Relatório- Card- 12 - Prática: Predição e a Base de Aprendizado de Máquina (II)

Jefferson korte junior

Predictive Models

1. 🎥 **Activity** Linear Regression

Regressão Linear: O que é?

A regressão linear é uma técnica estatística utilizada para encontrar a reta que melhor se ajusta a um conjunto de observações. O objetivo é modelar a relação entre uma variável independente (X) e uma variável dependente (Y), assumindo que essa relação seja aproximadamente linear.

Como saber se a regressão é boa?

Para avaliar a qualidade do ajuste da regressão linear, usamos o coeficiente de determinação (R^2).

- R^2 varia entre 0 e 1
- Um R^2 próximo de 1 indica que a linha explica bem a variação dos dados
- Um R^2 baixo significa que o modelo não representa bem os dados, ou seja, há muita dispersão em torno da linha.

Quanto maior o R^2, melhor o ajuste da regressão.

Predictive Models

2. 🎥 **Activity** Polynomial Regression

Regressão Polinomial

A regressão polinomial é uma forma de prever valores usando uma curva em vez de uma linha reta. Ela é útil quando os dados não seguem uma relação linear simples.

Nessa atividade, usamos dois conjuntos de dados: a velocidade com que as pessoas navegam numa página **(pageSpeeds)** e quanto elas gastam **(purchaseAmount)**. Fizemos um gráfico de dispersão e vimos que os pontos não formavam uma linha reta, então testamos uma regressão polinomial de grau 4 para ver se conseguíamos representar melhor essa relação.

Com isso, conseguimos ver a curva se ajustando bem aos dados, mostrando como a velocidade de navegação pode influenciar no valor que a pessoa gasta.

Predictive Models

3. 🎥 **Activity** Multiple Regression, and Predicting Car Prices

Regressão Múltipla a regressão múltipla é uma técnica estatística utilizada para prever um valor com base em **duas ou mais variáveis independentes**. Ao invés de considerar apenas um fator, como a quilometragem de um carro, ela incorpora múltiplas características — como ano de fabricação, potência do motor e tipo de combustível — para estimar, por exemplo, o preço de um automóvel com maior precisão.

OLS (*Ordinary Least Squares*) é um método estatístico usado para **ajustar uma linha ou um modelo de regressão** que melhor se encaixa aos dados. A ideia principal é **minimizar a soma dos quadrados dos resíduos**, ou seja, a diferença entre os valores observados e os valores previstos pelo modelo.

Imagine que você tem vários pontos espalhados em um gráfico e quer traçar uma linha que passe o mais próximo possível de todos eles. OLS encontra essa linha ideal.

Machine Learning with Python

Aprendizado de máquina supervisionado X Não supervisionado

Aprendizado Não supervisionado é quando não estamos dando ao modelo nenhuma resposta para aprender, é quando estamos apenas apresentando a ele um conjunto de dados e ele tenta dar sentido a isso sem nenhuma informação adicional

Já no **aprendizado supervisionado**, fornecemos ao modelo exemplos com entradas e saídas conhecidas (rótulos), para que ele aprenda a fazer previsões. Um exemplo comum é prever o preço de um carro com base em seus atributos.

Para avaliar o desempenho em aprendizado supervisionado, normalmente dividimos os dados em dois conjuntos: **treinamento** e **teste**.
Também podemos usar técnicas como a **validação cruzada K-Fold**, que melhora a avaliação dividindo os dados em várias partes e testando o modelo em diferentes combinações.

Um cuidado importante é evitar o **overfitting**, que acontece quando o modelo aprende demais os dados de treino, mas perde a capacidade de generalizar para novos dados.

Train/test

Treinamento e teste, consiste em um método de aprendizado em que o modelo, de aprendizado supervisionado ou não, é treinado utilizando certa porcentagem dos dados para treino e outra para testes. A fração do conjunto de dados mais utilizada é 80/20, 80% de dados para treinamento e 20% para testes. Train/Test é extremamente importante em modelos de aprendizado supervisionado, já que o modelo é utilizado para relacionar entradas e saídas, e testá-los para predição de saídas conhecidas, permite calcular a sua precisão, e prevenir overfitting e underfitting. É uma ótima forma de verificar a generalização do modelo

