MARATONA BEHIND THE CODE 2020

DESAFIO 2: PARTE 1

Introdução

Em projetos de ciência de dados visando a construção de modelos de *machine learning*, ou aprendizado estatístico, é muito incomum que os dados iniciais estejam já no formato ideal para a construção de modelos. São necessários vários passos intermediários de pré-processamento de dados, como por exemplo a codificação de variáveis categóricas, normalização de variáveis numéricas, tratamento de dados faltantes, etc. A biblioteca **scikit-learn** -- uma das mais populares bibliotecas de código-aberto para *machine learning* no mundo -- possui diversas funções já integradas para a realização das transformações de dados mais utilizadas. Entretanto, em um fluxo comum de um modelo de aprendizado de máquina, é necessária a aplicação dessas transformações pelo menos duas vezes: a primeira vez para "treinar" o modelo, e depois novamente quando novos dados forem enviados como entrada para serem classificados por este modelo.

Para facilitar o trabalho com esse tipo de fluxo, o scikit-learn possui também uma ferramenta chamada **Pipeline**, que nada mais é do que uma lista ordenada de transformações que devem ser aplicadas nos dados. Para auxiliar no desenvolvimento e no gerenciamento de todo o ciclo-de-vida dessas aplicações, alem do uso de Pipelines, as equipes de cientistas de dados podem utilizar em conjunto o **Watson Machine Learning**, que possui dezenas de ferramentas para treinar, gerenciar, hospedar e avaliar modelos baseados em aprendizado de máquina. Além disso, o Watson Machine Learning é capaz de encapsular pipelines e modelos em uma API pronta para uso e integração com outras aplicações.

Durante o desafio 2, você participante irá aprender a construir uma **Pipeline** para um modelo de classificação e hospedá-lo como uma API com o auxílio do Watson Machine Learning. Uma vez hospedado, você poderá integrar o modelo criado com outras aplicações, como assistentes virtuais e muito mais. Neste notebook, será apresentado um exemplo funcional de criação de um modelo e de uma pipeline no scikit-learn (que você poderá utilizar como template para a sua solução!).

Trabalhando com Pipelines do scikit-learn

```
# Primeiro, realizamos a instalação do scikit-learn versão 0.20.0 no Kernel deste notebook:
```

```
!pip install scikit-learn==0.20.0 --upgrade
```

```
Requirement already up-to-date: scikit-learn==0.20.0 in /usr/local/lib/python3.6/dist-packages (0.20.0)
Requirement already satisfied, skipping upgrade: scipy>=0.13.3 in /usr/local/lib/python3.6/dist-packages (from scikit-learn==0.20.0) (1.4.1)
Requirement already satisfied, skipping upgrade: numpy>=1.8.2 in /usr/local/lib/python3.6/dist-packages (from scikit-learn==0.20.0) (1.18.5)
```

```
# Em seguida iremos importar diversas bibliotecas que serão utilizadas:

# Pacote para trabalhar com JSON
import json

# Pacote para realizar requisições HTTP
import requests

# Pacote para exploração e análise de dados
import pandas as pd

# Pacote com métodos numéricos e representações matriciais
import numpy as np

# Pacote para construção de modelo baseado na técnica Gradient Boosting
import xgboost as xgb

# Pacotes do scikit-learn para pré-processamento de dados
# "SimpleImputer" é uma transformação para preencher valores faltantes em conjuntos de dados
from sklearn.impute import SimpleImputer

# Pacotes do scikit-learn para treinamento de modelos e construção de pipelines
# Método para separação de conjunto de dados em amostras de treino e teste
from sklearn.model_selection import train_test_split
# Método para criação de modelos baseados em árvores de decisão
from sklearn.tree import DecisionTreeClassifier
# Classe para a criação de uma pipeline de machine-learning
from sklearn.pipeline import Pipeline

# Pacotes do scikit-learn para avaliação de modelos
# Métodos para validação cruzada do modelo criado
from sklearn.model_selection import KFold, cross_validate
```

Importando um .csv de seu projeto no IBM Cloud Pak for Data para o Kernel deste notebook

Primeiro iremos importar o dataset fornecido para o desafio, que já está incluso neste projeto!

Você pode realizar a importação dos dados de um arquivo .csv diretamente para o Kernel do notebook como um DataFrame da biblioteca Pandas, muito utilizada para a manipulação de dados em Python.

