به نام خدا

علی فتحی

۹۴۱۰۹۲۰۵

گزارش تمرین سری اول شبکه‌های عمیق

دکتر فاطمی زاده

توضیحات اولیه:

شماره دانشجویی اینجانب: ۹۴۱۰۹۲۰۵ و چهار شماره آخر آن ۹۲۰۵ می‌باشد. لذا دو عدد کمتر آن ۰ و ۲ هستند و طبقه‌بند طراحی شده، طبقه بند این دو عدد است و برای اعدادی به جز ۰ و ۲ کارایی ندارد (مجموعه مرجع ما تنها شامل نمونه‌های Train و Test با اعداد ۰ و ۲ است).

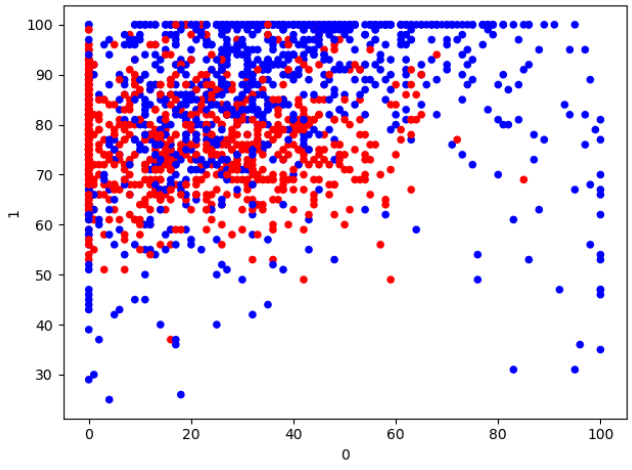
زبان برنامه نویسی، زبان python است و قسمت‌های مختلف ک دبا کامنت گذاری مشخص شده اند. برای خواندن داده‌ها از اکسل از کتابخانه pandas و برای اعمال ریاضی مانند رادیکال و توان از کتابخانه numpy استفاده شده است و کتابخانه دیگری مورد استفاده قرار نگرفته است.

توضیح مختصر متغیرها:

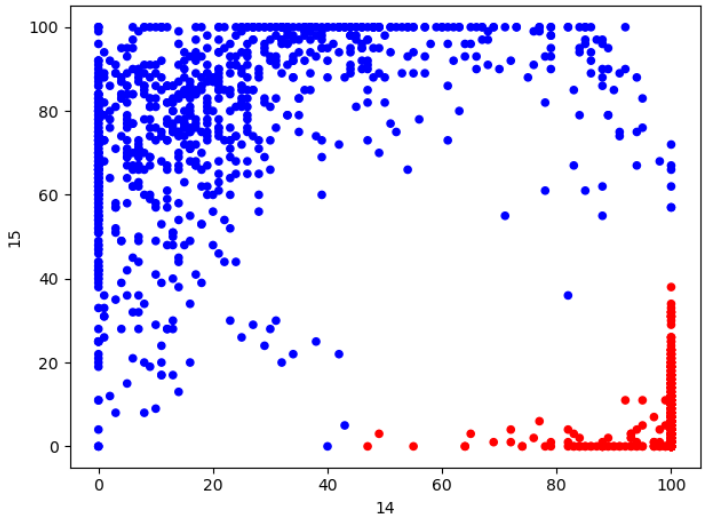
متغیر nF : تعداد ویژگی‌ها، که در اسنجا برابر ثابت ۱۶ است
مجموعه داده allData : تمام داده‌های سطری و ستونی اکسل (تمام داده‌ها)
ماتریس num0and2 : ماتریسی که سطرهایش نمونه های با برچسب ۰ یا ۲ هستند و ستون‌هایش ویژگی‌ها
ماتریس allDataLabeled : همان ماتریس num0and2 است که عدد ۲ با برچسب ۱ و عدد ۰ با برچسب ۱- جایگزین شده است (دسته بندی دو کلاسه)
بردار DataLebel : بردار برچسب‌ها
ماتریس allDataFeatures : ماتریس ویژگی‌ها که یک ۱ به انتهای آن اضافه شده است (متناظر با w0 ، عرض از مبدا خط جدا کننده پرسپترون)
ماتریس trainData : ویژگی داده‌های آموزش، که ۸۰ درصد اول نمونه‌ها هستند
ماتریس TestData : ویژگی داده‌های تست، که ۲۰ درصد آخر نمونه‌ها هستند
بردار trainLabel : برچسب داده‌های آموزش
بردار TestLabel : برچسب داده‌های تست

نکته: مقدار اپسیلون، گام حرکت در روش GradientDescend ، در اینجا برابر با مقدار ۰.۰۱ قرار داده شده است.

