حل تمرین درس یادگیری ماشین

سری ۴

نام و نام خانوادگی دانشجو:

همایون حیدرزاده (۹۵۱۳۱۰۷۰)

نام استاد: دکتر ناظرفرد

آبان ۹۶

سوال ۱

الف)

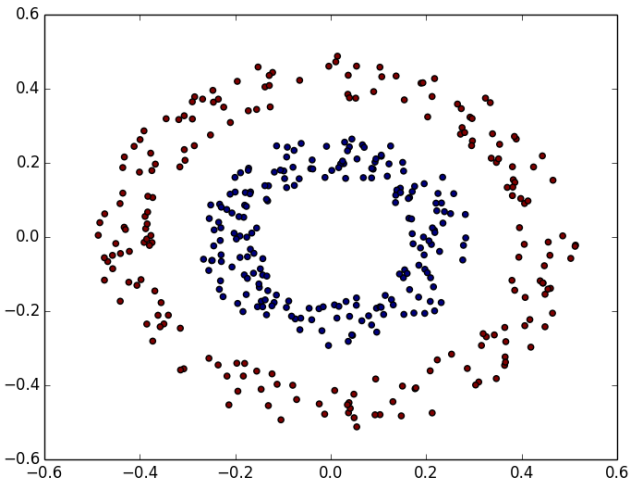
[کد این قسمت: main1.py]

همچنین خروجی‌های گزارش شده در این متن نیز در پوشه outputs موجود می‌باشند.

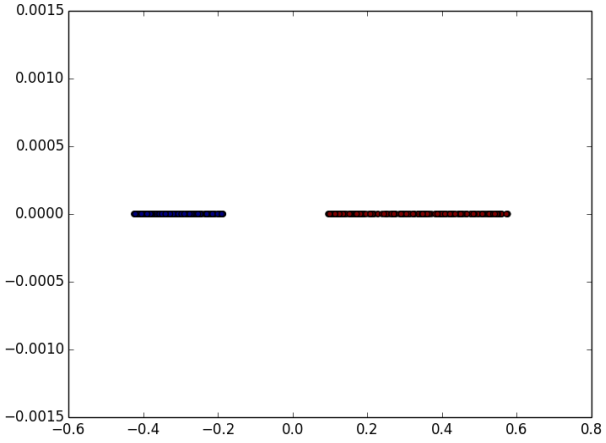
در این قسمت ابتدا تابع نگاشت را بر روی داده اولیه اعمال و سپس داده‌ها را به روش زیر نرمال کردیم:

$$X' = (X - E(X)) / (\max(X) - \min(X))$$

نمودار داده به صورت
زیر است:

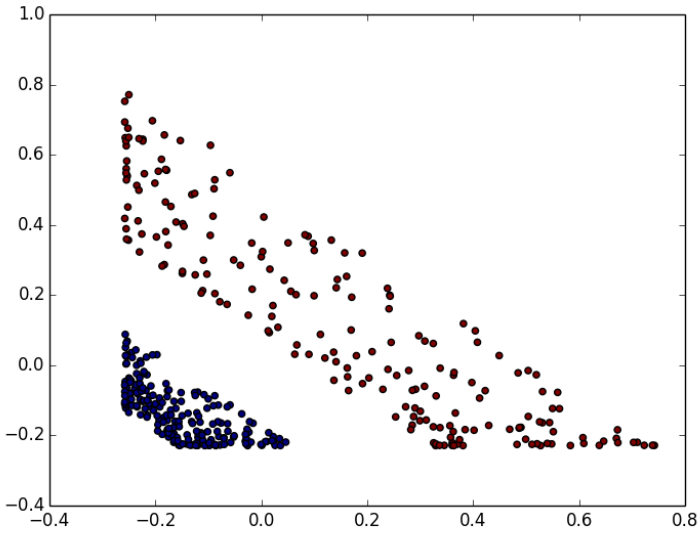


ب) چون توزیع داده‌ها دو دایره نویزی با شعاع‌های متفاوت است پس توسط ویژگی زیر جداپذیر خطی می‌شوند.



