Exercícios de Fixação - Modelos Lineares

Alunas: Flávia e Nathália

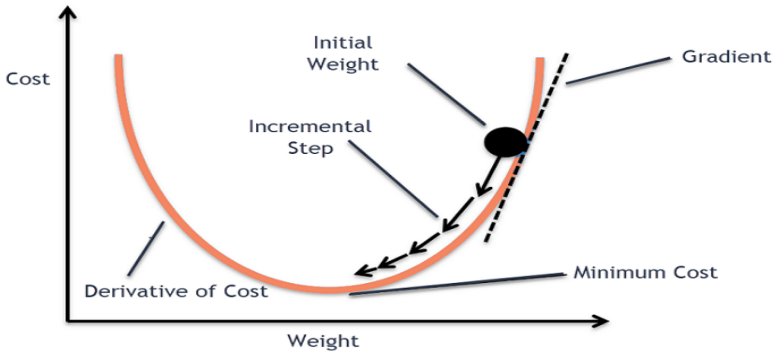
1- Para cada um dos algoritmos abaixo, faça uma pesquisa e o descreva, aponte quando cada um é indicado e porque, vantagens e desvantagens de cada um:

- Equação Normal, SVD, Batch GD, Stochastic GD e Mini-Batch GD.

Na **Equação Normal**, É preciso calcular o inverso de $X^T.X$, que pode ser uma matriz bem grande de ordem $(n + 1) \times (n + 1)$, ou seja complexidade computacional é quase igual a $O(n^3)$, acarretando que se o número de recursos aumentar, o tempo de cálculo aumentará drasticamente. Por exemplo, se você dobrar o número de recursos, o tempo de computação aumentará cerca de oito vezes. Além disso, a equação normal pode não funcionar quando $X^T.X$ é uma matriz singular.

A abordagem **SVD** é usada pela classe de regressão linear do Scikit-Learn. No método SVD, em vez de calcular o inverso, o pseudoinverso é calculado. A complexidade de computação da abordagem SVD é de cerca de $O(n^2)$. Portanto, ao dobrar o número de recursos, o tempo de cálculo aumenta quatro vezes, como o pseudoinverso é sempre definido para uma matriz e a complexidade de tempo de SVD é melhor do que a abordagem de Equação Normal. Mesmo assim, essa abordagem não é boa o suficiente. Ainda há espaço para otimização.

O gradiente descendente é um algoritmo de otimização iterativa de primeira ordem para encontrar o mínimo de uma função, o objetivo da descida gradiente é minimizar uma dada função.



No **Batch Gradient Descent**, todos os dados de treinamento são levados em consideração em uma única etapa, então é utilizada a média dos gradientes de todos os exemplos de treinamento e para atualizar os parâmetros. Ou seja, isso é apenas uma etapa da descida do gradiente em uma época. Batch Gradient Descent é ótimo para variedades de erros convexos ou relativamente suaves. Nesse caso, avançamos um pouco diretamente em direção a uma solução ótima.

Se o conjunto de dados for muito grande temos o **Stochastic Gradient Descent**. Os modelos de aprendizado profundo anseiam por dados, quanto mais dados, mais chances de um modelo ser bom. Suponha que nosso conjunto de dados tenha 5 milhões de exemplos, então, apenas para dar um passo, o modelo terá que calcular os gradientes de todos os 5 milhões de exemplos, não é uma maneira eficiente, para resolver este problema é usado o Stochastic Gradient Descent. Em Stochastic Gradient Descent (SGD), consideramos apenas um exemplo por vez para dar um único passo.

Vimos a descida do gradiente em lote. Também vimos a Descida do Gradiente Estocástico. O gradiente descendente em lote pode ser usado para curvas mais suaves. SGD pode ser usado quando o conjunto de dados é grande. A descida gradiente em lote converge diretamente para os mínimos. SGD converge mais rápido para conjuntos de dados maiores. Mas, como no SGD usamos apenas um exemplo por vez, não podemos implementar a implementação vetorizada nele. Isso pode retardar os cálculos. Para resolver este problema, uma mistura de Batch Gradient Descent e SGD é usada.

Nem usamos todo o conjunto de dados de uma vez nem usamos o único exemplo de cada vez. Usamos um lote de um número fixo de exemplos de treinamento que é menor do que o conjunto de dados real e o chamamos de mini lote(**Mini-Batch GD**). Fazer isso nos ajuda a obter as vantagens de ambas as variantes anteriores que vimos.

2- Qual regressão linear você usaria se tivesse um conjunto de treino com milhões de features?

Regressão Linear Múltipla

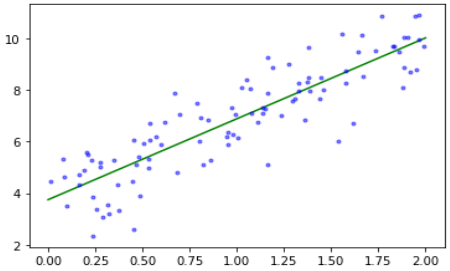
3- Suponha que as features no seu conjunto de treino possua escalas muito diferentes. Quais algoritmos podem ter problemas com isso e como? O que pode ser feito a respeito?

Regressão Linear e Regressão Logística, pois utilizam gradiente descendente, K-nearest neighbors(KNN), SVM, pois utilizam distância entre pontos de dados. A solução seria a Normalização ou Padronização. Dimensionar os dados utilizando técnicas de escalonamento, como a biblioteca *RobustScaler* . Pode ser utilizado também a biblioteca *Sklearn* aplicando o *MinMaxScalar* ou *StandardScalar.*

4- O algoritmo de Equação Normal foi desenvolvido em sala até o momento. Na aula passada o implementamos usando apenas o numpy. Agora use o mesmo gerador de dados (código abaixo) e implemente um dos outros algoritmos, a sua escolha. Compare os resultados gerando um gráfico com os dados e com os fits de cada modelo.

```
import numpy as np
import matplotlib.pyplot as plt

# Criação / aquisição dos dados
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# pré-proc
x_b = np.c_[np.ones((100, 1)), X]

# traino / fit()
beta = np.linalg.inv(x_b.T.dot(x_b)).dot(x_b.T).dot(y)

# predict()
X_new = np.array([[0], [2]])
x_new_b = np.c_[np.ones((2, 1)), X_new]
y_hat = x_new_b.dot(beta)
print(y_hat)

plt.plot(X_new, y_hat, 'g-')
plt.plot(X, y, 'b.', alpha=0.5)
plt.show()
```

```
[[ 3.74885691]
 [10.02734825]]
```



Referências:

https://medium.com/geekculture/quick-guide-gradient-descent-batch-vs-stochastic-vs-mini-batch-f657f48a3a0,

https://ichi.pro/pt/guia-rapido-gradiente-descendente-lote-x-estocastico-x-mini-lote-16928601580448,

https://towardsdatascience.com/difference-between-batch-gradient-descent-and-stochastic-gradient-descent-1187f1291aa1,