Vitor Galioti Martini

135543

Aula 09 - Exercício prático aprendizado supervisionado

Abaixo meu algoritmo KNN e sua análise descrita em comentários:

```python
# Vitor Galioti Martini
# 135543
import random
import csv
import math

# Algoritmo KNN
class Dados:

    # Construtor da classe
    def __init__(self, treino, testes):
        self.treino = treino
        self.testes = testes
        self.numero_acertos = 0

    # Algoritmo KNN
    def KNN(self):

        # Percorre os registros de teste
        i = 0
        while i < len(self.testes):
            distancia = []

            # Percorre os treinos
            j = 0
            while j < len(self.treino):

                # Faz o somatório daquele teste com o registro de treino atual
                k = 1
                soma = 0
                while k < 5:
                    soma = soma + (float(self.treino[j][k]) - float(self.testes[i][k]))**2
                    k += 1

                # Adiciona o valor na lista
                distancia.append([math.sqrt(soma), self.treino[j][k], self.treino[j][0]])
                j += 1

            melhores = []
            # Ordena os 3 registros que tiveram o melhor desempenho (K = 3)
            for linha in sorted(distancia, key=lambda x: x[0])[:3]:
                melhores.append(linha[1])

            # Se a classificação do registro teste for igual a classificação encontrada
            if self.testes[i][5] == max(set(melhores), key=melhores.count):
                self.numero_acertos += 1

            i += 1

        # Retorno
        precisao = float("{:.2f}".format(100 * self.numero_acertos / len(self.testes)))
        print("O algoritmo acertou: ", str(self.numero_acertos), " de ", str(len(self.testes)), " testes. Precisão: ", precisao, "%.")


def Importar():
    # Importa arquivo
    arquivo = open("Iris.csv")
    data_set = csv.reader(arquivo)

    # Cria lista com os dados
    lista_data_set = list(data_set)
    lista_data_set.pop(0) # Retirando cabeçalho
    tamanho_data_set = len(lista_data_set)

    # Separa 30% dos dados para testes e 70% para treino
    lista_treino = []
    lista_testes = []

    # Atribui aleatoriamente os registros para treino e teste com base no dataset
    i = 1
    while i < int(tamanho_data_set * 70 / 100):
        index = lista_data_set.index(random.choice(lista_data_set))
        lista_treino.append(lista_data_set[index])
        lista_data_set.pop(index)
        i += 1

    i = 0
    while i <= int(tamanho_data_set * 30 / 100):
        index = lista_data_set.index(random.choice(lista_data_set))
        lista_testes.append(lista_data_set[index])
        lista_data_set.pop(index)
        i += 1

    d = Dados(lista_treino, lista_testes)

    #Executa o algoritmo KNN
    d.KNN()

if __name__ == '__main__':
    Importar()
```

Para a implementação, segui a teoria vista em aula:

K-nearest neighbors (KNN)

- Para classificar um exemplo novo:
  1. Calcule a (dis)similaridade para todos os objetos de treinamento

Distância Euclidiana entre duas instâncias p_i e p_j definida como:

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

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

  2. Obtenha os K objetos mais similares (mais próximos)
  3. Classifique o objeto não visto na classe da maioria dos K vizinhos

Primeiramente a função “Importar()” é chamada. Nela importamos o arquivo CSV e alimentamos a classe com os dados:

```
def Importar():
    # Importa arquivo
    arquivo = open("Iris.csv")
    data_set = csv.reader(arquivo)

    # Cria lista com os dados
    lista_data_set = list(data_set)
    lista_data_set.pop(0) # Retirando cabeçalho
    tamanho_data_set = len(lista_data_set)
```

Depois, atribui aleatoriamente registros de teste e treino:

```
# Separa 30% dos dados para testes e 70% para treino
lista_treino = []
lista_testes = []

# Atribui aleatoriamente os registros para treino e teste com base no dataset
i = 1
while i < int(tamanho_data_set * 70 / 100):
    index = lista_data_set.index(random.choice(lista_data_set))
    lista_treino.append(lista_data_set[index])
    lista_data_set.pop(index)
    i += 1

i = 0
while i <= int(tamanho_data_set * 30 / 100):
    index = lista_data_set.index(random.choice(lista_data_set))
    lista_testes.append(lista_data_set[index])
    lista_data_set.pop(index)
    i += 1
```

A lógica usada foi a seguinte: Sorteio aleatoriamente um índice da lista principal (data_set), adiciono esse elemento na lista de treino/teste depois retiro esse da lista data_set, para que não seja sorteado de novo. Dessa forma a lista de testes fica diversa.

Inicialmente eu havia inserido os primeiros 104 registros na lista de treino e o restante na lista de teste, porém dessa forma percebi que o algoritmo ficava tendencioso, uma vez que até o registro 104 temos apenas três registros da classe “Iris-virginica”, o que é muito pouco para treinar o algoritmo.

