NHANES - PAD

Jupyter Example with Python

This example illustrates fitting and comparing Machine Learning algorithms

National Health and Nutrition Examination Survey (NHANES) - Peripheral Artery Disease (PAD)

Data collection:

Household screener, interview, and physical examination

Objectives:

Understand the survey data and create a predictive model to identify the main factors that are related to the disease. The model can also be useful to prioritize the physical exams and to support the diagnostics.

Activities:

- Start the session
- Prepare the data for Modelling
- Data Partition (Training and Validation)
- Feature Engineering (add additional features)
- Modelling
- Scoring

In [1]:

```
## Start the session and prepare the environment
import swat
import os
import pandas as pd
import matplotlib.pyplot as plt
from IPython.core.display import display, HTML
from swat.render import render_html #to visualize model results
%matplotlib inline

# Connect to CAS (this depends on the environment configuration)
conn = swat.CAS(
    hostname = ["link_to_your_host"],
    port = XXX,username="user",password="password")

# CAS Server connection details
#out = conn.serverstatus()
#print(out)
```

In [2]:

```
### Import action sets
conn.loadactionset(actionset="dataStep")
conn.loadactionset(actionset="dataPreprocess")
conn.loadactionset(actionset="cardinality")
conn.loadactionset(actionset="sampling")
conn.loadactionset(actionset="decisionTree")
conn.loadactionset(actionset="astore")
conn.loadactionset(actionset="percentile")
NOTE: Added action set 'dataStep'.
NOTE: Added action set 'dataPreprocess'.
NOTE: Added action set 'cardinality'.
NOTE: Added action set 'sampling'.
NOTE: Added action set 'decisionTree'.
NOTE: Added action set 'astore'.
NOTE: Added action set 'percentile'.
```

Out[2]:

§ actionset
percentile
elapsed 0.00906s · user 0.00216s · sys 0.00995s · mem 0.83MB

In [3]:

```
## Prepare the data for Modelling
conn.upload(path_to_data_folder/nhanes_nof.sas7bdat')
NOTE: Cloud Analytic Services made the uploaded file available as table NHANES_NOF in caslib CASUSER(sasadm).
NOTE: The table NHANES_NOF has been created in caslib CASUSER(sasadm) from binary data uploaded to Cloud Analytic Services.
```

Out[3]:

§ caslib
CASUSER(sasadm)

---

§ tableName
NHANES_NOF

---

§ casTable
CASTable('NHANES_NOF', caslib='CASUSER(sasadm)')
elapsed 0.432s · user 0.0764s · sys 0.158s · mem 34.4MB

In [4]:

```
## Assign variable to the table and check the columns
```

```
abt=conn.CASTable(name='NHANES_NOF', caslib='CASUSER(sasadm)')
#conn.columnInfo(table=abt)

## Create the target variable
conn.runCode(code="""
    data CASUSER.NHANES_PAD1 promote;
    set CASUSER.NHANES_NOF;
            if LEXRABPI = . then LEXRABPI = LEXLABPI;
            if ((LEXLABPI < 0.9) OR (LEXRABPI< 0.9 )) then PAD_Target = 1;
                else PAD_Target = 0;
run;
""")
```

§ InputCasTables

	casLib	Name	Rows	Columns	casTable
0	CASUSER(sasadm)	NHANES_NOF	6929	47	CASTable('NHANES_NOF', caslib='CASUSER(sasadm)')

---

§ OutputCasTables

	casLib	Name	Rows	Columns	Append	Promoted	casTable
0	CASUSER(sasadm)	NHANES_PAD1	6929	48	NaN	N	CASTable('NHANES_PAD1', caslib='CASUSER(sasadm)')
1	CASUSER(sasadm)	promote	6929	48	NaN	N	CASTable('promote', caslib='CASUSER(sasadm)')

elapsed 0.291s · user 0.0913s · sys 0.167s · mem 28MB

```
## Reassign variable to the table and check the columns
abt=conn.CASTable(name='NHANES_PAD1', caslib='CASUSER')
#conn.columnInfo(table=abt)

## Data Partition (Training and Validation)
conn.sampling.srs(
     table = abt,
     samppct = 30,
     partind = True,
     output = dict(casout = dict(name = 'abt_part', replace = True), copyVars = 'ALL')
)
conn.CASTable('abt_part').freq(inputs='PAD_Target')
NOTE: Simple Random Sampling is in effect.
NOTE: Using SEED=1414455854 for sampling.
```

