**Semana 03 – Exercício em Aula**

PRO6006 - Métodos de Otimização Não Linear com aplicações em aprendizado de máquina

Nathan Sampaio Santos – 8988661

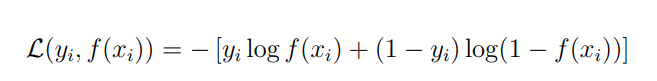
IMPLEMENTAÇÃO REGRESSÃO LOGÍSTICA

O método principal desta classe é chamado *fit*. Nele, os parâmetros w e b da função *f(x)* são calculados até ser atingido a convergência.

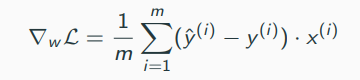
A mathematical equation with numbers and symbols

AI-generated content may be incorrect.

Primeiramente, é feita a previsão de probabilidade binária pela função sigmoide f(x). Após isso, é definida uma variável custo, de acordo com a seguinte fórmula:



Também são calculados os fatores do vetor gradiente desta função de custo, de acordo com as seguintes fórmulas:

A mathematical equation with numbers and symbols

AI-generated content may be incorrect.

De posse deste vetor e do passo de aprendizado definido como um parâmetro, a função calcula novos valores de *w* e *b* até serem atingidos os critérios de convergência, que são: o número de interações ou a norma do vetor gradiente ser menor do que o parâmetro de tolerância.

    def fit(self, X, y):

        n\_samples, n\_features = X.shape

        grad\_norm = np.inf

        self.w    = np.zeros(n\_features)

        self.b    = 0

        self.cost\_history = []

        while self.n\_iterations <= self.n\_iterations\_max and \

              grad\_norm > self.tol:

            self.n\_iterations += 1

            y\_predicted = self.predict\_proba(X)

            epsilon = 1e-9

            cost = (-1 / n\_samples) \* np.sum(y \* np.log(y\_predicted + epsilon) + (1 - y) \* np.log(1 - y\_predicted + epsilon))

            self.cost\_history.append(cost)

            dw = (1 / n\_samples) \* np.dot(X.T, (y\_predicted - y))

            db = (1 / n\_samples) \* np.sum(y\_predicted - y)

            grad\_norm = np.linalg.norm(np.concatenate((dw, np.array([db]))))

            self.w -= self.learning\_rate \* dw

            self.b -= self.learning\_rate \* db

USO REGRESSÃO LOGÍSTICA

Para testar o uso dessa função, utilizamos uma base de diagnóstico de câncer, a qual possui 30 parâmetros listados a seguir que caracterizam quantitativamente um tumor. Além desses parâmetros, também é obtido a coluna *target*, binária, que indica a confirmação ou não de um câncer.

* *mean radius*
* *mean texture*
* *mean perimeter*
* *mean area*
* *mean smoothness*
* *mean compactness*
* *mean concavity*
* *mean concave points*
* *mean symmetry*
* *mean fractal dimension*
* *radius error*
* *texture error*
* *perimeter error*
* *area error*
* *smoothness error*
* *compactness error*
* *concavity error*
* *concave points error*
* *symmetry error*
* *fractal dimension error*
* *worst radius*
* *worst texture*
* *worst perimeter*
* *worst area*
* *worst smoothness*
* *worst compactness*
* *worst concavity*
* *worst concave points*
* *worst symmetry*
* *worst fractal dimension*

Todos esses dados podem ser encontrados publicamente na base de dados UCI ou dentro da biblioteca sklearn do Python.

from LogisticRegression      import LogisticRegression

from sklearn.datasets        import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing   import StandardScaler

from sklearn.metrics         import accuracy\_score

import matplotlib.pyplot as plt

cancer\_data = load\_breast\_cancer()

X = cancer\_data.data

y = cancer\_data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)

scaler         = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled  = scaler.transform(X\_test)

model = LogisticRegression(learning\_rate=0.01, n\_iterations\_max=1000, tol=0.001)

model.fit(X\_train\_scaled, y\_train)

y\_predictions = model.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_predictions)

Após obter a base de dados, separamos ela em dois conjuntos: teste e treinamento, com a proporção 20/80, respectivamente. Isso é feito para que a análise da acurácia não seja contaminada por teste em dados que foram utilizados no treinamento do modelo.

Além disso, normalizamos os dados, isto é, subtraímos a média e dividimos pelo desvio padrão, para que todos os coeficientes dos parâmetros tenham pesos equivalentes na busca de uma função que minimize a função custo previamente determinada. O impacto deste passo será visto na seção a seguir.

ANÁLISE DE RESULTADOS E CONVERGÊNCIA

Com a execução do código descrito acima, a função convergiu para os parâmetros *w* e *b* listados a seguir, com uma acurácia de 97.37%. Estes valores de acurácia podem variar conforme a divisão aleatória dos conjuntos de teste e treinos feita, porém o alto valor de acurácia neste caso já dá ótimos indicativos de que a função está bem calibrada.

A screenshot of a computer program

AI-generated content may be incorrect.

Podemos ainda analisar a curva da função custo ao longo das consecutivas interações. Vemos que, para uma tolerância de 0.001 do módulo do gradiente, o processo foi interrompido no segundo critério de parada que era 1000 interações. Mesmo assim, o valor final da função de custo é baixo, indicando que a solução está próxima do seu ótimo global, convergindo de modo exponencial e contínuo.

A graph with a blue line

AI-generated content may be incorrect.

Por fim, vemos o efeito que a normalização dos dados trouxe para curva de custo. Sem normalizarmos os parâmetros, a função de custo torna-se instável sem convergir para um valor baixo desejável.

A graph showing a graph

AI-generated content may be incorrect.