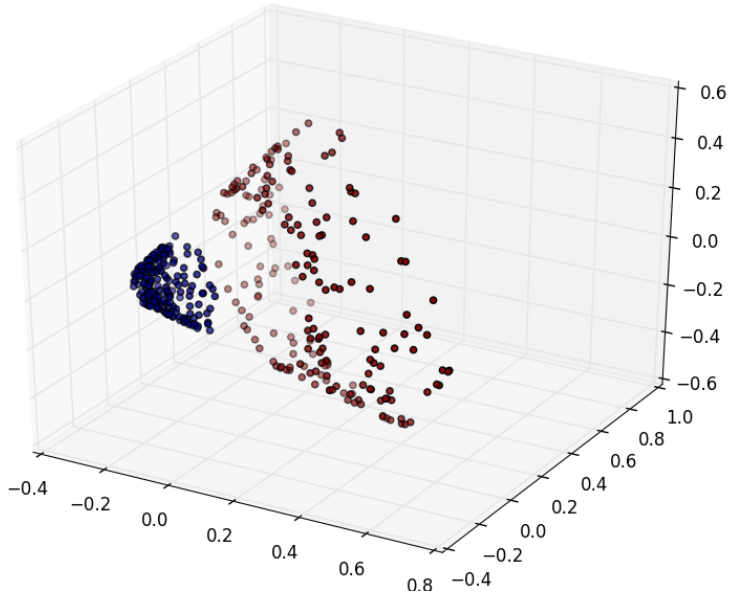
$$Phi1(x1, x2) = (x1^2+x2^2)$$

ج) چون توزیع داده‌ها دو دایره نویزی با شعاع‌های متفاوت است پس توسط دو ویژگی زیر نیز جداپذیر خطی می‌شوند.



$$Phi2(x1, x2) = (x1^2, x2^2)$$

د) همچنین اگر به مجموعه ویژگی‌های بالا ویژگی حاصل‌ضرب را نیز اضافه کنیم، داده‌های جدید نیز جداپذیر خطی هستند:



$$Phi3(x1, x2) = (x1^2, x2^2, x1.x2)$$

ه) ماتریس‌های کرنل برای هر نگاشت در فایل‌هایی با مسیرهای زیر ذخیره شده است:

outputs/main1/Phi1.csv
outputs/main1/Phi2.csv
outputs/main1/Phi3.csv

تابع‌های کرنل نیز به صورت زیر محاسبه شده است:

$$K1(x_i, x_j) = \varphi_1(x_i)^T \varphi_1(x_j) = (x_{i1}^2 + x_{i2}^2)(x_{j1}^2 + x_{j2}^2)$$
$$= (x_{i1}x_{j1})^2 + (x_{i2}x_{j1})^2 + (x_{i1}x_{j2})^2 + (x_{i2}x_{j2})^2$$

$$K2(x_i, x_j) = \varphi_2^T(x_i).\varphi_2(x_j) = [x_{i1}^2 \quad , \quad x_{i2}^2] \begin{bmatrix} x_{j1}^2 \\ x_{j2}^2 \end{bmatrix} = (x_{i1}x_{j1})^2 + (x_{i2}x_{j2})^2$$

$$K3(x_i, x_j) = \varphi_3^T(x_i)\varphi_3(x_j) = [x_{i1}^2 \quad x_{i2}^2 \quad x_{i1}x_{i2}] \begin{bmatrix} x_{j1}^2 \\ x_{j2}^2 \\ x_{j1}x_{j2} \end{bmatrix} =$$

$$(x_{i1}x_{j1})^2 + (x_{i2}x_{j2})^2 + x_{i1}x_{i2}x_{j1}x_{j2}$$

و) نتایج به دست آمده از آموزش SVM برای داده‌های قسمت قبل در جدول ۱ آمده است. ستون دوم مربوط به SVM خطی و ستون سوم مربوط به SVM کرنلی می‌باشد. همچنین تمامی دقت‌ها با استفاده از کراس ولیدیشن

