به نام خدا

گزارش سوال 3 و 4:

سوال 3:

در ابتدا روی داده پیش پردازش انجام میدهیم. مثلا در stalk-root داده ی بی ربط علامت سوال ؟ را داریم که من جای آن را با حرفی که بیشترین تکرار در این فیچر را داشت پر کردم. برای فیچر ring-number (بیانگر تعداد ring) حروف را به اعداد متناظرش تبدیل کردم. برای باقی فیچر ها label encoding را انجام میدهیم یعنی به هر مقدار category یک عدد نسبت دادم (البته از one hot encoding هم میشد استفاده کرد ولی با توجه به اینکه قسمت مهم این سوال پیاده سازی الگوریتم ها بود بنابراین روی قسمت پیش پردازش به حد مقعول وقت گذاشتم).

قسمت پیاده سازی Logistic Regression و NaiveBayes در کد لیبل گذاری شده است.

در Logistic Regression قسمت مهم بهینه سازی است که اگر از gradient descent استفاده کنیم آپدیت وزن ها با توجه به این فرمول:

```
self.param -= self.learning_rate * -(y - y_pred).dot(X)
```

میباشد که این فرمول در سوال 2 با استفاده از مشتق تابع هزینه نسبت به پارامتر ها به دست آمده است.

اگر از gradient descent استفاده نکردیم بجای آن از batch optimization با حداقل مربع خطا استفاده میکنیم.

در NaiveBayse باید prior و likelihood محاسبه شود و در آخر با ضرب این دو احتمال پسین یا همان posterior به دست می آید و کلاسی انتخاب میشود که بیشترین احتمال پسین را دارد.

نتیجه accuracy، precision ، recall و f1 برای هر دو الگوریتم یکسان به دست آمد. البته در confusion matrix تفاوت مشاهده شد که آن را نمایش دادم.

برای محاسبه precision ، recall و f1 از کتابخانه sklearn استفاده کردم ولی این کتابخانه در هیچ کجای دیگر جز این قسمت استفاده نشده است.

سوال4:

برای کامپیوتر حرف X را در نظر گرفتم و برای یوزر حرف O. سپس برای برنده شده کامپیوتر 5 قانون در نظر گرفتم و در تابع def computer() کامپیوتر به ترتیب از شماره 1 شروع به اجرای هر کدام از قوانینی که شرایطش ارضا شده باشد میکند.

1- اگر دو خانه به صورت افقی با حرف X پر شده بود خانه افقی سوم را هم پر کن تا برنده شوی. (برای حالت عمودی و قطری نیز همین کد پیاده سازی شده)
2- اگر دو خانه به صورت افقی با حرف O توسط حریف پر شده بود خانه افقی سوم را پر کن تا مانع برنده شدن حریف شوی. (برای حالت عمودی و قطری نیز همین کد پیاده سازی شده)
3- اگر گوشه ها خالی بود یکی از آن ها را پر کن.
4- اگر مرکز خالی بود آن را پر کن.
5- اگر خانه های لبه خالی بود آن ها را پر کن.

تابع def user() از کاربر ورودی یکی از 9 خانه را میگیرد. ضمن اینکه چک میکند شماره وارد شده از بین عدد 1 تا 9 باشد و اینکه خانه انتخابی پر نشده باشد.

تابع check_rows() ، check_columns() و check_diagonals() برای چک کردن حالت های برنده شدن پیاده سازی شده است. هر جایی که سه خانه متوالی توسط یک بازیکن پر شده باشد برنده اعلام و بازی خاتمه میابد.

تابع check_for_tie() برای این پیاده سازی شده است که اگر هیچ سه خانه متوالی ای توسط یک حرف یکسان پر نشده باشد و تمام خانه ها هم پر شده باشند اعلام کند بازی برنده ندارد و به اصطلاح tie رخ داده است.

از دیگر توابع مهم flip_player() و handle_player(player) میباشند که برای مدیریت نوبت بازیکن پیاده سازی شده اند.