UNIVERSIDADE FEDERAL DO ABC

Camila das Mercês Silva

Projeto01 - *K-nearest neighbors*, KNN

Projeto apresentado à Universidade Federal do ABC,
como parte dos requisitos para a disciplina
INF-317B - Tópicos em Inteligência Artificial: Machine Learning.

Prof. Dr. André Kazuo Takahata
e Prof. Dr. Ricardo Suyama

São Paulo
2019

RESUMO

K - Nearest Neighbors (KNN) é um algoritmo de aprendizagem supervisionada usado em *machine learning*. Ele é um classificador no aprendizado é baseado similaridade entre os dados, ou seja, na menor distância euclidiana das amostras vizinhas, em que a escolha de K varia de acordo com a base de dados. Neste trabalho o algoritmo foi implementado na linguagem *Python*, com o método de validação cruzada *Holdout*, dividindo os *datasets* entre conjunto de treino (70%) e conjunto de teste (30%). Os valores de K vizinhos mais próximos escolhidos foram sempre ímpares e/ou primos dependendo se o número de rótulo era par ou ímpar. Para dados mais complexos, o método se mostrou menos acurado para valores de K maiores (escolha de mais vizinhos). Para dados simples, o KNN é muito eficiente, com atenção na escolha do valor de K, baseado na quantidade de dados e de classes.

INTRODUÇÃO

K - Nearest Neighbors (KNN) é um dos clássicos algoritmos de aprendizagem supervisionada, usado em *machine learning*, proposto por Fukunaga e Narendra em 1975. Ele é um classificador em que o aprendizado é baseado similaridade entre os dados e, assim, seu treinamento é formado por vetores de n dimensões. A partir do conjunto de treinamento, o rótulo de classificação de uma amostra é determinado baseado nas amostras vizinhas, no qual a escolha do método do cálculo da distância é muito importante para o desempenho do método.

Neste trabalho, foi usado a distância Euclidiana, na qual a distância entre duas instâncias p_{ik} *e* p_{jk} é definida como:

$$d = \sqrt{\sum_{k=1}^{n} (p_{ik} - p_{jk})^2}$$

Em que p_{ik} *e* p_{jk} para k =1, .., **n** são **n** atributos que descrevem as instâncias p_i *e* p_j , respectivamente.

A escolha de K varia de acordo com a base de dados. Nas bases em que a quantidade de rótulos é par, é recomendável utilizar valores de K ímpares ou primos. Uma maneira é testar um conjunto de valores e encontrar o valor de K empiricamente. O algoritmo K segue os seguinte passos (pseudocódigo):

1. Início
    a. Receber e preparar o conjunto de dados de entrada (com os vetores de características e o rótulo)
    b. Inicializar o valor de K
2. Para cada uma nova amostra com vetor de características, fazer:
    a. Calcular as distâncias para todas as amostras
    b. Obter os K's vetores com as distâncias mais próximas
    c. Encontrar o rótulo mais frequente no conjunto de K's vizinhos
3. Fim
    a. Retornar o conjunto de rótulos de classificação

Por fim, é importante avaliar o modelo e os resultados gerados, a fim de confirmar a veracidade do método pela acurácia obtida.

MATERIAIS E MÉTODOS

A implementação do KNN foi feita na linguagem *Python* com o import seguintes pacotes csv, random, math e operator. A primeira função, "dividir_dados", faz o carregamento do *dataset* e tratamento dos dados, percorrendo sua linhas e colunas, e separando-o em duas listas, a lista de treino e a lista de teste a partir da proporção utilizada (70% para treino e 30% para teste). A validação cruzada foi a do método *Holdout*. Além disso, foram feitos *loops* para garantir que os dados do *dataset* sejam *floats* e não *strings*, transformando-os em dados numéricos. Por fim, a função retorna duas listas, uma de treino e a outra de teste.

Já a função "distancia_euclidiana" calcula a distância de uma instância em relação a todas as instâncias de treino, sem levar em conta o último dado de cada linha já que se trata da classificação em si. Essa distância é armazenada em uma variável e por fim, a função retorna a raiz quadrada dessa variável, calculando, assim, a distância euclidiana.