Phi	Acc(Linear)	Acc(Kernel)	Function
PhiIdentity	49.75	49	"Phi(x1, x2 --> x1, x2)"
Phi1	100	100	"Phi(x1, x2 --> x1^2 + x2^2)"
Phi2	100	100	"Phi(x1, x2 --> x1^2, x2^2)"
Phi3	100	100	"Phi(x1, x2 --> x1^2, x2^2, x1.x2)"

محاسبه شده‌اند.

منظور از PhiIdentity تابع نگاشت همانی است. همانطور که مشاهده می‌شود این مجموعه داده‌ها با SVM خطی جداپذیر نیستند و هر خطی که رسم شود در بهترین حالت مانند یک دسته‌بند تصادفی عمل می‌کند.

ز) همان‌طور که از جدول ۱ مشاهده می‌شود ویژگی‌های به دست آمده توسط نگاشت‌ها به صورت خطی جدا شده‌اند و خطای دسته‌بندی ۰ شده است.

ح) با توجه به ستون ۳ دقت‌های به دست آمده مشابه حالت قبل است زیرا تمامی فرمول‌های مربوط به SVM داده‌ها به صورت ضرب داخلی آن‌هل ظاهر می‌شوند. بنابرین نگاشت‌های حالت خطی در هم ضرب شده و همان تابع کرنل که در قسمت ز استفاده شده است را محاسبه می‌کنند.

در عمل SVM خطی قسمت و داده‌ها را ابتدا به یک فضای دیگر برده و آن‌ها را با یک خط جدا می‌کند ولی حالت کرنلی آن بدون رفتن به فضای جدید داده‌ها را در همان فضای اولیه به صورت غیرخطی دسته‌بندی می‌کند.

سوال ۲

برای این قسمت مجموعه داده پارکینسون به صورت ۷۰ درصد آموزشی، ۱۵ درصد ارزیابی و ۱۵ درصد آزمایشی تقسیم شد، کد این قسمت در فایل py.parkinson_main موجود است.

الف)

۱) در این قسمت فضای هایپرپارامترها به صورت یک grid بررسی شد. این فضا برای پارامترها به صورت زیر مشخص می‌شود:

lg(gama): (-15, 3)
lg(c): (-3, 7)
degree: (1, 5)
coef0: (-3, 3)

۲) نتایج حاصل از تمامی آزمایش‌های این بخش در جدول زیر آمده است:

Model	Acc(Test)	Acc(Validati	#SV	parameters
Best model(Coarse grain	100	96.551724	68	"{'c': 64, 'gama': 4}"
Best model(Fine grain)	96.666667	96.551724	81	"{'c': 74.0, 'gama': 5.428571428571428
Random model(1)	80	89.655172	64	"{'c': 32, 'gama': 0.00390625}"
Random model(2)	86.666667	89.655172	57	"{'c': 16, 'gama': 0.03125}"
Random model(3)	76.666667	65.517241	62	"{'c': 2, 'gama': 0.000244140625}"
Random model(4)	76.666667	65.517241	63	"{'c': 0.5, 'gama': 0.03125}"
Random model(5)	76.666667	65.517241	62	"{'c': 4, 'gama': 0.0001220703125}"
Random model(6)	76.666667	65.517241	62	"{'c': 2, 'gama': 3.0517578125e-05}"
Random model(7)	93.333333	86.206897	52	"{'c': 16, 'gama': 1}"
Random model(8)	76.666667	65.517241	62	"{'c': 1, 'gama': 3.0517578125e-05}"
Random model(9)	83.333333	89.655172	60	"{'c': 2, 'gama': 0.25}"
Random model(10)	90	86.206897	56	"{'c': 4, 'gama': 1}"
Random model(11)	76.666667	65.517241	63	"{'c': 64, 'gama': 6.103515625e-05}"
Random model(12)	76.666667	65.517241	62	"{'c': 4, 'gama': 3.0517578125e-05}"
Random model(13)	80	89.655172	67	"{'c': 0.5, 'gama': 0.5}"
Random model(14)	76.666667	65.517241	63	"{'c': 8, 'gama': 0.000244140625}"
Random model(15)	76.666667	65.517241	63	"{'c': 0.5, 'gama': 0.015625}"
Random model(16)	90	86.206897	41	"{'c': 64, 'gama': 0.5}"

