K-Nearest Neighbors (kNN)

Mateus Tomoo Yonemoto Peixoto, Renan Kodama Rodrigues

Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87301-899 – Campo Mourão – PR – Brasil

Ciências da Computação – (UTFPR)

Inteligência Artificial – Aprendizagem de Maquina
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – Brasil
{mateus_tomoo,renan_kdm_rdg}@hotmail.com

Abstract. *In this article, we will analyze how the kNN classification algorithm works with predefined test and training files, we will also show how the metric was calculated to calculate the k nearest neighbors of a given instance and the results obtained together with the confusion matrix. The objective of this article is to explain the operation of kNN together with the data and algorithm obtained.*

Resumo. *Neste artigo, iremos analisar o funcionamento do algoritmo de classificação kNN com arquivos de teste e treino predefinidos, mostraremos tambem como foi realizado a métrica para calcular os k vizinhos mais próximos de uma dada instância e os resultados obtidos juntamente com a matriz de confusão. O objetivo deste artigo é conseguir explicar o funcionamento do kNN juntamente com os dados e algoritmo obtidos.*

1. Infomações Gerais

A descrição para a execução do algoritmo *"kNN.py"* encontra -se no arquivo denominado *"READ-ME"*. Para este experimento foram utilizados dois arquivos, sendo eles, um de treino("*training.data"*) e outro de teste("*testing.data"*), contendo 10 classes, 132 características e 1000 instâncias em ambos arquivos. As classes são representadas pelo conjunto *C={0,1,2,3,4,5,6,7,8,9},* sendo definido pelo último caractere de uma uma linha. Os resultados gerados pela execução do algoritmo estão na forma de matriz de confusão.

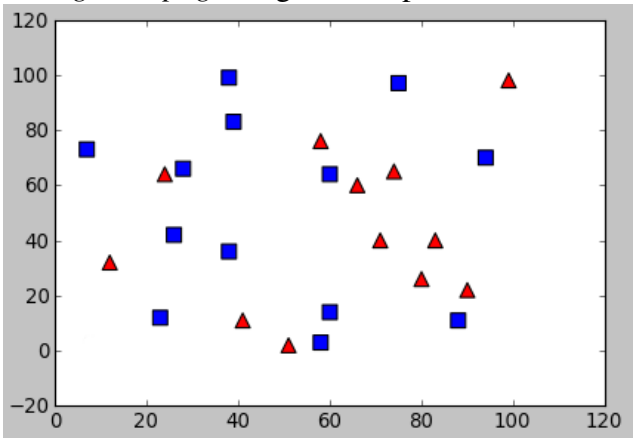
O algoritmo foi implementado usando a linguagem *python,* foi escolhido por trazer maior flexibilidade para a implementação, para o calculo da distância optamos pelo método da ditsância Euclidiana e para a normalização dos dados usamos o *mim* e *max*, pois sabemos qual valor o mínimo e máximo que uma característica poderá ter.

2. Introdução

A ideia do funcionamento do algoritmo kNN em forma mais abstrata, consiste em plotar as instâncias em um plano *x* e *y* de acordo com alguma função de característica que retorna os valores para *x* e *y,* sendo assim, classificando novas instâncias de acordo com os *k* vizinhos mais próximos a esta instância. Para tal feito sempre é utilizado valores ímpares para não ocorrer empates.

3. Aprendizagem

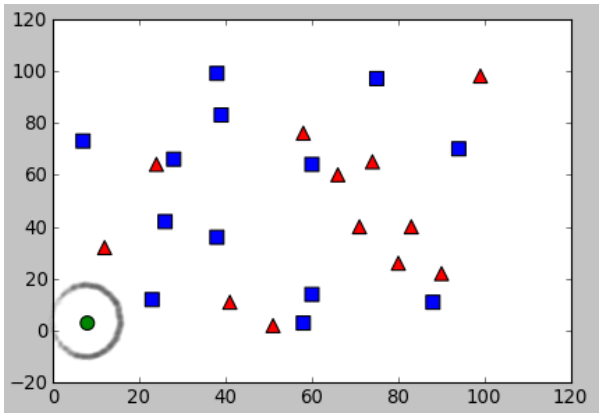
A fase de aprendizagem é definido pelo arquivo "*training.data"* onde o algoritmo irá criar um modelo de plano de acordo com as coordenadas *x* e *y* baseadas nos valores das características das instâncias, então serão plotadas do arquivo "*training.data"* para o plano, usando um exemplo onde as instâncias são plotadas como quadrados ou triângulos no plano de acordo com os valores de características, sabemos disso pois no caso treino a classe a qual pertence determinada instância encontra -se na última posição da linha. A imagem "*imagem01.png*"a seguir exemplifica a ideia.