Para encontrar os K's vizinhos mais próximos de uma instância teste, foi feita a função "achar_vizinhos" que percorre a lista de treino e armazena a distância da instância em relação a todas a outras instâncias de teste, organizando as que estão mais próximas nos primeiros índices. Depois, para o K escolhido, é retornada as K's instâncias que estão mais próximas.

A função "decidir" usa a lista retornada da função anterior, selecionando o último elemento de cada lista, já que essa é a classe esperada. Assim, um contador com o valor de ocorrências é salvo e organizado de forma crescente, e é retornada a moda (elemento com maior ocorrência).

Por fim, a função "correto" percorre uma lista e armazenar as predições corretas em uma variável, retornado os certos em números brutos. Esse retorno será usada para calcular a acurácia do modelo em porcentagem, comparando a lista de teste com a predição.

Finalmente o algoritmo é chamado e são impressos na tela algumas informações, tais quais: o nome do *dataset*, número de dados na lista de treino, número de dados na lista de teste, número de classes do conjunto, a proporção de divisão entre treino e teste, o valor de K utilizado, o número de dados corretos e a acurácia do modelo em porcentagem.

O método de validação cruzada usado, o Método Holdout, é o mais simples de se implementar e funciona bem, mas possui as desvantagens, que são (i) nem todos os dados são usados no treinamento, já que parte dele fica apenas para o teste, (ii) quanto maior for o conjunto de treinamento, menor é a qualidade do teste (o viés), e (iii) quanto menor for o conjunto de treinamento, maior é a variância do modelo.

Os nomes, números de amostras e número de classes dos *datasets* usados neste modelo estão relacionados abaixo (Tabela 1).

Tabela 1: *Datasets* utilizados.

Nome do *dataset*	**Número de amostras**	**Número de classes**
dataset1.txt	20	2
dataset2.txt	25	2
dataset3.txt	200	2
dataset4.txt	130	2
dataset5.txt	150	3
dataset6.txt	150	3
twospirals.txt	194	2

A proporção usada para dividir o conjunto entre lista de teste e lista de treino foi de 0.7 e 0.3, respectivamente. Métrica de desempenho usada foi acurácia que compara o conjunto de teste com as predições, retornando um valor em porcentagem, a partir da fórmula:

$$(\textit{predições corretas} \,/\, \textit{total de elementos do conjunto de treino}) * 100$$

O gráfico de dispersão *Two spirals* (Figura 1) descreve os vetores de características no espaço de características do *dataset twospirals.txt*, que é um conjunto de dados bidimensionais mais complexo.

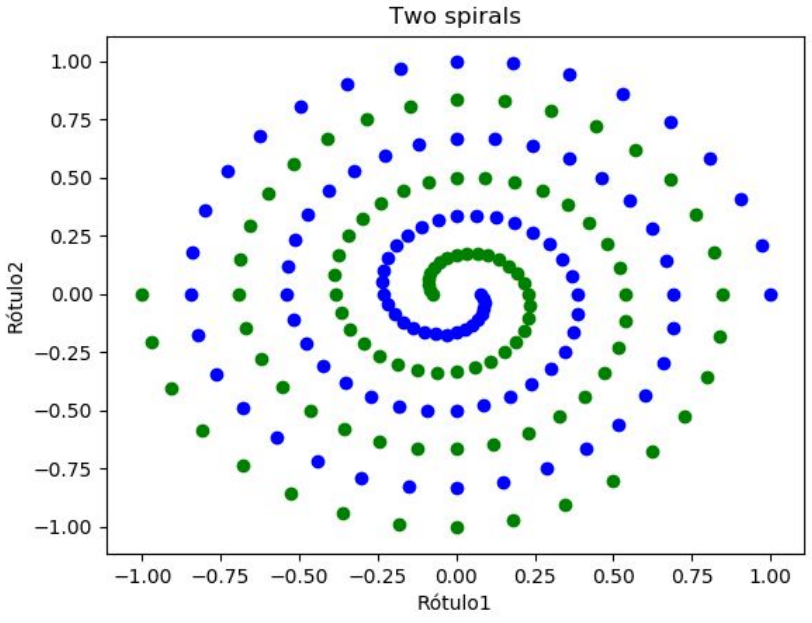

Figura 1: gráfico de dispersão Two spirals.

