Data Science with Python
Trabalho 3

Prof Luiz Barboza
Vivian Medina
RM345379

Data Science with Python

Arv Decisao

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
df = pd.read_csv('/Users/vivianmedina/Downloads/heart.csv')
```

[39] Python

```
df.info()
```

[40] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
```

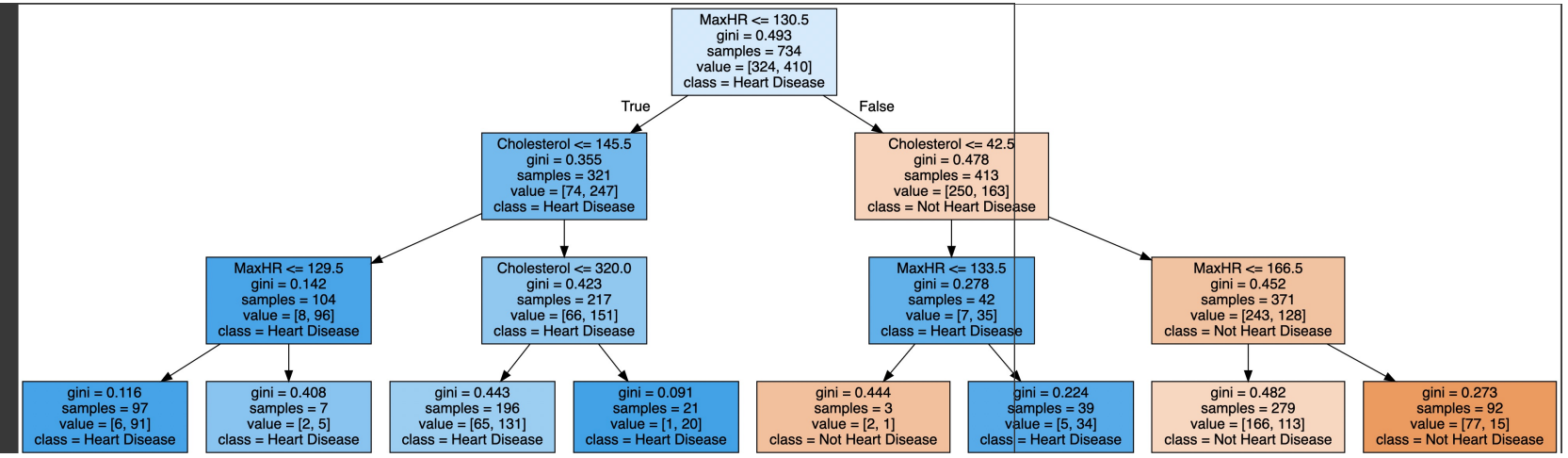
Data Science with Python

```
X_treino, X_teste, y_treino, y_teste = train_test_split(df[['Cholesterol','MaxHR']],df['HeartDisease'],test_size=0.2)
arv = DecisionTreeClassifier(max_depth=3)
arv.fit(X_treino,y_treino)
acuracia_treino = arv.score(X_treino,y_treino)
acuracia_treino*100
```

[43] Python

```
70.84468664850137
```

```
from sklearn.tree import export_graphviz
from graphviz import Source
dot_data = export_graphviz(arv,filled=True,
                           feature_names=['Cholesterol','MaxHR'],
                           class_names=['Not Heart Disease','Heart Disease'])
grafico = Source(dot_data)
grafico
```



Data Science with Python

H2O

```
[45]  !pip install h2o
```

Python

```
zsh:1: command not found: pip
```

```
[46]  import h2o
      h2o.init()
```

Python

```
Checking whether there is an H2O instance running at http://localhost:54321 ..... not found.
Attempting to start a local H2O server...
  Java Version: java version "18.0.1.1" 2022-04-22; Java(TM) SE Runtime Environment (build 18.0.1.1+2-6); Java HotSpot(TM) 64-Bit Server VM (build 18.0.1.1+2-6, mixed mode, sharing)
  Starting server from /Users/vivianmedina/Library/Python/3.8/lib/python/site-packages/h2o/backend/bin/h2o.jar
  Ice root: /var/folders/64/lkwh0p2s2vg5v8d2tdgz5ltm0000gn/T/tmpogbwsh9v
  JVM stdout: /var/folders/64/lkwh0p2s2vg5v8d2tdgz5ltm0000gn/T/tmpogbwsh9v/h2o_vivianmedina_started_from_python.out
  JVM stderr: /var/folders/64/lkwh0p2s2vg5v8d2tdgz5ltm0000gn/T/tmpogbwsh9v/h2o_vivianmedina_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

</>

H2O_cluster_uptime:	02 secs
H2O_cluster_timezone:	America/Sao_Paulo
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.38.0.2
H2O_cluster_version_age:	3 days
H2O_cluster_name:	H2O_from_python_vivianmedina_uzxoxs
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	2 Gb
H2O_cluster_total_cores:	8
H2O_cluster_allowed_cores:	8
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.8.9 final

Data Science with Python

