CheckAUC

May 26, 2023

[1]:
```
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import seaborn as sns
import rpy2
import pandas as pd
import numpy as np
from pathlib import Path
%load_ext rpy2.ipython
```

[2]:
```
Path("./r_output/").mkdir(parents=True, exist_ok=True)
```

[3]:
```
%%R
library('glmnet')
for (i in 1:100){
    set.seed(i)
    n <- 1000
    df <- data.frame("x" = rep(c(0,1), n),
    "y" = rbinom(n, 1, 0.25))
    df$x_cr <- resid(lm(x~y, data=df))


    df$y_hat_x <- predict(glm(y~x, data=df, family='binomial'))
    df$y_hat_x_cr <- predict(glm(y~x_cr, data=df, family='binomial'))


    library(MLmetrics)

    auc_x = MLmetrics::AUC(df$x, df$y)
    auc_x_cr = MLmetrics::AUC(df$x_cr, df$y)

    df_scores = data.frame(auc_x, auc_x_cr )
    write.csv(df_scores, paste("./r_output/scores_",i, ".csv",  sep = ""))
    write.csv(df, paste("./r_output/df_",i, ".csv",  sep = ""))
}
```

```
R[write to console]: Loading required package: Matrix

R[write to console]: Loaded glmnet 4.1-6

R[write to console]:
Attaching package: 'MLmetrics'


R[write to console]: The following object is masked from 'package:base':

    Recall
```

[4]:
```
df_source = pd.concat([pd.read_csv(f"./r_output/df_{i}.csv",index_col=0).
 ↪assign(iteration=i)
                 for i in range(1,101)]
                )


df_scores = pd.concat([pd.read_csv(f"./r_output/scores_{i}.csv",index_col=0).
 ↪assign(iteration=i)
                 for i in range(1,101)]
                )
```

[5]:
```
df_source.head()
```

[5]:

	x	y	x_cr	y_hat_x	y_hat_x_cr	iteration
1	0	0	-0.503989	-1.077391	-1.109308	1
2	1	0	0.496011	-1.141746	-1.109308	1
3	0	0	-0.503989	-1.077391	-1.109308	1
4	1	1	0.512097	-1.141746	-1.109308	1
5	0	0	-0.503989	-1.077391	-1.109308	1

[6]:
```
df_scores.head()
```

[6]:

	auc_x	auc_x_cr	iteration
1	0.491957	0.741909	1
1	0.466708	0.715843	2
1	0.492317	0.742271	3
1	0.497227	0.747221	4
1	0.496092	0.746080	5

[7]:
```
df_train = df_source.drop(columns=["x_cr"])
df_train.head()
```

[7]:

	x	y	y_hat_x	y_hat_x_cr	iteration
1	0	0	-1.077391	-1.109308	1

```
2  1  0 -1.141746   -1.109308          1
3  0  0 -1.077391   -1.109308          1
4  1  1 -1.141746   -1.109308          1
5  0  0 -1.077391   -1.109308          1
```

[8]:
```
def compute_aucs(df_train):
    confound_model = LinearRegression().fit(df_train[["y"]], df_train["x"])
    df_train["x_cr"] = df_train["x"] - confound_model.predict(df_train[["y"]])

    prediction_model = LogisticRegression(penalty="none", solver='newton-cg',␣
 ↪class_weight=None ).fit(df_train[["x_cr"]], df_train["y"] )
    dt = DecisionTreeClassifier().fit(df_train[["x_cr"]], df_train["y"] )

    df_train["y_hat_x"] = prediction_model.predict_proba(df_train[["x"]])[:, 1]
    df_train["y_hat_x_cr"] = prediction_model.
 ↪predict_proba(df_train[["x_cr"]])[:, 1]
    df_train["rf_y_hat_x"] = dt.predict_proba(df_train[["x"]])[:, 1]
    df_train["rf_y_hat_x_cr"] = dt.predict_proba(df_train[["x_cr"]])[:, 1]
    return pd.Series(
        dict(
        auc_x = roc_auc_score(df_train["y"], df_train["x"], ),
        auc_x_cr = roc_auc_score(df_train["y"], df_train["x_cr"], ),
        auc_y_hat_x = roc_auc_score(df_train["y"], df_train["y_hat_x"]),
        auc_y_hat_x_cr = roc_auc_score(df_train["y"], df_train["y_hat_x_cr"]),
        dt_auc_y_hat_x = roc_auc_score(df_train["y"], df_train["rf_y_hat_x"]),
        dt_auc_y_hat_x_cr = roc_auc_score(df_train["y"],␣
 ↪df_train["rf_y_hat_x_cr"])
    ))
```

