MINERAÇÃO DE DADOS COMPLEXOS

EXTECAMP

Dataset:
Medical Appointments

Grupo “E Éramos Quatro”:
Felipe Pereira
Anderson Rocha
Alexandre Guidin
~~Miguel Filho~~

Introdução

O foco deste trabalho é a realização da análise do dataset e a criação do modelo de predição para o problema proposto.

O problema proposto é o de Consultas Médicas. Neste conjunto, temos uma série de atributos e, como target, devemos saber se o paciente comparecerá à consulta ou não.

Para isto, realizamos o trabalho que se segue.

1. Análise dos dados

Dimensionalidade :

O dataset proposto possui as seguintes dimensões:

110527 registros
14 atributos

Estatísticas:

Através da análise sumária dos atributos, pudemos verificar que não existe nenhum valor nulo no dataset. Também não parece existir nenhum valor corrompido ou incorreto, pois todos os valores estão dentro de faixas esperadas para o atributo (exemplo: a idade máxima é de 115 anos, o que parece ser razoável, embora não tão comum). Detalhamento do dataset:

	PatientId	AppointmentID	Age	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received
count	1.105270e+05	1.105270e+05	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000
mean	1.474963e+14	5.675305e+06	37.088874	0.098266	0.197246	0.071865	0.030400	0.022248	0.321026
std	2.560949e+14	7.129575e+04	23.110205	0.297675	0.397921	0.258265	0.171686	0.161543	0.466873
min	3.921784e+04	5.030230e+06	-1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.172614e+12	5.640286e+06	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	3.173184e+13	5.680573e+06	37.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	9.439172e+13	5.725524e+06	55.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	9.999816e+14	5.790484e+06	115.000000	1.000000	1.000000	1.000000	1.000000	4.000000	1.000000

Balanceamento da variável target:

Foi verificado que o dataset está desbalanceado. Cerca de 80% do dataset é composto de registros de pessoas que compareceram à consulta (No-Show=No) e apenas 20% é composto de pessoas que faltaram às consultas (No-Show=Yes). Os dados precisos estão abaixo:

No: 28.707
Yes: 6.115

Testamos o balanceamento do dataset (tanto com under sampling quanto com over sampling), e ficamos com o oversampling para testar:

No: 28.707
Yes: 28.707

Porém, os resultados finais indicaram modelos melhores para os dados originais.

Valores Nulos:

Confirmando a não existência de valores nulos, extraímos mais uma visão do dataset que comprova isso:

```
PatientId         False
AppointmentID     False
Gender            False
ScheduledDay      False
AppointmentDay    False
Age               False
Neighbourhood     False
Scholarship       False
Hipertension      False
Diabetes          False
Alcoholism        False
Handcap           False
SMS_received      False
No-show           False
dtype: bool
```

Transformação de dados:

Através da análise do dataset, foi verificado que algumas transformações de dados eram necessárias. As transformações realizadas foram:

- Extração do mês da data agendada;
- Extração do dia da data agendada;
- Extração e cálculo de diferença de dias entre o agendamento e a consulta;
- Drop de colunas;
- Mapeamento de variáveis categóricas para features discretas.

Normalização:

Os dados de Idade, Dia do mês e Mês foram normalizados para ficarem na mesma faixa dos outros valores. Isto influenciou positivamente a acurácia.

Correlação:

Abaixo segue a correlação entre os atributos. O target mostra maior correlação com os atributos Scholarship e SMS_Received:

	PatientId	AppointmentID	Age	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
PatientId	1.000000	0.004039	-0.004139	-0.002880	-0.006441	0.001605	0.011011	-0.007916	-0.009749	-0.001461
AppointmentID	0.004039	1.000000	-0.019126	0.022615	0.012752	0.022628	0.032944	0.014106	-0.256618	-0.162602
Age	-0.004139	-0.019126	1.000000	-0.092457	0.504586	0.292391	0.095811	0.078033	0.012643	-0.060319
Scholarship	-0.002880	0.022615	-0.092457	1.000000	-0.019729	-0.024894	0.035022	-0.008586	0.001194	0.029135
Hipertension	-0.006441	0.012752	0.504586	-0.019729	1.000000	0.433086	0.087971	0.080083	-0.006267	-0.035701
Diabetes	0.001605	0.022628	0.292391	-0.024894	0.433086	1.000000	0.018474	0.057530	-0.014550	-0.015180
Alcoholism	0.011011	0.032944	0.095811	0.035022	0.087971	0.018474	1.000000	0.004648	-0.026147	-0.000196
Handcap	-0.007916	0.014106	0.078033	-0.008586	0.080083	0.057530	0.004648	1.000000	-0.024161	-0.006076
SMS_received	-0.009749	-0.256618	0.012643	0.001194	-0.006267	-0.014550	-0.026147	-0.024161	1.000000	0.126431
No-show	-0.001461	-0.162602	-0.060319	0.029135	-0.035701	-0.015180	-0.000196	-0.006076	0.126431	1.000000

2. Criação do modelo

Separamos o dataset em 80% para treinamento (sendo 60% treinamento e 20% validação) e 20% para o teste final:

```
#Separação do dataset
from sklearn.model_selection import train_test_split

X = data.drop('Noshow',1)
y = data.Noshow

#Separação de conjunto de testes
X_model, X_test, y_model, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

#Separação de conjunto de validação
X_train, X_val, y_train, y_val = train_test_split(X_model, y_model, test_size=0.2, random_state=0)
```