Para realizar a importação, basta selecionar a próxima célula e seguir as instruções na imagem abaixo:

 alt text

Após a seleção da opção "**Insert to code**", a célula abaixo será preenchida com o código necessário para importação e leitura dos dados no arquivo .csv como um DataFrame Pandas.

```
#pd.read_csv("/content/dataset_desafio_2.csv", nrows=5)
```

```
#df_teste = pd.read_csv("/content/dataset_desafio_2.csv", nrows=5)
```

```
df_data_1 = pd.read_csv("/content/dataset_desafio_2.csv")
df_data_1.head(5)
```

	MATRICULA	NOME	REPROVACOES_DE	REPROVACOES_EM	REPROVACOES_MF	REPROVACOES_GO	NOTA_DE	NOTA_EM	NOTA_MF	NOTA_GO	INGLES	H_AULA_
0	502375	Márcia Illiglener	0	0	0	0	6.2	5.8	4.6	5.9	0.0	
1	397093	Jason Jytereoman Izoimum	0	0	0	0	6.0	6.2	5.2	4.5	1.0	
2	915288	Bartolomeu Inácio da Gama	0	0	0	0	7.3	6.7	7.1	7.2	0.0	
3	102652	Fernanda	1	2	1	1	0.0	0.0	0.0	0.0	1.0	

```
df_data_1.sample(10)
```

```
MATRICULA       - número de matrícula do estudante
NOME            - nome completo do estudante
REPROVACOES_DE  - número de reprovações na disciplina de ``Direito Empresarial``
REPROVACOES_EM  - número de reprovações na disciplina de ``Empreendedorismo``
REPROVACOES_MF  - número de reprovações na disciplina de ``Matemática Financeira``
REPROVACOES_GO  - número de reprovações na disciplina de ``Gestão Operacional``
NOTA_DE         - média simples das notas do aluno na disciplina de ``Direito Empresarial`` (0-10)
NOTA_EM         - média simples das notas do aluno na disciplina de ``Empreendedorismo`` (0-10)
NOTA_MF         - média simples das notas do aluno na disciplina de ``Matemática Financeira`` (0-10)
NOTA_GO         - média simples das notas do aluno na disciplina de ``Gestão Operacional`` (0-10)
INGLES          - variável binária que indica se o estudante tem conhecimento em língua inglesa (0 -> sim ou 1 -> não).
H_AULA_PRES     - horas de estudo presencial realizadas pelo estudante
TAREFAS_ONLINE  - número de tarefas online entregues pelo estudante
FALTAS          - número de faltas acumuladas do estudante (todas disciplinas)
```

A variável-alvo é:

```
PERFIL              - uma *string* que indica uma de cinco possibilidades:
    "EXCELENTE"     - Estudante não necessita de mentoria
    "MUITO BOM"     - Estudante não necessita de mentoria
    "HUMANAS"       - Estudante necessita de mentoria exclusivamente em matérias com conteúdo de ciências humanas
    "EXATAS"        - Estudante necessita de mentoria apenas em disciplinas com conteúdo de ciências exatas
    "DIFICULDADE"   - Estudante necessita de mentoria em duas ou mais disciplinas
```

Com um modelo capaz de classificar um estudante em uma dessas categorias, podemos automatizar parte da mentoria estudantil através de assistentes virtuais, que serão capazes de recomendar práticas de estudo e conteúdo personalizado com base nas necessidades de cada

Explorando os dados fornecidos

Podemos continuar a exploração dos dados fornecidos com a função `info()`:

```
df_data_1.info()
```

	MATRICULA	NOME	REPROVACOES_DE	REPROVACOES_EM	REPROVACOES_MF	REPROVACOES_GO	NOTA_DE	NOTA_EM	NOTA_MF	NOTA_GO	INGLES	H_AU
8843	539109	Lígia Elisa Egasov Disnaz de Gusmões	0	0	0	0	7.1	7.0	7.4	6.7	1.0	
6584	421925	Reginaldo de Álvares Valverde	0	0	0	0	6.5	5.7	5.1	NaN	NaN	
2184	174531	Natália Alessandra de Junqueira	1	1	1	1	0.0	0.0	0.0	0.0	1.0	
2474	312647	Alan Lumal Udrasece	0	0	0	0	5.3	4.4	4.9	NaN	NaN	
7045	474573	Saulo Nanwomvic Otávio	3	3	1	1	0.0	0.0	0.0	0.0	1.0	

Quantidades por perfil

```
df_data_1.groupby('PERFIL').size()
```

```
PERFIL
DIFICULDADE    3508
EXATAS         4190
EXCELENTE       306
HUMANAS        1555
MUITO_BOM       440
dtype: int64
```

Temos 15 colunas presentes no dataset fornecido, sendo dezessete delas variáveis características (dados de entrada) e um delas uma variável-alvo (que queremos que o nosso modelo seja capaz de prever).