§ Frequency

Frequency for ABT_PART

	Column	NumVar	FmtVar	Level	Frequency
0	PAD_Target	0.0	0	1	6470.0
1	PAD_Target	1.0	1	2	459.0

elapsed 0.0756s · user 0.0377s · sys 0.0202s · mem 39.6MB

```
## Feature Engineering (add additional features)
conn.runCode(code="""
    data CASUSER.NHANES_PAD1(replace=yes);
    set CASUSER.abt_part;
            PulsePreassure = BPXSAR - BPXDAR;
            TC_HDL = LBXTC / LBDHDL;
            IF ((DIQ010 In ('Yes','Borderline')) OR (DIQ050 In ('Yes')) OR (LBXGH > 6.5))
                   then Diabetes = 1;
                   else Diabetes = 0;
            IF ( BPXSAR >= 140 OR BPXDAR >= 90 )
                   then Hypertension = 1;
                   else Hypertension = 0;
    run;
""")
abt_pf=conn.CASTable(name='NHANES_PAD1', caslib='CASUSER')
#conn.columnInfo(table=abt_pf)
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      48 at 0:107   99 at 0:144
      41 at 0:107    103 at 0:144
      63 at 0:107    111 at 0:144
NOTE: Duplicate messages output by DATA step:
NOTE: Missing values were generated as a result of performing an operation on missing values.  (occurre
      Each place is given by: (Number of times) at (Line):(Column).  (occurred 3 times)

## Modelling
```

```
gb = conn.decisionTree.gbtreeTrain(
    table={"name":abt_pf, "where":"strip(put(_PartInd_, best.))='0'"},
    target=target,
    inputs=class_inputs + interval_inputs,
    nominals=class_vars,
    nTree=150,  m=7,  lasso=0.777,  learningrate=1,  subsamplerate=0.883,  ridge=6.03,  seed=1634211770
    leafsize=5,  maxbranch=2,  binorder=True,  encodename=True,  mergebin=True,  nBins=20,  maxLevel=6,
    varImp=True,  missing="USEINSEARCH",
    casOut={"name":"gb_model", "replace":True}
)

# Output model statistics
render_html(gb)
```

Decision Tree for NHANES_PAD1			
Analysis Variable	**Importance**	**Std**	**Count**
INDHHINC	1.0269520699	2.1867208776	505
RIDAGEMN_Recode	0.8730783408	6.4782277548	304
PulsePreassure	0.5064843554	1.1654919978	399
BMXBMI	0.4975143589	0.8839628693	414
TC_HDL	0.4851795726	1.0263028907	430
LBXGH	0.463693539	0.8823927015	376
SMQ040	0.3637655607	2.1579940124	149
DMDEDUC2	0.3379145309	0.7056114026	260
RIDRETH1	0.2985500822	1.0016570844	192
DIQ150	0.1674961201	0.8626071012	125
DIQ110	0.1604200599	0.3233105761	180
RIAGENDR	0.0633082012	0.1814010498	48
ALQ100	0.0557084372	0.1613893225	71
Diabetes	0.0469505021	0.1756860513	39
Hypertension	0.0318877837	0.1678827969	26

Gradient Boosting Tree for NHANES_PAD1	
Descr	**Value**
Number of Trees	150
Distribution	2
Learning Rate	1
Subsampling Rate	0.883
Number of Selected Variables (M)	7
Number of Bins	20
Number of Variables	15
Max Number of Tree Nodes	63
Min Number of Tree Nodes	21
Max Number of Branches	2
Min Number of Branches	2
Max Number of Levels	6
Min Number of Levels	6
Max Number of Leaves	32
Min Number of Leaves	11