Random model(17)	76.666667	65.517241	63	"{'c': 32, 'gama': 6.103515625e-05}"
Random model(18)	76.666667	65.517241	62	"{'c': 64, 'gama': 3.0517578125e-05}"
Random model(19)	76.666667	65.517241	62	"{'c': 1, 'gama': 6.103515625e-05}"
Random model(20)	76.666667	65.517241	63	"{'c': 16, 'gama': 0.0009765625}"

۳) روش هوشمندانه‌تر می‌تواند این باشد که ابتدا فضای کلی را در بازه‌هایی بزرگ‌تر جستجو کنیم (جستجوی عمومی) و سپس با بازه‌های کوچکتر حول جواب بدست آمده جستجوی محلی انجام دهیم و پارامترهای دقیق‌تر به دست آوریم. این روش در کد ارسالی پیاده‌سازی شده است و جواب graine-Fine نیز در جدول بالا آورده شده است. البته **چون در این روش دقت داده ارزیابی بررسی می‌شود ممکن است در نهایت بعد از جستجوی محلی مدلی graine-Fineای که خطای ارزیابی کمتری دارد، خطای آزمایش بیشتری** داشته باشد.

ب) نتایج در جدول زیر آورده شده است.

Model	Acc(Test)	Acc(Valid	#SV	parameters
Best model(Coarse	96.666667	89.655172	38	"{'c': 64, 'coef0': 2, 'degree': 4, 'gama': 4}"
Best model(Fine gra	96.666667	89.655172	40	"{'c': 74.0, 'coef0': 3.0, 'degree': 4.5, 'gama': 9.714
Random model(1)	93.333333	79.310345	39	"{'c': 32, 'coef0': 1, 'degree': 3, 'gama': 4}"
Random model(2)	76.666667	62.068966	30	"{'c': 8, 'coef0': -3, 'degree': 3, 'gama': 1}"
Random model(3)	90	82.758621	48	"{'c': 2, 'coef0': 2, 'degree': 4, 'gama': 0.125}"
Random model(4)	76.666667	65.517241	62	"{'c': 0.25, 'coef0': -3, 'degree': 4, 'gama': 3.051757
Random model(5)	56.666667	51.724138	62	"{'c': 2, 'coef0': -3, 'degree': 4, 'gama': 0.00195312
Random model(6)	76.666667	65.517241	63	"{'c': 8, 'coef0': 2, 'degree': 4, 'gama': 6.103515625
Random model(7)	76.666667	65.517241	63	"{'c': 64, 'coef0': -1, 'degree': 1, 'gama': 0.0001220
Random model(8)	90	75.862069	40	"{'c': 64, 'coef0': 2, 'degree': 4, 'gama': 0.0625}"
Random model(9)	83.333333	89.655172	60	"{'c': 2, 'coef0': 2, 'degree': 1, 'gama': 0.25}"
Random model(10)	90	82.758621	49	"{'c': 32, 'coef0': 1, 'degree': 3, 'gama': 0.0625}"
Random model(11)	76.666667	65.517241	63	"{'c': 0.5, 'coef0': 2, 'degree': 2, 'gama': 0.03125}"
Random model(12)	76.666667	65.517241	63	"{'c': 32, 'coef0': 2, 'degree': 2, 'gama': 3.05175781
Random model(13)	76.666667	65.517241	64	"{'c': 4, 'coef0': 1, 'degree': 3, 'gama': 0.0078125}"
Random model(14)	76.666667	65.517241	62	"{'c': 0.125, 'coef0': 0, 'degree': 1, 'gama': 0.00012
Random model(15)	76.666667	65.517241	62	"{'c': 16, 'coef0': 0, 'degree': 2, 'gama': 0.00024414
Random model(16)	96.666667	79.310345	37	"{'c': 1, 'coef0': 1, 'degree': 3, 'gama': 4}"
Random model(17)	66.666667	55.172414	62	"{'c': 32, 'coef0': -1, 'degree': 4, 'gama': 0.0004882
Random model(18)	80	65.517241	37	"{'c': 8, 'coef0': -2, 'degree': 3, 'gama': 0.125}"

