

Terceiro trabalho prático de reconhecimento de padrões

K vizinhos mais próximos

Introdução

Neste trabalho, será implementado o método dos K vizinhos mais próximos, e serão criados testes para avaliação qualitativa de desempenho, variando parâmetros do agrupamento.

In [2]:

```
import numpy as np
from numpy.random import normal, uniform
import matplotlib.pyplot as plt
from sklearn.metrics import pairwise_distances
from matplotlib import cm
from matplotlib.ticker import LinearLocator
```

1- O desenvolvimento do método KNN será abordado posteriormente, após a criação de entradas necessárias para executar a implementação.

2 - Implementação gaussiana 2d

Aqui é implementado e testado o método de gerar uma gaussiana 2d, utilizando 2 distribuições unidimensionais, para cada dimensão.

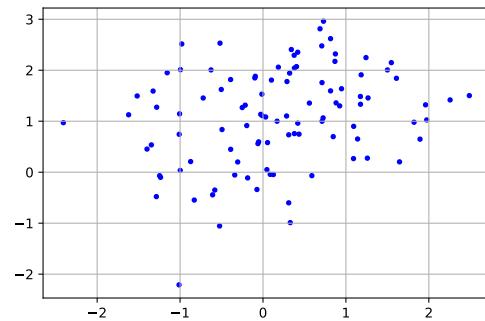
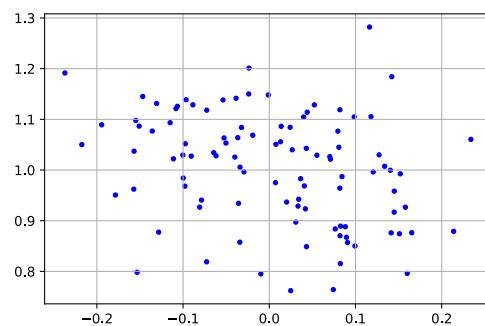
In [3]:

```
def normal_2d(N =100, mu_x=0, mu_y=0, sigma_x=1, sigma_y=1):
    return (normal(mu_x, sigma_x, N), normal(mu_y, sigma_y, N))
```

In [4]:

```
mu_x, mu_y, sigma, N = 0, 1, 0.1, 100 # mean and standard deviation
X, Y = normal_2d(mu_x, sigma, N), normal(mu_y, sigma, N)
fig,ax = plt.subplots(1)
ax.grid()
ax.scatter(X,Y,marker='.',color='b',linewidths=1)
plt.show()

X, Y = normal_2d(N, mu_x, mu_y)
fig,ax = plt.subplots(1)
ax.grid()
ax.scatter(X,Y,marker='.',color='b',linewidths=1)
plt.show()
```



3 - Quatro grupos de gaussianas bidimensionais

Neste ponto são criadas as quatro gaussianas bidimensionais, cada uma possuindo uma cor. Possuem desvio padrão de 0.3 e seus centros são:

- grupo 1, de centro (0,0) classificado como 1
- grupo 2, de centro (1,1) classificado como 1
- grupo 3, de centro (0,1) classificado como -1
- grupo 4, de centro (1,0) classificado como -1

Optou-se em criar somente 2 labels, pois na métrica de KNN, é utilizado função sinal do somatório, o que possibilita somente dois grupos.

Para um número maior de labels como {0,1,2,3}, necessita uma alteração do cálculo da métrica de classificação, como utilizar média em vez de função sinal.

