گزارش کار تمرین چهارم

درس یادگیری ماشین

استاد درس: دکتر کمندی

دانشجویان گروه:

محیا معتمدی ۸۱۰۸۹۷۰۵۳

مینو احمدی ۸۱۰۸۹۷۰۳۲

دانشکده علوم مهندسی، دانشگاه تهران

بهار ۱۴۰۱

در قسمت اول ابتدا دیتا ها را به نسبت ۸۰ به ۲۰ برای اموزش و تست مدل جدا کردیم
سپس میانگین و واریانس و احتمال پیشین را محاسبه کردیم سپس این احتمال ها را بر اساس قوی بودن sort کردیم .

```
means = train.groupby(["diagnosis"]).mean() # Estimate mean of each class, feature
var = train.groupby(["diagnosis"]).var() # Estimate variance of each class, feature
prior = (train.groupby("diagnosis").count() / len(train)).iloc[:,1] # Estimate prior probabilities
classes = np.unique(train["diagnosis"].tolist()) # Storing all possible classes
```

فرآیند طبقه‌بندی در یک مدل ساده بیز با محاسبه احتمال پسین برای همه کلاس‌ها با توجه به داده‌های فعلی انجام می‌شود.

$$P(C_i \mid x) = \frac{P(x \mid C_i)P(C_i)}{P(x)}$$

$$\propto P(x \mid C_i)P(C_i) \quad \dots \text{Line 2}$$

$$\propto P(C_i)\prod_{k=1}^{K} P(x_k \mid C_i) \quad \dots \text{Line 3}$$

$$\text{Where, } P(x_k \mid C_i) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}}\exp\left\{-\frac{(x-\mu_{ik})^2}{2\sigma^2}\right\} \quad \dots \text{Assuming gaussian distribution}$$

در خط 2،P(x)حذف می شود. این به این دلیل است که احتمال مشاهده x در همه کلاس ها یکسان است. این روی محاسبه تاثیری نخواهد داشت زیرا ما نیازی به پیدا کردن احتمال واقعی پسین کلاس ها نداریم. در عوض، فقط باید ببینیم کدام کلاس بیشترین احتمال پسین را دارد. تقسیم همه احتمالات پسین بر یک ثابت همان نتایجی را به دست می دهد که مقدار آنها بزرگترین است.

در خط 3، P(x | C) بسط داده شده است تا هر ستون را شامل شود.

P(x | C)اولیه احتمال مشاهده x (کل ردیف)، با توجه به کلاس فعلی است.
نسخه توسعه یافته احتمال هر ستون از ردیف و کلاس داده شده (داده های فردی) را محاسبه می کند.

$$P(x \mid C_i) = P(x_1 \cap x_2 \cap \cdots \cap x_K \mid C_i)$$

$$= P(x_1 \mid C_i)P(x_2 \mid C_i) \ \ldots \ P(x_K \mid C_i)$$

$$= \prod_{k=1}^{K} P(x_k \mid C_i)$$

این یکی از مفروضات اساسی مدل Naïve Bayes است، که فرض می کند همه ویژگی ها مستقل از یکدیگر هستند. در عمل، به ندرت این طوری است . به همین دلیل نام مدل را Naïve Bayes گذاشتند.

درطبقه بندی، میتوان به جای خود احتمال پسین، log-posterior را محاسبه کرد. این کار پیش بینی را تغییر نمی دهد چون(x)logیک تابع اکیدا صعودی است یعنی اگر b < a داشته باشیم، log(a)> log(b) است. بنابراین ترتیب احتمال پسین بر اساس اندازه تغییر نخواهد کرد. ما از تبدیل log در کد خود استفاده کردیم.

