Aula 5

1 Introdução: o problema da recomendação

Uma das principais aplicações de machine learning é treinar modelos que sejam capazes de fazer boas recomendações, considerando a necessidade das empresas atuais em conseguirem cativar seus clientes. Como resolver esse problema de forma efetiva é uma área atual de muita pesquisa, sendo a aula de hoje uma brevíssima introdução a esse extenso campo na literatura.

A princípio, os sistemas de recomendação ('recommender systems') são modelos responsáveis por, através do feedback dos usuários, serem capazes de recomendá-los produtos que tem a maior chance de atraí-los. Considere por exemplo o Spotify: é por meio dos dados das suas interações, e também das interações de outros users similares a você (como veremos em collaborative filtering), que o app te recomenda albúns específicos, que julga como mais prováveis de te satisfazer.

Em suma, o objetivo é fazer boas recomendações com base em dados dos usuários, que podem vir de avaliações dos itens que você quer recomendar ('explicit feedback') ou de dados relacionados, mas não diretamente avaliativos em sua natureza, como número de clicks, histórico de interação com produtos, etc ('implicit feedback').

1.1 Necessidade de novos Modelos

Como veremos a seguir, na recomendação, é insensato escolher features para prever os gostos e tendências: é uma fonte de viés humano que não consegue capturar a complexidade do processo caótico que é a interação de milhões ou bilhões de users com os serviços. Desse modo, alguns modelos aprendidos até agora no curso se tornam obsoletos, pelo menos da forma como os vimos. A regressão linear tradicional (apresentada na 3° aula), por exemplo, se torna inútil para nós nesse cenário. A solução é buscar modelos que sejam capazes de enfrentar esse novo problema.

2 K-Nearest-Neighbors (KNN)

2.1 Panorama Geral

Vamos nos voltar agora para um algoritmo muito conhecido pela sua surpreendente eficácia, dada sua simplicidade: o KNN. Como o próprio nome sugere, esse algoritmo analisa os 'K' primeiros vizinhos de um determinado ponto para classificá-lo, seguindo a máxima: "se a maioria dos K pontos ao meu redor são da classe X, então eu provavelmente sou da classe X também". Desse modo, fica

fácil perceber que esse modelo não requer qualquer 'aprendizado', ou seja, ele é só um algoritmo que é aplicado para cada ponto novo que se deseja classificar ("lazy classifier").

A distância para determinar os vizinhos mais próximos pode ser calculada de diversos modos, porém o mais simples e intuitivo é usar a distância Euclidiana. Desse modo, para um ponto novo (P) de dimensão N, são escolhidos como vizinhos os K primeiros pontos (P') no dataset que tem o menor valor de d, sendo sua definição:

$$d(P, P') = \sqrt{(P_x - P'_x)^2 + (P_y - P'_y)^2 + ... + (P_n - P'_n)^2} \tag{1}$$

