به نام خدا

راضیه مسعودی 9935598

شیرین محبی 9935537

تمرین سوم یادگیری ماشین

پارت اول (bagging):

ابتدا کتابخانه های مورد نیاز را اضافه می کنیم سپس دیتاست ها را لود کرده آن ها را به float تبدیل کرده و 30٪ از داده ها را به عنوان داده ی تست در نظر می گیریم و بقیه را داده ی ترین در نظر می گیریم، همچنین داده های دیتاست های BreastTissue، Diabetes، Glass، Ionosphere، Sonar و Wine را به ترتیب با شماره های از یک تا شش نشان می دهیم (هم داده تست هم ترین).

تابع classifier(): این تابع با استفاده از تابع آماده DecisionTreeClassifier() مدل را می سازد سپس x_train و y_train را بر روی آن فیت می کند سپس با استفاده از x_test لیبل ها را پیشبینی می کند و آن ها را بر می گرداند.

تابع new_data(): این تابع داده ی ترین را می گیرد و با استفاده از random.choice یک دیتاست جدید که ممکن است سمپل های تکراری نیز داشته باشد، تولید می کند و آن را به x_train و y_train تبدیل کرده و با فرا خوانی تابع classifier() لیبل های پیشبینی شده توسط این تابع و داده ی ترین جدید را بر می گرداند.

تابع majority(): این تابع k (تعداد classifier ها)، داده ی ترین و x و y داده ی تست را می گیرد و سپس با فراخوانی تابع new_data() به تعداد k تا classifier می سازد و سپس بین تک تک لیبل های این k تا classifier لیبلی که بیشترین تکرار را داشته به عنوان لیبل آن سمپل بر می گرداند و در لیست majority_list می ریزد و آن را به آرایه ی

majority_array تبدیل می کند و در نهایت با استفاده از y_test و majority_array و متد accuracy_score دقت را بر می گرداند.

تابع ()accuracy: این تابع مقدار دقت را بر حسب تعداد classifier ها که در لیست classifiers قرار دارند پرینت می کند.

تابع ()noisy: این تابع داده (داده ی ترین) را می گیرد و k% تعداد فیچر ها را در count ریخته و از بین فیچر ها به تعداد count به صورت رندوم انتخاب می کند و با جایگزین کردن فیچر های انتخابی با داده هایی که از یک توزیع نرمال به وجود آمده اند داده ی ترین را نویزی می کند و در نهایت ستون لیبلی که در ابتدای تابع از داده ی ترین جدا کرده بودیم را دوباره به آن می چسبانیم و تحت عنوان data_train آن را بر می گردانیم.

تابع ()acc_noisy: در این تابع ابتدا داده ی ترین برای نویزی شدن به تابع ()noisy به همراه درصد نویزی شدن فرستاده می شود سپس داده ی نویزی شده را برای به دست آوردن دقت و پرینت آن به تابع accuracy می فرستیم این تابع تمام این مراحل را برای همه ی دیتاست ها انجام می دهد.

در نهایت مقادیر دقت برای داده ی بدون نویز و نویزی را به ترتیب با فراخوانی تابع ()accuracy به ازای تمام دیتاست ها و acc_noisy به ازای تمام درصد های نویز، نشان می دهیم.

پارت 2 (boosting):

ابتدا کتابخانه های مورد نیاز را اضافه می کنیم سپس دیتاست ها را لود کرده آن ها را به float تبدیل کرده و 30٪ از داده ها را به عنوان داده ی تست در نظر می گیریم و بقیه را داده ی ترین در نظر می گیریم، همچنین داده های دیتاست های Diabetes،Ionosphere و Sonar را به ترتیب با شماره های دو، چهار و پنج نشان می دهیم (هم داده تست هم ترین).

تابع ()classifier: این تابع با استفاده از تابع آماده ()DecisionTreeClassifier مدل را می سازد سپس x_train و y_train را بر روی آن فیت می کند سپس با استفاده ازx_train و x_test لیبل ها را پیشبینی می کند و آن ها به ترتیب در h و pre ریخته و بر می گرداند.