As variáveis características são:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   MATRICULA        9999 non-null   int64  
 1   NOME             9999 non-null   object 
 2   REPROVACOES_DE   9999 non-null   int64  
 3   REPROVACOES_EM   9999 non-null   int64  
 4   REPROVACOES_MF   9999 non-null   int64  
 5   REPROVACOES_GO   9999 non-null   int64  
 6   NOTA_DE          9999 non-null   float64
 7   NOTA_EM          9999 non-null   float64
 8   NOTA_MF          9999 non-null   float64
 9   NOTA_GO          8099 non-null   float64
 10  INGLES           8165 non-null   float64
 11  H_AULA_PRES      9999 non-null   int64  
 12  TAREFAS_ONLINE   9999 non-null   int64  
 13  FALTAS           9999 non-null   int64  
 14  PERFIL           9999 non-null   object 
dtypes: float64(5), int64(8), object(2)
memory usage: 1.1+ MB
```

É notado que existem variáveis do tipo `float64` (números "decimais"), variáveis do tipo `int64` (números inteiros) e do tipo `object` (nesse caso são *strings*, ou texto).

Como a maioria dos algoritmos de aprendizado estatístico supervisionado só aceita valores numéricos como entrada, é necessário então o pré-processamento das variáveis do tipo "object" antes de usar esse dataset como entrada para o treinamento de um modelo. Também é notado que existem valores faltantes em várias colunas. Esses valores faltantes também devem ser tratados antes de serem construídos modelos com esse conjunto de dados base.

A função `describe()` gera várias informações sobre as variáveis numéricas que também podem ser úteis:

```
df_data_1.describe()
```

	MATRICULA	REPROVACOES_DE	REPROVACOES_EM	REPROVACOES_MF	REPROVACOES_GO	NOTA_DE	NOTA_EM	NOTA_MF	NOTA_GO	
count	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	8099.000000	8165.
mean	551593.966797	0.241224	0.238224	0.297730	0.290129	5.188009	5.067017	4.795240	4.524534	0.
std	259458.695128	0.612447	0.603762	0.675542	0.656064	2.511396	2.511394	2.709779	2.493729	0.
min	100003.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	326239.500000	0.000000	0.000000	0.000000	0.000000	5.200000	4.900000	4.700000	4.500000	0.

Visualizações

Para visualizar o dataset fornecido, podemos utilizar as bibliotecas `matplotlib` e `seaborn`:

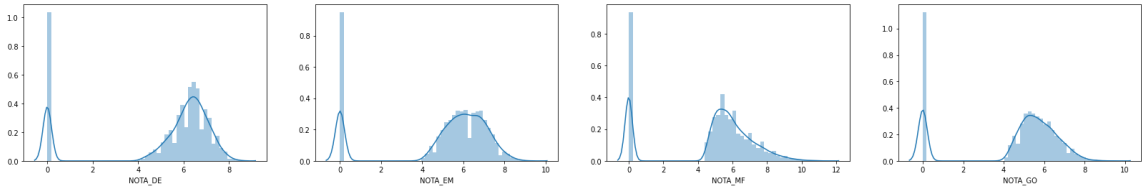
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))

sns.countplot(ax=axes[0], x='REPROVACOES_DE', data=df_data_1)
sns.countplot(ax=axes[1], x='REPROVACOES_EM', data=df_data_1)
sns.countplot(ax=axes[2], x='REPROVACOES_MF', data=df_data_1)
sns.countplot(ax=axes[3], x='REPROVACOES_GO', data=df_data_1)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fcc14398a90>

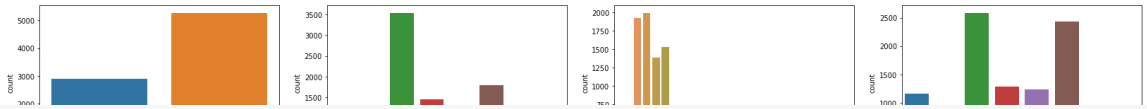
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))

sns.distplot(df_data_1['NOTA_DE'], ax=axes[0])
sns.distplot(df_data_1['NOTA_EM'], ax=axes[1])
sns.distplot(df_data_1['NOTA_MF'], ax=axes[2])
sns.distplot(df_data_1['NOTA_GO'].dropna(), ax=axes[3])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fcc141ba668>



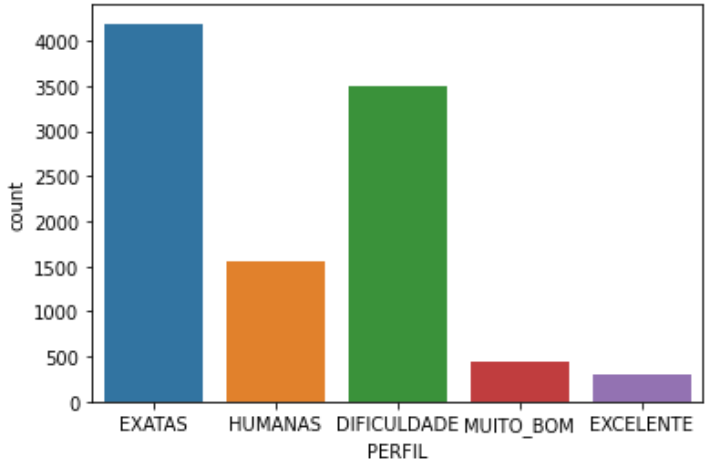