برای شهود بیشتر، مقادیر دو ویژگی بر حسب هم برای اعداد ۰ و ۲ رسم شده است:



ویژگی اول و دوم از میان ۱۶ ویژگی



ویژگی ۱۴ ام و ۱۶ ام از میان ۱۶ ویژگی

که صرفا برای نشان دادن ویژگی‌های جداکننده‌تر است (و استفاده‌ای از آن نشده است، و تمام ۱۶ ویژگی با هم در طبقه‌بندی استفاده شده اند). این نمودار (نمودار دوم) نشان می‌دهد داده‌ها خطی تفکیک پذیر هستند و پرسپترون حتما پس از تعداد گام محدودی همگرا خواهد شد. بردار ضرایب w اولیه رندوم است.

نکته: تنها تفاوت کدهای چهار بخش، نهوه گام برداری آنهاست که تنها در یک خط از کد تفاوت می‌کند. توابع لازم برای محاسبه گام لازم (مثل تابع مشتق tanh یا تابع مشتق Huber) در ابتدای کد ها بصورت تابع تعریف شده‌اند و باقی کد تفاوتی ندارد.

خروجی کد:

خروجی کد، تعداد داده‌های اشتباه دسته‌بندی شده در داده‌های تست،
به همراه گام‌های طی شده تا رسیدن به خطای آموزش صفر است.

1. پرسپترون ساده:

```
** Perceptron with Simple Sign Function **
Number Of Moves:  312
Test Error: 1 In 458 Test Data = 0.218340611354 %
```

(تعداد گام‌های حرکت بین ۲۶۰ تا ۴۲۰ اتفاق افتاده است)

2. پرسپترون tanh:

```
** Perceptron With Tanh as Sign **
Number Of Moves:  313
Test Error: 1 In 458 Test Data = 0.218340611354 %
```

(تعداد گام‌های حرکت بین ۲۶۰ تا ۴۲۰ اتفاق افتاده است)

3. پرسپترون Huber:

```
** Perceptron With Huber Function as Absolute Value **
Number Of Moves:  214
Test Error: 1 In 458 Test Data = 0.218340611354 %
```

(تعداد گام‌های حرکت بین ۱۹۱ تا ۳۵۰ اتفاق افتاده است)

4. پرسپترون پیشنهادی:

```
** Perceptron With Sqrt(x2 + c2) Function as Absolute Value **
Number Of Moves:  265
Test Error: 1 In 458 Test Data = 0.218340611354 %
```

(تعداد گام‌های حرکت بین ۲۶۵ تا ۴۳۳ اتفاق افتاده است)

نکته: خطای دسته‌بندی روی داده تست برای تمام روش‌ها تنها ۱ عدد است و ماتریس Confusion هر چهار روش بصورت زیر است:

	پیش‌بینی شده: ۰	پیش‌بینی شده: ۲	
۲۲۶ عدد	۱	۲۲۵	برچشب واقعی: ۲
۲۳۲ عدد	۲۳۲	۰	برچسب واقعی: ۰
	۲۳۳ عدد	۲۲۵ عدد	

مقایسه روش‌ها: به دلیل زیاد بودن تعداد feature ها و اینکه این ویژگی‌ها به خوبی این دو عدد را از هم جدا می‌کنند، خطای تمام روش‌ها بسیار کم، و همگی برابر با ۱ نمونه هستند پس متاسفانه امکان مقایسه مناسبی با این معیار وجود ندارد (!!!!!!!).
پس مقایسه را با تعداد گام‌ها انجام می‌دهیم. با انجام تست‌های مختلف، این نتیجه حاصل شد که سرعت همگرایی روش Huber (همه روش ها اپسیلون برابر دارند) از بقیه روش‌ها بهتر است. روش معمولی نیز مزیت خاصی ندارد مگر از نظر راحتی محاسبات و پردازش کامپیوتری، به شکلی که در بقیه روش‌ها یک تابعی محاسبه میگردد اما در پرسپترون ساده خیر.

پایان گزارش