تابع boost(): این تابع تعدادclassifier ها (k) و داده های تست و ترین را می گیرد و الگوریتم adaboost را پیاده سازی می کند بدین صورت که ابتدا وزن همه داده ها را طبق فرمول 1/m که m تعداد داده ها است به دست آورده و در D می ریزد سپس یک ماتریس به ابعاد تعداد کلاسیفایر ها و m می سازد و آن را با D پر می کند؛ با یک حلقه ی for به تعداد k می سازد و تابع classifier را فراخوانی می کند و لیبل های پیشبینی شده توسط داده ی تست و ترین بر می گرداند و به ترتیب در pre و h و مقدار pre را در لیست pred_list می ریزد، مقدار error را برای داده های misclassify شده در مقایسه ی h و y_train به دست آورده و مقدار آلفا را طبق فرمول اسلاید ها محاسبه کرده سپس به کمک آن مقدار وزن ها را آپدیت می کند در نهایت یک ماتریس به نام pred و با ابعاد y_test می سازد و طبق فرمول زیر آن را پر می کند و در نهایت مقدار دقت را بر اساس pred و y_test به دست می آورد و بر می گرداند.

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

تابع accuracy(): این تابع مقدار دقت را بر حسب تعداد classifier ها که در لیست T قرار دارند پرینت می کند.

تابع noisy(): این تابع داده (داده ی ترین) را می گیرد و k% تعداد فیچر ها را در count ریخته و از بین فیچر ها به تعداد count به صورت رندوم انتخاب می کند و با جایگزین کردن فیچر های انتخابی با داده هایی که از یک توزیع نرمال به وجود آمده اند داده ی ترین را نویزی می کند و در نهایت ستون لیبلی که در ابتدای تابع از داده ی ترین جدا کرده بودیم و سمپل های نویزی شده را بر می گرداند.

تابع acc_noisy(): در این تابع ابتدا داده ی ترین برای نویزی شدن به تابع noisy() به همراه درصد نویزی شدن فرستاده می شود سپس داده ی نویزی شده را برای به دست آوردن دقت و پرینت آن به تابع accuracy می فرستیم این تابع تمام این مراحل را برای همه ی دیتاست ها انجام می دهد.

در نهایت مقادیر دقت برای داده ی بدون نویز و نویزی را به ترتیب با فراخوانی تابع accuracy() به ازای تمام دیتاست ها و acc_noisy به ازای تمام درصد های نویز و توسط یک حلقه ی for، نشان می دهیم.

Datasets	Algorithms	
	Bagging	AdaBoost.M1
Wine	98.14%	
Glass	80%	
BreastTissue	68.75%	
Diabetes	77.92%	78.35%
Sonar	87.30%	79.36%
Ionosphere	93.39%	91.50%

Datasets	Algorithms					
	Bagging + noise 10%, 20%, 30%			AdaBoost.M1 + noise 10%, 20%,		
	10%	20%	30%	10%	30%	20%
Wine	100%	94.44%	96.29%			
Glass	83.07%	76.92%	69.23%			
BreastTissue	65.62%	62.5%	62.5%			
Diabetes	77.05%	74.02%	71.86%	76.19%	72.29%	76.62%
Sonar	87.30%	87.30%	85.71%	74.60%	76.19%	80.95%
Ionosphere	95.28%	96.22%	92.45%	89.62%	86.79%	90.56%

Why should we set max_depth parameter in AdaBoost.M1 so that the base classifiers become a little better than random?

به این دلیل max_depth می گذاریم که درخت تصمیم گیری بعد از انتخاب هر فیچر، داده ی split شده را classify کند و این کار باعث می شود که بتواند لیبل ها را با دقتی بهتر از random classifier تشخیص دهد.

What do we mean by stable, unstable, and weak classifier?

اگر کلاسیفایر stable باشد تغییر جزئی در داده های ورودی روی hypothesis اثر نمی گذارد اما اگر unstable باشد تاثیر می گذارد و weak classifier ها نیز کمی بهتر از رندوم عمل می کنند و قابلیت تشخیص لیبل ها را دارند اما دقت آن ها بالا تر از random classifier نیست.

What kind of classifiers should be used in Bagging? How about AdaBoost.M1?

هر classifier ای که از رندوم عمل کند و مستقل از دیگر classifier ها باشد، بهتر است؛ از انواع کلاسیفایر هایی که می توان در bagging استفاده کرد می توان decision tree، SVM، linear regression و ... را نام برد همچنین برای boosting بهتر است از weak classifierها مثل decision tree استفاده کرد.

Compare the results in noiseless and noisy settings and say which algorithm’s performance degrades with noisy features. And why?

Boosting در مقابل نویز robust است به همین دلیل مقدار دقت در داده های نویزی شده و بدون نویز خیلی تفاوتی ندارد اما در bagging این طور نیست.