Instead of this,

$$P(C_i \mid x) \propto P(C_i) \prod_{k=1}^{K} P(x_k \mid C_i)$$

We will use this,

$$\log P(C_i \mid x) \propto \log P(C_i) + \sum_{k=1}^{K} \log P(x_k \mid C_i)$$

فرآیند طبقه‌بندی این مدل به سادگی با محاسبه log-posterior و مشاهده اینکه کدام کلاس بیشترین مقدار را دارد انجام می‌شود.
ما به کمک دو تابع normal و Predictاین مراحل را در کد پیاده سازی کردیم .

```
def Normal(n, mu, var):

    # Function to return pdf of Normal(mu, var) evaluated at x
    sd = np.sqrt(var)
    pdf = (np.e ** (-0.5 * ((n - mu)/sd) ** 2)) / (sd * np.sqrt(2 * np.pi))

    return pdf

def Predict(X,x_tr):
    Predictions = []

    for i in X.index: # Loop through each instances

        ClassLikelihood = []
        instance = X.loc[i]

        for cls in classes: # Loop through each class

            FeatureLikelihoods = []
            FeatureLikelihoods.append(np.log(prior[cls])) # Append log prior of class 'cls'

            for col in x_tr.columns: # Loop through each feature

                data = instance[col]

                mean = means[col].loc[cls] # Find the mean of column 'col' that are in class 'cls'
                variance = var[col].loc[cls] # Find the variance of column 'col' that are in class 'cls'

                Likelihood = Normal(data, mean, variance)

                if Likelihood != 0:
                    Likelihood = np.log(Likelihood) # Find the log-likelihood evaluated at x
                else:
                    Likelihood = 1/len(train)

                FeatureLikelihoods.append(Likelihood)

            TotalLikelihood = sum(FeatureLikelihoods) # Calculate posterior
            ClassLikelihood.append(TotalLikelihood)

        MaxIndex = ClassLikelihood.index(max(ClassLikelihood)) # Find largest posterior position
        Prediction = classes[MaxIndex]
        Predictions.append(Prediction)

    return Predictions
```

در خطی که با نارنجی مشخص شده log prior , log P(ci) محاسبه شده است .

در خطی که با سبز مشخص شده log P(xk/ci) محاسبه شده است .

در خطی که با بنفش مشخص شده است log-posterior با جمع کردن - log prior و log-likelihoods محاسبه شده است .

در خطی که با فلش ابی مشخص شده است طبقه ای که بزرگ ترین log-posterior را دارد را پیدا میکنند و متغیر Predictions تمام پیش بینی های انجام شده توسط مدل را ذخیره می کند.

هم چنین در ادامه کد با پیاده سازی تابع accuracy میزان accuracy را محاسبه کردیم. و هم چنین ماتریس confusion را نیز چاپ کردیم نتیجه به صورت زیر است:

```
0.8947368421052632
[[34 11]
 [ 1 68]]
```

سپس PCA را انجام دادیم

ابتدا ماتریس کواریانس را تشکیل دادیم سپس مقادیر ویژه و بردار ویژه ها را محاسبه کردیم و تاپل هایی از مقدار و بردار ها تشکیل دادیم ان ها را بر اساس بزرگی مقدار ویژه مرتب کردیم و به صورت درصد میزان اهمیت و تاثیر ان ها را چاپ کردیم که به صورت زیر حاصل شد.

```
[63.44682538735288, 20.421829719538756, 15.771593100364715, 0.3318488492015147, 0.027902943542144225]
```

همان طور که مشخص است چون درصد تاثیر بردار ویژه ۴ و ۵ بسیار کم است از ان دو صرف نظر کرده و ابعاد ویژگی ها را از ۵ به ۳ کاهش میدهیم .

سپس matrix_w را تشکیل داده و در دیتای خود ضرب داخلی میکنیم و ابعاد دیتا را کاهش میدهیم.

```
matrix_w = np.hstack((eig_pairs[0][1].reshape(5,1),
                      eig_pairs[1][1].reshape(5,1),
                      eig_pairs[2][1].reshape(5,1)))
X_reduce = np.dot((X_std),matrix_w)
```

سپس تمام مراحل قبلی بیز ساده را بر دیتایی که ابعاد ان کاهش یافته است پیاده سازی میکنیم .