Por fim, executo a função KNN():

```
d = Dados(lista_treino, lista_testes)

#Executa o algoritmo KNN
d.KNN()
```

Ela percorre cada elemento de teste, e para cada um, percorre todos os elementos de treino fazendo os cálculos de distância e os armazena na lista “distancia”:

```
# Percorre os registros de teste
i = 0
while i < len(self.testes):
    distancia = []

    # Percorre os treinos
    j = 0
    while j < len(self.treino):

        # Faz o somatório daquele teste com o registro de treino atual
        k = 1
        soma = 0
        while k < 5:
            soma = soma + (float(self.treino[j][k]) - float(self.testes[i][k]))**2
            k += 1

        # Adiciona o valor na lista
        distancia.append([math.sqrt(soma), self.treino[j][k], self.treino[j][0]])
        j += 1
```

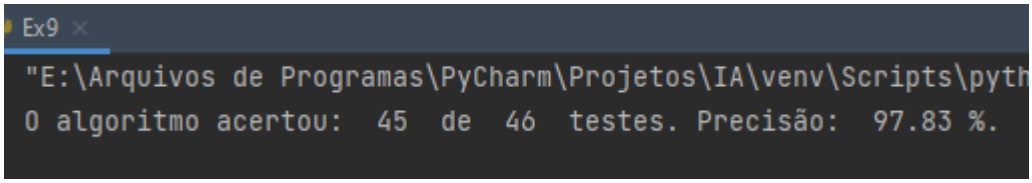
Depois de percorrido os elementos de treino, seleciona os três (pois o K utilizado é 3) elementos de distância que tiveram o menor resultado e verifica qual a classe mais recorrente neles. Depois disso, verifica se o resultado encontrado é correto:

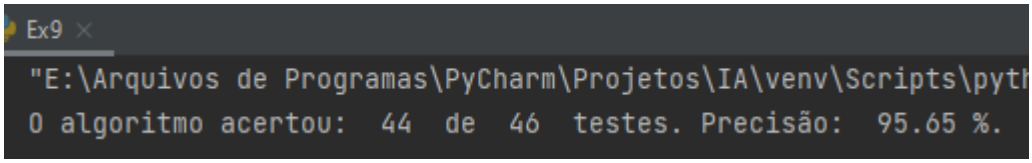
```
melhores = []
# Ordena os 3 registros que tiveram o melhor desempenho (K = 3)
for linha in sorted(distancia, key=lambda x: x[0])[:3]:
    melhores.append(linha[1])

# Se a classificação do registro teste for igual a classificação encontrada
if self.testes[i][5] == max(set(melhores), key=melhores.count):
    self.numero_acertos += 1

i += 1
```

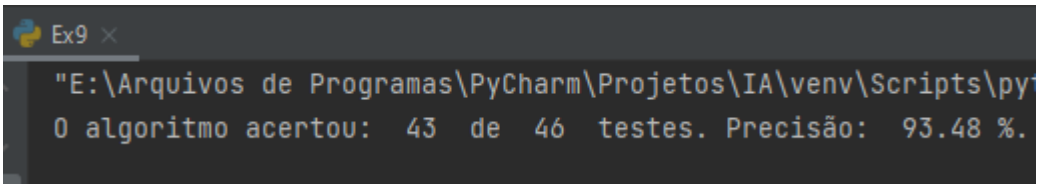
Depois de percorrido todos os testes, retorna para o usuário qual a precisão do algoritmo:

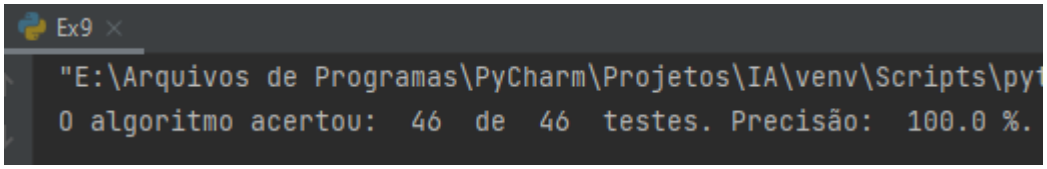
```
# Retorno
precisao = float("{:.2f}".format(100 * self.numero_acertos / len(self.testes)))
print("O algoritmo acertou: ", str(self.numero_acertos), " de ", str(len(self.testes)),
" testes. Precisão: ", precisao, "%.")
```

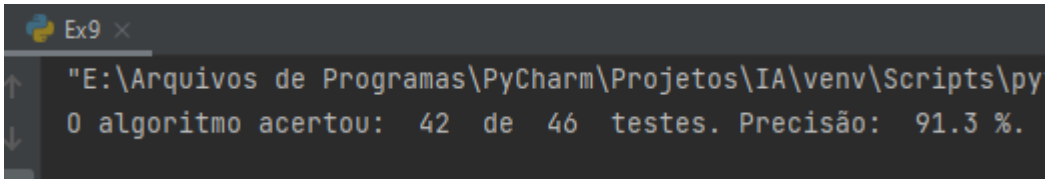
Como os elementos de treino são escolhidos aleatoriamente, a cada execução o algoritmo tem um desempenho diferente. A seguir 5 exemplos de prints de saída:











Nota-se que os resultados encontrados foram todos acima de 90%.