```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))

sns.countplot(ax=axes[0], x='INGLES', data=df_data_1)
sns.countplot(ax=axes[1], x='FALTAS', data=df_data_1)
sns.countplot(ax=axes[2], x='H_AULA_PRES', data=df_data_1)
sns.countplot(ax=axes[3], x='TAREFAS_ONLINE', data=df_data_1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc13e309b0>
```



```
fig = plt.plot()
sns.countplot(x='PERFIL', data=df_data_1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc144d7c18>
```



Realizando o pré-processamento dos dados

Para o pré-processamento dos dados serão apresentadas duas transformações básicas neste notebook, demonstrando a construção de uma Pipeline com um modelo funcional. Esta Pipeline funcional fornecida deverá ser melhorada pelo participante para que o modelo final alcance a maior acurácia possível, garantindo uma pontuação maior no desafio. Essa melhoria pode ser feita apenas no pré-processamento dos dados, na escolha de um algoritmo para treinamento de modelo diferente, ou até mesmo na alteração do *framework* usado (entretanto só será fornecido um exemplo pronto de integração do Watson Machine Learning com o *scikit-learn*).

A primeira transformação (passo na nossa Pipeline) será a exclusão da coluna "NOME" do nosso dataset, que além de não ser uma variável numérica, também não é uma variável relacionada ao desempenho dos estudantes nas disciplinas. Existem funções prontas no scikit-learn para a realização dessa transformação, entretanto nosso exemplo irá demonstrar como criar uma transformação personalizada do zero no scikit-learn. Se desejado, o participante poderá utilizar esse exemplo para criar outras transformações e adicioná-las à Pipeline final :)

Transformação 1: excluindo colunas do dataset

Para a criação de uma transformação de dados personalizada no scikit-learn, é necessária basicamente a criação de uma classe com os métodos `transform` e `fit`. No método transform será executada a lógica da nossa transformação.

Na próxima célula é apresentado o código completo de uma transformação `DropColumns` para a remoção de colunas de um DataFrame pandas.

```
from sklearn.base import BaseEstimator, TransformerMixin


# All sklearn Transforms must have the `transform` and `fit` methods
class DropColumns(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Primeiro realizamos a cópia do dataframe 'X' de entrada
        data = X.copy()
        # Retornamos um novo dataframe sem as colunas indesejadas
        return data.drop(labels=self.columns, axis='columns')
```

Para aplicar essa transformação em um DataFrame pandas, basta instanciar um objeto *DropColumns* e chamar o método transform().