"imagem01.png" - Exemplo de plano gerado a partir do caso treino.

4. Teste

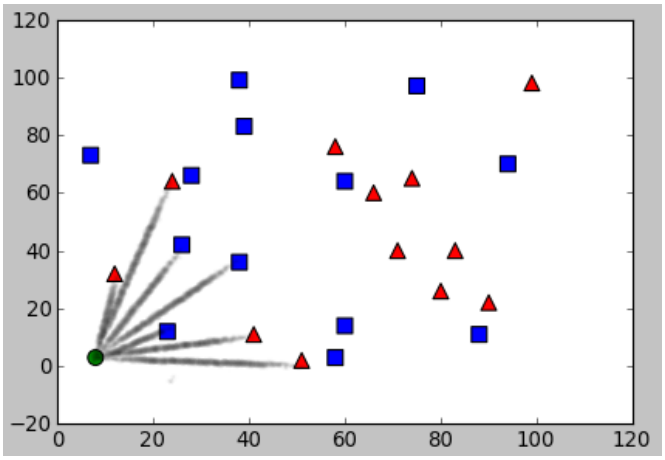
A fase de teste é definido pelo arquivo "*testing.data"* onde as novas instâncias serão plotadas no plano já treinado definido pelo arquivo "*training.data"*. As novas instâncias serão plotadas no plano de acordo com seus valores de caracteristicas.Podemos assumir uma forma desconhecida para as novas instâncias plotadas no plano, já que não sabemos a qual classe pertence tais instâncias, como por exemplo um círculo denotado pelo objeto circulado na imagem ao lado.



"imagem02.png" - Exemplo para o caso teste.

4. Classificação

A fase de classificação consiste em classificar as instâncias do arquivo "*testing.data*" com base nos k vizinhos mais próximos em relação à uma dada instância. A distância é calculada pela função *"euclidianDistance()"* que computa para todas as outras instâncias contidas no plano e seleciona aquela de menor valor , em outras palavras, os mais próximos. Na figura *"imagem03.png"*, podemos analisar que o resultado da classificação de "círculo" dependerá do valor de k. Para k=1, temos que "círculo" pertence a classe "quadrado", enquanto para k=7 pertence a classe "triângulo".



"imagem03.png" - Exemplo para a classificação.

5. Resultados

Os resultados obtidos ao executarmos o *"kNN.py"* com os arquivos de entrada "*testing.data" e* "*training.data"* com variação do k, serão apresentados logo abaixo.

K = 1 [92, 0, 0, 1, 0, 3, 1, 2, 1, 0]-> 0 [0, 87, 3, 1, 1, 3, 2, 0, 3, 0]-> 1 [0, 2, 84, 7, 0, 2, 0, 0, 5, 0]-> 2 [1, 0, 10, 81, 0, 5, 1, 0, 0, 2]-> 3 [0, 1, 0, 0, 92, 1, 1, 0, 1, 4]-> 4 [1, 1, 2, 1, 0, 95, 0, 0, 0, 0]-> 5 [2, 0, 0, 1, 0, 1, 96, 0, 0, 0]-> 6 [0, 0, 0, 2, 1, 0, 0, 95, 0, 2]-> 7 [4, 2, 1, 3, 2, 5, 1, 2, 76, 4]-> 8 [0, 1, 0, 0, 3, 3, 0, 3, 0, 90]-> 9 Accuracy: 88.8%	K = 3 [93, 0, 0, 1, 1, 1, 1, 2, 0, 1]-> 0 [1, 87, 3, 0, 2, 2, 1, 0, 4, 0]-> 1 [0, 0, 86, 8, 0, 2, 1, 0, 2, 1]-> 2 [4, 0, 7, 81, 0, 2, 3, 0, 0, 3]-> 3 [2, 1, 0, 1, 91, 0, 1, 0, 3, 1]-> 4 [1, 1, 0, 2, 0, 91, 1, 0, 3, 1]-> 5 [1, 0, 0, 1, 0, 1, 97, 0, 0, 0]-> 6 [0, 0, 0, 3, 0, 0, 0, 95, 1, 1]-> 7 [6, 2, 2, 3, 1, 2, 5, 0, 76, 3]-> 8 [0, 1, 0, 2, 1, 1, 0, 4, 0, 91]-> 9 Accuracy: 88.8%
K = 5 [93, 0, 0, 1, 1, 2, 0, 1, 0, 2]-> 0 [1, 86, 3, 0, 1, 2, 1, 0, 5, 1]-> 1 [0, 0, 87, 5, 2, 2, 1, 0, 3, 0]-> 2 [2, 0, 7, 83, 0, 2, 2, 0, 1, 3]-> 3 [1, 0, 0, 1, 93, 0, 1, 0, 0, 4]-> 4 [0, 0, 0, 1, 0, 94, 1, 0, 3, 1]-> 5 [1, 0, 0, 1, 0, 1, 97, 0, 0, 0]-> 6 [0, 0, 0, 0, 2, 0, 0, 96, 1, 1]-> 7 [4, 1, 2, 2, 1, 2, 2, 1, 80, 5]-> 8 [0, 0, 0, 0, 5, 1, 1, 4, 0, 89]-> 9 Accuracy: 89.8%	K = 7 [93, 0, 0, 0, 2, 1, 0, 1, 0, 3]-> 0 [1, 87, 3, 0, 1, 1, 1, 1, 3, 2]-> 1 [0, 0, 86, 5, 2, 1, 2, 0, 4, 0]-> 2 [1, 1, 9, 83, 0, 0, 2, 0, 1, 3]-> 3 [0, 0, 0, 1, 94, 0, 1, 0, 0, 4]-> 4 [0, 0, 0, 3, 0, 93, 1, 0, 0, 3]-> 5 [0, 0, 0, 1, 0, 1, 97, 0, 0, 1]-> 6 [0, 0, 0, 0, 2, 0, 0, 96, 1, 1]-> 7 [7, 1, 2, 1, 2, 2, 2, 1, 73, 9]-> 8 [0, 0, 0, 0, 5, 1, 1, 4, 0, 89]-> 9 Accuracy: 89.1%