$$= \|P - P'\| \tag{2}$$

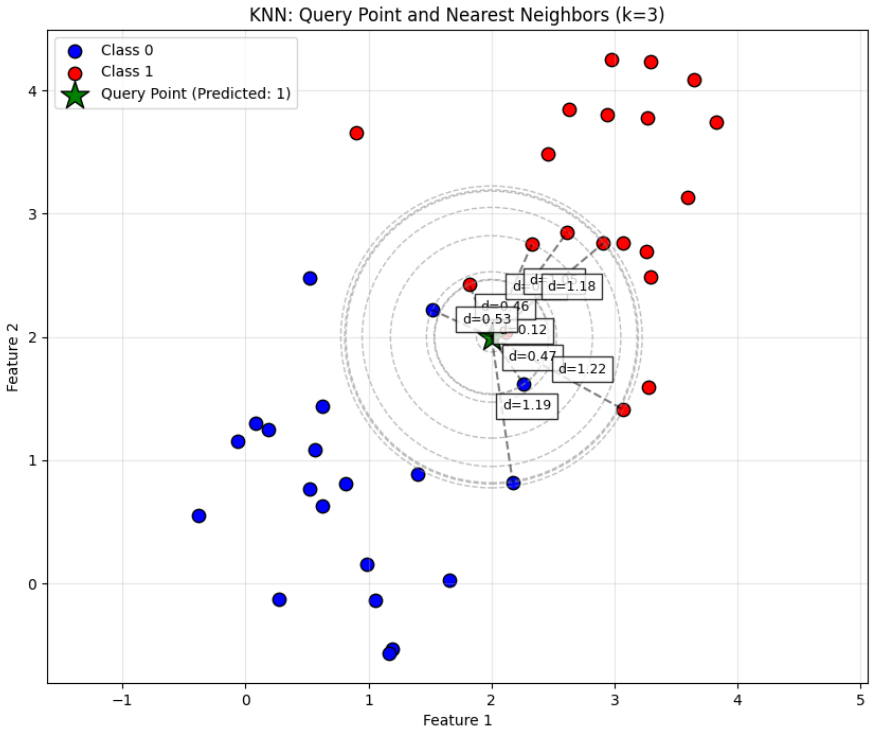


Figura 1: Algoritmo do KNN usando distância Euclidiana (K = 9)

2.2 Aplicação e limitações

No contexto de sistemas de recomendações (recommender systems), o KNN é utilizado para recomendar produtos ao usuário com base em ou produtos similares que ele gostou no passado, ou outros users com preferências similares ao analisado. Para ambos os casos, o algoritmo é bem simples:

- os dados são processados e angariados
- dados ordenados em relação ao ponto a ser analisado (o usuário a quem desejamos recomendar), sendo que a distância é calculada segundo alguma métrica que analise similaridade entre vetores (por exemplo, a de Pearson)

- são escolhidos os K primeiros itens ou usuários como crivo para recomendar
- recomendar o produto presente nos K primeiros itens

Apesar desse algoritmo ser bastante eficiente, o KNN apresenta alguns problemas que o tornam muitas vezes indesejável para nossa aplicação. Por ser um modelo que favorece itens que aparecem frequentemente no dataset, ele só recomendará produtos populares (pense na Netflix recomendando "Vingadores"para todo mundo); além disso, o custo computacional do KNN é alto, já que para todo user, é necessário avaliar os K vizinhos dele para determinar recomendações (complexidade $O(n^2)$). Portanto, é necessário buscar um modelo de ML que seja mais eficiente para a recomendação.

3 Collaborative Filtering

3.1 Motivação

Um dos principais problemas quando falamos de problemas de recomendação é que não conseguimos definir features para os nossos dados, dessa forma, caso tentemos, de forma arbitrária definir características para os nossos dados, podemos estar, na verdade, limitando o nosso modelo.

Dessa forma, o Collaborative Filtering usa uma das coisas mais práticas das técnicas de machine learning, ele deixa a máquina decidir as features! Afinal, aprendizado de máquina é isso, temos que treinar o nosso modelo para que ele consiga por ele mesmo fazer suas predições e produzir seus resultados, por isso, deixaremos também a cargo dele a tarefa de definir as features.

3.2 Ideia por trás

Agora, como podemos fazer isso? Na verdade a resposta é bem simples, imagine que nos estamos no caso clássico de uma matriz que contem nas linhas os nossos usuários e cada coluna é a nota que aquele usuário deu para um certo filme, assim, como com qualquer outra matriz 2D ela tem um número **n** de linhas e **m** de colunas.