```
[47]
from h2o.estimators import H2ORandomForestEstimator
dados = h2o.import_file('/Users/vivianmedina/Downloads/heart.csv')
dados['HeartDisease'] = dados['HeartDisease'].asfactor()
treino, teste = dados.split_frame(ratios=[0.8])
tree =H2ORandomForestEstimator(max_depth=3)
tree.train(['Cholesterol','MaxHR'],'HeartDisease',treino)
tree.model_performance(teste)
```

```
Parse progress: |████████████████████████████████████████████████| (done) 100%
drf Model Build progress: |███████████████████████████████████████| (done) 100%

ModelMetricsBinomial: drf
** Reported on test data. **

MSE: 0.19636314868455942
RMSE: 0.44312881725809644
LogLoss: 0.5794671349352659
Mean Per-Class Error: 0.335973955507325
AUC: 0.7789473684210526
AUCPR: 0.7804406594946213
Gini: 0.5578947368421052
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.36826948821544647

	0	1	Error	Rate
0	39.0	56.0	0.5895	(56.0/95.0)
1	8.0	89.0	0.0825	(8.0/97.0)
Total	47.0	145.0	0.3333	(64.0/192.0)

Data Science with Python

Gains/Lift Table: Avg response rate: 50.52 %, avg score: 52.13 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_rate	cumulative_capture_rate	
1	0.015625	0.9181818	1.9793814	1.9793814	1.0	0.9181818	1.0	0.9181818	0.0309278	0.0309278	97.93
2	0.0416667	0.9131431	1.5835052	1.7319588	0.8	0.9131431	0.875	0.9150326	0.0412371	0.0721649	58.35
3	0.0416667	0.9114433	0.0	1.7319588	0.0	0.0	0.875	0.9150326	0.0	0.0721649	.
4	0.0572917	0.9104872	1.9793814	1.7994377	1.0	0.9104872	0.9090909	0.9137929	0.0309278	0.1030928	97.93
5	0.109375	0.8827379	1.5835052	1.6966127	0.8	0.9003975	0.8571429	0.9074142	0.0824742	0.1855670	58.35
6	0.1510417	0.7747725	1.7319588	1.7063633	0.875	0.8157230	0.8620690	0.8821201	0.0721649	0.2577320	73.19
7	0.203125	0.6950434	1.9793814	1.7763680	1.0	0.7122042	0.8974359	0.8385519	0.1030928	0.3608247	97.93
8	0.3020833	0.6581611	1.2501356	1.6039815	0.6315789	0.6755103	0.8103448	0.7851417	0.1237113	0.4845361	25.01
9	0.4010417	0.6167525	1.2501356	1.5166689	0.6315789	0.6462315	0.7662338	0.7508651	0.1237113	0.6082474	25.01
10	0.5	0.4311513	1.0417797	1.4226804	0.5263158	0.4662348	0.71875	0.6945320	0.1030928	0.7113402	4.17
11	0.609375	0.3911757	0.7540501	1.3026698	0.3809524	0.4069755	0.6581197	0.6429193	0.0824742	0.7938144	-24.59
12	0.6979167	0.3758763	0.5821710	1.2112633	0.2941176	0.3849061	0.6119403	0.6101863	0.0515464	0.8453608	-41.78
13	0.796875	0.3429126	0.9376017	1.1772792	0.4736842	0.3652152	0.5947712	0.5797651	0.0927835	0.9381443	-6.23
14	0.8958333	0.3049726	0.2083559	1.0702469	0.1052632	0.3231327	0.5406977	0.5514161	0.0206186	0.9587629	-79.16
15	1.0	0.2041349	0.3958763	1.0	0.2	0.2625141	0.5052083	0.5213222	0.0412371	1.0	-60.41