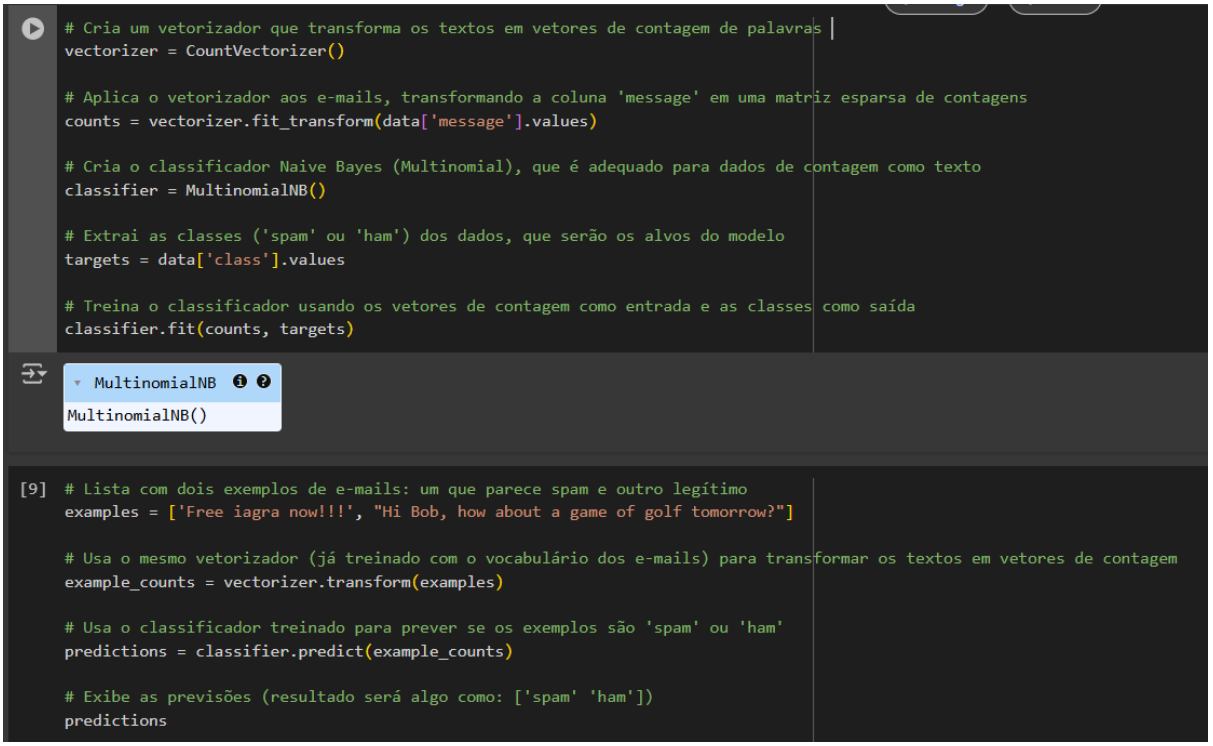
---

Naive Bayes é um algoritmo de aprendizado supervisionado baseado no Teorema de Bayes, que estima a probabilidade de uma classe ser correta com base nas características dos dados de entrada. Sua principal característica é a suposição de que os atributos são independentes entre si, o que raramente é verdade na prática, mas ainda assim o modelo costuma oferecer bons resultados.

Trata-se de um método útil em tarefas de classificação, como a detecção de spam em e-mails, onde ele pode identificar automaticamente mensagens indesejadas com base na frequência de palavras..

Código realizado para verificar se Email é Spam

```python
# Importa os módulos necessários
import os          # Para manipular caminhos e diretórios
import io          # Para leitura de arquivos com codificações específicas
import numpy
import pandas as pd
from pandas import DataFrame
from sklearn.feature_extraction.text import CountVectorizer  # Para converter texto em vetores numéricos
from sklearn.naive_bayes import MultinomialNB                # Algoritmo de classificação Naive Bayes


# Função para ler os arquivos de texto dos e-mails

def readFiles(path):
    # Percorre todos os arquivos dentro do diretório fornecido
    for root, dirnames, filenames in os.walk(path):
        for filename in filenames:
            filepath = os.path.join(root, filename)  # Monta o caminho completo do arquivo

            # Lê o arquivo linha por linha, procurando o corpo do e-mail (após a primeira linha em branco)
            inBody = False
            lines = []
            f = io.open(filepath, 'r', encoding='latin1')
            for line in f:
                if inBody:
                    lines.append(line)  # Lê apenas o corpo do e-mail
                elif line == '\n':
                    inBody = True  # O corpo começa após a primeira linha em branco
            f.close()

            # Junta as linhas do corpo em um único texto
            message = '\n'.join(lines)

            # Retorna o caminho do arquivo e a mensagem
            yield filepath, message


# Função para criar um DataFrame com os e-mails e sua classificação

def dataFrameFromDirectory(path, classification):
    rows = []   # Armazena os dados (mensagem + classe)
    index = []  # Armazena os nomes dos arquivos (como índice)

    # Lê todos os arquivos do diretório e armazena suas informações
    for filename, message in readFiles(path):
        rows.append({'message': message, 'class': classification})
        index.append(filename)

    # Retorna um DataFrame com os dados lidos
    return DataFrame(rows, index=index)


# Define os caminhos para os diretórios com os e-mails extraídos

# OBS: Esses caminhos foram ajustados após verificar a estrutura real dentro do .zip extraído
spam_path = "emails/emails/spam"  # Caminho para e-mails classificados como spam
ham_path  = "emails/emails/ham"   # Caminho para e-mails legítimos (ham)


# Cria o DataFrame principal e preenche com dados de spam e ham

data = pd.DataFrame({'message': [], 'class': []})  # Cria DataFrame vazio com colunas 'message' e 'class'

# Adiciona os e-mails de spam
data = pd.concat([data, dataFrameFromDirectory(spam_path, "spam")])

# Adiciona os e-mails de ham (legítimos)
data = pd.concat([data, dataFrameFromDirectory(ham_path, "ham")])
```



```python
# Cria um vetorizador que transforma os textos em vetores de contagem de palavras
vectorizer = CountVectorizer()

# Aplica o vetorizador aos e-mails, transformando a coluna 'message' em uma matriz esparsa de contagens
counts = vectorizer.fit_transform(data['message'].values)

# Cria o classificador Naive Bayes (Multinomial), que é adequado para dados de contagem como texto
classifier = MultinomialNB()

# Extrai as classes ('spam' ou 'ham') dos dados, que serão os alvos do modelo
targets = data['class'].values

# Treina o classificador usando os vetores de contagem como entrada e as classes como saída
classifier.fit(counts, targets)
```

```
MultinomialNB
MultinomialNB()
```

```python
[9] # Lista com dois exemplos de e-mails: um que parece spam e outro legítimo
examples = ['Free iagra now!!!', "Hi Bob, how about a game of golf tomorrow?"]

# Usa o mesmo vetorizador (já treinado com o vocabulário dos e-mails) para transformar os textos em vetores de contagem
example_counts = vectorizer.transform(examples)

# Usa o classificador treinado para prever se os exemplos são 'spam' ou 'ham'
predictions = classifier.predict(example_counts)

# Exibe as previsões (resultado será algo como: ['spam' 'ham'])
predictions
```

