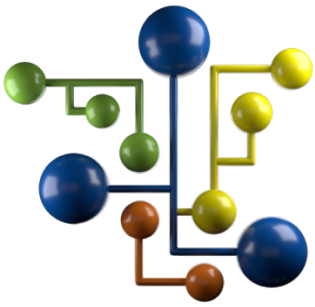

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Machine Learning do Zero – Algoritmo de Aprendizagem Supervisionada

Nas duas últimas aulas do curso de Introdução à Lógica de Programação eu não vou aliviar!!! Se quiser realmente aprender você terá que sair da zona de conforto e se dedicar! Mas, é claro, a escolha é sua!

O que eu vou mostrar nas duas últimas aulas não deveria fazer parte de um curso de introdução, mas tudo será explicado tão passo a passo, mas tão passo a passo (padrão DSA), que tenho certeza que a maioria dos alunos conseguirá acompanhar. Dúvidas são normais e fazem parte do processo de aprendizagem! Siga em frente, mesmo se algo não ficar claro em um primeiro momento. A explicação pode estar na próxima curva!

Vamos praticar algo parecido ao que é feito nos Labs do curso de Matemática Para Machine Learning (usando exemplos diferentes, claro)! Não tenho como explicar todos os conceitos matemáticos aqui, mas não é o foco. Queremos olhar para a programação!

De qualquer forma, recomendo a leitura dos capítulos sobre gradiente descendente do Deep Learning Book, para aqueles que não estejam fazendo os cursos de Machine Learning ou Deep Learning aqui na DSA. Também vou considerar que você fez pelo menos nosso curso gratuito de Introdução à Ciência de Dados 2.0 e conhece alguns termos básicos de Machine Learning. Os 5 primeiros capítulos do curso de Python Fundamentos também serão úteis.

Com relação ao que expliquei na Aula 13, vamos trabalhar nas duas primeiras áreas. Que comecem os jogos, ou melhor, as aulas! 😄

Área 1

Imagine que você esteja participando de um projeto de Data Science e um dos requerimentos é desenvolver um algoritmo de Machine Learning a partir do zero. Não é uma opção usar os algoritmos prontos do Scikit-Learn em Python ou Caret em R. Embora esse não seja um cenário comum, é um cenário possível. Por onde você começa?

O primeiro passo é definir se estamos falando de aprendizagem supervisionada ou não supervisionada (ou ainda aprendizagem por reforço, se for um projeto mais avançado). Vamos considerar aqui aprendizagem supervisionada (temos dados históricos de entrada e de saída).

O segundo passo é definir se iremos prever uma classe (Classificação) ou um valor numérico (Regressão). Para este exemplo vamos considerar Classificação.

O terceiro passo é escolher o algoritmo a ser usado. Temos dúzias de opções e podemos testar diversos algoritmos ao mesmo tempo conforme ensinamos na Formação Cientista de Dados. Mas um bom algoritmo para começar é a Regressão Logística (não se deixe enganar pelo nome, este é um algoritmo de Classificação).

Com essas definições, podemos começar a construir nosso algoritmo a partir do zero.

Mas espere! Cadê o pseudo-código? Onde estão os passos do algoritmo para que possamos programar em Python? E o problema de negócio a ser resolvido!

Boas perguntas! Você está atento!

Ao construir um algoritmo a partir do zero, esse será seu primeiro grande desafio! Buscar a especificação do algoritmo. Nem sempre será fácil encontrar e muita pesquisa deverá ser feita. Como a Regressão Logística é um algoritmo amplamente conhecido, conseguimos encontrar detalhes até na Wikipedia:

https://en.wikipedia.org/wiki/Logistic_regression

Ao encontrar a especificação do algoritmo você deve ler e compreender as fórmulas e expressões matemáticas, que envolvem Cálculo e Probabilidade. É aqui onde esse conhecimento é importante e por isso criamos a Formação Análise Estatística Para Cientistas de Dados.

Por fim, vamos definir um problema de negócio. Sem isso, estaríamos vagando sem rumo. Precisamos ter uma meta clara para saber onde chegar!