[9]:
```
df_train.head()
```

[9]:
```
   x  y   y_hat_x  y_hat_x_cr  iteration
1  0  0 -1.077391   -1.109308          1
2  1  0 -1.141746   -1.109308          1
3  0  0 -1.077391   -1.109308          1
4  1  1 -1.141746   -1.109308          1
5  0  0 -1.077391   -1.109308          1
```

[10]:
```
df_scores_both = pd.concat(
    [
        df_scores.assign(program="R"),
        (df_train
         .drop(columns=['y_hat_x', 'y_hat_x_cr'])
         .groupby("iteration").apply(compute_aucs)
         .reset_index()
         .assign(program="Python")
        )
```

```
    ]
)
df_scores_both.head()
```

[10]:

	auc_x	auc_x_cr	iteration	program	auc_y_hat_x	auc_y_hat_x_cr	dt_auc_y_hat_x	dt_auc_y_hat_x_cr
1	0.491957	0.741909	1	R	NaN	NaN	NaN	NaN
1	0.466708	0.715843	2	R	NaN	NaN	NaN	NaN
1	0.492317	0.742271	3	R	NaN	NaN	NaN	NaN
1	0.497227	0.747221	4	R	NaN	NaN	NaN	NaN
1	0.496092	0.746080	5	R	NaN	NaN	NaN	NaN

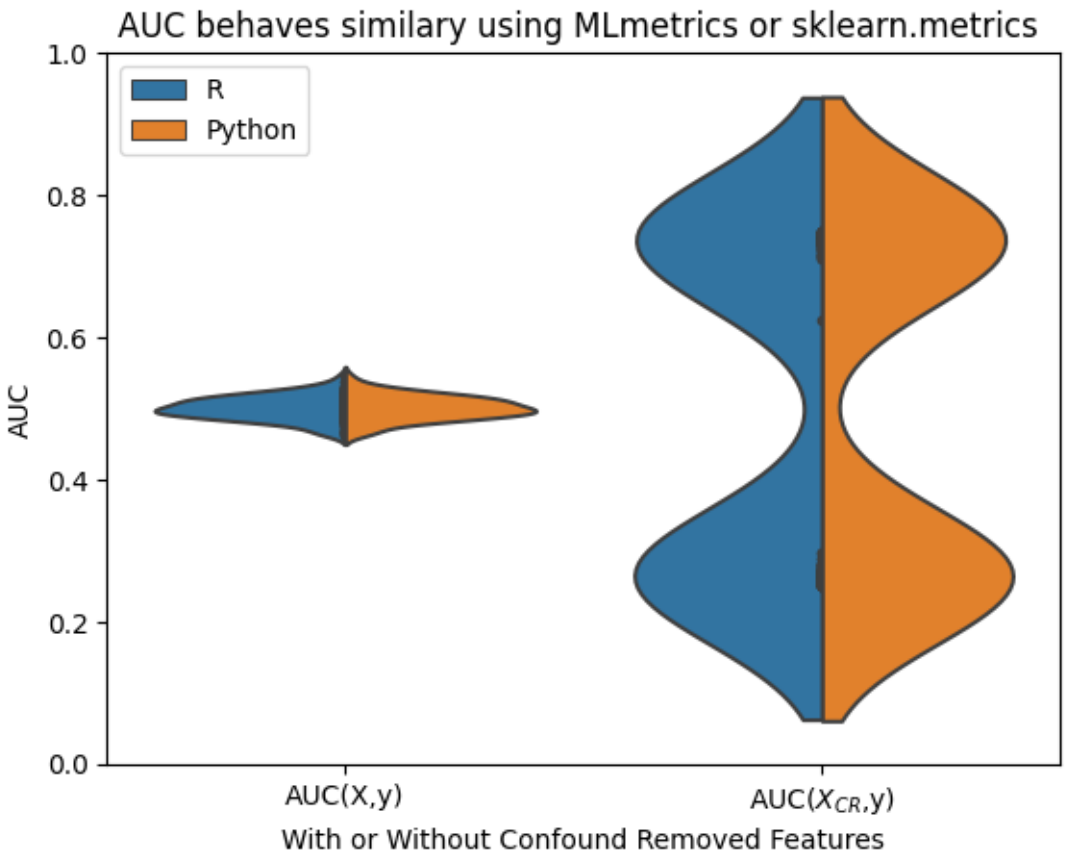
[11]:
```
df_scores_both_long = (df_scores_both
                       .drop(columns=["auc_y_hat_x", "auc_y_hat_x_cr"])
                       .melt(
                           value_vars=["auc_x","auc_x_cr"],
                           var_name="score",
                           id_vars=["program"]
                       )
       )
df_scores_both_long.head()
```