Também decidimos testar vários modelos diferentes para verificar a eficiência de cada um. Para cada modelo escolhido, realizamos 2 experimentos, um com o conjunto de dados original e outro com o conjunto de dados resampled com oversampling:

```python
#Criação dos modelos
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='auto', random_state=0)
lr = LogisticRegression(random_state=0, penalty='l2', C=1, fit_intercept=True, solver='liblinear')
dt = DecisionTreeClassifier(random_state=0, criterion='entropy', splitter='best')

algs = []
algs.append(rf)
algs.append(lr)
algs.append(dt)

rf1 = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='auto', random_state=0)
lr1 = LogisticRegression(random_state=0, penalty='l2', C=1, fit_intercept=True, solver='liblinear')
dt1 = DecisionTreeClassifier(random_state=0, criterion='entropy', splitter='best')

algsSampled = []
algsSampled.append(rf1)
algsSampled.append(lr1)
algsSampled.append(dt1)

for alg in algs:
    print('Fitting: ', type(alg).__name__)
    alg.fit(X_train, y_train)

for alg in algsSampled:
    print('Fitting: ', type(alg).__name__, ' resampled')
    alg.fit(x_resampled, y_resampled)
```

```
Fitting:  RandomForestClassifier
Fitting:  LogisticRegression
Fitting:  DecisionTreeClassifier
Fitting:  RandomForestClassifier  resampled
Fitting:  LogisticRegression  resampled
Fitting:  DecisionTreeClassifier  resampled
```

Para cada modelo, testamos alguns hiper-parâmetros diferentes:

- **RandomForestClassifier**:
    - **N_estimators**: A quantidade de árvores a serem colocadas na floresta. Testamos com 10, 100 e 1000. A quantidade de 100 obteve uma melhor relação de custo/benefício.
    - **Criterion**: O tipo de função usado para medir a qualidade de um split (qual feature é mais relevante). Gini e Entropy são muito parecidas, com um ligeiro ganho para Entropy.
    - **Max_Features**: Testado com auto (raiz quadrada da quantidade total) e também com a quantidade total. O teste com auto ficou melhor.

- **LogisticRegression**:

- **Penalty**: a penalidade para evitar overfit. Tanto L1 quanto L2 possuem performance igual neste caso.

- **C**: Foram testados valores de 0.001 a 100, e o resultado não foi alterado. Continuamos com o padrão (1).

- **fit_intercept**: Utilizado para colocar um bias na função de classificação. Não influenciou nos resultados.

- **solver**: Foram testados outros modelos de classificação (sag e saga), mas ficamos com o padrão de liblinear.

- **Decision Tree**:

- **Criterion**: O tipo de função usado para medir a qualidade de um split (qual feature é mais relevante). Gini e Entropy são muito parecidas, com um ligeiro ganho para Entropy.

Testamos os modelos em validação para verificar as melhoras obtidas:

```
#Acurácia do modelo em VALIDAÇÃO
from sklearn.metrics import accuracy_score, confusion_matrix

for alg in algs:
    predicted = alg.predict(X_val)
    accuracy = accuracy_score(y_val, predicted)
    print('Acc: ', type(alg).__name__, f' = {accuracy:.3}')
    print(confusion_matrix(y_val, predicted))
    print()
    print()
```

3. Acurácia de modelos

Para a validação dos modelos, fizemos o cálculo da acurácia, baseado na matriz de confusão, com um conjunto de teste separado. Os resultados seguem abaixo:

	Name	Type	Resampled	ACC
0	RandomForestClassifier	Training	N	0.995175
1	LogisticRegression	Training	N	0.823388
2	DecisionTreeClassifier	Training	N	0.995233
3	RandomForestClassifier	Training	Y	0.997109
4	LogisticRegression	Training	Y	0.610200
5	DecisionTreeClassifier	Training	Y	0.997109
6	RandomForestClassifier	Validation	N	0.813347
7	LogisticRegression	Validation	N	0.821617
8	DecisionTreeClassifier	Validation	N	0.744199
9	RandomForestClassifier	Validation	Y	0.778544
10	LogisticRegression	Validation	Y	0.644843
11	DecisionTreeClassifier	Validation	Y	0.738571
12	RandomForestClassifier	Test	N	0.818783
13	LogisticRegression	Test	N	0.824848
14	DecisionTreeClassifier	Test	N	0.741684
15	RandomForestClassifier	Test	Y	0.784047
16	LogisticRegression	Test	Y	0.652086
17	DecisionTreeClassifier	Test	Y	0.743981

Tanto a Random Forest quanto a Decision Tree obtiveram um resultado excepcional no treinamento e depois um resultado bem diferente na Validação e no Teste (para os modelos que receberam os conjuntos de dados originais e não os resampled).

Este comportamento indica que houve overfit tanto da Random Forest quanto da Decision Tree. Já a Logistic Regression manteve o mesmo nível de acurácia em todas as fases, mostrando-se um modelo mais maduro. O melhor modelo da Logistic Regression foi o que foi treinado com os dados originais.