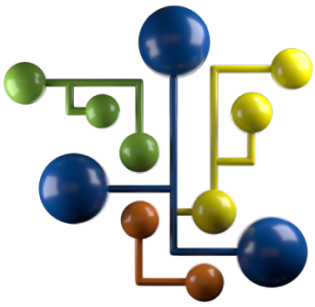

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Machine Learning do Zero – Algoritmo de Aprendizagem Não Supervisionada

Na última aula do curso de Introdução à Lógica de Programação faremos o mesmo exercício da Aula 14, mas com um diferente problema a ser resolvido e consequentemente um diferente algoritmo de Machine Learning.

Vem comigo!

Imagine que uma empresa esteja interessada em fazer previsões de vendas para 2020. Como resolvemos isso? Obtemos as vendas realizadas (por exemplo) em 2017, 2018 e 2019. Esses são dados históricos e temos variáveis de entrada (características de uma venda, como informação do cliente ou produto) e uma variável de saída que representa o valor de cada venda. Esse é tipicamente um problema que pode ser resolvido construindo um modelo de Machine Learning para aprendizagem supervisionada, pois temos dados de entrada e saída.

Agora imagine que essa mesma empresa queira criar um programa de fidelidade para seus clientes atuais com diferentes níveis de descontos e benefícios. Clientes mais antigos teriam mais benefícios de acordo com os valores gastos em compras e por diferentes faixas. Clientes mais novos teriam benefícios diferentes e assim por diante.

Mas como a empresa cria esses segmentos de clientes? Quem não conhece Machine Learning vai tentar fazer isso manualmente. Alguns vão dizer que resolvem isso com uma query SQL. Mas quais seriam os grupos para segmentar os clientes? Quantos grupos? Se tivermos milhões de registros como encontramos padrões? Manualmente?

O fato é que podemos agrupar os clientes por características semelhantes de forma automática. Perceba que não sabemos de antemão quais seriam os grupos de clientes. Isso é o que queremos descobrir! Logo, não temos a variável de saída, somente as variáveis de entrada. Se é o que temos, trabalhamos com o que temos.

Entregaremos as variáveis de entrada a um algoritmo de Machine Learning e ele deverá ser capaz de encontrar os padrões nos dados e agrupá-los por similaridade. Como não sabemos quantos grupos podemos encontrar, definiremos um número arbitrário definido por K. Podemos testar o algoritmo com K igual a 3, 4 ou 10 e investigar em quantos grupos os dados melhor se ajustam.

Pois bem! Acabei de descrever para você a aprendizagem supervisionada, especificamente o algoritmo K-Means.

Em geral esse tipo de aprendizagem tem algumas complicações adicionais, pois não temos a variável de saída para avaliar a performance do nosso modelo! Por outro lado, o algoritmo vai nos ajudar a encontrar padrões nos dados de forma automática.

E adivinhe o que está por trás de tudo isso? Acertou se você respondeu: Matemática.

Aqui você encontra um paper com o pseudo-código do K-means:

https://arxiv.org/pdf/1002.2425.pdf

Vamos implementá-los a partir do zero em Python!

Essencialmente o que faz o K-means é calcular a distância entre os pontos de dados. Definimos um dos pontos de dados como centroide para cada cluster (inicialmente escolhemos um dos pontos de dados de forma aleatória) e vamos calculando a distância Euclidiana de cada ponto de dado para o centroide e assim criando grupos (ou clusters) com pontos de dados que contenham distâncias similares para o centroide. O centroide pode mudar à medida que vamos calculando as similaridades. Com isso, o algoritmo é capaz de agrupar os dados por similaridade. E isso funciona muito bem na maioria dos casos.

Já encontramos a solução para um dos problemas e usaremos o K-means para o agrupamento e segmentação de clientes. Mas agora temos outro problema (isso é o que nós fazemos: usamos ciência para resolver problemas).

Vamos trabalhar com K = 3 para encontrar 3 segmentos de clientes. Poderíamos trabalhar com 4, 5 ou mesmo 10 clusters. Usar 1 ou 2 não faria muito sentido!

Leia com bastante atenção! Isso é importante!

Mas temos (ou podemos ter) diversas variáveis de entrada, certo? Isso significa que temos dados com alta dimensionalidade. Se quisermos apresentar graficamente o resultado do agrupamento para os tomadores de decisão que não possuem conhecimento técnico em Machine Learning (e nem precisam ter), como faríamos isso? Um gráfico de 4 ou 5 dimensões é incompreensível e ainda ter que representar os clusters, deixaria tudo muito mais complicado.