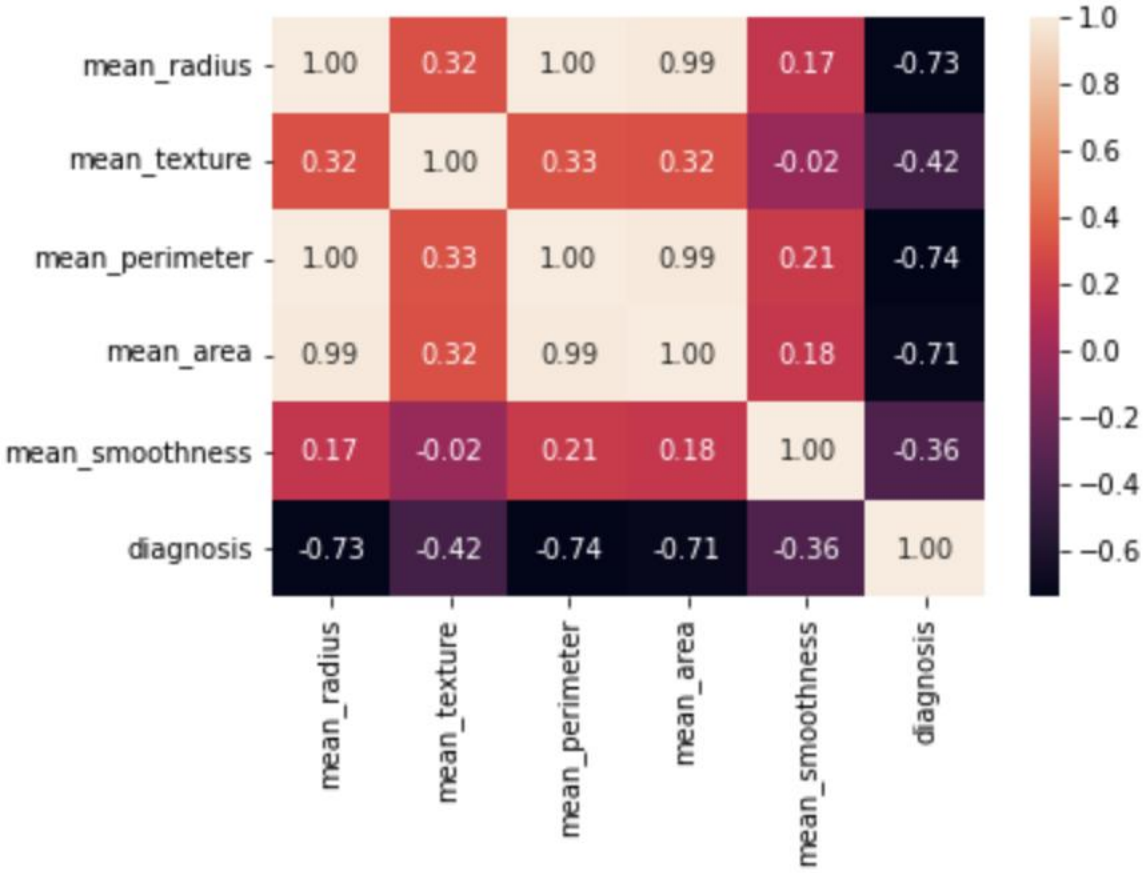
نتیجه accuracy و ماتریس confusion برای ان به شکل زیرحاصل شد:

```
[[35 10]
 [ 1 68]]
0.9035087719298246
```

همانطور که مشاهده میشود accuracy افزایش یافته است که این موضوع نشان میدهد که correlation بین ویژگی ها بالا بوده است که با اعمال PCA این accuracy افزایش یافته است .

```
sns.heatmap(df.corr(),annot=True,fmt=".2f")
```

نمودار به شکل زیر است :



وقتی ابعاد کاهش بیابد ۳ ویژگی انتخاب میشود که دیده میشود ۳ ویژگی correlationهای بسیار بالایی در حد ۰.۹۹ و ۱ دارند که این موضوع دلیل افزایش accuracy بعد از اعمال PCA می باشد.

منابع در جهت افزایش accuracy :

https://algotech.netlify.app/blog/time-and-accuracy-/improvement-using-pca

https://stats.stackexchange.com/questions/55034/how-does-pca-improve-the-accuracy-of-a-predictive-model

تمام این پروژه به کمک حالتی که به جای پیاده سازی دستی از تابع پیش ساخته شده guassianNB استفاده شود نیز در فایل دوم پیاده سازی شده است که با اجرای ان دقت کاهش یافته که به دلیل تقلیل ابعاد می باشد.

با وجود تغییر بسیار اندک در دقت حالت بدون اعمال و با اعمال PCA در accuracy ولی میزان سرعت بعد از اعمال PCA در هر دو حالت بسیار بالاتر میرود و از فواید اعمال PCA افزایش سرعت به کمک تقلیل ابعاد می باشد.
برای محاسبه ی سرعت ران کردن هر cell در کولب ما از کتابخانه استفاده کردیم که در خود کولب اماده نیست روش افزودن این کتابخانه به صورت زیر است :

```
!pip install ipython-autotime

%load_ext autotime
```

با استفاده از ان بعد از ران شدن هر cell به صورت خودکار زمان مورد نیاز ران شدن ان نوشته میشود.
در حالت قبل از اعمال PCA زمان ۳۹۲ میلی ثانیه و اما بعد از اعمال ان۲۵۰ میلی ثانیه زمان برد که نشان دهنده ی افزایش سرعت میباشد

```
time: 392 ms (started: 2022-06-10 13:05:38 +00:00)
time: 252 ms (started: 2022-06-10 13:05:38 +00:00)
```