```
# Instanciando uma transformação DropColumns
rm_columns = DropColumns(
    columns=["NOME"]  # Essa transformação recebe como parâmetro uma lista com os nomes das colunas indesejadas
)

print(rm_columns)
```

```
DropColumns(columns=['NOME'])
```

```
# Visualizando as colunas do dataset original
```

```
print("Colunas do dataset original: \n")
print(df_data_1.columns)
```

```
Colunas do dataset original:

Index(['MATRICULA', 'NOME', 'REPROVACOES_DE', 'REPROVACOES_EM',
       'REPROVACOES_MF', 'REPROVACOES_GO', 'NOTA_DE', 'NOTA_EM', 'NOTA_MF',
       'NOTA_GO', 'INGLES', 'H_AULA_PRES', 'TAREFAS_ONLINE', 'FALTAS',
       'PERFIL'],
      dtype='object')
```

```
# Aplicando a transformação ``DropColumns`` ao conjunto de dados base
rm_columns.fit(X=df_data_1)

# Reconstruindo um DataFrame Pandas com o resultado da transformação
df_data_2 = pd.DataFrame.from_records(
    data=rm_columns.transform(
        X=df_data_1
    ),
)
```

```
# Visualizando as colunas do dataset transformado
print("Colunas do dataset após a transformação ``DropColumns``: \n")
print(df_data_2.columns)
```

```
Colunas do dataset após a transformação ``DropColumns``:

Index(['MATRICULA', 'REPROVACOES_DE', 'REPROVACOES_EM', 'REPROVACOES_MF',
       'REPROVACOES_GO', 'NOTA_DE', 'NOTA_EM', 'NOTA_MF', 'NOTA_GO', 'INGLES',
       'H_AULA_PRES', 'TAREFAS_ONLINE', 'FALTAS', 'PERFIL'],
      dtype='object')
```

Nota-se que a coluna "NOME" foi removida e nosso dataset agora poossui apenas 17 colunas.

Transformação 2: tratando dados faltantes

Para tratar os dados faltantes em nosso conjunto de dados, iremos agora utilizar uma transformação pronta da biblioteca scikit-learn, chamada **SimpleImputer**.

Essa transformação permite diversas estratégias para o tratamento de dados faltantes. A documentação oficial pode ser encontrada em: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

Neste exemplo iremos simplesmente transformar todos os valores faltantes em zero.

```
# Criação de um objeto ``SimpleImputer``
si = SimpleImputer(
    missing_values=np.nan,  # os valores faltantes são do tipo ``np.nan`` (padrão Pandas)
    strategy='constant',  # a estratégia escolhida é a alteração do valor faltante por uma constante
    fill_value=0,  # a constante que será usada para preenchimento dos valores faltantes é um int64=0.
    verbose=0,
    copy=True
)
```

```
# Visualizando os dados faltantes do dataset após a primeira transformação (df_data_2)
print("Valores nulos antes da transformação SimpleImputer: \n\n{}\n".format(df_data_2.isnull().sum(axis = 0)))
```

```
Valores nulos antes da transformação SimpleImputer:

MATRICULA            0
REPROVACOES_DE       0
REPROVACOES_EM       0
REPROVACOES_MF       0
REPROVACOES_GO       0
NOTA_DE              0
NOTA_EM              0
NOTA_MF              0
NOTA_GO           1900
INGLES            1834
H_AULA_PRES          0
TAREFAS_ONLINE       0
FALTAS               0
PERFIL               0
dtype: int64
```

```
# Aplicamos o SimpleImputer ``si`` ao conjunto de dados df_data_2 (resultado da primeira transformação)
si.fit(X=df_data_2)

# Reconstrução de um novo DataFrame Pandas com o conjunto imputado (df_data_3)
```

```
df_data_3 = pd.DataFrame.from_records(
    data=si.transform(
        X=df_data_2
    ),  # o resultado SimpleImputer.transform(<<pandas dataframe>>) é lista de listas
    columns=df_data_2.columns  # as colunas originais devem ser conservadas nessa transformação
)
```

```
# Visualizando os dados faltantes do dataset após a segunda transformação (SimpleImputer) (df_data_3)
print("Valores nulos no dataset após a transformação SimpleImputer: \n\n{}\n".format(df_data_3.isnull().sum(axis = 0)))
```

```
Valores nulos no dataset após a transformação SimpleImputer:

MATRICULA         0
REPROVACOES_DE    0
REPROVACOES_EM    0
REPROVACOES_MF    0
REPROVACOES_GO    0
NOTA_DE           0
NOTA_EM           0
NOTA_MF           0
NOTA_GO           0
INGLES            0
H_AULA_PRES       0
TAREFAS_ONLINE    0
FALTAS            0
PERFIL            0
dtype: int64
```

Nota-se que não temos mais nenhum valor faltante no nosso conjunto de dados :)

Vale salientar que nem sempre a alteração dos valores faltantes por 0 é a melhor estratégia. O participante é incentivado a estudar e implementar estratégias diferentes de tratamento dos valores faltantes para aprimorar seu modelo e melhorar sua pontuação final.

Treinando um modelo de classificação

Finalizado o pré-processamento, já temos o conjunto de dados no formato necessário para o treinamento do nosso modelo:

