IA – 1° SEMESTRE DE 2022

9. EXERCÍCIO PRÁTICO – APRENDIZADO SUPERVISIONADO

Nome: Luiz Gustavo Alves Assis da Silva
RA: 149115

(1,0) Implemente o algoritmo KNN, Naive Bayes e Hunt e aplique no dataset IRIS: Não use bibliotecas prontas, implemente toda a lógica do algoritmo.

Separe aleatoriamente 70% dos dados para treino e 30% para teste e reporte com um print da saída qual a acurácia do algoritmo (número de acertos).

```
'''

#
#   AULA 9
#   APRENDIZADO SUPERVISIONADO
#   LINGUAGEM DE PROGRAMAÇÃO: Python
#
#   NOME: LUIZ GUSTAVO ALVES ASSIS DA SILVA
#   RA: 149115
#

'''

# BIBLIOTECAS UTILIZADAS

import pandas as pd
import numpy as np
import math
from sklearn.preprocessing import normalize
```

Temos um programa que implementa os seguintes algoritmos:

KNN (K-Vizinho mais próximo)
Naive Bayes (Classificador probabilístico)

O objetivo deste programa é utilizar o dataset Iris e separar aleatoriamente 70% dos dados para treino e 30% para teste e reportar com um print da saída qual a acurácia do algoritmo (número de acertos).

```python
def pre_processamento(data):

    x = data.drop([data.columns[-1]], axis = 1)
    y = data[data.columns[-1]]

    return x, y

def split(x, y, test_size, random_state):

    x_test = x.sample(frac = test_size, random_state = random_state)
    y_test = y[x_test.index]

    x_train = x.drop(x_test.index)
    y_train = y.drop(y_test.index)

    return x_train, x_test, y_train, y_test
```

Primeiramente, foram criadas duas funções que realizam o pré-processamento dos dados e o que separa os dados para treinamento e teste (responsável para predição).

```python
if __name__ == '__main__':

    data = pd.read_csv('Iris.csv')
    X, y = pre_processamento(data)

    x_train, x_test, y_train, y_test = split(X, y, test_size = 0.7, random_state = 21)

    print("Selecione a opção: 1 - KNN | 2 - Naive Bayes")
    input_op = int(input())
```

Na função main, a variável *data* recebe o dataset *Iris.*csv e são criadas quatro variáveis sendo elas: duas para treinamento (*x_train* e *y_train*) e duas para testagem (*x_test* e *y_test*)

Além disso, foi feito um menu no terminal para auxiliar o usuário a escolher os parâmetros dos algoritmos de acordo com a escolha inicial de opção, sendo a opção 1 – algoritmo KNN, opção 2 – algoritmo Naive Bayes.

```python
if (input_op == 1): # EXECUTAR ALGORITMO KNN

    print("Insere o seguinte parâmetro: k")
    k = int(input())
    acertos = 0

    x_train = normalize(x_train, axis=0)
    x_test = normalize(x_test, axis=0)

    y_train = [coluna for coluna in y_train]
    y_test = [coluna for coluna in y_test]

    for i in range(len(x_test)):
        KVizinhos = [0] * k

        for j in range(k):
            KVizinhos[j] = KNN(x_test[i], x_train, y_train)

        if (max(set(KVizinhos))) == y_test[i]:
            acertos += 1

    print('Acertos: ', acertos,'Precisão: ', acertos / len(y_test))
```

Caso a opção do usuário seja a de utilizar o algoritmo KNN, é preciso que o mesmo especifique o parâmetro K que será utilizado no algoritmo. Além disso, os dados *x_train* e *x_test* são normalizados e é feita uma manipulação nos dados *y_train* e *y_test* para falicitar o desenvolvimento do algoritmo adiante.

Em seguida, para cada iteração do loop é criado um vetor que armazena os K-Vizinhos mais próximos do conjunto de teste. Após isso, é feito uma checagem para verificar se a classe (Species) dos K-Vizinhos mais próximos é a mesma do conjunto de teste na posição i, caso verdadeiro o numero de acertos é incrementado. Por fim o número de acertos e da precisão é impresso na tela.

```python
def distancia_euclidiana(x_teste, x_train):

    dist = 0
    for i in range(len(x_teste)):
        dist += pow((x_teste[i] - x_train[i]), 2)
    return math.sqrt(dist)

def KNN(x_teste, x_train, y_train):

    dist_min = math.inf
    for i in range(len(x_train)):
        dist_atual = distancia_euclidiana(x_teste, x_train[i])

        if dist_atual < dist_min:
            dist_min = dist_atual
            classe = y_train[i]

    return classe
```

A função distancia_euclidiana, como o nome diz, calcula a distância euclidiana do conjunto de teste (*x_test*) e do conjunto de treinamento (*x_train*) e retorna o valor da distância.

