Projeto Integrador V: K Nearest Neighbor

Danilo Mative[1], **Fernando Alves**[1], **Victor Eleuterio**[1]

[1]Departamento de Ciência da Computação – Centro Universitário Senac (SENAC)
São Paulo – SP – Brasil

danilo.amative/eleuteriotrindade@gmail.com, fernandoimp@outlook.com.br

1. Ferramentas

Tendo como foco a aplicação do algoritmo de K Nearest Neighbor, foi imprescindível verificar ao início do projeto o planejamento de construção para cada uma das fases, tanto no que se refere à linguagem e macro utilizada, quanto ao conjunto de dados para teste.

1.1. Python

O desenvolvimento do algoritmo e sua demonstração necessitavam de grandes operações com trocas de valores entre vetores e matrizes, além da leitura e demonstração de dados. Verificando o caso, foi realizada uma pesquisa afim de obter a melhor e mais simples forma de realizar estes tipos de operações, tendo tido o Python como o ideal para curto prazo.
Em primeira via a linguagem demonstrou devera simplicidade, além de trazer com suas bibliotecas padrões uma gama enorme de possibilidades, tais como: A identificação de valores em variados pontos em um vetor/matriz, a fácil obtenção de métodos de ordenação, randomização e iteração, além da fácil conversão de arquivos txt e csv para diversos formatos.

1.2. Conjunto de Dados Utilizados

Para realização do trabalho, se fez necessária a utilização de seis conjuntos de dados utilizados para Machine Learning, sendo eles: Abalone, Adults, Breast-Cancer-Winsconsin, Iris, Wine e Wine Quality, todos retirados do UCI Machine Learning Repository [A. Asuncion 2007].
Uma vez que os dados apresentavam blocos com texto, nulo ou classes, foi preciso efetuar a normalização dos dados, transformando todos os valores em valores do intervalo [0;1] e classes e texto em números de [1;n], sendo n o número de instancias de texto diferentes.
Além da normalização padrão, definiu-se ainda que todas as bases de dados utilizadas teriam a classe como último valor/coluna, assim facilitando a identificação da instancia no software independente da base utilizada.

2. Versão do Algoritmo Knn

Uma vez normalizadas todas as bases a serem utilizadas, foi feita a implementação do algoritmo K Nearest Neighbor, no qual dado um conjunto de dados e uma entrada sem classe definida, organiza e identifica valores aproximados afim de obter a classe do dado inserido[Jason Brownlee 2014].
Foram feitas implementações do algoritmo com diferentes números de vizinhos, uma vez que o tipo de classe do dado inserido era definida pela maior quantidade de comparações com um mesmo valor de classe.

Os valores utilizados para os vizinhos foram: 1, M+2, M*10+1 e Q/2(ou (Q/2)+1 caso Q/2 par), onde M é a quantidade de classes ímpar (soma-se 1 no caso par), e Q a quantidade da instâncias.
Os valores para os vizinhos foram testados simultaneamente durante as buscas, de modo a obter todos os resultados no menor número de chamadas para facilitar o cálculo estatístico das fases seguintes.

2.1. Divisão Euclidiana

Para medir a distância entre os pontos do espaço de características, foi aplicado o cálculo de Divisão Euclidiana, definida por:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Onde p e q são pontos de um espaço com n dimensões(em nosso caso, n linhas de instâncias e n linhas de características.

3. Cross-Validation

O Cross-Validation é uma técnica utilizada para estimar a performance de modelos preditivos, sendo assim, estabelece comparações com toda a base de dados a fim de verificar todos os resultados possíveis com valores internos, retornando porcentagens preditivas de acertos.
Para que a técnica seja aplicada, se faz necessário dividir a base em conjuntos de treino e teste, de modo que os dados de teste são valores testados pelo algoritmo, gerindo respostas corretas ou errôneas, e os dados de treino é um conjunto em que os dados de teste são inseridos como entrada para a verificação.
Neste projeto os dados de teste foram aplicados após as instâncias Q serem divididas em 10, método conhecido pelo nome 10-fold Cross Validation, que consiste em realizar 10 rodadas de teste, sendo cada uma com uma das partes separadas como teste e o restante como treino[Anonymous Author 2007].

4. Testes e Resultados

Após a finalização do algoritmo e execução do Cross-Validation, foram realizados testes para verificar a precisão de cada modelo de vizinhos, sendo estes os testes binários e multi-classe.
Com estes testes, buscou-se compreender quais os desvios de código, ou seja, para onde o erro se encaminhava, passando-nos a noção dos valores mais comuns a causarem a falta de precisão.

4.1. Teste Binário

Foca em modelos de dados com duas classes descritivas, onde considera-se uma como positiva e outra como negativa(0 ou 1). Para cada resultado apresentado, o teste checa

e verifica se o valor condiz ou não com a informação correta, demonstrando se a resposta obtida é verdadeira ou falso[Gary Ericson]. Ao final de todas as checagens, se faz possível realizar a criação de uma tabela com valores representativos sobre acertos e falhas para os dois casos.

4.2. Teste multi-classes

O teste multi-classes é aplicado ao caso do número de quantidades das características ser maior do que dois.
Uma vez que o multi-classes passa a ser aplicado, verifica-se em seu todo quais valores foram calculados de maneira correta e incorreta, gerando ao fim desse processo uma matriz de confusão multinível, sendo de NxN e demonstrando, para cada classe característica, a quantidade de acertos e para onde foram os erros(em quais classes caíram).

5. Conclusão

Com a aplicação do algoritmo de Knn em diferentes configurações de vizinhos, pode-se notar por meio da verificação dos resultados que, para pequenos conjuntos de dados, o erro para o menor número de vizinhos era muito próximos às outras configurações NN. Por outro lado, para grandes quantidades de instâncias, um número maior de vizinhos oferecia uma precisão maior.
Embora o 1NN fosse aplicável para pequenas bases e o Q/2 NN para classes maiores, notou-se que a melhor média de precisão se valiam para o M*2 NN e M*10+1 NN, que nunca estavam muito acima ou abaixo das médias de erro.
Com a validação de testes, pôde-se notar ainda que bases de dados com uma maior quantidade de instâncias, embora obtivessem resultados em intervalos maiores e demorados, obtiam também um conjunto mais preciso de vizinhos, com dados de um valor limite muito próximo ao inserido.
No que se refere a ferramenta de criação, pode ser notada incrível lentidão por parte do Python, que embora facilitasse o desenvolvimento, tinha um alto custo de processamento com suas funções pré-definidas. Mas não só problemas foram relacionados à linguagem, dado que seu uso agilizou o processo de criação e tornou menos abstrata a demonstração da codificação e dos respectivos resultados finais.
Sendo assim, concluímos que dependendo do tipo de aplicação do algoritmo e da base a ser estudada, diferentes configurações deverão ser aplicadas, levando em consideração a quantidade de colunas e instâncias.

Referências

A. Asuncion, D. N. (2007). UCI machine learning repository. Access date: 10 mar. 2018.

Anonymous Author (2007). Cross-validation: evaluating estimator performance. Access date: 12 mar. 2018.

Gary Ericson. Como avaliar o desempenho do modelo no azure machine learning. Access date: 13 mar. 2018.

Jason Brownlee (2014). Tutorial to implement k-nearest neighbors in python from scratch. Access date: 10 mar. 2018.