[11]:

	program	score	value
0	R	auc_x	0.491957
1	R	auc_x	0.466708
2	R	auc_x	0.492317
3	R	auc_x	0.497227
4	R	auc_x	0.496092

[12]:
```
ax = sns.violinplot(x="score",y="value", hue="program",
             data=df_scores_both_long,split=True, inner="point")
ax.set_xticklabels(["AUC(X,y)", "AUC($X_{CR}$,y)"])
ax.set_xlabel("With or Without Confound Removed Features")
ax.set_ylabel("AUC")
ax.set_title("AUC behaves similary using MLmetrics or sklearn.metrics ")
ax.legend(loc="upper left")
ax.set_ylim(0,1)
```

[12]: (0.0, 1.0)



[13]:
```
df_scores_python = (df_scores_both
                      .query("program=='Python'")
                      .melt(
                            value_vars=[
                                        "auc_y_hat_x", "auc_y_hat_x_cr",
                                        "dt_auc_y_hat_x", "dt_auc_y_hat_x_cr"
                            ],
                            var_name="score",
                            id_vars=["program"]
                        )

)
```

[14]:
```
df_scores_python.query('score  == "dt_auc_y_hat_x"').mean()
```

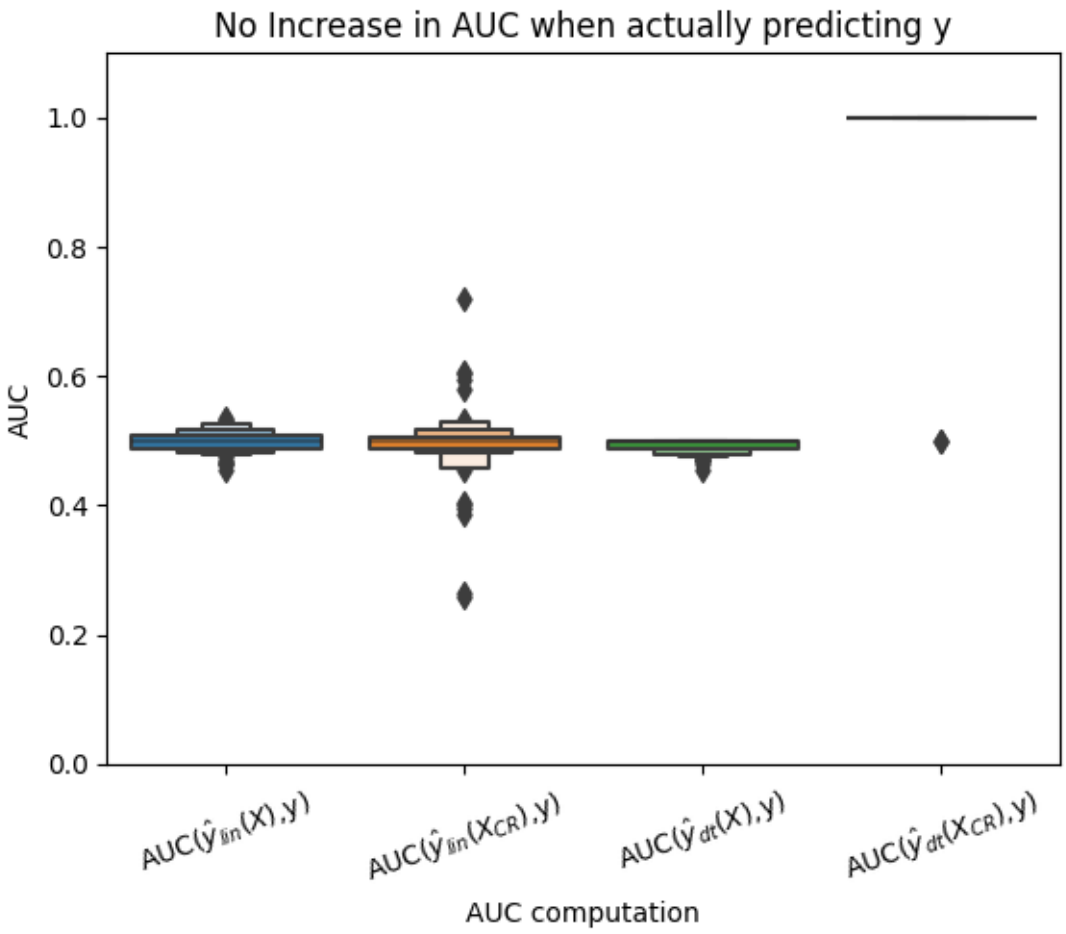
