Respostas da lista 2

Nome: Celso Henrique de Souza Lopes

1 – Eu utilizaria o algoritmo do **gradiente descendente**, tendo em vista que ele escalona melhor comparado ao método da equação normal para grandes conjuntos de dados. A solução da equação normal envolve o cálculo da inversa de uma matriz, o qual tem alta complexidade. Dependendo do número de exemplos e *features*, a matriz pode consumir muita memória. Portanto, essa abordagem não é escalonável.

2 - A diferença de magnitude entre os atributos pode fazer com que alguns se tornem dominantes sobre os demais, exercendo assim grande influência sobre o erro cometido pelo modelo, o que pode afetar o desempenho dos algoritmos de ML. Esse problema ocorre com todo algoritmo que se baseia no cálculo da distância durante a fase do treinamento. Para evitar esse problema, a variação de todos os atributos deve ser escalonada para que cada atributo contribua com a mesma importância/peso para o cálculo da distância (ou seja, o erro quadrático médio). Essa técnica é chamada de escalonamento de *features* e possibilita comparar o peso/influência de cada *feature* do modelo, além de melhorar seu desempenho e estabilidade do treinamento.

3 - Nesse caso, provavelmente o passo de aprendizagem está muito grande e o algoritmo está divergindo. Para resolver esse problema, deve-se utilizar um passo de aprendizagem ótimo, ou seja, que convirja rapidamente. Há diversas maneiras para escolher esse passo, por exemplo, através de um ajuste manual, que é feito por tentativa e erro até que o passo ótimo seja encontrado, ou utilizando esquemas de variação programado ou adaptativo. No programado, o passo de aprendizagem tem seu valor diminuído ao longo do tempo (época), ou seja, ao longo do processo de treinamento. No adaptativo, o passo é ajustado de acordo com a performance do modelo. Além disso, possui pesos diferentes para cada parâmetro do modelo e os atualiza independentemente.

4 - O Gradiente Descendente Estocástico converge mais rapidamente, porém o Batch tem a convergência garantida, dado que o passo de aprendizagem seja ótimo. Para que o Estocástico e o Mini-batch convirjam, basta utilizar esquemas de variação do passo de aprendizagem.

5 - F) Pode-se concluir que o algoritmo GD em Batch:

• Segue diretamente para o mínimo global;

• Nesse caso específico, segue linha reta entre a0 e a1, pois a taxa de decrescimento da superfície de erro é igual para os dois parâmetros (contornos são circulares);

• Não fica "oscilando" em torno do mínimo após alcançá-lo;

• Algoritmo para no mínimo pois o vetor gradiente no ponto ótimo é praticamente nulo.

O algoritmo GD Estocástico:

• Devido à sua natureza estocástica, não apresenta um caminho regular/direto para o mínimo, mudando de direção várias vezes;

• O algoritmo não diminui suavemente até atingir o mínimo, fica "oscilando" ou "ricocheteando" em torno dele;

• Quando o algoritmo para os valores finais dos parâmetros são bons, mas não são ótimos;

• A convergência ocorre apenas na média;

• Tempo de treinamento é menor.

O algoritmo GD em Mini-batch:

• O progresso do algoritmo no espaço de parâmetros é menos irregular do que com o GD estocástico, especialmente com mini-batches grandes o suficiente;

• Como resultado, o mini-batch se aproxima mais do mínimo global do que o GDS;

• Tem comportamento mais próximo do GD em batelada;

• Oscilação em torno do mínimo diminui conforme o tamanho do minibatch aumenta.

7 - A. A função hipótese que aproxima melhor a função alvo (target) é h = a0 + a1*x + a2*x^2.

B. A função hipótese que aproxima melhor a função alvo (target) continua sendo h = a0 + a1*x + a2*x^2. Pode-se concluir que após o treinamento, através desses novos exemplos, conseguiu-se validar esse modelo.

8 - e) Como há uma grande diferença de magnitude entre x1 e x2, pode-se notar na letra (a) que, sem o escalonamento de *features*, a superfície de erro plotada na letra (a) está em forma de U, com maior taxa de variação do erro na direção de a2. Nesse caso, a taxa de variação do erro é praticamente constante na direção de a1. Portanto, é evidente que x2 contribui muito mais no valor do erro do que x1, fazendo com que a2 seja rapidamente atualizado, o que não acontece com o parâmetro a1, cujo gradiente é muito pequeno. Também é possível notar isso no gráfico de contorno, onde há retas ao invés de círculos, devido às inclinações diferentes na superfície de erro. O resultado disso é um treinamento lento (demora mais de 1874 iterações para convergir), ou seja, um impacto na performance do algoritmo de ML.

Ao aplicar a normalização mín-máx, onde para cada atributo do conjunto de treinamento, retira-se o mínimo do conjunto e depois divide-se pela subtração do máximo pelo mínimo. Nesse caso, houve uma pequena diferença em relação a letra a. Nota-se que o algoritmo convergiu mais rapidamente, porém a superfície de erro continua em formato de U. Ao aplicar a padronização, onde retira-se a média e divide-se pelo desvio padrão do conjunto de atributos de treinamento, a superfície de erro se aproxima mais da forma de uma tigela. O gráfico de contorno é mais circular, denotando que a superfície tem inclinação similar em todas as direções (i.e. nas direções de a1 e a2), dado que os atributos agora tem variações similares. Nesse caso, o treinamento é rápido (30 iterações) pois a inclinação é íngreme em todas as direções.

O escalonamento de *features* é vantajoso, pois:

• Possibilita comparar o peso/influência de cada *feature* no modelo;

• Melhora o desempenho e a estabilidade do treinamento do modelo, pois ajuda a acelerar a convergência do gradiente descendente ao deixar as curvas de nível da superfície de erro mais circulares.