Gradient Boosting Tree for NHANES_PAD1	
Descr	**Value**
Actual Number of Trees	150
Average number of Leaves	24.453333333

Output CAS Tables				
CAS Library	**Name**	**Number of Rows**	**Number of Columns**	**Table**
CASUSER(sasadm)	gb_model	7186	47	CASTable('gb_model', caslib='CASUSER(sasadm)')

```
## Scoring
conn.decisionTree.gbtreeScore(
     table={"name":abt_pf},
     modelTable={"name":"gb_model"},
     casOut={"name":'scored_gb', "replace":True},
     copyVars={"PAD_Target", "_PartInd_"},
     encodename = True,
     assessonerow = True
)
```

§ EncodedName

	LEVNAME	LEVINDEX	VARNAME
0	1	0	P_PAD_Target1
1	0	1	P_PAD_Target0

§ EncodedTargetName

	LEVNAME	LEVINDEX	VARNAME
0		0	I_PAD_Target

§ ErrorMetricInfo

	TreeID	Trees	NLeaves	MCR	LogLoss	ASE	RASE	MAXAE
0	0.0	1.0	29.0	0.066243	0.222444	0.061337	0.247663	0.969904
1	1.0	2.0	56.0	0.068985	0.205645	0.056629	0.237968	0.985768
2	2.0	3.0	85.0	0.066099	0.197244	0.054045	0.232475	0.992778
3	3.0	4.0	115.0	0.065233	0.191524	0.052455	0.229031	0.996211
4	4.0	5.0	139.0	0.062491	0.187500	0.051015	0.225866	0.997660
...	...	...	...	...	...	...	...	...
145	145.0	146.0	3594.0	0.027998	0.125489	0.023797	0.154264	0.999995
146	146.0	147.0	3619.0	0.027998	0.125530	0.023809	0.154301	0.999994
147	147.0	148.0	3636.0	0.027998	0.125626	0.023815	0.154321	0.999994
148	148.0	149.0	3652.0	0.027854	0.125299	0.023784	0.154221	0.999993
149	149.0	150.0	3668.0	0.027710	0.125312	0.023723	0.154022	0.999994

150 rows × 8 columns

§ OutputCasTables

	casLib	Name	Rows	Columns	casTable
0	CASUSER(sasadm)	scored_gb	6929	6	CASTable('scored_gb', caslib='CASUSER(sasadm)')