Data Science with Python

```
[48]
from h2o.estimators import H2ORandomForestEstimator
dados = h2o.import_file('/Users/vivianmedina/Downloads/heart.csv')
dados['HeartDisease'] = dados['HeartDisease'].asfactor()
tree =H2ORandomForestEstimator(nfolds = 5)
tree.train(['Cholesterol','MaxHR'],'HeartDisease',dados)
tree.model_performance()
```

```
Parse progress: |████████████████████████████████████████████████| (done) 100%
drf Model Build progress: |███████████████████████████████████████| (done) 100%

ModelMetricsBinomial: drf
** Reported on train data. **

MSE: 0.22223795247927025
RMSE: 0.47142120495292766
LogLoss: 1.189031755409206
Mean Per-Class Error: 0.3014211638179374
AUC: 0.7236052429421932
AUCPR: 0.7119648019803915
Gini: 0.4472104858843864
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.42368743533179876

	0	1	Error	Rate
0	250.0	160.0	0.3902	(160.0/410.0)
1	108.0	400.0	0.2126	(108.0/508.0)
Total	358.0	560.0	0.2919	(268.0/918.0)

Data Science with Python

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.4236874	0.7490637	241.0
max f2	0.0	0.8610169	399.0
max f0point5	0.4563311	0.7293233	231.0
max accuracy	0.4333333	0.7080610	238.0
max precision	0.9061814	0.7724138	53.0
max recall	0.0	1.0	399.0
max specificity	1.0	0.9829268	0.0
max absolute_mcc	0.4333333	0.4050624	238.0
max min_per_class_accuracy	0.5490196	0.6853659	201.0
max mean_per_class_accuracy	0.4333333	0.6992846	238.0
max tns	1.0	403.0	0.0
max fns	1.0	498.0	0.0
max fps	0.0	410.0	399.0
max tps	0.0	508.0	399.0
max tnr	1.0	0.9829268	0.0
max fnr	1.0	0.9803150	0.0
max fpr	0.0	1.0	399.0
max tpr	0.0	1.0	399.0

Data Science with Python

FIAP

Gains/Lift Table: Avg response rate: 55.34 %, avg score: 54.35 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_rate	cumulative_capture_rate	
1	0.0185185	1.0	1.0629921	1.0629921	0.5882353	1.0	0.5882353	1.0	0.0196850	0.0196850	6.29
2	0.0206972	0.9844493	0.9035433	1.0462080	0.5	0.9864597	0.5789474	0.9985747	0.0019685	0.0216535	-9.64
3	0.0305011	0.9674692	1.2047244	1.0971597	0.6666667	0.9755076	0.6071429	0.9911603	0.0118110	0.0334646	20.47
4	0.0403050	0.9597254	1.2047244	1.1233241	0.6666667	0.9627332	0.6216216	0.9842456	0.0118110	0.0452756	20.47
5	0.0501089	0.9512761	1.4055118	1.1785347	0.7777778	0.9558996	0.6521739	0.9786996	0.0137795	0.0590551	40.5
6	0.1013072	0.9285714	1.4994974	1.3407417	0.8297872	0.9380250	0.7419355	0.9581436	0.0767717	0.1358268	49.94
7	0.1503268	0.9091895	1.4858268	1.3880520	0.8222222	0.9182753	0.7681159	0.9451431	0.0728346	0.2086614	48.58
8	0.2004357	0.8809194	1.2963882	1.3651361	0.7173913	0.8961167	0.7554348	0.9328865	0.0649606	0.2736220	29.63
9	0.3006536	0.7961302	1.3749572	1.3684098	0.7608696	0.8440114	0.7572464	0.9032615	0.1377953	0.4114173	37.49
10	0.3997821	0.7053870	1.4099247	1.3787037	0.7802198	0.7544703	0.7629428	0.8663677	0.1397638	0.5511811	40.99
11	0.5	0.5873094	1.0803235	1.3188976	0.5978261	0.6411609	0.7298475	0.8212282	0.1082677	0.6594488	8.03
12	0.6002179	0.4356303	1.1785347	1.2954614	0.6521739	0.5107590	0.7168784	0.7693895	0.1181102	0.7775591	17.85
13	0.6993464	0.3161550	0.6354590	1.2019096	0.3516484	0.3771301	0.6651090	0.7137888	0.0629921	0.8405512	-36.45
14	0.7995643	0.1994444	0.5892674	1.1251207	0.3260870	0.2582080	0.6226158	0.6566861	0.0590551	0.8996063	-41.07
15	0.8997821	0.0970497	0.4910561	1.0544985	0.2717391	0.1432489	0.5835351	0.5994994	0.0492126	0.9488189	-50.89
16	1.0	0.0	0.5106984	1.0	0.2826087	0.0404987	0.5533769	0.5434775	0.0511811	1.0	-48.93

Data Science with Python