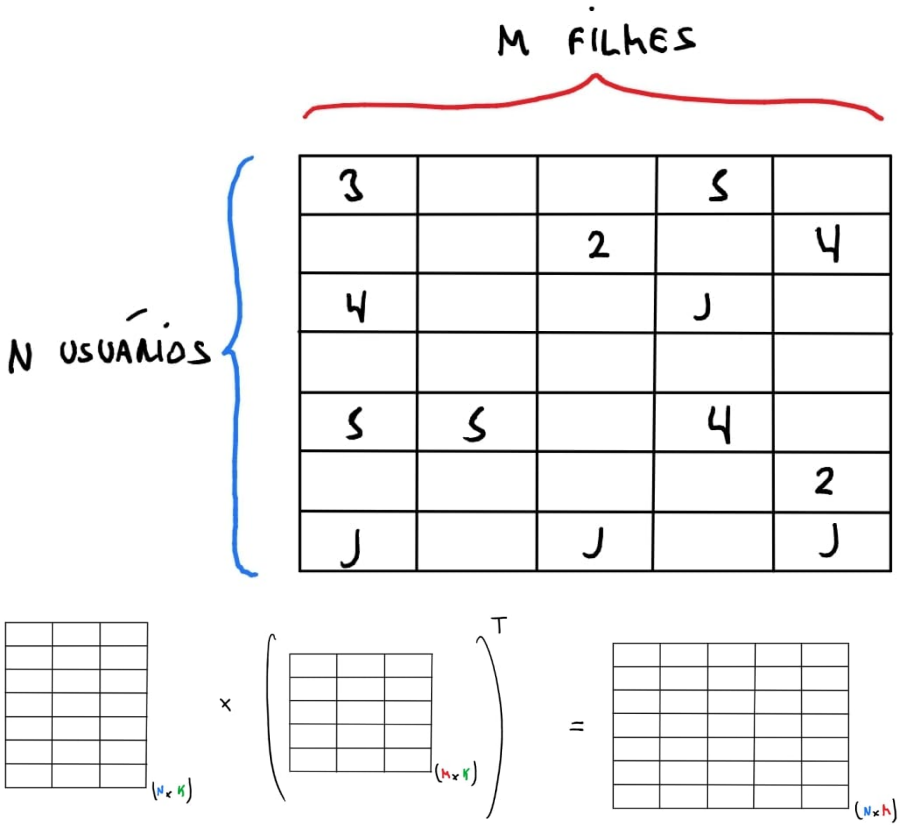
Aqui que entra a parte interresante, nos vamos fatorar essa matriz agora como a multiplicação de duas outras matrizes, pois, sabemos que o resultado de uma multiplicação entre duas matrizes $(n \times k)$ e $(m \times k)^T$ tem dimenções $n \times m$, que são as dimensões da nossa matriz.

Mas o que significa cada uma dessas matrizes? A primeira matriz, $(n \times h)$, é a matriz que define as k features dos nossos n usuários, em que k é o hiperparâmetro que define o número de features que a máquina irá criar. O mesmo raciocínio é usado para a matriz $(m \times k)$, sendo que ela, nesse caso, representa as features dos nossos filmes.

Dessa maneira, a nota final de um certo filme é apenas o produto escalar entre uma linha $\vec{u_i}$ da matriz de usuários (U) e uma linha $\vec{f_a}$ da matriz de filmes (F).

3.3 Função Loss

Agora, como nos nossos outros problemas, também precisamos saber o quão bem o nosso modelo está indo, por isso definimos as seguintes funções loss para cada um dos nossos parâmetros:



$$J(\vec{u_i}) = \sum_{a \in D} \frac{1}{2}(y_{ia} - \vec{u_i} \cdot \vec{f_a})^2 + \frac{\lambda}{2}\|\vec{u_i}\|^2 \tag{3}$$

$$J(\vec{f_a}) = \sum_{i \in D} \frac{1}{2}(y_{ia} - \vec{u_i} \cdot \vec{f_a})^2 + \frac{\lambda}{2}\|\vec{f_a}\|^2 \tag{4}$$

Onde D é o conjunto de dados que já tem uma nota.

3.4 Otimização alternada

A forma com que iremos otimizar nossos parametros, ou seja, achar as melhores features para os nossos usuários e filmes, será a seguinte:

1. Inicializar a matriz U e F com valores aleatórios
2. Fixamos F e calculamos a derivada parcial de cada $\vec{u_i}$ em J
3. Igualamos essa derivada parcial à zero e achamos os novos valores para cada $\vec{u_i}$
4. Fixamos U e calculamos a derivada parcial de cada $\vec{f_a}$ em J
5. Igualamos essa derivada parcial à zero e achamos os novos valores para cada $\vec{f_a}$
6. Repete

Assim, nossos parâmetros estaram treinados e poderemos executar a tarefa do problema de recomendação que é preencher os valores vazios!