Machine Learning with Python

K-Means Clustering

O K-means Clustering é um algoritmo de dispersão supervisionado que busca agrupar os dados em k clusters baseados em características semelhantes entre si. Isso é feito a partir de centróides que representam pontos centrais em clusters, escolhidos aleatoriamente, onde os pontos são atribuídos pela distância. Ou seja, um ponto de dado pertencerá ao cluster cujo centróide está mais próximo. Esse processo de atribuição de centróides é feito até que não haja mais mudanças significativas. A distância entre pontos pode ser calculada a partir de fórmulas já conhecidas da distância, como a euclidiana.

Exemplo de aplicaçoes que podem ser usadas usando K-Means Clustering:

1 - Agrupar **clientes com perfis parecidos** em marketing (ex: clientes que compram muito vs. pouco).

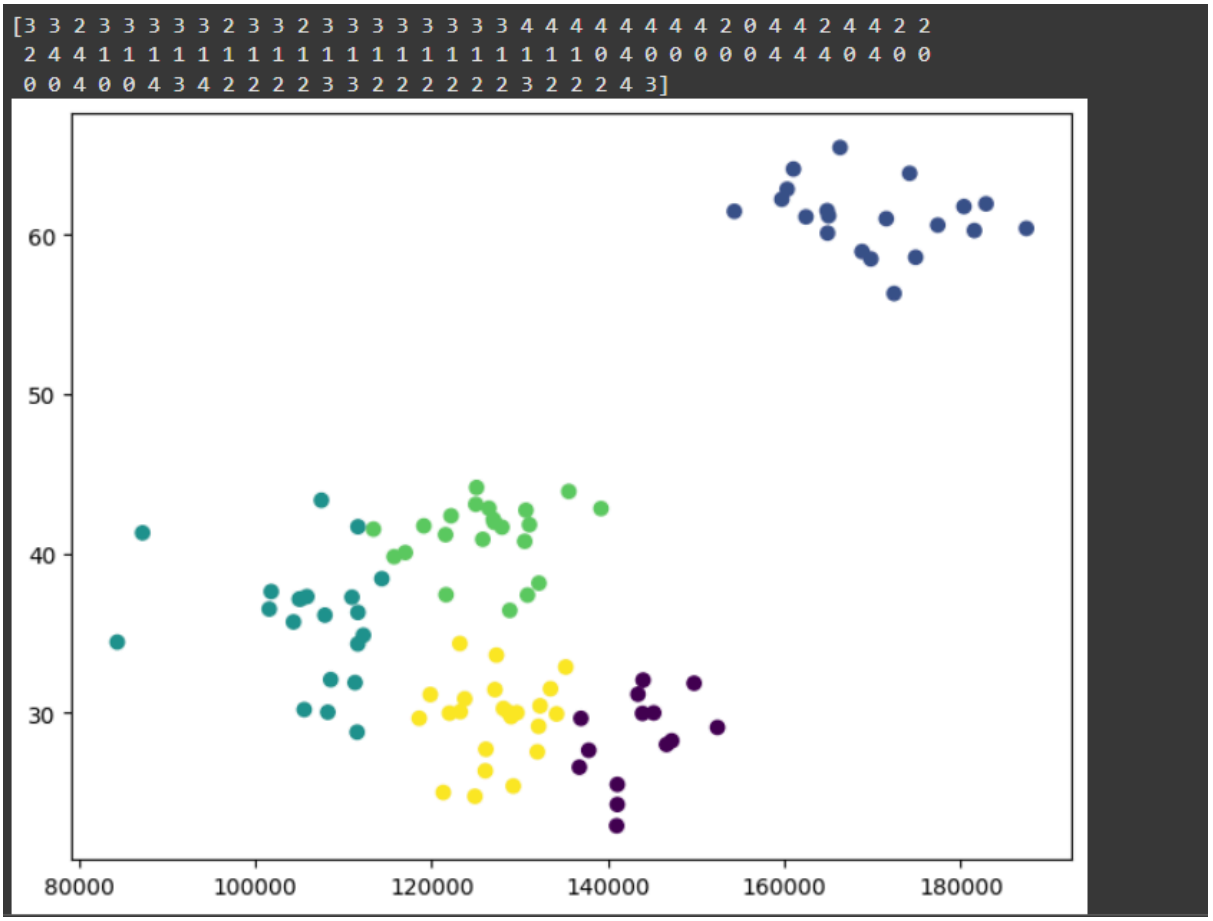
2 - Agrupar **notícias semelhantes** automaticamente.

