن این ها را حذف می کرد چون جزو حروف به حساب نمی آیند. همچنین ویژگی های capital_run_length_average ، capital_run_length_longest و capital_run_length_total که ربطی به خود کلمه ندارند و از نظر مفهومی در Naïve Bayes جایگاهی ندارند، حذف شدند.

نتیجه ی حاصل این بود که دقت بسیار پایین آمد چون ویژگی حذف کردیم و فرض استقلال بین ویژگی ها را در نظر گرفتیم که در بحث تشخیص ایمیل spam درست نمی باشد باید ۱۰ ویژگی ای که مجبور به حذفش شدیم را در نظر می گرفتیم اما در نظر گرفتن این ویژگی ها با فرض های Naïve Bayes در تناقض بود. بنابراین ما خطای زیادی را محتمل شدیم.

شکل مقایسه ی این خطا ها :

```
bita@bita-K401UQK:~/programming/Logistic Regression$ python evaluation.py
Evaluating Logistic Regression :
error rate of logistic regression :  9.338%

Evaluating Naive Bayes classifier ;
error rate of Naive Bayes classifier  :  37.568%
```

خطای ما به ۳۷ درصد رسید که اصلا خوب نمی باشد.

References:

[1] https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8

[2] https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac]

[3] https://stats.stackexchange.com/questions/48360/is-standardization-needed-before-fitting-logistic-regression

[4] https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e