Random model(19)	76.666667	65.517241	62	"{'c': 0.125, 'coef0': -2, 'degree': 1, 'gama': 3.05175
Random model(20)	90	82.758621	49	"{'c': 2, 'coef0': 2, 'degree': 1, 'gama': 2}"

ج) نتایج در جدول زیر آورده شده است.

Model	Acc(Test	Acc(Valida	#SV	parameters
Best model(Coarse gra	86.66666	89.655172	55	"{'c': 64, 'coef0': 0, 'gama': 0.5}"
Best model(Fine grain)	96.66666	89.655172	35	"{'c': 74.0, 'coef0': 1.0, 'gama': 0.5}"
Random model(1)	83.33333	65.517241	33	"{'c': 32, 'coef0': -2, 'gama': 0.125}"
Random model(2)	76.66666	65.517241	63	"{'c': 0.5, 'coef0': 2, 'gama': 0.00390625}"
Random model(3)	76.66666	65.517241	62	"{'c': 1, 'coef0': 0, 'gama': 0.00048828125}"
Random model(4)	90	82.758621	50	"{'c': 32, 'coef0': 1, 'gama': 0.125}"
Random model(5)	73.33333	79.310345	36	"{'c': 0.25, 'coef0': -1, 'gama': 2}"
Random model(6)	83.33333	65.517241	39	"{'c': 4, 'coef0': -3, 'gama': 0.125}"
Random model(7)	76.66666	65.517241	63	"{'c': 0.5, 'coef0': 2, 'gama': 0.001953125}"
Random model(8)	86.66666	62.068966	30	"{'c': 64, 'coef0': -3, 'gama': 0.125}"
Random model(9)	76.66666	65.517241	62	"{'c': 32, 'coef0': 0, 'gama': 3.0517578125e-05}
Random model(10)	76.66666	65.517241	62	"{'c': 2, 'coef0': 0, 'gama': 0.0078125}"
Random model(11)	76.66666	65.517241	63	"{'c': 0.125, 'coef0': -1, 'gama': 0.5}"
Random model(12)	76.66666	65.517241	62	"{'c': 1, 'coef0': 0, 'gama': 0.000244140625}"
Random model(13)	76.66666	65.517241	62	"{'c': 4, 'coef0': -1, 'gama': 6.103515625e-05}"
Random model(14)	83.33333	89.655172	57	"{'c': 64, 'coef0': 2, 'gama': 0.0009765625}"
Random model(15)	76.66666	65.517241	62	"{'c': 0.25, 'coef0': 0, 'gama': 6.103515625e-05
Random model(16)	83.33333	89.655172	57	"{'c': 4, 'coef0': -2, 'gama': 0.015625}"
Random model(17)	76.66666	65.517241	63	"{'c': 16, 'coef0': -2, 'gama': 6.103515625e-05}"
Random model(18)	76.66666	65.517241	62	"{'c': 4, 'coef0': 0, 'gama': 0.0078125}"
Random model(19)	76.66666	65.517241	62	"{'c': 32, 'coef0': 0, 'gama': 0.0001220703125}"
Random model(20)	80	75.862069	64	"{'c': 4, 'coef0': -2, 'gama': 0.00390625}"