```
[49]
from h2o.grid.grid_search import H2OGridSearch
from h2o.estimators import H2ORandomForestEstimator
dados = h2o.import_file('/Users/vivianmedina/Downloads/heart.csv')
dados['HeartDisease'] = dados['HeartDisease'].asfactor()
params = {'max_depth': [2,3,4]}
grid = H2OGridSearch(model=H2ORandomForestEstimator,hyper_params=params)
grid.train(['Cholesterol','MaxHR'],'HeartDisease',dados, nfolds = 5)
print(grid.get_grid(sort_by='accuracy', decreasing=True))
grid.models[0].model_performance()
```

Python

```
Parse progress: |████████████████████████████████████████████████████████| (done) 100%
drf Grid Build progress: |████████████████████████████████████████████████| (done) 100%
Hyper-Parameter Search Summary: ordered by decreasing accuracy
    max_depth    model_ids                                                  accuracy
--  -----------  ---------------------------------------------------------  ----------
    4            Grid_DRF_py_6_sid_ab8f_model_python_1667177360899_615_model_3  0.718954
    2            Grid_DRF_py_6_sid_ab8f_model_python_1667177360899_615_model_1  0.711329
    3            Grid_DRF_py_6_sid_ab8f_model_python_1667177360899_615_model_2  0.711329


ModelMetricsBinomial: drf
** Reported on train data. **

MSE: 0.19548607900451495
RMSE: 0.44213807685440865
LogLoss: 0.5764499785445105
Mean Per-Class Error: 0.3088006529671596
AUC: 0.7621735164202035
AUCPR: 0.7801256813557387
Gini: 0.524347032840407
```

Data Science with Python

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3977022401743099

	0	1	Error	Rate
0	227.0	183.0	0.4463	(183.0/410.0)
1	87.0	421.0	0.1713	(87.0/508.0)
Total	314.0	604.0	0.2941	(270.0/918.0)

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.3977022	0.7571942	271.0
max f2	0.2583520	0.8646228	374.0
max f0point5	0.5088524	0.7504181	209.0
max accuracy	0.4401712	0.7178649	240.0
max precision	0.9398882	1.0	0.0
max recall	0.2192720	1.0	399.0
max specificity	0.9398882	1.0	0.0
max absolute_mcc	0.5088524	0.4311922	209.0
max min_per_class_accuracy	0.4818136	0.7125984	217.0
max mean_per_class_accuracy	0.5088524	0.7167611	209.0
max tns	0.9398882	410.0	0.0
max fns	0.9398882	507.0	0.0
max fps	0.2204453	410.0	398.0
max tps	0.2192720	508.0	399.0
max tnr	0.9398882	1.0	0.0
max fnr	0.9398882	0.9980315	0.0
max fpr	0.2204453	1.0	398.0
max tpr	0.2192720	1.0	399.0

Data Science with Python

FIAP

Gains/Lift Table: Avg response rate: 55.34 %, avg score: 55.26 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_rate	cumulative_capture_rate	
1	0.0108932	0.9186038	1.6263780	1.6263780	0.9	0.9286851	0.9	0.9286851	0.0177165	0.0177165	62.637
2	0.0206972	0.9055955	1.6062992	1.6168670	0.8888889	0.9127069	0.8947368	0.9211165	0.0157480	0.0334646	60.629
3	0.0305011	0.8956800	1.6062992	1.6134702	0.8888889	0.9006127	0.8928571	0.9145260	0.0157480	0.0492126	60.629
4	0.0403050	0.8911113	1.2047244	1.5140455	0.6666667	0.8931762	0.8378378	0.9093328	0.0118110	0.0610236	20.472
5	0.0501089	0.8885251	1.4055118	1.4928107	0.7777778	0.8895148	0.8260870	0.9054553	0.0137795	0.0748031	40.55
6	0.1002179	0.8727628	1.6499486	1.5713797	0.9130435	0.8818393	0.8695652	0.8936473	0.0826772	0.1574803	64.994
7	0.1503268	0.7976208	1.5320952	1.5582848	0.8478261	0.8426841	0.8623188	0.8766596	0.0767717	0.2342520	53.205
8	0.2004357	0.7339284	1.5713797	1.5615585	0.8695652	0.7674511	0.8641304	0.8493575	0.0787402	0.3129921	57.137
9	0.3006536	0.6987018	1.2374615	1.4535262	0.6847826	0.7138135	0.8043478	0.8041761	0.1240157	0.4370079	23.746
10	0.3997821	0.6725448	1.2113438	1.3934755	0.6703297	0.6861606	0.7711172	0.7749134	0.1200787	0.5570866	21.134
11	0.5	0.5889592	1.3356727	1.3818898	0.7391304	0.6418507	0.7647059	0.7482429	0.1338583	0.6909449	33.56
12	0.6002179	0.4268770	0.9035433	1.3020207	0.5	0.4639619	0.7205082	0.7007767	0.0905512	0.7814961	-9.645
13	0.6993464	0.3850469	0.7148914	1.2187983	0.3956044	0.4023017	0.6744548	0.6584695	0.0708661	0.8523622	-28.510
14	0.7995643	0.3317928	0.6481941	1.1472784	0.3586957	0.3617887	0.6348774	0.6212834	0.0649606	0.9173228	-35.180
15	0.8997821	0.2803409	0.4714139	1.0720005	0.2608696	0.3047323	0.5932203	0.5860259	0.0472441	0.9645669	-52.85
16	1.0	0.2192720	0.3535604	1.0	0.1956522	0.2526272	0.5533769	0.5526134	0.0354331	1.0	-64.643

Data Science with Python