In [5]:

```
x = y = np.arange(5)

fig,ax = plt.subplots(1)
ax.grid()
sigma=0.4

X,Y = normal_2d(N =100, mu_x=0, mu_y=0, sigma_x=sigma, sigma_y=sigma)
ax.scatter(X,Y,marker='.',color='b',linewidths=1)

G0 = np.dstack((X,Y))[0]
classification_0 = np.ones(G0.shape[0])
G0_classified = np.column_stack((G0,classification_0))

X,Y = normal_2d(N =100, mu_x=1, mu_y=1, sigma_x=sigma, sigma_y=sigma)
ax.scatter(X,Y,marker='.',color='g',linewidths=1)

G1 = np.dstack((X,Y))[0]
classification_1 = np.ones(G1.shape[0])
G1_classified = np.column_stack((G1,classification_1))

X,Y = normal_2d(N =100, mu_x=0, mu_y=1, sigma_x=sigma, sigma_y=sigma)
ax.scatter(X,Y,marker='.',color='r',linewidths=1)

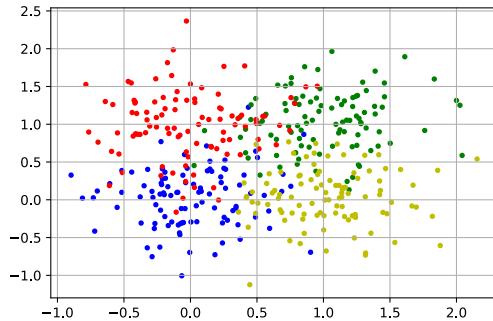
G2 = np.dstack((X,Y))[0]
classification_2 = -1 * np.ones(G2.shape[0])
G2_classified = np.column_stack((G2,classification_2))

X,Y = normal_2d(N =100, mu_x=1, mu_y=0, sigma_x=sigma, sigma_y=sigma)
ax.scatter(X,Y,marker='.',color='y',linewidths=1)

G3 = np.dstack((X,Y))[0]
classification_3 = -1 * np.ones(G3.shape[0])
G3_classified = np.column_stack((G3,classification_3))

plt.show()

scatter = np.concatenate([G0, G1, G2, G3])
classification_all_neighbours = np.concatenate([classification_0, classification_1, classification_2, classification_3])
y_train = classification_all_neighbours
x_train = scatter
scatter.shape
```



Out[5]:

(400, 2)

4 - Conjunto de testes

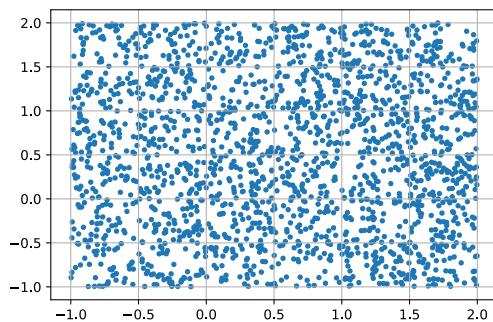
Aqui é criado o conjunto de teste utilizando distribuição uniforme entre $x,y \in [-1, 2]$.

Optou-se em utilizar mais amostras de teste (2000), para que seja melhor identificável as fronteiras de classificação.

In [6]:

```
N_test = 2000

x=uniform(-1,2,N_test)
y=uniform(-1,2,N_test)
test_samples = np.dstack((x,y))[0]
X_test = test_samples
fig,ax = plt.subplots(1)
ax.grid()
ax.scatter(x,y,marker='.',linewidths=1)
plt.show()
```



1- Implementação do método KNN

Aqui é implementado o método KNN. Foram utilizadas operações vetorizadas para ganho de desemnho. Logo se opera com matrizes, evitando-se loops.

Consiste em:

- 0 - Início
- 1 - Criação da matriz de distâncias par a par das amostras de teste e amostras de treino;
- 2 - ordenamento das k amostras de treino mais proximas e obtenção de seus índices;
- 3 - obtenção da afinidade das k amostras de treino mais proximas das de teste de cada ponto;
- 4 - predição, dada pelo sinal da soma da classificação dos k vizinhos mais proximos, multiplicado por suas afinidades;

$$prediction = \text{sign} \sum^K \alpha_i y_{train}$$

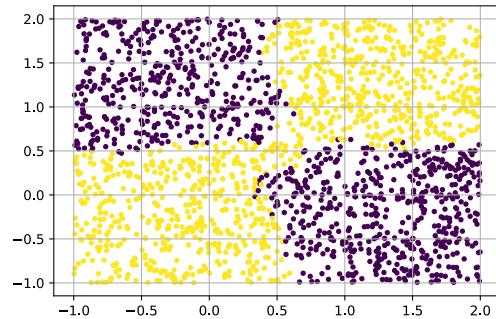
- 5 - Retorna predição

O teste demonstra o correto funcionamento da implementação, seguindo o agrupamento dos grupos.

Para um número maior de labels como {0,1,2,3}, necessita uma alteração do calculo da métrica de classificação, como utilizar média em vez de função sinal.