د) برای SVM با داده نویزی، تابع هزینه به صورت زیر تعریف می‌شود که در آن زتاها مقادیر خطای داده‌های نویزی و C میزان تاثیر مجموع این خطاها را مشخص می‌کند. اگر مقدار C بسیار بزرگ باشد تاثیر مجموع خطای داده نویزی زیاد شده و تمایل جداساز برای انتخاب ابر صفحه‌ای با حاشیه کوچکتر که بتواند

Find **w** and b such that
$\mathbf{\Phi(w)} = \frac{1}{2}\,\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\,(\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

خطای زتا را کاهش دهد، افزایش می‌یابد. و به طور عکس اگر مقدار C کوچک باشد، مدل به سمتی می‌رود که بزرگ‌ترین حاشیه ممکن را انتخاب کند، حتی اگر تعداد زیادی هم خطای دسته‌بندی پیش بیاید. اگر مقدار C خیلی کوچک باشد، حتی ممکن است که یک داده جداپذیر خطی نیز به درستی دسته‌بندی نشود.

به عبارت دیگر ممکن است C خیلی بزرگ باعث واریانس بالا و C خیلی کوچک باعث بایاس بالا بشود.

نتایج مقادیر بسیار بزرگ و بسیار کوچک C برای کرنل گاوسی در زیر آورده شده است:

Cross Validation Acc(RBF) for C:2^26 ==> 81.0256410256

Cross Validation Acc(RBF) for C:2^-20 ==> 75.3846153846

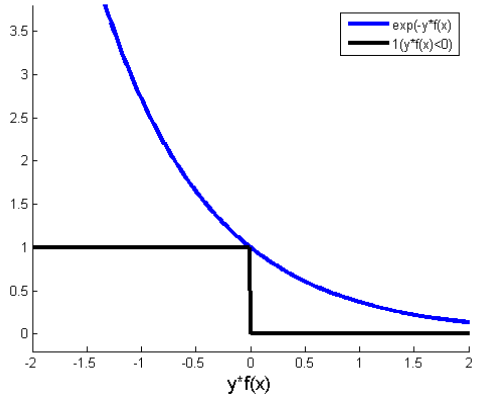
این در حالی است که مدلی بهتر با خطای ۰.۹۶ به دست آمده است. البته می‌توان گفت که پارامتر C رابطه مستقیمی با ویژگی‌های مجموعه داده دارد، برای مثال در یک مجموعه داده جداپذیر خطی از یک حدی به بعد هر چه C را افزایش دهیم دقت همان ۱۰۰ باقی می‌ماند.