K = 9 [93, 0, 0, 2, 1, 1, 0, 0, 0, 3]-> 0 [0, 87, 3, 0, 1, 1, 2, 1, 3, 2]-> 1 [0, 0, 88, 4, 1, 1, 2, 0, 4, 0]-> 2 [1, 0, 9, 82, 1, 1, 2, 0, 0, 4]-> 3 [0, 0, 0, 1, 94, 0, 1, 0, 0, 4]-> 4 [0, 0, 0, 2, 1, 93, 1, 0, 0, 3]-> 5 [0, 0, 0, 1, 0, 1, 97, 0, 0, 1]-> 6 [0, 0, 0, 0, 2, 0, 0, 96, 0, 2]-> 7 [7, 1, 1, 2, 2, 2, 2, 1, 73, 9]-> 8 [0, 0, 0, 0, 5, 0, 1, 6, 0, 88]-> 9 Accuracy: 89.1%	K = 11 [93, 0, 0, 1, 0, 2, 2, 0, 0, 2]-> 0 [0, 87, 3, 0, 0, 2, 2, 1, 3, 2]-> 1 [0, 0, 85, 6, 0, 1, 4, 0, 4, 0]-> 2 [1, 0, 9, 83, 1, 0, 3, 0, 0, 3]-> 3 [0, 0, 0, 0, 96, 0, 1, 0, 0, 3]-> 4 [0, 0, 0, 2, 1, 93, 1, 0, 0, 3]-> 5 [0, 0, 0, 1, 0, 1, 97, 0, 0, 1]-> 6 [0, 0, 0, 0, 1, 0, 1, 97, 0, 1]-> 7 [8, 1, 1, 2, 1, 2, 3, 1, 72, 9]-> 8 [0, 0, 0, 0, 5, 1, 1, 6, 0, 87]-> 9 Accuracy: 89.0%
K = 13 [93, 0, 0, 1, 0, 2, 2, 0, 1, 1]-> 0 [0, 87, 3, 0, 1, 2, 2, 1, 4, 0]-> 1 [0, 0, 85, 7, 0, 1, 3, 0, 4, 0]-> 2 [1, 0, 9, 84, 1, 0, 3, 0, 0, 2]-> 3 [0, 0, 0, 0, 97, 0, 1, 0, 0, 2]-> 4 [0, 0, 0, 2, 1, 94, 1, 0, 0, 2]-> 5 [0, 0, 0, 1, 0, 1, 97, 0, 0, 1]-> 6 [0, 0, 0, 0, 1, 0, 0, 97, 0, 2]-> 7 [10, 1, 1, 3, 1, 2, 3, 0, 72, 7]-> 8 [0, 0, 0, 0, 7, 0, 1, 4, 0, 88]-> 9 Accuracy: 89.4%	K = 15 [93, 0, 0, 1, 0, 2, 1, 0, 1, 2]-> 0 [0, 87, 3, 0, 1, 1, 2, 1, 4, 1]-> 1 [0, 0, 87, 5, 0, 1, 3, 0, 4, 0]-> 2 [1, 0, 9, 83, 1, 0, 4, 0, 0, 2]-> 3 [0, 0, 0, 0, 97, 0, 1, 0, 0, 2]-> 4 [0, 0, 0, 2, 1, 94, 1, 0, 0, 2]-> 5 [0, 0, 0, 0, 0, 2, 97, 0, 0, 1]-> 6 [0, 0, 0, 0, 1, 0, 0, 97, 0, 2]-> 7 [10, 2, 2, 2, 1, 3, 3, 1, 69, 7]-> 8 [0, 0, 0, 0, 7, 0, 1, 6, 0, 86]-> 9 Accuracy: 89.0%
K = 17 [93, 0, 0, 2, 0, 2, 1, 0, 0, 2]-> 0 [0, 87, 3, 0, 1, 3, 1, 0, 4, 1]-> 1 [0, 0, 86, 6, 0, 1, 3, 0, 4, 0]-> 2 [1, 0, 9, 83, 2, 0, 4, 0, 0, 1]-> 3 [0, 0, 0, 0, 97, 0, 1, 0, 0, 2]-> 4 [0, 0, 0, 2, 1, 95, 0, 0, 0, 2]-> 5 [0, 0, 0, 1, 0, 1, 97, 0, 0, 1]-> 6 [0, 0, 0, 1, 1, 0, 0, 97, 0, 1]-> 7 [10, 0, 1, 2, 2, 6, 0, 2, 70, 7]-> 8 [0, 0, 0, 0, 8, 0, 0, 5, 0, 87]-> 9 Accuracy: 89.2%	K = 19 [93, 0, 1, 1, 0, 2, 1, 0, 0, 2]-> 0 [0, 87, 3, 0, 2, 2, 1, 0, 4, 1]-> 1 [0, 0, 86, 6, 0, 1, 3, 0, 4, 0]-> 2 [1, 0, 10, 82, 2, 0, 4, 0, 0, 1]-> 3 [0, 0, 0, 0, 98, 0, 1, 0, 0, 1]-> 4 [0, 0, 0, 3, 1, 94, 0, 0, 0, 2]-> 5 [0, 0, 1, 0, 0, 2, 96, 0, 0, 1]-> 6 [0, 0, 0, 2, 1, 0, 0, 97, 0, 0]-> 7 [10, 1, 2, 3, 2, 5, 1, 1, 68, 7]-> 8 [0, 0, 0, 0, 5, 1, 0, 8, 0, 86]-> 9 Accuracy: 88.7%