```
df_data_3.head()
```

```
df_data_3.head()
```

	MATRICULA	REPROVACOES_DE	REPROVACOES_EM	REPROVACOES_MF	REPROVACOES_GO	NOTA_DE	NOTA_EM	NOTA_MF	NOTA_GO	INGLES	H_AULA_PRES	TAREFA
0	502375	0	0	0	0	6.2	5.8	4.6	5.9	0.0	2	
1	397093	0	0	0	0	6.0	6.2	5.2	4.5	1.0	2	
2	915288	0	0	0	0	7.3	6.7	7.1	7.2	0.0	5	
3	192652	1	3	1	1	0.0	0.0	0.0	0.0	1.0	4	
4	949491	1	3	1	1	0.0	0.0	0.0	0.0	1.0	5	

No exemplo fornecido, iremos utilizar todas as colunas, exceto a coluna **LABELS** como *features* (variáveis de entrada).

A variável **LABELS** será a variável-alvo do modelo, conforme descrito no enunciado do desafio.

Definindo as features do modelo

```
# Definição das colunas que serão features (nota-se que a coluna NOME não está presente)
features = [
    "MATRICULA", 'REPROVACOES_DE', 'REPROVACOES_EM', "REPROVACOES_MF", "REPROVACOES_GO",
    "NOTA_DE", "NOTA_EM", "NOTA_MF", "NOTA_GO",
    "INGLES", "H_AULA_PRES", "TAREFAS_ONLINE", "FALTAS",
]

# Definição da variável-alvo
target = ["PERFIL"]

# Preparação dos argumentos para os métodos da biblioteca ``scikit-learn``
X = df_data_3[features]
y = df_data_3[target]
```

O conjunto de entrada (X):

```
X.head()
```

As variáveis-alvo correspondentes (y):

```
y.head()
```

Separando o dataset em um conjunto de treino e um conjunto de teste

Iremos separar o dataset fornecido em dois grupos: um para treinar nosso modelo, e outro para testarmos o resultado através de um teste cego. A separação do dataset pode ser feita facilmente com o método *train_test_split()* do scikit-learn:

```
# Separação dos dados em um conjunto de treino e um conjunto de teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=337)
```

```
y_train
```

Criando um modelo baseado em árvores de decisão

No exemplo fornecido iremos criar um classificador baseado em **árvores de decisão**.

O primeiro passo é basicamente instanciar um objeto *DecisionTreeClassifier()* da biblioteca scikit-learn.

```
# Criação da árvore de decisão com a biblioteca ``scikit-learn``:
dtc_model = DecisionTreeClassifier()  # O modelo será criado com os parâmetros padrões da biblioteca
```

Material teórico sobre árvores de decisão na documentação oficial do scikit-learn: https://scikit-learn.org/stable/modules/tree.html

Um guia para iniciantes no mundo do machine learning: https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/

Execução do evento de treino de uma árvore de decisão

```
# Treino do modelo (é chamado o método *fit()* com os conjuntos de treino)
dtc_model.fit(
    X_train,
    y_train
)
```

Execução de predições e avaliação do modelo criado

```
# Realização de teste cego no modelo criado
y_pred = dtc_model.predict(X_test)
```

```
X_test.head()
```

```
print(y_pred)
```

```
from sklearn.metrics import accuracy_score

# Acurácia alcançada pela árvore de decisão
```

```
print("Acurácia: {}%".format(100*round(accuracy_score(y_test, y_pred), 2)))
```

Neste notebook foi demonstrado como trabalhar com transformações e modelos com a biblioteca scikit-learn. É recomendado que o participante realize seus experimentos editando o código fornecido aqui até que um modelo com acurácia elevada seja alcançado.

Quando você estiver satisfeito com seu modelo, pode passar para a segunda etapa do desafio -- encapsular seu modelo como uma API REST pronta para uso com o Watson Machine Learning!

O notebook para a segunda etapa já se encontra neste projeto, basta acessar a aba **ASSETS** e inicializá-lo! Não se esqueca de antes desligar o Kernel deste notebook para reduzir o consumo de sua camada grátis do IBM Cloud Pak for Data.