UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática U·F·P·E

IF699 – Aprendizagem de Máquina

Lista de exercícios 1

Adilson Angelo da Silva Junior (aasj2)

A questão foi implementada em Python3 utilizando as bibliotecas **csv, math, json, time** e **matplotlib** para plotagem dos gráficos. Os datasets escolhidos foram o cm1 e kc2 e armazenados como csv para melhor manipulação dos dados em python. Ambos têm as mesmas 21 variáveis numéricas mais sua classe booleana ao final.

Primeiramente implementei as funções básicas necessárias para a execução dos algoritmos de *Nearest Neighbor*, que podem ser encontradas no arquivo **knn_functions.py** (*euclidean, get_neighbors, get_response*), que foram posteriormente modificandas para aceitarem os algoritmos Adaptativo e k-nn com peso. A função *get_neighbors* recebeu mais um parâmetro para definir se usará o algoritmo adaptativo e a função *get_response* recebeu mais outro parâmetro para definir se o k-nn será com pesos ou não.

Foram adicionadas também as funções *get_min_radius* para auxiliar no cálculo do algoritmo adaptativo e a *get_accuracy* para o cálculo da acurácia a cada conjunto de teste.

Com as funções em mãos, passei a implementar a lógica do k-fold cross-validation no arquivo **main.py** para manipulação dos datasets e ser capaz de calcular a taxa de acerto média. Para uma otimização do tempo de treinamento realizei os testes dos 3 algoritmos na mesma iteração, mas calculando o tempo de processamento de cada algoritmo separadamente.

O resultado final é condensado em um json (**metadata.json**) com os seguintes dados:

No qual para cada dataset, para cada valor de k, temos:

n_acc: acurácia média k-nn

w_acc: acurácia média k-nn com pesos

a_acc: acurácia média k-nn adaptativo

n_time: tempo total de processamento do k-nn

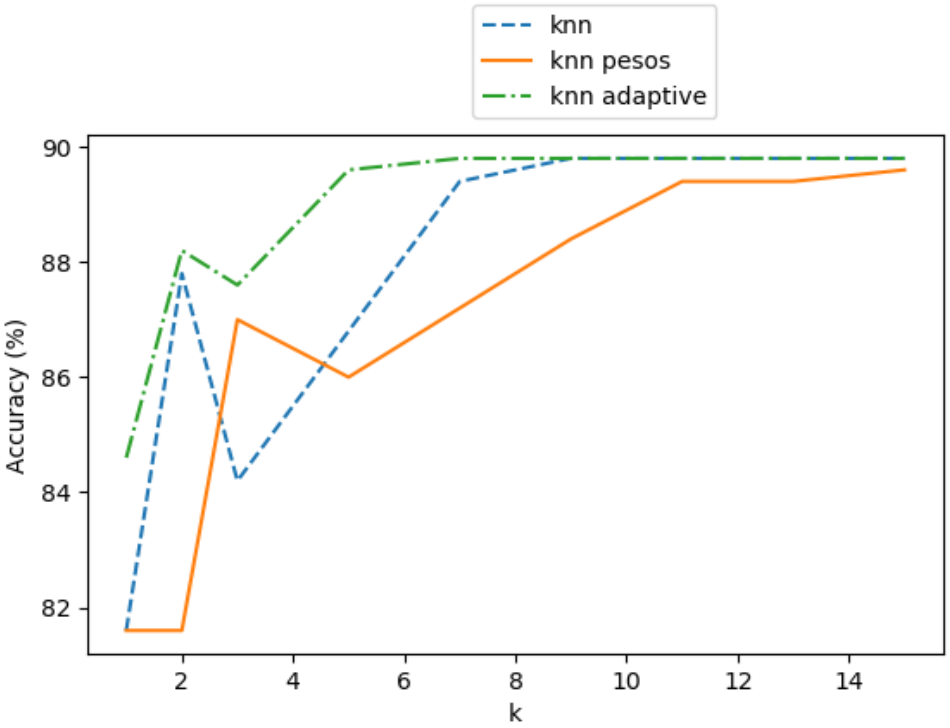
w_time: tempo total de processamento do k-nn com pesos

a_time: tempo total de processamento do k-nn adaptativo


```
"cm1": {
    "1": {
        "n_acc": 0.8160000000000001,
        "w_acc": 0.8160000000000001,
        "a_acc": 0.8460000000000001,
        "n_time": 2250,
        "w_time": 2250,
        "a_time": 190742
    },
    "2": {
        "n_acc": 0.8780000000000001,
        "w_acc": 0.8160000000000001,
        "a_acc": 0.882,
        "n_time": 2209,
        "w_time": 2225,
        "a_time": 190102
    },
    "3": {
```