A função KNN encontra a distância mínima de todos atributos do conjunto de treinamento e retorna uma classe que será atribuída aos KVizinhos na função principal.

```
Selecione a opção: 1 - KNN | 2 - Naive Bayes
1
Insere o seguinte parâmetro: k
5
Acertos:  76 Precisão:  0.7238095238095238

Process finished with exit code 0
```

```
elif(input_op == 2): # EXECUTAR ALGORITMO NAIVE BAYES

    y_pred = naive_bayes_gaussiano(x_test, y_test)

    acertos = sum(y_pred == y_test)
    print('Acertos: ', acertos,'Precisão: ', acertos / len(y_test))
```

Caso a opção do usuário seja a de utilizar o algoritmo Naive Bayes, é criado uma variável *y_pred* que armazena os resultados das classes previstas na função *naive_bayes_gaussiano*. Por último, o número de acertos e da precisão é impresso na tela.

```
def priori(y):

    prior = []
    for classe in np.unique(y):
        prior.append(sum(y == classe) / len(y))

    return prior

def probabilidade_condicional(x, y):

    probabilidade_media = {}
    probabilidade_var = {}

    for atributo in x:
        probabilidade_media[atributo] = {}
        probabilidade_var[atributo] = {}

        for classe in np.unique(y):
            media = x[atributo][y[y == classe].index.values.tolist()].mean()
            var = x[atributo][y[y == classe].index.values.tolist()].var()
            probabilidade_media[atributo][classe] = media
            probabilidade_var[atributo][classe] = var

    return probabilidade_media, probabilidade_var
```

A função *priori* calcula as probabilidades a priori P(c) de todas as classes do dataset e, por fim, é retornado essas probabilidades.

A função *probabilidade_condicional* calcula as probabilidades condicionais P(c | x) dos atributos do dataset e, por fim, é retornado essas probabilidades

```
def distribuicao_normal(media, var, valor_atributo):
    return (1 / math.sqrt(2 * math.pi * var)) * np.exp(-(valor_atributo - media)**2 / (2*var))
```

A função *distribuicao_normal* calcula a distribuição normal (ou gaussiana) dado os parâmetros: média e variância de uma dada probabilidade condicional e o valor do atributo dessa probabilidade.

```python
def naive_bayes_gaussiano(X, y):

    Y_pred = []
    prior = priori(y)
    probabilidade_media, probabilidade_var = probabilidade_condicional(X, y)
    tabela = np.array(X)
    for linha in tabela:
        resultado = {}

        for classe in np.unique(y):
            probabilidade = 1
            indice_prior = 0

            for atributo, valor_atributo in zip(X, linha):
                media = probabilidade_media[atributo][classe]
                var = probabilidade_var[atributo][classe]
                probabilidade *= distribuicao_normal(media, var, valor_atributo)

            resultado[classe] = probabilidade * prior[indice_prior]
            indice_prior += 1

        classe = max(resultado, key = resultado.get)
        Y_pred.append(classe)

    return Y_pred
```

Temos a função *naive_bayes_gaussiano* que implementa o algoritmo naive bayes para dados contínuos.

Primeiramente, calculamos as probabilidades a priori e condicionais das classes e atributos, respectivamente, e o algoritmo inicializa o loop que itera cada linha do dataset. Em seguida, a cada iteração loop é calculado a probabilidade de todos atributos de uma linha do dataset por meio da distribuição normal e após isso é armazenado o resultado do produto da probabilidade acumulada e a probabilidade a priori.

Por último, a variável *classe* armazena a classe resultante da probabilidade máxima dos resultados encontrados no passo anterior.

```
Selecione a opção: 1 - KNN | 2 - Naive Bayes
2
Acertos:  104 Precisão:  0.9904761904761905

Process finished with exit code 0
```