3 - Separar **imagens com temas parecidos** em um sistema de organização automática de fotos.

Exemplo de Aplicação:

```
[8]  import random
     import numpy as np

     def createClusteredData(N, K):
         random.seed(10)
         pointsPerCluster = float(N) / K
         X = []
         for i in range (K):
             incomeCentroid = random.uniform(20000.0, 200000.0)
             ageCentroid = random.uniform(20.0, 70.0)
             for j in range(int(pointsPerCluster)):
                 X.append([np.random.normal(incomeCentroid, 10000.0), np.random.normal(ageCentroid, 2.0)])
         X = np.array(X)
         return X
```

```
[11] from sklearn.cluster import KMeans
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import scale

     data = createClusteredData(100, 5)

     model = KMeans(n_clusters=5)

     model = model.fit(scale(data))

     print(model.labels_)

     plt.figure(figsize=(8, 6))
     plt.scatter(data[:,0], data[:,1], c=model.labels_.astype(float))
     plt.show()
```



Entropia:

Entropia mede a imprevisibilidade e impureza do conjunto de dados. Quanto mais misturadas estão as classes em um conjunto de dados, maior é a entropia.

Vamos usar o exemplo do Email:

Suponha um conjunto com 10 e-mails:

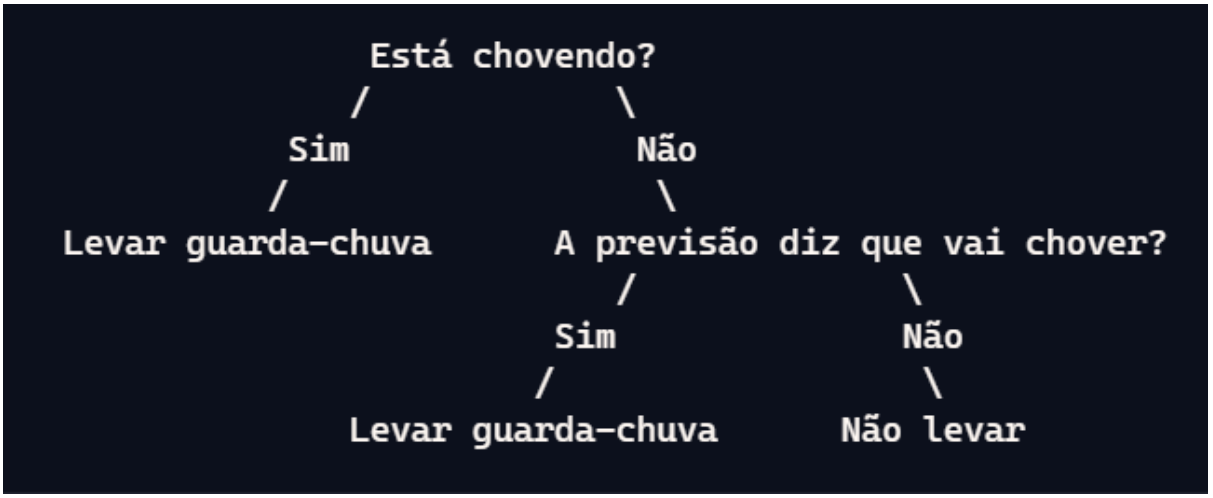
5 são **spam**
5 são **ham**
A **entropia** desse conjunto é máxima, porque está bem equilibrado (máxima incerteza para classificar).

Se fosse 10 são **spam**, 0 **ham**
Entropia = 0 (nenhuma incerteza — sabemos que tudo é spam)

Árvore de decisão:

Árvore de decisão é um modelo de aprendizado de máquina supervisionado composto por uma coleção de "perguntas", as folhas, organizadas hierarquicamente na forma de uma árvore. As perguntas geralmente são chamadas de condição.

Exemplo de uma:

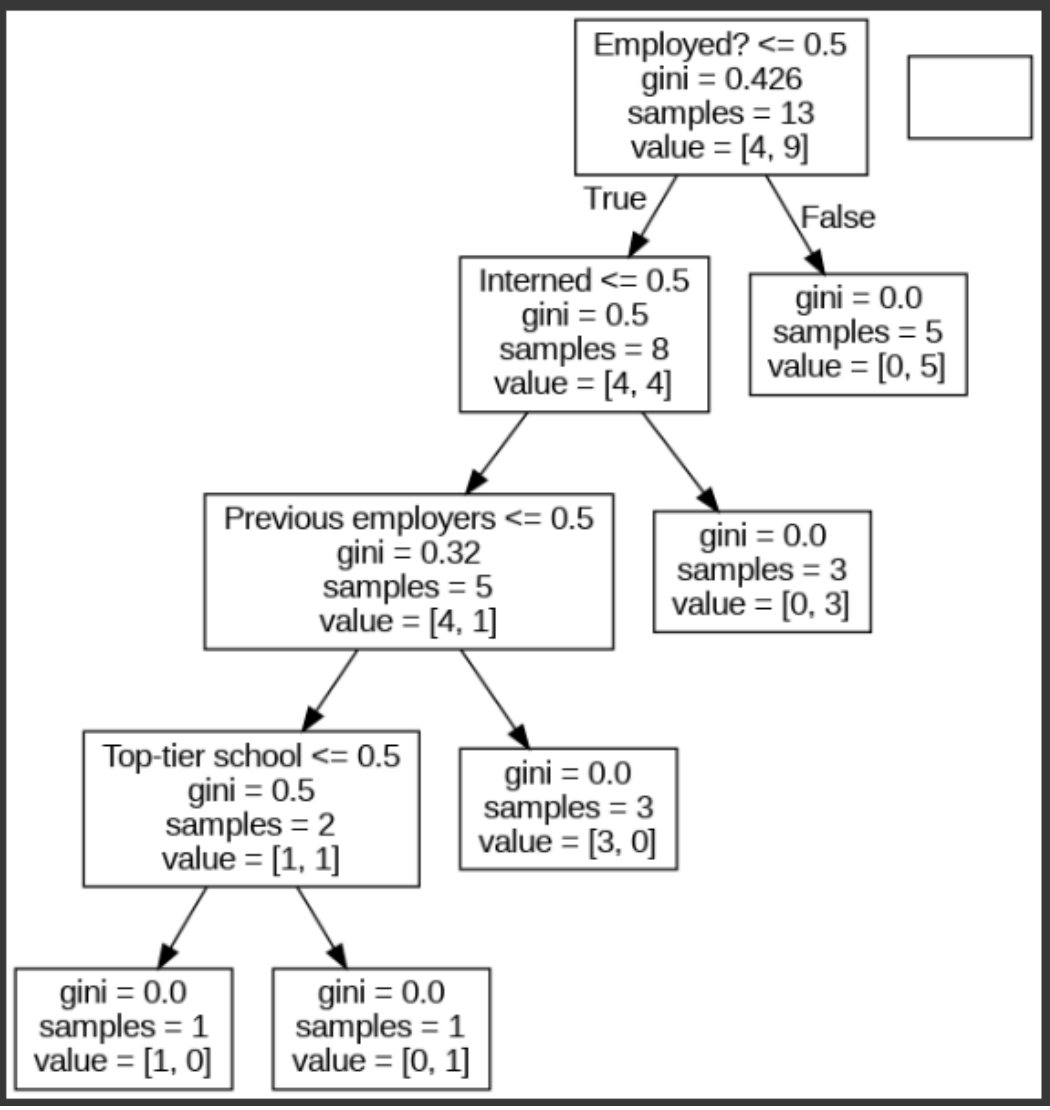


Nesse exemplo simples, é possível observar o funcionamento de uma **árvore de decisão**, onde cada condição gera uma ramificação com duas respostas possíveis: **sim (verdadeiro)** ou **não (falso)**. Esse caminho de decisões leva a uma **predição** no final. No contexto de aprendizado de máquina, as árvores podem ser de dois tipos:

- **Árvores de Classificação**, que predizem **categorias**, como "spam ou não spam".
- **Árvores de Regressão**, que predizem **valores numéricos contínuos**, como o preço de um carro.

Ambas funcionam com a mesma lógica de divisão por condições, mas o tipo de resposta final muda conforme o problema.