RESULTADOS

Os resultados obtidos conforme o *dataset* utilizado, as proporções de divisão entre conjunto de treino e conjunto de teste, valores de K (sempre ímpares e primos para *datasets* com duas classes e primos para três classes), as predições e a acurácia calculadas em cada rodada estão listadas na tabela 2.

Tabela 2: Resultados KNN

Dataset	**Conjunto de treino**	**Conjunto de teste**	**K**	**Predições**	**Acurácia**
dataset1.txt	13	7	1	7	100%
	14	6	3	6	100%
dataset2.tx	18	7	1	6	85.71%
	17	8	3	7	87.5%
dataset3.txt	140	60	1	49	81.66%
	144	56	3	43	76.78%
	138	62	5	49	79.03%
	138	62	7	53	85.48%
dataset4.txt	89	41	1	28	68.29%
	88	42	3	30	71.42%
	92	38	5	32	84.21%
dataset5.txt	106	44	1	43	97.72%
	107	43	2	42	97.67%
	106	44	3	43	97.72%
	105	45	4	45	100%
	104	46	5	46	100%
dataset6.txt	103	47	1	33	70.21%
	108	42	2	30	71.42%
	106	44	3	36	81.81%
	107	43	4	36	83.72%
	108	42	5	35	83.33%
twospirals.txt	132	60	1	53	85.48%
	132	62	3	40	64.51%
	137	57	5	23	40.35%
	133	61	7	21	34.42%

Foram feitos testes com mudanças nas proporções de divisão entre conjunto de teste e conjunto de treino, porém não ocorreram diferenças significativas nos acertos e nas acurácias.

CONCLUSÕES

Com os resultados mostrados acima, é possível destacar que a proporção entre o valor de treinamento e de teste não muda consideravelmente nas respostas finais. O valor de K, entretanto, é essencial para a correta aplicação do modelo, já que ele depende da quantidade de rótulos da base de dados, assim como a quantidade e complexidade desses dados.

Para as menores bases de dados (*dataset1* e *dataset2*), valores de K iguais a 1 e a 3 foram suficientes para uma acurácia de mais de 80%, e no caso do *dataset1*, 100% para os dois K's. Para os outros *datasets*, com maior quantidade de dados, valores maiores de K se mostraram mais acurados, especialmente nos que tinham classes ímpares. Para os conjuntos de dados com dois rótulos (número par), sempre foi usado K ímpar, e par de três classes, sempre K primo.

Para dados mais complexos, como o *twospirals*, a escolha de um valor de K maior, resultou numa menor acurácia. Para exemplificar, faremos uma comparação desse conjunto de dados (*twospirals*) com o *dataset3*, que possuem quase a mesma quantidade de dados e a mesma quantidade de classes. Para o *dataset3*, mais simples, um maior valor de K resulta numa maior acurácia, exceto quando K é igual a 1. Já para *twospirals*, mais complexo, quanto maior o valor de K, menor é a acurácia. Isso acontece porque as distâncias euclidianas dos dados dessa última base de dados estão "menos organizadas" em comparação com o do *dataset3*.

Por fim, o algoritmo KNN se mostrou ser um eficiente modelo de tomada de decisão para dados menos complexos, possuindo potencialidades e fragilidades de acordo com o banco de dados utilizado. Os *datasets* propostos para o item b, não funcionavam no meu algoritmo, então pesquisei e utilizei um bem parecido (*twospirals*). O conjunto de dados *olivetti faces* não abria no meu *notebook*, fazendo o sistema travar, e por isso não foi usado no projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

Fukunaga, K.; Narendra, P. M. A branch and bound algorithm for computing k-nearest neighbors. IEEE Transactions on Computers, v. 100, 1975.

APÊNDICES

O código fonte utilizado está disponível em: https://drive.google.com/file/d/1iGFjQApK__Z6WPoQJoAr_JE1Edt10XIN/view?usp=sharing

Os *datasets* utilizados estão disponíveis em: https://drive.google.com/drive/folders/1dbBiUBVQ0XiSzgnRtmI7d8r7PhTwX9q8?usp=sharing