```
/tmp/ipykernel_25138/984044208.py:1: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
```

```
df_scores_python.query('score  == "dt_auc_y_hat_x"').mean()
```

[14]:
```
value    0.492835
dtype: float64
```

[15]:
```
ax = sns.boxenplot(x="score",y="value",
              data=df_scores_python, )



ax.set_xticklabels(["AUC($\hat{y}_{lin}(X)$,y)",␣
 ↪"AUC($\hat{y}_{lin}(X_{CR}$),y)",
                    "AUC($\hat{y}_{dt}(X)$,y)", "AUC($\hat{y}_{dt}(X_{CR}$),y)"



                   ])
ax.set_xlabel("AUC computation")
ax.set_ylabel("AUC")
ax.set_title("No Increase in AUC when actually predicting y")
ax.set_ylim(0,1.1)
ax.set_xticklabels(ax.get_xticklabels(),rotation=20)
```

[15]:
```
[Text(0, 0, 'AUC($\\hat{y}_{lin}(X)$,y)'),
 Text(1, 0, 'AUC($\\hat{y}_{lin}(X_{CR}$),y)'),
 Text(2, 0, 'AUC($\\hat{y}_{dt}(X)$,y)'),
 Text(3, 0, 'AUC($\\hat{y}_{dt}(X_{CR}$),y)')]
```


No Increase in AUC when actually predicting y
AUC
1.0
0.8
0.6
0.4
0.2
0.0
AUC($\hat{y}_{lin}$(X),y)
AUC($\hat{y}_{lin}$(X_{CR}),y)
AUC($\hat{y}_{dt}$(X),y)
AUC($\hat{y}_{dt}$(X_{CR}),y)
AUC computation