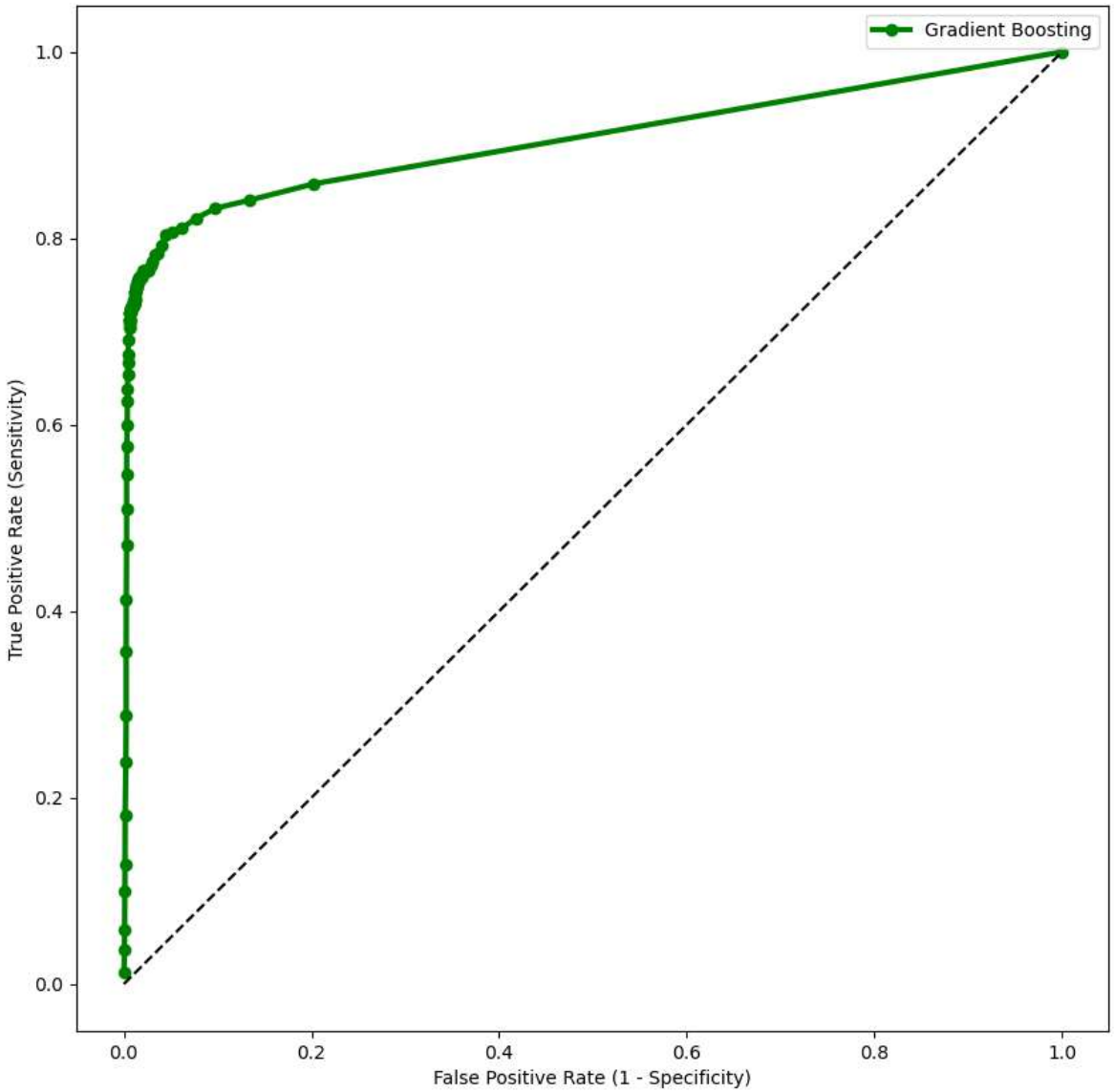
§ ScoreInfo

	Descr	Value
0	Number of Observations Read	6929
1	Number of Observations Used	6929
2	Misclassification Error (%)	2.7709626209