```
from h2o.automl import H2OAutoML
dados = h2o.import_file('/Users/vivianmedina/Downloads/heart.csv')
dados['HeartDisease'] = dados['HeartDisease'].asfactor()
aml = H2OAutoML(max_models=6, nfolds = 5)
aml.train(['Cholesterol','MaxHR'],'HeartDisease',dados)
print(aml.leaderboard.head(6))
aml.get_best_model().model_performance()
```

Python

```
Parse progress: |████████████████████████████████████████████████████████| (done) 100%
AutoML progress: |
21:54:50.884: AutoML: XGBoost is not available; skipping it.

████████████████████████████████████████████████████████| (done) 100%
```

model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse
GBM_1_AutoML_1_20221030_215450	0.766238	0.57515	0.796339	0.309228	0.442409	0.195725
StackedEnsemble_AllModels_1_AutoML_1_20221030_215450	0.762661	0.575535	0.793675	0.345319	0.442868	0.196132
StackedEnsemble_BestOfFamily_1_AutoML_1_20221030_215450	0.761559	0.57727	0.792171	0.300307	0.443531	0.19672
GLM_1_AutoML_1_20221030_215450	0.746757	0.589309	0.777474	0.366992	0.449657	0.202191
GBM_2_AutoML_1_20221030_215450	0.746274	0.590518	0.778303	0.338175	0.450082	0.202573
GBM_4_AutoML_1_20221030_215450	0.744126	0.593243	0.774708	0.341319	0.451245	0.203622

[6 rows x 7 columns]