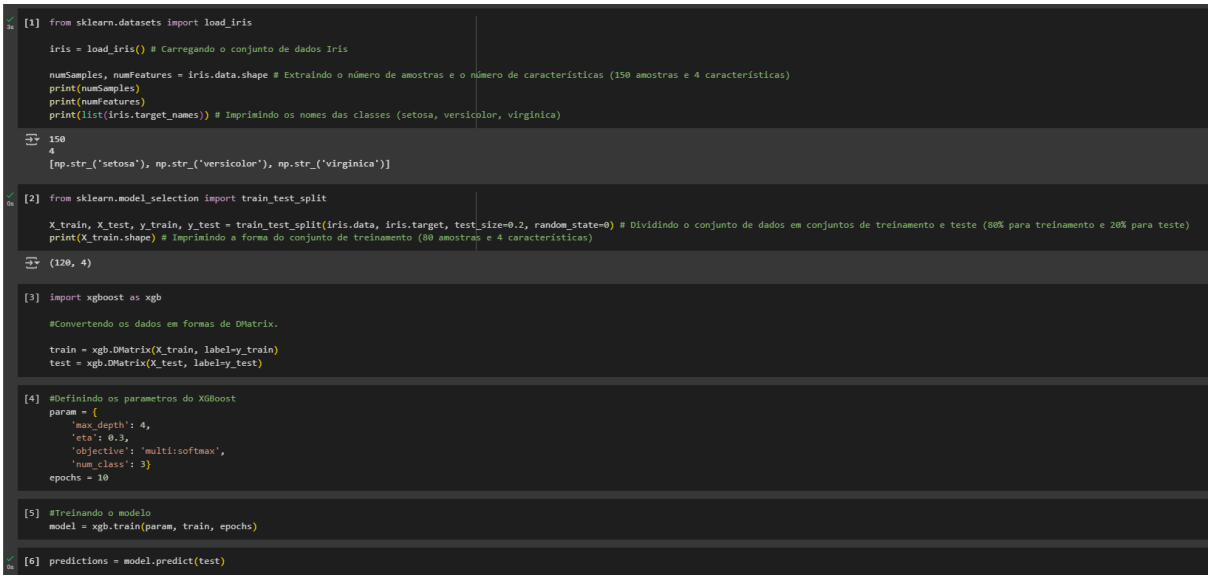
```
from IPython.display import Image
from io import StringIO
import pydotplus

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                     feature_names=features)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



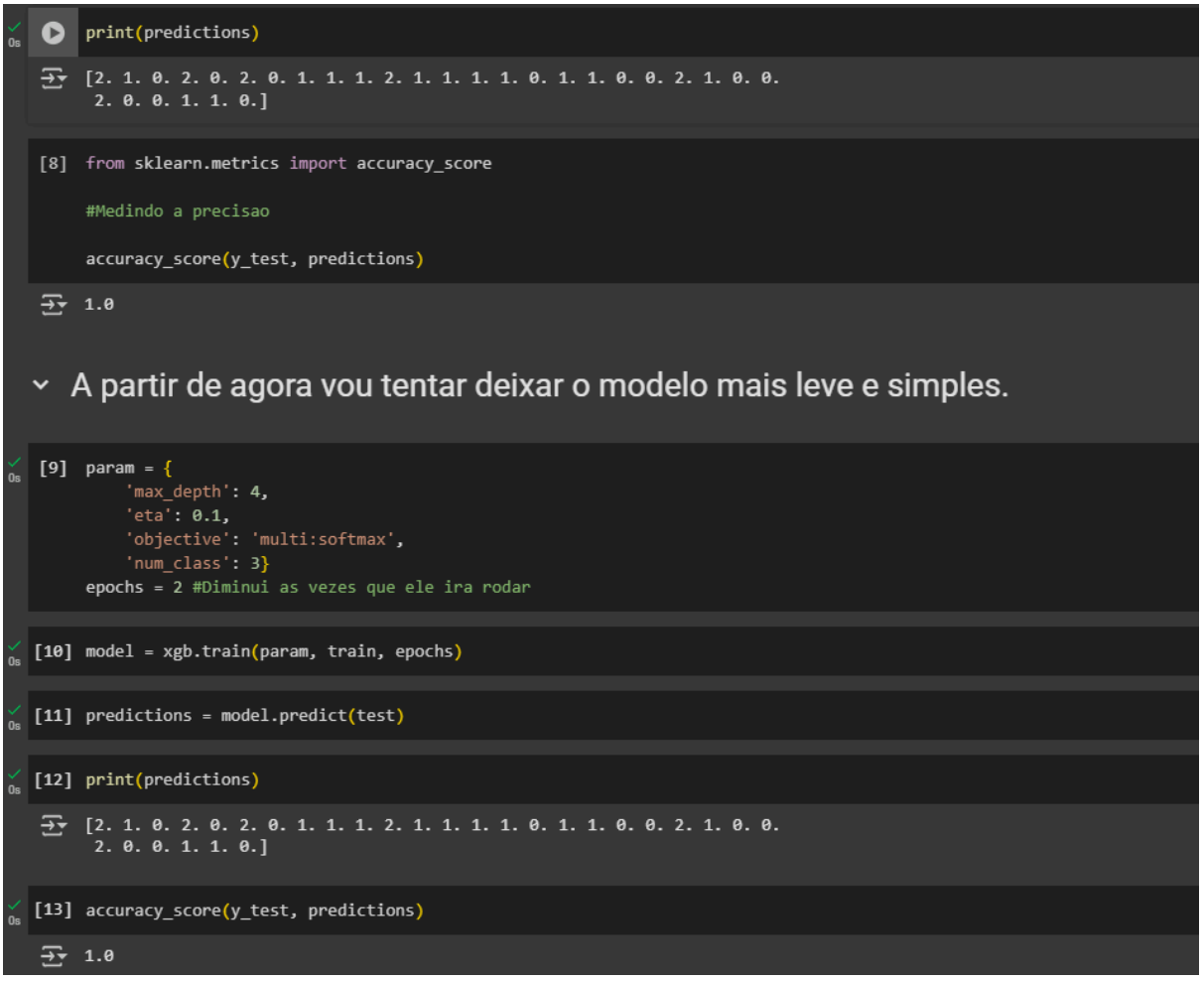
Ensemble Learning - XGBoost é uma técnica de aprendizado de máquina que combina múltiplos modelos (geralmente fracos) para criar um modelo mais robusto e preciso. A ideia central é que, ao unir várias previsões, os erros individuais tendem a se cancelar, melhorando o desempenho geral. Existem diferentes métodos de ensemble, como Bagging (ex: Random Forest), Boosting (ex: XGBoost, AdaBoost) e Stacking. Essas abordagens são especialmente eficazes para reduzir o overfitting e aumentar a capacidade de generalização

dos modelos. Uma coisa muito boa do XGBoost é que ele pode lidar com valores ausentes automaticamente.

Exemplo de uso:



```
[1] from sklearn.datasets import load_iris

    iris = load_iris() # Carregando o conjunto de dados Iris

    numSamples, numFeatures = iris.data.shape # Extraindo o número de amostras e o número de características (150 amostras e 4 características)
    print(numSamples)
    print(numFeatures)
    print(list(iris.target_names)) # Imprimindo os nomes das classes (setosa, versicolor, virginica)

    150
    4
    [np.str_('setosa'), np.str_('versicolor'), np.str_('virginica')]

[2] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=0) # Dividindo o conjunto de dados em conjuntos de treinamento e teste (80% para treinamento e 20% para teste)
    print(X_train.shape) # Imprimindo a forma do conjunto de treinamento (80 amostras e 4 características)

    (120, 4)

[3] import xgboost as xgb

    #Convertendo os dados em formas de DMatrix.

    train = xgb.DMatrix(X_train, label=y_train)
    test = xgb.DMatrix(X_test, label=y_test)

[4] #Definindo os parametros do XGBoost
    param = {
        'max_depth': 4,
        'eta': 0.3,
        'objective': 'multi:softmax',
        'num_class': 3}
    epochs = 10

[5] #Treinando o modelo
    model = xgb.train(param, train, epochs)

[6] predictions = model.predict(test)
```



```
print(predictions)

[2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 2. 1. 1. 1. 1. 0. 1. 1. 0. 0. 2. 1. 0. 0.
 2. 0. 0. 1. 1. 0.]