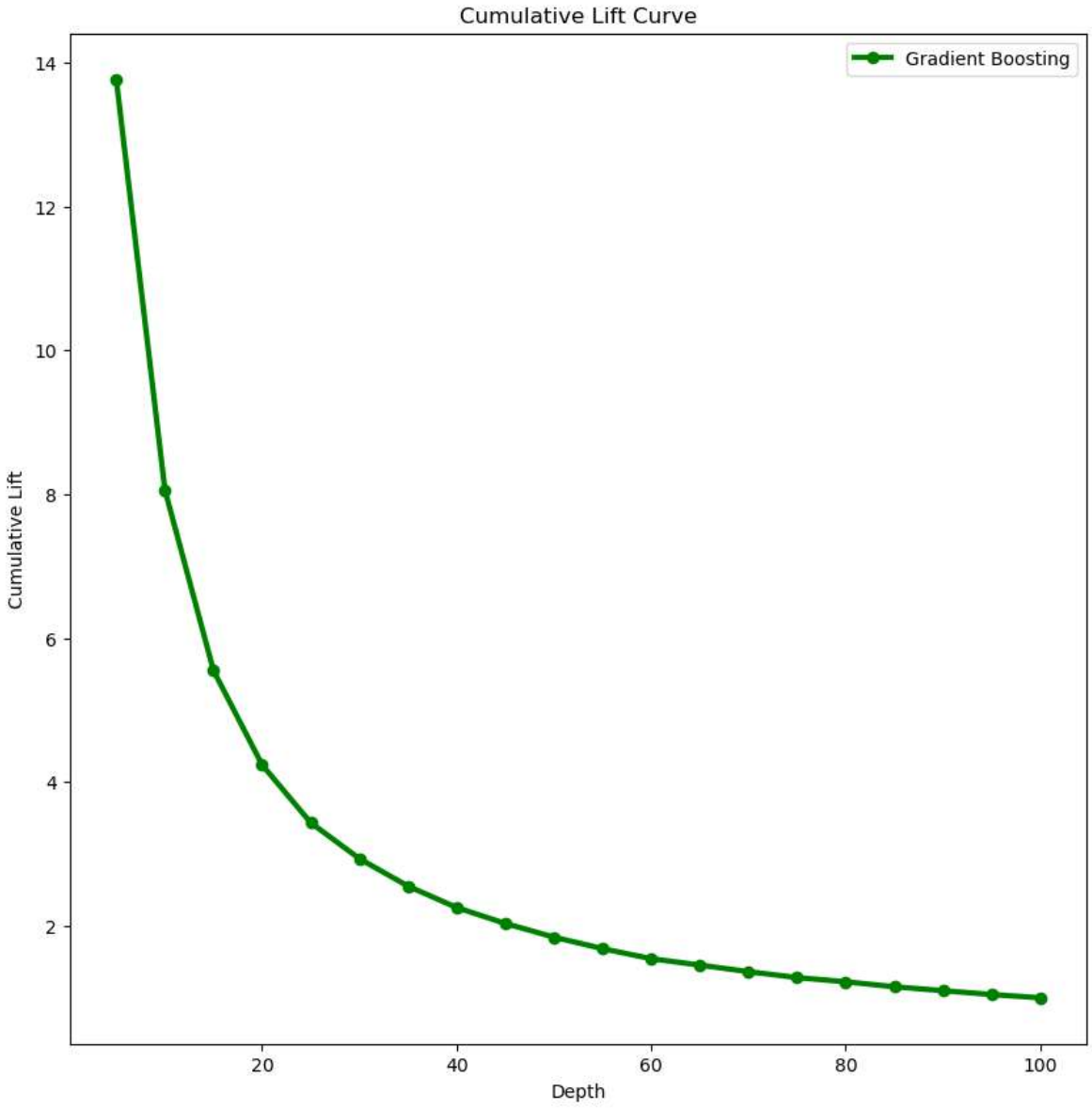
elapsed 1.05s · user 0.863s · sys 0.0932s · mem 78.3MB

```
assessed = conn.percentile.assess(
     table = 'scored_gb',
     inputs = 'P_PAD_Target1',
     casout = dict(name = 'assessed', replace = True),
```

```
# Plot ROC
plt.figure(figsize = (10,10))
plt.plot(1-ROC_pandas_gb['_Specificity_'],
           ROC_pandas_gb['_Sensitivity_'], 'go-', linewidth = 3)
plt.plot(pd.Series(range(0,11,1))/10, pd.Series(range(0,11,1))/10, 'k--')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(['Gradient Boosting'])
plt.show()
```




```
# Plot Lift Locally
plt.figure(figsize = (10,10))
plt.plot(Lift_pandas_gb['_Depth_'], Lift_pandas_gb['_CumLift_'], 'go-', linewidth = 3)
plt.xlabel('Depth')
plt.ylabel('Cumulative Lift')
plt.title('Cumulative Lift Curve')
plt.legend(['Gradient Boosting'])
plt.show()
```




```
conn.session.endSession()
```

elapsed 0.0063s · user 0.00643s · sys 0.00512s · mem 0.851MB