---

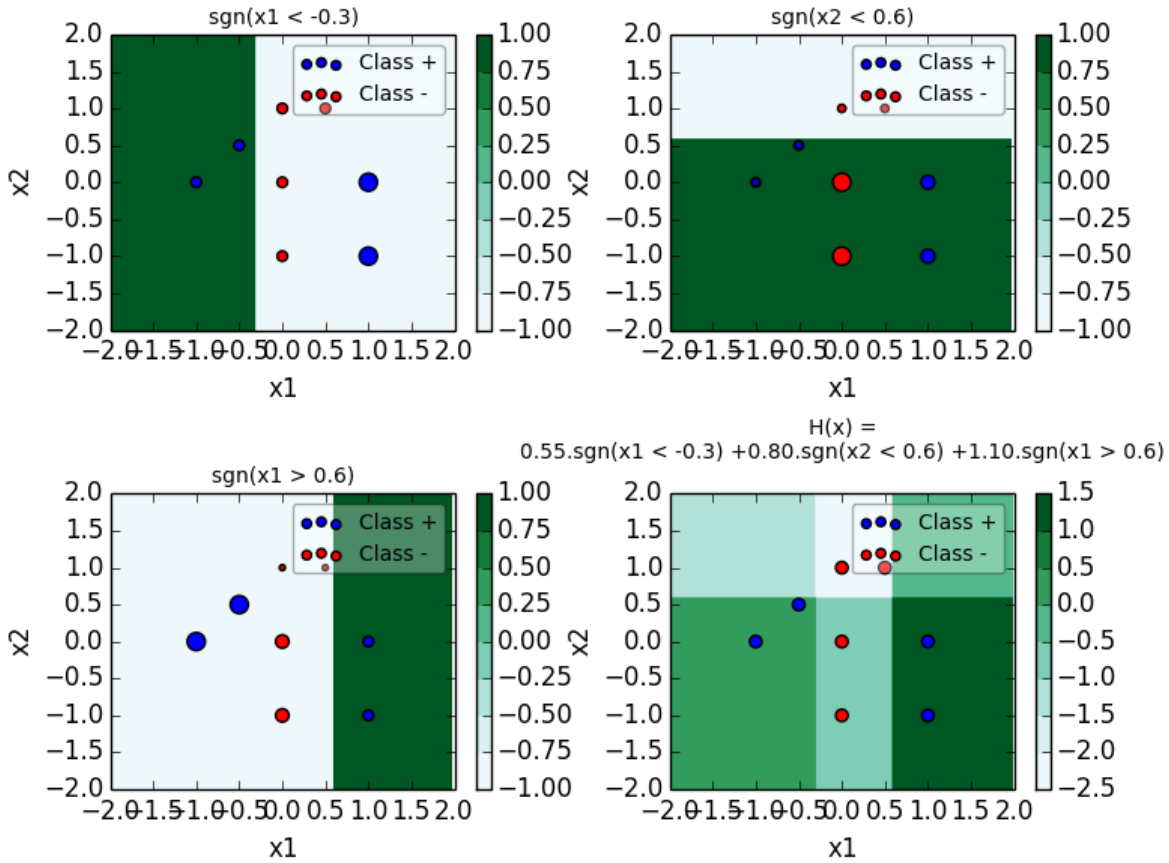
۵) تعداد بردارهای پشتیبان در جدول‌ها آورده شده است.