[8] from sklearn.metrics import accuracy_score

    #Medindo a precisao

    accuracy_score(y_test, predictions)

    1.0
```

A partir de agora vou tentar deixar o modelo mais leve e simples.

```
[9] param = {
        'max_depth': 4,
        'eta': 0.1,
        'objective': 'multi:softmax',
        'num_class': 3}
    epochs = 2 #Diminui as vezes que ele ira rodar

[10] model = xgb.train(param, train, epochs)

[11] predictions = model.predict(test)

[12] print(predictions)

     [2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 2. 1. 1. 1. 1. 0. 1. 1. 0. 0. 2. 1. 0. 0.
      2. 0. 0. 1. 1. 0.]

[13] accuracy_score(y_test, predictions)

     1.0
```

Máquinas de Vetores de Suporte (Support Vector Machines (SVM))

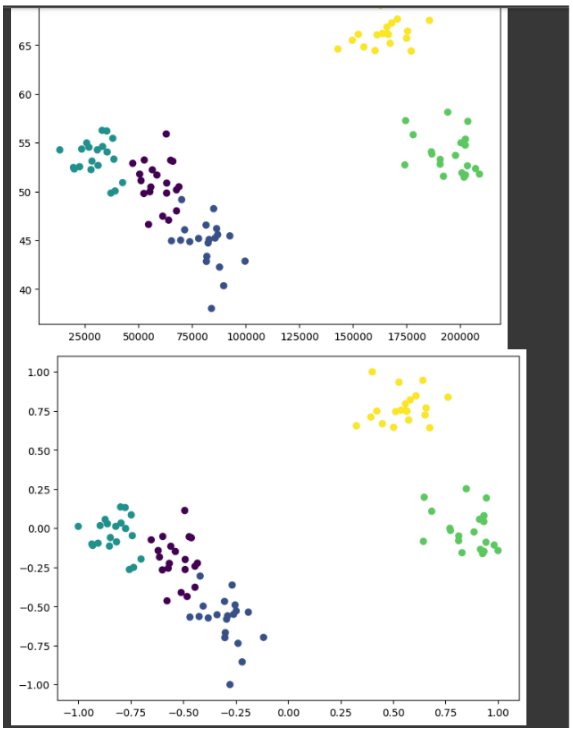
Máquinas de Vetores de Suporte (SVM) são algoritmos de aprendizado supervisionado usados principalmente para classificação e regressão. A ideia principal é encontrar um hiperplano ótimo que separa os dados em classes distintas da melhor forma possível. Esse hiperplano é definido com base nos vetores de suporte, que são os pontos mais próximos da fronteira de separação.

se quisermos separar e-mails entre spam e não spam, o SVM tentará traçar uma linha (ou plano) que divida os dois grupos com a maior margem possível, ou seja, mantendo uma boa distância dos pontos dos dois lados da linha.

SVMs também podem ser usados com dados não linearmente separáveis, graças ao uso de kernels, que transformam os dados para um espaço de dimensão mais alta, onde a separação é possível.

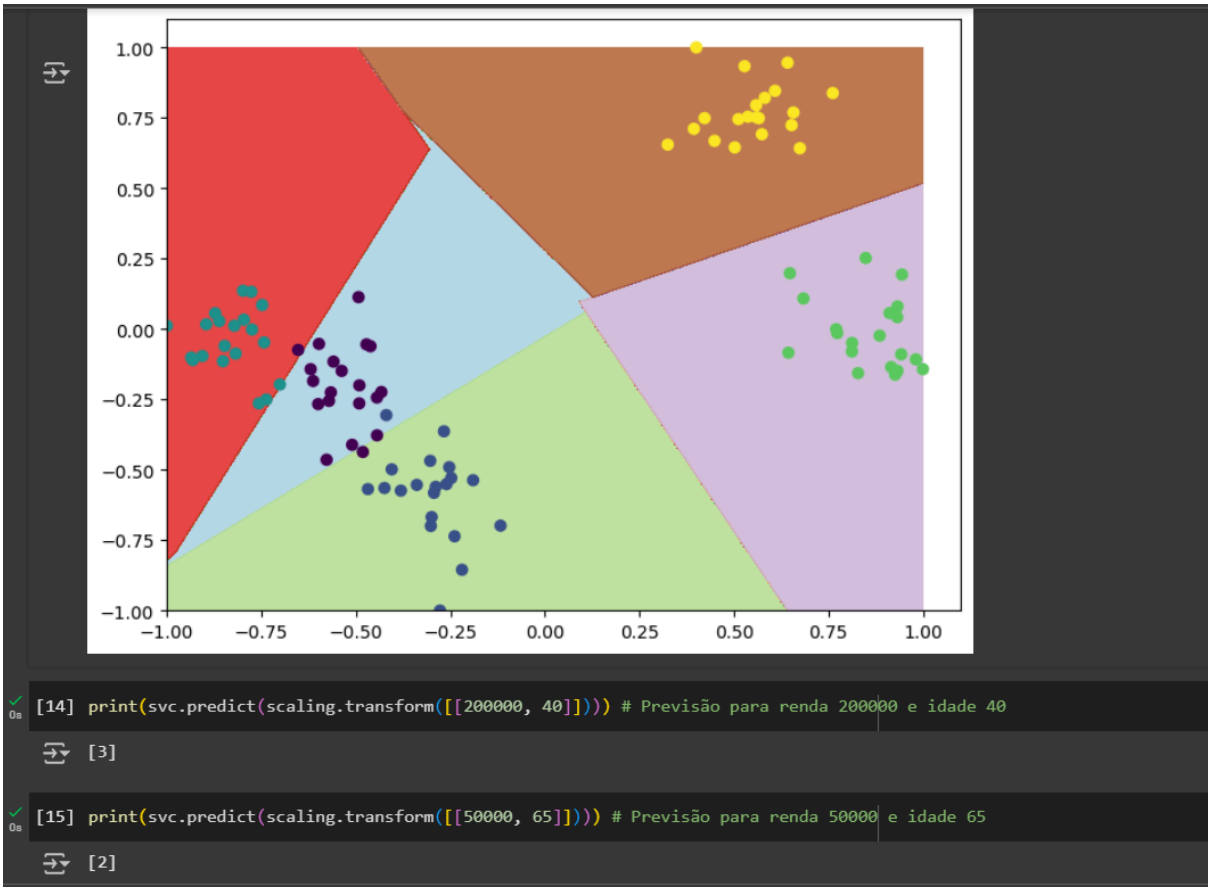
```python
import numpy as np  # Importa a biblioteca NumPy, usada para cálculos numéricos e manipulação de arrays