Como o foco aqui é na programação, vamos manter as coisas simples e usar o dataset iris. O dataset possui dados de flores para 3 categorias e nosso modelo deverá receber dados das flores e prever a qual das 3 categorias a flor pertence. Esse é um problema de Classificação Multiclasse. Aqui o dataset:

https://archive.ics.uci.edu/ml/datasets/Iris

Ufa! O trabalho de programação ainda não começou e já estou exausto!! Mas vamos em frente! Agora eu compreendo porque o Daniel diz que um Cientista de Dados deve ser muito bem remunerado. 😄

A melhor forma de criar o algoritmo em Python é usando uma Classe, pois assim poderemos definir atributos e métodos e definir o comportamento do algoritmo.

O algoritmo está prontinho no código abaixo. Leia os comentários e as linhas de código com muita atenção. São operações relativamente simples, pois a Regressão Logística é um algoritmo básico.

```
# Pacotes para computação e matemática
import numpy as np
import math
```

```
# Classe para a função de Ativação Sigmoide
class Sigmoide():
--def __call__(self, x):
----return 1 / (1 + np.exp(-x))

# Algoritmo de Machine Learning - Regressão Logística
class RegressaoLogistica():

--# Construtor da Classe - define os atributos
--def __init__(self, learning_rate = .1, gradient_descent = True):
----self.param = None
----self.learning_rate = learning_rate
----self.gradient_descent = gradient_descent
----self.sigmoid = Sigmoide()

--# Inicializa os parâmetros
--def _inicializa_parametros(self, X):
----n_features = np.shape(X)[1]
----limit = 1 / math.sqrt(n_features)
----self.param = np.random.uniform(-limit, limit, (n_features,))

--# Converte uma matriz x para diagonal a fim de realizar os cálculos
--def make_diagonal(x):
----m = np.zeros((len(x), len(x)))
----for i in range(len(m[0])):
------m[i, i] = x[i]
----return m

--# Função para o treinamento
--# Aqui está o coração do algoritmo, suas operações matemáticas
--# Usamos o gradiente descendente a fim de encontrar o melhor valor dos
--# coeficientes, aquilo que o modelo aprende no treinamento
--def treinamento(self, X, y, n_iterations = 4000):
----self._inicializa_parametros(X)
----for i in range(n_iterations):
------y_pred = self.sigmoid(X.dot(self.param))
------if self.gradient_descent:
--------self.param -= self.learning_rate * -(y - y_pred).dot(X)
------else:
--------diag_gradient = make_diagonal(self.sigmoid.gradient(X.dot(self.param)))
--------self.param =
np.linalg.pinv(X.T.dot(diag_gradient).dot(X)).dot(X.T).dot(diag_gradient.dot(X).dot(self.param)
+ y - y_pred)
```

```
--# Método para previsão com o modelo
--def previsao(self, X):
----y_pred = np.round(self.sigmoid(X.dot(self.param))).astype(int)
----return y_pred
```

Você acreditaria se eu contasse que todas as linhas acima poderiam ser substituídas por isso:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X, y)
clf.predict(X[:2, :])
```

É um grande equívoco começar a aprender Machine Learning desenvolvendo a partir do zero. Usando o algoritmo pronto o aluno aprende a "Big Picture", aprende o processo! Depois será muito (realmente muito) mais fácil ir aprendendo os detalhes.

Por essa razão recomendamos fazer a FCD antes da FAECD, conforme explicamos aqui:

http://datascienceacademy.com.br/blog/qual-trilha-de-aprendizagem-devo-escolher-na-data-science-academy/

Área 2

Achou que o trabalho tinha terminado? Você está de brincadeira, não é? 😄

Depois de construir o algoritmo, vem o trabalho "braçal". Preparar os dados para que possamos treinar o algoritmo. Todo o código necessário está logo abaixo.

O que fazemos essencialmente é:

1- Carregar os dados
2- Normalizar os dados de entrada (colocá-los na mesma escala)
3- Carregar os dados de saída
4- Dividir os dados em treino e teste
5- Definir o modelo
6- Treinar o algoritmo e criar o modelo
7- Usar o modelo treinado para fazer previsões
8- Calcular a acurácia comparando valores previstos com valores observados

E então, finalizamos o primeiro round! Se a acurácia não estiver ok, temos que voltar, fazer modificações e seguir nesse ciclo até alcançar o objetivo, que em geral foi definido no começo do projeto.