E se reduzíssemos a dimensionalidade dos dados para que pudéssemos criar um gráfico de duas dimensões independente da quantidade de variáveis em nosso dataset? Lindo, não? Podemos usar PCA (Análise de Componentes Principais), uma das principais técnicas para redução de dimensionalidade. Perfeito. Mais um problema resolvido.

Não, espere!

Tem outro problema (isso é o que nós fazemos: resolvemos problemas). Para desenvolver o PCA a partir do zero também temos que montar toda a estrutura matemática por trás do PCA, o que implica calcular as matrizes de co-variância e correlação, a fim de encontrar autovalores e autovetores (o que você acha que fazemos no curso de Matemática Para Machine Learning? Estudamos tudo isso, claro!).

Podemos aplicar o PCA com uma linha de código usando o Scikit-Learn. Mas se a proposta é programar tudo na unha, então vamos à diversão.

Abaixo você encontra o programa completo, como sempre comentado linha a linha. Alguns conceitos matemáticos podem não ficar claros se você não souber os fundamentos. Faça uma pesquisa adicional se necessário (embora tudo isso seja estudados nos cursos da DSA). Algumas operações matemáticas não são triviais, mas toda a programação é relativamente simples.

Cuidado com a identação e use o Jupyter Notebook.

```
# Pacotes
import math
import numpy as np
import matplotlib.cm as cmx
import matplotlib.pyplot as plt
import matplotlib.colors as colors


##### Funções Auxiliares #####

# Função para normalizar os dados
def normaliza_dados(X, axis = -1, order = 2):
--l2 = np.atleast_1d(np.linalg.norm(X, order, axis))
--l2[l2 == 0] = 1
--return X / np.expand_dims(l2, axis)

# Função para calcular a distância euclidiana entre 2 vetores
def calcula_distancia_euclidiana(x1, x2):
--distance = 0
--for i in range(len(x1)):
----distance += pow((x1[i] - x2[i]), 2)
--return math.sqrt(distance)


##### Algoritmo K-means #####


# Classe para o algoritmo K-means
# Aprendizagem não supervisionada
class KMeans():

--# Construtor da classe
--def __init__(self, k=3, max_iterations=500):
----self.k = k
```

```
----self.max_iterations = max_iterations

--# Inicializa os centróides com k amostras randômicas de x
--def _init_random_centroids(self, X):
----n_samples, n_features = np.shape(X)
----centroids = np.zeros((self.k, n_features))
----for i in range(self.k):
------centroid = X[np.random.choice(range(n_samples))]
------centroids[i] = centroid
----return centroids

--# Retorna o índice mais próximo do centróide da amostra
--def _closest_centroid(self, sample, centroids):
----closest_i = 0
----closest_dist = float('inf')
----for i, centroid in enumerate(centroids):
------distance = calcula_distancia_euclidiana(sample, centroid)
------if distance < closest_dist:
--------closest_i = i
--------closest_dist = distance
----return closest_i

--# Associa as amostras de dados aos centróides mais próximos para criar os clusters (grupos)
--def _create_clusters(self, centroids, X):
----n_samples = np.shape(X)[0]
----clusters = [[] for _ in range(self.k)]
----for sample_i, sample in enumerate(X):
------centroid_i = self._closest_centroid(sample, centroids)
------clusters[centroid_i].append(sample_i)
----return clusters

--# Calcula novos centróides como a média das amostras em cada cluster
--def _calculate_centroids(self, clusters, X):
----n_features = np.shape(X)[1]
----centroids = np.zeros((self.k, n_features))
----for i, cluster in enumerate(clusters):
------centroid = np.mean(X[cluster], axis=0)
------centroids[i] = centroid
----return centroids

--# Classifica as amostras com o índice dos seus clusters
--def _get_cluster_labels(self, clusters, X):
----y_pred = np.zeros(np.shape(X)[0])
----for cluster_i, cluster in enumerate(clusters):
```

```
------for sample_i in cluster:
--------y_pred[sample_i] = cluster_i
----return y_pred

--# Faz a previsão de cada cluster e retorna os índices dos clusters
--def predict(self, X):

----# Inicializa centróides como k amostras aleatórias de X
----centroids = self._init_random_centroids(X)