# Função para gerar dados artificiais agrupados (clusters)
def createClusteredData(N, k):
    np.random.seed(1234)  # Define a semente

    pointsPerCluster = float(N) / k  # Calcula quantos pontos haverá por cluster (N = total de pontos, k = número de grupos)
    X = []
    y = []

    for i in range(k):
        # Define aleatoriamente o centro de renda e idade desse cluster
        incomeCentroid = np.random.uniform(20000.0, 200000.0)  # renda entre 20 mil e 200 mil
        ageCentroid = np.random.uniform(20.0, 70.0)            # idade entre 20 e 70 anos

        for j in range(int(pointsPerCluster)):  #
            # Cria um ponto com distribuição normal em torno do centro
            X.append([
                np.random.normal(incomeCentroid, 10000.0),  # Renda com desvio padrão de 10 mil
                np.random.normal(ageCentroid, 2.0)          # Idade com desvio padrão de 2
            ])
            y.append(i)  # Rótulo

    X = np.array(X)  # Converte a lista de pontos para um array
    y = np.array(y)  # Converte os rótulos para um array

    return X, y  # Retorna os dados e seus respectivos rótulos
```

```python
[10] %matplotlib inline
     from pylab import *
     from sklearn.preprocessing import MinMaxScaler

     (X, y) = createClusteredData(100, 5) # gerando 100 pontos em 5 clusters

     # Plotando os dados gerados

     plt.figure(figsize=(8, 6))
     plt.scatter(X[:,0], X[:,1], c=y.astype(float))
     plt.show()

     scaling = MinMaxScaler(feature_range=(-1,1)).fit(X)
     X = scaling.transform(X)

     plt.figure(figsize=(8, 6))
     plt.scatter(X[:,0], X[:,1], c=y.astype(float))
     plt.show()
```




```
[12] from sklearn import svm, datasets

     C = 1.0 # Ajuste o parâmetro de regularização
     svc = svm.SVC(kernel='linear', C=C).fit(X, y) # Treina o modelo SVM com kernel linear
```

```
def plotPredictions(clf):
    # Cria uma grade densa de pontos para amostragem
    xx, yy = np.meshgrid(np.arange(-1, 1, .001),
                         np.arange(-1, 1, .001))

    # Converte para arrays Numpy
    npx = xx.ravel()
    npy = yy.ravel()

    # Converte para uma lista de pontos 2D (renda, idade)
    samplePoints = np.c_[npx, npy]

    # Gera rótulos previstos (números dos clusters) para cada ponto
    Z = clf.predict(samplePoints)

    plt.figure(figsize=(8, 6))
    Z = Z.reshape(xx.shape) # Redimensiona os resultados para corresponder à dimensão de xx
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8) # Desenha o contorno
    plt.scatter(X[:,0], X[:,1], c=y.astype(float)) # Desenha os pontos
    plt.show()

plotPredictions(svc)
```



Conclusão:

A partir desse card, eu fui capaz de compreender os principais conceitos, modelos e algoritmos de Aprendizado de Máquina, além de aplicá-los na prática utilizando python. Pode-se afirmar que python é extremamente útil para a aplicação de modelos preditivos, já que ele possui uma gama muito grande de bibliotecas.

Referências:

Curso disponibilizado pelo bootcamp -LAMIA:

Predictive Models

Machine Learning with Python