"Tabela01" - Resultado das amostras pelo algoritmo kNN.py.

Os resultados apresentados estão no formato de matriz de confusão onde a diagonal principal representa as instâncias que o algoritmo acertou, e os valores que não estão nessa diagonal são as instâncias que o algoritmo errou, então para saber o quanto o algoritmo errou ou acertou, basta pegar o elemento da diagonal principal para computar o acerto ou somar os demais elementos para computar o erro. Por exemplo, na primeira linha extraida do resultado para k = 3 temos, [93,0,0,1,1,1,1,2,0,1]→0, onde o número 93 significa o numero de instâncias que o algoritmo acertou pois este, encontra -se na posição (*"matriz[0][0]"*) da matriz de confusão, como os índices *"i"* e *"j"* da matriz são iguais, logo é um elemento da diagonal principal da matriz, os demais elementos significam o quanto o algoritmo errou, por exemplo, para os elementos que não pertencem a diagonal principal e são diferentes de zero como no caso o "1"(mais próximo de "93"), está na posição (*"matriz[0][3]"*), onde o valor de "*j*" significa a classe que estou comparando, como no exemplo, o algoritmo errou uma vez quando foi classificar a instância "*0*" com a instância "*3*".

6. Conclusão

Analisando os resultados obtidos, podemos concluir que nem sempre aumentar os k vizinhos para analise trará uma taxa de acerto maior. Podemos analisar isso para os valores de k={1,7,13,17 e 19} onde a taxa de acerto começa a subir até o k= 5, para o k a partir de 7 as taxas de acerto diminuiu. Podemos concluir que nem sempre o algoritmo kNN terá bons resultados para um dado valor de k, para alguns casos, talvez os melhores resultados estejam contidos em um k de baixo valor, enquanto para outros tipos de problemas o melhor resultado está em um k de alto valor. Para este problema em questão denotado nos arquivos de entrada, tanto para teste quanto para treino, aquele que obteve melhor resultado foi para o k = 5, atingindo um precisão de 89.80% de acerto. Observando a matriz de confusão podemos notar quanto o algoritmo está confundindo uma dada instância com outra, denotado pela soma dos demais elementos fora da diagonal principal.

7. Referências

Deepak Sinwar, Rahul Kaushik "Study of Euclidean and Manhattan Distance Metrics using Simple K-Means Clustering".