----# Iterar até convergência ou para iterações máximas
----for _ in range(self.max_iterations):

------# Atribuir amostras aos centróides mais próximos (criar clusters)
------clusters = self._create_clusters(centroids, X)

------# Salvar os centróides atuais para verificação de convergência
------prev_centroids = centroids

------# Calcular novos centróides a partir dos clusters
------centroids = self._calculate_centroids(clusters, X)

------# Se nenhum centróide mudou => convergência
------diff = centroids - prev_centroids
------if not diff.any():
--------break

----return self._get_cluster_labels(clusters, X)


##### Funções Auxiliares Para o PCA #####

# Calcula a matriz de co-variância
def calculate_covariance_matrix(X, Y=None):
--if Y is None:
----Y = X
--n_samples = np.shape(X)[0]
--covariance_matrix = (1 / (n_samples-1)) * (X - X.mean(axis=0)).T.dot(Y - Y.mean(axis=0))

--return np.array(covariance_matrix, dtype=float)

# Calcula a matriz de correlação
def calculate_correlation_matrix(X, Y=None):
--if Y is None:
```

```
----Y = X
--n_samples = np.shape(X)[0]
--covariance = (1 / n_samples) * (X - X.mean(0)).T.dot(Y - Y.mean(0))
--std_dev_X = np.expand_dims(calculate_std_dev(X), 1)
--std_dev_y = np.expand_dims(calculate_std_dev(Y), 1)
--correlation_matrix = np.divide(covariance, std_dev_X.dot(std_dev_y.T))

--return np.array(correlation_matrix, dtype=float)


##### Classe Para o Plot dos Clusters em 2D #####


# Classe para criar o plot
class Plot():

--# Construtor da classe
--def __init__(self):
----self.cmap = plt.get_cmap('viridis')

--# Função para transformar os dados
--def _transform(self, X, dim):
----covariance = calculate_covariance_matrix(X)
----eigenvalues, eigenvectors = np.linalg.eig(covariance)
----idx = eigenvalues.argsort()[::-1]
----eigenvalues = eigenvalues[idx][:dim]
----eigenvectors = np.atleast_1d(eigenvectors[:, idx])[:, :dim]
----X_transformed = X.dot(eigenvectors)

----return X_transformed

--# Plot do dataset X e seus correspondentes labels y em 2D usando PCA.
--def plot_in_2d(self, X, y=None, title=None, accuracy=None, legend_labels=None):
----X_transformed = self._transform(X, dim=2)
----x1 = X_transformed[:, 0]
----x2 = X_transformed[:, 1]
----class_distr = []

----y = np.array(y).astype(int)

----colors = [self.cmap(i) for i in np.linspace(0, 1, len(np.unique(y)))]

----# Plot de diferentes distribuições de classe
----for i, l in enumerate(np.unique(y)):
```

```
------ _x1 = x1[y == l]
------_x2 = x2[y == l]
------_y = y[y == l]
------class_distr.append(plt.scatter(_x1, _x2, color=colors[i]))

----# Plot da legenda
----if not legend_labels is None:
------plt.legend(class_distr, legend_labels, loc=1)

----# Plot do título
----if title:
------if accuracy:
--------perc = 100 * accuracy
--------plt.suptitle(title)
--------plt.title("Acurácia: %.1f%%" % perc, fontsize=10)
------else:
--------plt.title(title)

----# Axis labels
----plt.xlabel('Componente Principal 1')
----plt.ylabel('Componente Principal 2')

----plt.show()

##### Execução do Programa #####

from sklearn.datasets import make_blobs

# Função para execução principal do programa
def main():

--# Carrega o dataset
--X, y = make_blobs()

--# Executa o algoritmo para k = 3
--clf = KMeans(k = 3)
--y_pred = clf.predict(X)

--# Projeta os dados com 2 componentes principais
--p = Plot()
--p.plot_in_2d(X, y_pred, title = "Segmentação de Clientes com K-Means")


if __name__ == "__main__":
```

--main()

Ao executar o programa encontramos o gráfico abaixo, com todos os dados e entrada separados em 3 grupos (K = 3).

##

Com isso concluímos este curso de Introdução à Lógica de Programação! Esse é um assunto incrível, pois nos leva para a essência do processo de solução de problemas, desenvolvendo nossa capacidade de raciocínio!

Neste sábado publicaremos todos os detalhes sobre a avaliação final e como acessar todo o curso de forma única, o que estará disponível para os alunos matriculados nas Formações DSA.

Muito Obrigado!

#aula15