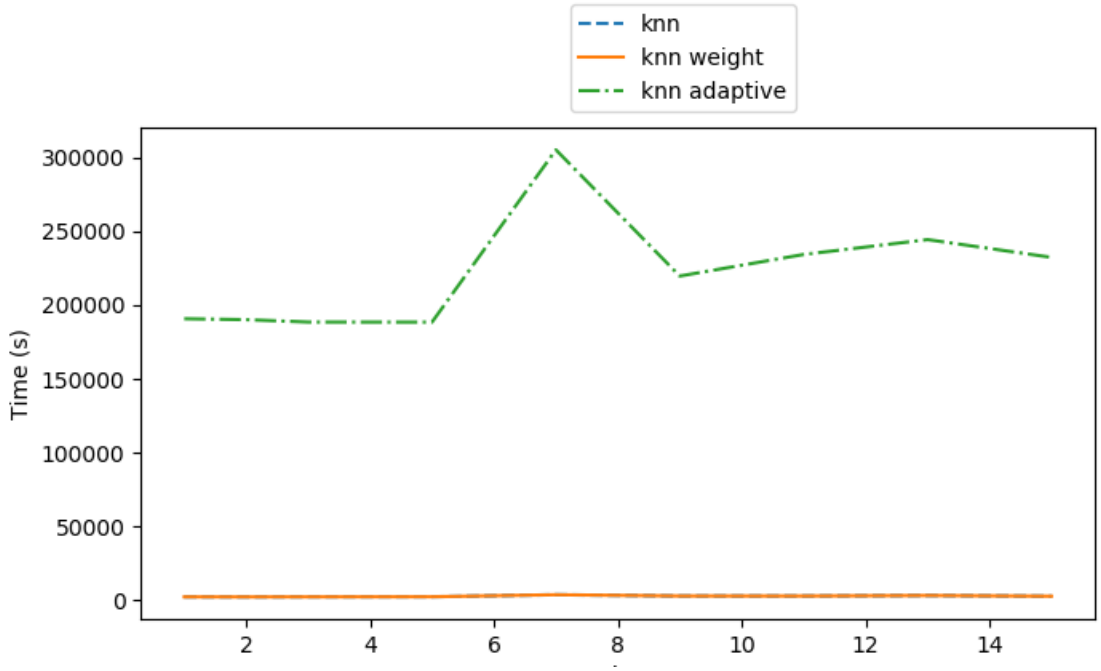
A partir desse json é possível plotar os gráficos executando o script **plt.py**. Dessa forma o gráfico de acurácia por valor de k para cada algoritmo e dataset:

Base: cm1



Dessa forma, é possível observar que para valores de k mais baixos, a taxa de acertos é menor, e que mesmo para os valores mais altos, o algoritmo k-nn com pesos ainda perde para o k-nn para este dataset.

Levando agora em consideração o tempo de execução de cara algoritmo para este dataset, dados pelo gráfico:



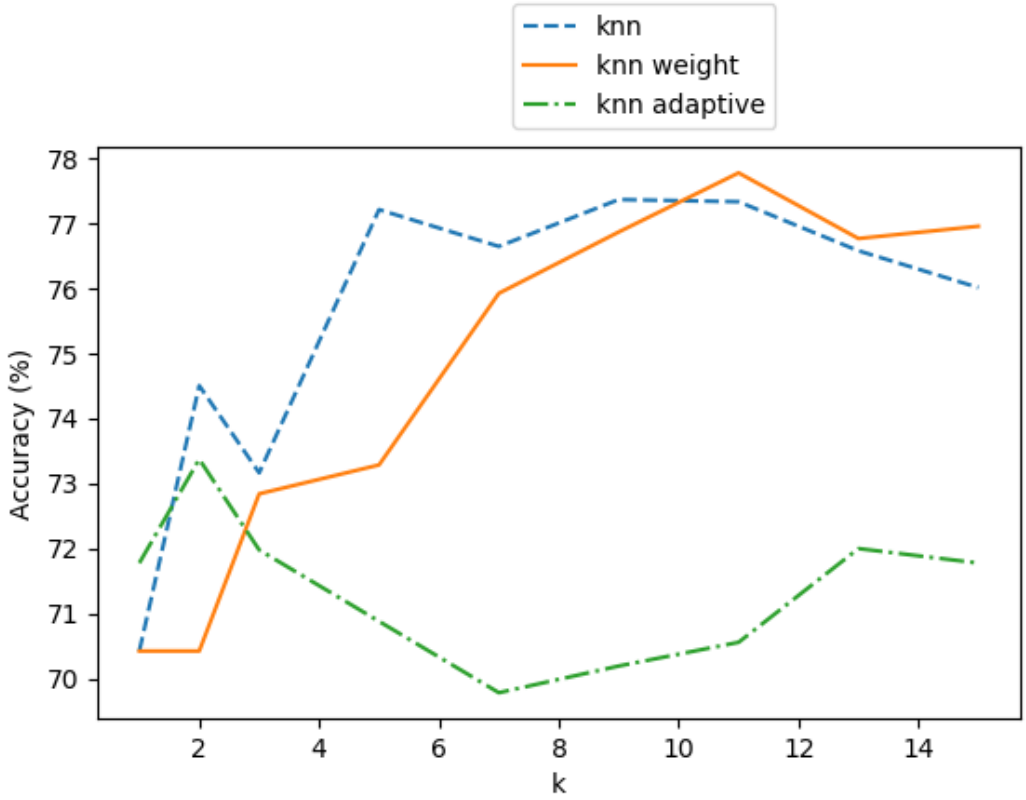
É possível notar que o algoritmo adaptativo leva da ordem de 10^5 mais tempo que os demais para executar devido as constantes consultas para achar o maior raio dentro de cada classe, o que poderia ser solucionado com a criação de uma tabela de raios por instancias a princípio, mas que, apesar de disposição e curiosidade para tal, não pôde ser implementada por falta de tempo disponível no cotidiano do aluno.

Assim, o knn e o knn adaptativo se mostram mais vantajosos em relação ao knn com pesos para esta base de dados, alcançando melhores resultados com o k no valor de 9 ou mais, mas o knn ainda leva menos tempo da forma que foi implementado. Caso houvesse uma tabela de raios calculada em um pre-processamento, creio que o knn adaptativo se mostraria ainda melhor.

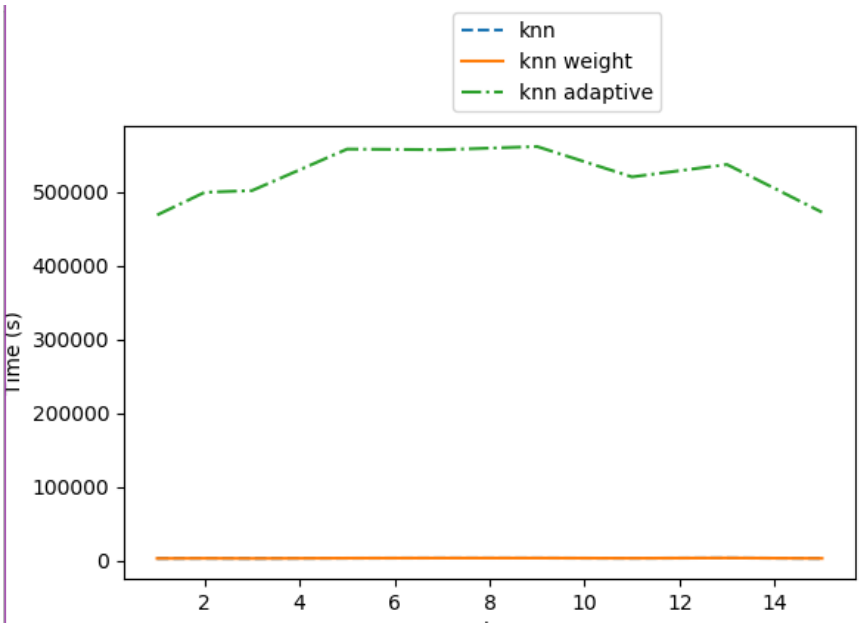
Base: kc2

Na base kc2, o knn adaptativo se mostrou bastante oneroso em comparação com os demais algoritmos, com baixa taxa de acertos e um custo de tempo enorme, por motivos supracitados.

É o que pode ser observado no gráfico abaixo:



O melhor resultado observado foi o do knn com pesos. Esse obteve uma taxa de acertos de mais de 77% com um k = 11 e toma virtualmente o mesmo tempo de processamento que o knn, como mostra o gráfico abaixo.


knn
knn weight
knn adaptive
Time (s)
500000
400000
300000
200000
100000
0
2
4
6
8
10
12
14