```
# Pacotes
from sklearn.datasets import load_iris
import numpy as np

# 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 randomizar os dados
def randomiza_dados(X, y, seed=None):
--if seed:
----np.random.seed(seed)
--idx = np.arange(X.shape[0])
--np.random.shuffle(idx)
--return X[idx], y[idx]

# Função para calcular a acurácia
def calcula_acuracia(y_true, y_pred):
--accuracy = np.sum(y_true == y_pred, axis=0) / len(y_true)
--return accuracy

# Função para dividir os dados em treino e teste
def train_test_split(X, y, test_size=0.5, shuffle=True, seed=None):
--if shuffle:
----X, y = randomiza_dados(X, y, seed)
--split_i = len(y) - int(len(y) // (1 / test_size))
--X_train, X_test = X[:split_i], X[split_i:]
--y_train, y_test = y[:split_i], y[split_i:]

--return X_train, X_test, y_train, y_test

# Função para execução do programa
def main():

--# Carregar os dados
--data = load_iris()

--# Normalizar os dados de entrada
```

```
--X = normaliza_dados(data.data[data.target != 0])

--# Carregar os dados de saída
--y = data.target[data.target != 0]
--y[y == 1] = 0
--y[y == 2] = 1

--# Dividir os dados em treino e teste
--X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, seed=1)

--# Definir o modelo
--modelo = RegressaoLogistica(gradient_descent=True)

--# Treinar o algoritmo e criar o modelo
--modelo.treinamento(X_train, y_train)

--# Usar o modelo treinado para fazer previsões
--y_pred = modelo.previsao(X_test)

--# Calcular a acurácia comparando valores previstos com valores observados
--acuracia = calcula_acuracia(y_test, y_pred)

--# Imprime a acurácia do modelo
--print ("Acurácia do Modelo:", acuracia)


# Executa o programa
if __name__ == "__main__":
   main()
```

```
Acurácia do Modelo: 0.93
```

Acabamos de construir e treinar um algoritmo de Machine Learning a partir do zero. E você pode até não concordar, mas não há nada de programação avançada. Nada! Criamos uma sequência de operações com loops, condicionais, atribuição de variáveis, operações matemáticas, chamadas a funções, instanciação de classes e indexação. Tudo isso é programação e ensinamos nos 5 primeiros capítulos do curso gratuito de Python Fundamentos.

O objetivo desta aula foi mostrar a você que na grande maioria das vezes o conhecimento de programação em Ciência de Dados é diferente do conhecimento em desenvolvimento de software. O que criamos aqui é uma sequência lógica de operações para representar um algoritmo de aprendizagem, com todo o tratamento que os dados precisam para o treinamento do algoritmo!

Área 3

Lembra na Aula 13 quando mencionei a Área 3? Com o modelo treinado, teríamos que construir uma aplicação para consumir o modelo e fazê-lo realizar as previsões para as quais ele foi treinado. Isso, em geral, não é trabalho do Cientista de Dados, mas sim do Engenheiro de Dados ou do Engenheiro de Machine Learning. Se for necessário o desenvolvimento de uma aplicação completa, um Engenheiro de Software será necessário.

O trabalho do Engenheiro de Dados será prover toda a infraestrutura para que os dados possam ser coletados, armazenados, tratados e então possam alimentar o modelo já treinado. Isso tudo com segurança, escalabilidade e performance. O Engenheiro de Dados não terá que desenvolver nenhum software, mas poderá usar programação para criar um fluxo de operação para os dados. Microsserviços, Docker, Cluster Hadoop/Spark, Apache NiFi, Apache Kafka, etc... são ferramentas usadas aqui.

O Engenheiro de Machine Learning é um perfil híbrido, mas com forte especialização em Machine Learning para fazer alterações no modelo, retreiná-lo, modificá-lo, etc...

O Engenheiro de Software vai empregar as boas práticas de programação e desenvolver uma aplicação que possa usar o modelo. A aplicação teria que receber os dados, alimentar o modelo, obter o resultado e apresentar ao usuário final, resumidamente falando, claro!

Ou então o Cientista de Dados Full-Stack, capaz de trabalhar nas 3 áreas!

Ufa! Aula 14 concluída!

#aula14