```
ModelMetricsBinomial: gbm
** Reported on train data. **

MSE: 0.18565835495907218
RMSE: 0.4308809057722008
LogLoss: 0.5522064357272938
Mean Per-Class Error: 0.2666746687151911
AUC: 0.7955372575379297
AUCPR: 0.8217424272952698
Gini: 0.5910745150758594
```

Data Science with Python

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.4766043290594123

	0	1	Error	Rate
0	268.0	142.0	0.3463	(142.0/410.0)
1	95.0	413.0	0.187	(95.0/508.0)
Total	363.0	555.0	0.2582	(237.0/918.0)

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.4766043	0.7770461	231.0
max f2	0.2747529	0.8698702	348.0
max f0point5	0.5803787	0.7667877	164.0
max accuracy	0.4944818	0.7461874	222.0
max precision	0.8936821	0.9636364	3.0
max recall	0.2002780	1.0	382.0
max specificity	0.9095876	0.9951220	0.0
max absolute_mcc	0.4944818	0.4841092	222.0
max min_per_class_accuracy	0.5406124	0.7243902	195.0
max mean_per_class_accuracy	0.4944818	0.7400855	222.0
max tns	0.9095876	408.0	0.0
max fns	0.9095876	465.0	0.0
max fps	0.2002780	410.0	382.0
max tps	0.2002780	508.0	382.0
max tnr	0.9095876	0.9951220	0.0
max fnr	0.9095876	0.9153543	0.0
max fpr	0.2002780	1.0	382.0
max tpr	0.2002780	1.0	382.0

Data Science with Python

FIAP

Gains/Lift Table: Avg response rate: 55.34 %, avg score: 55.47 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_rate	cumulative_capture_rate	
1	0.0490196	0.9095876	1.7267717	1.7267717	0.9555556	0.9095876	0.9555556	0.9095876	0.0846457	0.0846457	72.67
2	0.0501089	0.8943034	1.8070866	1.7285176	1.0	0.8946280	0.9565217	0.9092624	0.0019685	0.0866142	80.70
3	0.1132898	0.8776190	1.6201466	1.6680800	0.8965517	0.8822648	0.9230769	0.8942060	0.1023622	0.1889764	62.01
4	0.1525054	0.7971479	1.5561024	1.6392857	0.8611111	0.8351779	0.9071429	0.8790274	0.0610236	0.25	55.61
5	0.2015251	0.7449213	1.4456693	1.5921898	0.8	0.7691079	0.8810811	0.8522902	0.0708661	0.3208661	44.56
6	0.3006536	0.6806553	1.3702085	1.5190003	0.7582418	0.7117277	0.8405797	0.8059453	0.1358268	0.4566929	37.02
7	0.3997821	0.6209147	1.2709181	1.4574868	0.7032967	0.6516574	0.8065395	0.7676886	0.1259843	0.5826772	27.09
8	0.5043573	0.5548494	1.2047244	1.4050781	0.6666667	0.5876235	0.7775378	0.7303533	0.1259843	0.7086614	20.47
9	0.6013072	0.4856475	1.0355215	1.3454938	0.5730337	0.5168749	0.7445652	0.6959338	0.1003937	0.8090551	3.55
10	0.6993464	0.4106086	0.5622047	1.2356870	0.3111111	0.4477794	0.6838006	0.6611458	0.0551181	0.8641732	-43.77
11	0.7995643	0.3529879	0.6089096	1.1571263	0.3369565	0.3838962	0.6403270	0.6263951	0.0610236	0.9251969	-39.10
12	0.9008715	0.2691754	0.5052070	1.0838149	0.2795699	0.3056271	0.5997582	0.5903233	0.0511811	0.9763780	-49.47
13	1.0	0.2002780	0.2382971	1.0	0.1318681	0.2305229	0.5533769	0.5546568	0.0236220	1.0	-76.17

+ Code + Markdown

- three pre-specified XGBoost GBM (Gradient Boosting Machine) models
- a fixed grid of GLMs
- a default Random Forest (DRF)
- five pre-specified H2O GBMs
- a near-default Deep Neural Net
- an Extremely Randomized Forest (XRT)
- a random grid of XGBoost GBMs
- a random grid of H2O GBMs
- a random grid of Deep Neural Nets

[51]

```
lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

Python

model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse
GBM_1_AutoML_1_20221030_215450	0.766238	0.57515	0.796339	0.309228	0.442409	0.195725
StackedEnsemble_AllModels_1_AutoML_1_20221030_215450	0.762661	0.575535	0.793675	0.345319	0.442868	0.196132
StackedEnsemble_BestOfFamily_1_AutoML_1_20221030_215450	0.761559	0.57727	0.792171	0.300307	0.443531	0.19672
GLM_1_AutoML_1_20221030_215450	0.746757	0.589309	0.777474	0.366992	0.449657	0.202191
GBM_2_AutoML_1_20221030_215450	0.746274	0.590518	0.778303	0.338175	0.450082	0.202573
GBM_4_AutoML_1_20221030_215450	0.744126	0.593243	0.774708	0.341319	0.451245	0.203622
GBM_3_AutoML_1_20221030_215450	0.743269	0.592688	0.777482	0.372532	0.451317	0.203687
DRF_1_AutoML_1_20221030_215450	0.737608	0.882322	0.740989	0.343994	0.46304	0.214406

[8 rows x 7 columns]

Data Science with Python

A árvore de decisão apresentou uma acurácia de 70,84% utilizando as variáveis Colesterol e MaxHR para prever uma doença cardíaca. Ao rodar o Modelo de Auto ML com as mesmas variáveis, vemos no quadro acima que a acurácia melhora um pouco, mas não fica muito distante.