سوال ۳

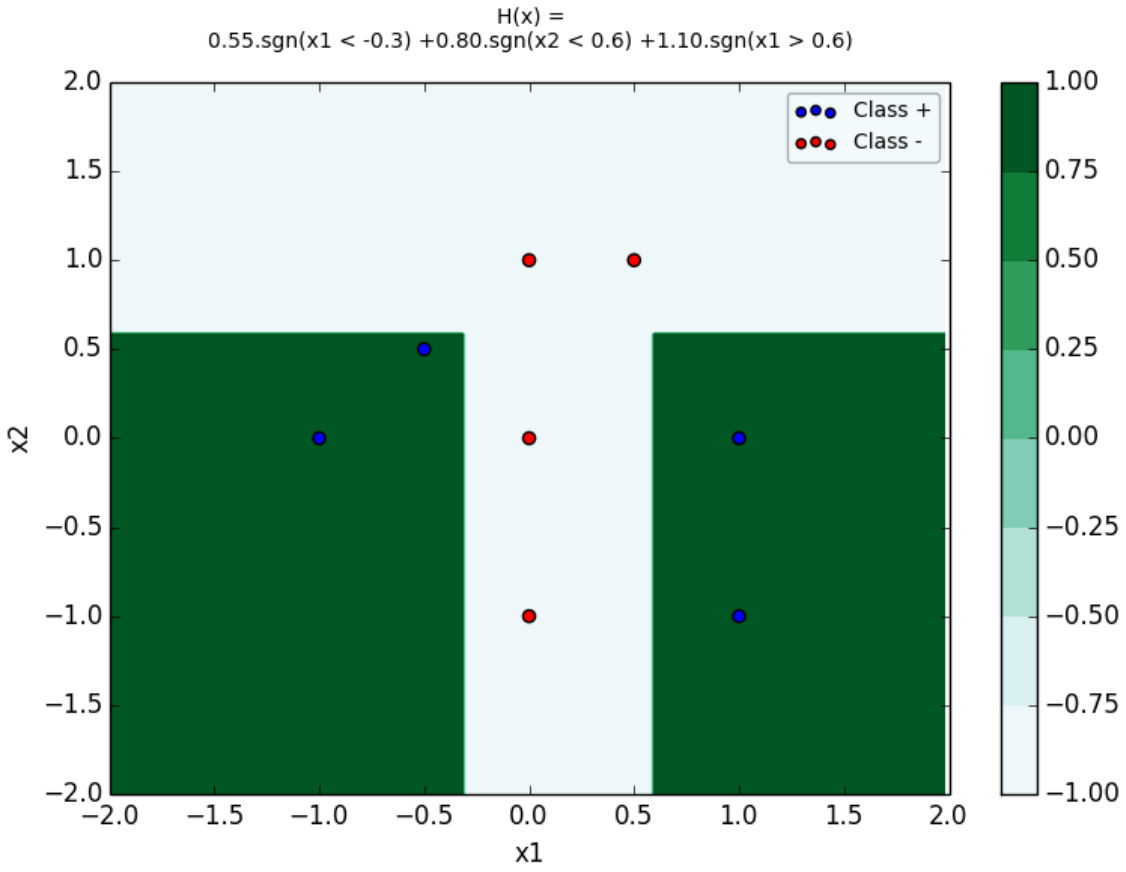
الف) الگوریتم آدابوست نسبت به داده نویزی یا پرت حساس است، زیرا از یک تابع جریمه نمایی استفاده می‌کند. فرض کنید که یک داده مثبت وجود دارد که در میان تعداد خیلی زیادی داده منفی قرار گرفته است. طبق الگوریتم آدابوست این داده هر دفعه به صورت نمایی $\exp(-f(x)*y)$ شامل جریمه خواهد شد و چون هدف مینیمم کردن این جریمه است الگوریتم مدل مناسبی را آموزش نخواهد دید. زیرا داده پرت تاثیر بسیار زیادی در یادگیری نهایی خواهد گذاشت. شکل زیر تفاوت جریمه دسته‌بندی نادرست را برای تابع نمایی و تابع شمارش تعداد خطا مقایسه می‌کند.



ب) کد این قسمت با پایتون بدون استفاده از کتاب‌خانه در فایل py.adaboost_main پیاده‌سازی شد. Desicion stump و مقادیر مرحله‌ای متغیرها در زیر آورده شده است:



مراحل اجرای الگوریتم از چپ به راست است و در مرحله چهارم (ردیف دوم سمت راست) تابع عددی آنسامبل آورده شده است. همچنین در شکل زیر تابع علامت آنسامبل رسم شده است:



نتایج مرحله‌ای نیز در جدول زیر آورده شده است:

t	h	eps	alpha	W								Z	err(H)
1	sgn(x1 < -0.3)	0.25	0.549	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.12	0.866	0.25
2	sgn(x2 < 0.6)	0.167	0.805	0.083	0.083	0.083	0.083	0.250	0.250	0.083	0.08	0.745	0.25
3	sgn(x1 > 0.6)	0.1	1.099	0.050	0.050	0.050	0.050	0.150	0.150	0.250	0.25	0.6	0

با توجه به جدول پس از سه مرحله اجرای الگوریتم آدابوست خطای آموزشی صفر شده است و همانطور که مشاهده می‌شود مدل نهایی به دست آمده احتمالا دارای قابلیت تعمیم خوبی باشد، زیرا مدلی ساده است.