Machine Learning

E se der empate na votação do KNN?

O KNN é um daqueles algoritmos que são muito simples de entender, mas que funcionam incrivelmente bem na prática. Também é surpreendentemente versátil e suas aplicações variam da detecção de padrões à geometria computacional. Muitos profissionais aprendem o algoritmo e não o utilizam, embora o KNN possa tornar algumas tarefas de construção de modelos preditivos muito simples. E talvez você não saiba, mas o KNN é considerado um dos 10 melhores algoritmos de mineração de dados.

KNN é um algoritmo de aprendizagem preguiçoso (lazy) não paramétrico. Essa é uma definição bastante concisa. Quando você diz que uma técnica é não paramétrica, isso significa que não faz nenhuma suposição sobre a distribuição de dados. Isso é bastante útil, pois no mundo real, a maioria dos dados não obedece às suposições teóricas típicas (por exemplo, misturas gaussianas, linearmente separáveis, etc.).

É também um algoritmo preguiçoso. O que isto significa é que ele não usa os pontos de dados de treinamento para fazer qualquer generalização. Em outras palavras, não há uma fase de treinamento explícita ou, quando há, ela é mínima. Isso permite que a fase de treinamento seja bastante rápida. A falta de generalização significa que o KNN mantém todos os dados de treinamento. Mais exatamente, todos os dados de treinamento são necessários durante a fase de teste. (Bem, isso é um exagero, mas não muito longe da verdade). Isso é um contraste com outras técnicas como SVM onde você pode descartar todos os “não vetores de suporte” sem qualquer problema. A maioria dos algoritmos preguiçosos - especialmente KNN - toma a decisão com base em todo o conjunto de dados de treinamento (ou no melhor dos casos, um subconjunto deles).

A dicotomia é bastante óbvia aqui - Há uma fase de treinamento inexistente ou mínima, mas uma fase de teste bastante intensa em termos de computação. O custo é em termos de tempo e memória. Mais tempo pode ser necessário, quando no pior caso, todos os pontos de dados podem se tornar pontos de decisão. Mais memória é necessária, pois precisamos armazenar todos os dados de treinamento.

O KNN assume que os dados estão em um espaço de características (feature space). Mais exatamente, os pontos de dados estão em um espaço métrico. Os dados podem ser escalares ou possivelmente vetores multidimensionais. Uma vez que os pontos estão no espaço de características, eles têm uma noção de distância - Esta não precisa ser necessariamente distância euclidiana, embora seja o comumente usado.

Cada um dos dados de treinamento consiste em um conjunto de vetores e labels de classe associada a cada vetor. No caso mais simples, será + ou - (para classes positivas ou negativas). Mas KNN, pode trabalhar igualmente bem com número arbitrário de classes.

Também definimos um único número "k". Esse número decide quantos vizinhos (onde os vizinhos são definidos com base na métrica de distância) influenciam a classificação. Este é

geralmente um número ímpar se o número de classes é 2. Se k = 1, então o algoritmo é simplesmente chamado de algoritmo do vizinho mais próximo. De acordo com o valor definido para k, o KNN fará a comparação com o número k de vizinhos mais próximos. Ocorre normalmente um processo de votação, de modo a garantir que o novo ponto de dado seja classificado de acordo com a maioria de vizinhos similares.

Mas o que acontece quando esta votação dá empate? O que acontece se não houver um vencedor claro na votação da maioria? Por exemplo. Todos os k vizinhos mais próximos são de classes diferentes, ou para k = 4 existem 2 vizinhos da classe A e 2 vizinhos da classe B? O que acontece se não for possível determinar exatamente k vizinhos mais próximos por que há mais vizinhos que têm a mesma distância? Por exemplo. Para a lista de distâncias (x1; 2), (x2; 3.5), (x3; 4.8), (x4; 4.8), (x5; 4.8), (x6; 9.2) **não seria possível** determinar k = 3 ou k = 4 vizinhos mais próximos, porque os vizinhos x3, x4 e x5 todos têm mesma distância.

Ao trabalharmos com KNN é preciso compreender que ele não é um algoritmo estritamente matemático, mas um simples classificador / regressor baseado em uma intuição - a função alvo não muda muito quando os argumentos não mudam muito. Ou, em outras palavras, a função alvo é localmente quase constante. Com essa suposição, você pode estimar o valor da função alvo em qualquer ponto, por uma média (possivelmente ponderada) dos valores de k pontos mais próximos.

Tendo isso em mente, você pode perceber que não há uma regra pré-definida sobre o que fazer quando não há um vencedor claro na votação por maioria. Você pode sempre usar um k ímpar ou usar alguma ponderação. Outra opção é reduzir o valor de k em uma unidade, até atingir o valor ideal.

No caso dos vizinhos x3, x4 e x5 do exemplo acima, que estão à mesma distância do ponto de interesse, você pode usar apenas dois ou usar todos os 5. Novamente, tenha em mente KNN não é um algoritmo derivado de análise matemática complexa, mas apenas uma intuição simples. Cabe a você, Cientista de Dados, decidir como quer lidar com esses casos especiais.

Quando se trata de ponderação, você baseia seu algoritmo na intuição de que a função alvo não muda muito quando os atributos não mudam muito. Podemos dar pesos maiores para pontos que estão mais perto do ponto de interesse. Uma boa ponderação seria, por exemplo:

$$(1 / x – y)2$$

Ou qualquer outra que seja relativamente grande quando a distância é pequena e relativamente pequena quando a distância entre os pontos é grande (Inversa de alguma função métrica contínua).

A intuição geral é que a função alvo varia de forma diferente em diferentes direções (ou seja, suas diferentes derivadas parciais são de magnitude diferente), portanto, seria sábio, em

algum sentido, mudar as métricas / ponderação de acordo com essa intuição. Este truque geralmente melhora o desempenho do KNN, requerendo uma customização do algoritmo implementado nas principais linguagens de programação como R ou Python.

Mas então como encontrar o valor ideal de k, quando houver empate? Você provavelmente precisará testar diferentes grupos de hiperparâmetros e ver quais funcionam bem em algum conjunto de validação. Se você tem recursos computacionais (computador com bastante memória e processador) e deseja chegar aos parâmetros corretos automaticamente em um bom conjunto de hiperparâmetros, há uma ideia recente de usar processos gaussianos para otimização sem derivada nessa configuração.

Encontrar o conjunto de hiperparâmetros (isto é, que minimizam o erro nos dados de validação), pode ser visto como um problema de otimização. Infelizmente, nessa configuração, não podemos obter o gradiente da função que tentamos otimizar (que é o que geralmente queremos fazer, executar a descida de gradiente ou alguns métodos mais avançados). Os processos gaussianos podem ser usados nesta configuração, para encontrar conjuntos de hiperparâmetros, que têm grandes chances, de se comportarem melhor do que os melhores encontrados até o momento. Assim, você pode iterativamente executar o algoritmo com um conjunto de hiperparâmetros, então pergunte ao processo gaussiano para qual deles seria melhor tentar em seguida e assim por diante.

Ou seja, é mais fácil reduzir o valor de k em uma unidade até encontrar o valor ideal!!!

Referências:

Gradient Weights help Nonparametric Regressors
http://www.columbia.edu/~skk2175/Papers/gradientWeights_longVersion.pdf

Practical Bayesian Optimization of Machine Learning Algorithms
https://hips.seas.harvard.edu/files/snoek-bayesopt-nips-2012.pdf