In [7]:

```
def knn(classification, samples, test, k):  
    # vector of euclidean distance between test point and each of all training points  
    dist_matrix = pairwise_distances(samples, test).T  
  
    # Sort k nearest neighbours of training set  
    k_neighbours = np.argsort(dist_matrix)[:,0:k]  
  
    # Get weight of each neighbour, given by affinity  
    k_dist = [ 1/dist_matrix[i,v] for i, v in enumerate(k_neighbours)]  
    prediction = [np.sign(np.sum(classification[t]*k_neighbours)) for t in k_neighbours]  
fig,ax = plt.subplots(1)  
ax.grid()  
ax.scatter(test[:,0],test[:,1],c=prediction, marker='.',linewidths=1)  
plt.show()  
return prediction  
y_prediction = knn(y_train, x_train, test_samples, k=20)
```



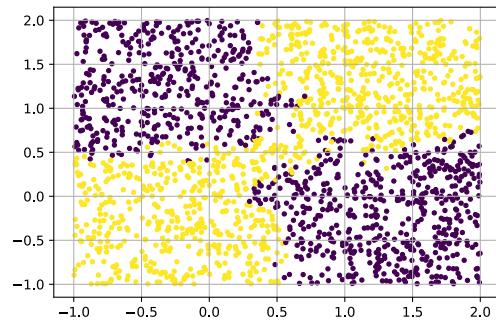
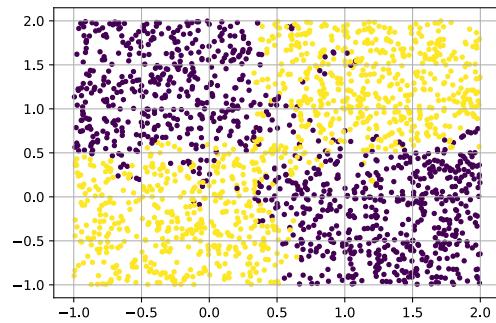
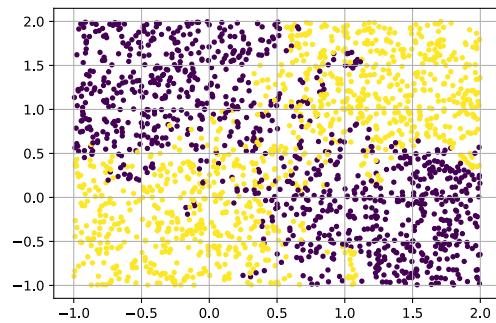
7 - Criação de loop variando K

Neste experimento é realizado um loop variando K entre os valores de K ∈ {2, 4, 8}

É possível observar que aumentando K, há uma melhora na separação dos pontos, classificando aos clusters que pertencem.

In [8]:

```
for k in (2,4,8):
    y_prediction = knn(y_train, x_train, test_samples, k=k)
```



8 - Experimento variando o desvio padrão

Aqui é realizado o experimento variando tanto K quanto sigma.
A consequência é o aumento de região de interseção entre as distribuições de pontos de classificação dos agrupamentos, dificultando a classificação de amostras de testes próximas às fronteiras.

Contudo para $K = 8$ identificou-se satisfatoriamente, mesmo para onde há interseções.

Um

In [9]:

```
def generate_train_test(sigma):
    x = y = np.arange(5)

    fig,ax = plt.subplots(1)
    ax.grid()
    sigma=0.4

    X,Y = normal_2d(N =100, mu_x=0, mu_y=0, sigma_x=sigma, sigma_y=sigma)
    ax.scatter(X,Y,marker='.',color='b',linewidths=1)

    G0 = np.dstack((X,Y))[0]
    classification_0 = np.ones(G0.shape[0])
    G0_classified = np.column_stack((G0,classification_0))

    X,Y = normal_2d(N =100, mu_x=1, mu_y=1, sigma_x=sigma, sigma_y=sigma)
    ax.scatter(X,Y,marker='.',color='g',linewidths=1)

    G1 = np.dstack((X,Y))[0]
    classification_1 = np.ones(G1.shape[0])
    G1_classified = np.column_stack((G1,classification_1))

    X,Y = normal_2d(N =100, mu_x=0, mu_y=1, sigma_x=sigma, sigma_y=sigma)
    ax.scatter(X,Y,marker='.',color='r',linewidths=1)

    G2 = np.dstack((X,Y))[0]
    classification_2 = -1 * np.ones(G2.shape[0])
    G2_classified = np.column_stack((G2,classification_2))

    X,Y = normal_2d(N =100, mu_x=1, mu_y=0, sigma_x=sigma, sigma_y=sigma)
    ax.scatter(X,Y,marker='.',color='y',linewidths=1)

    G3 = np.dstack((X,Y))[0]
    classification_3 = -1 * np.ones(G3.shape[0])
    G3_classified = np.column_stack((G3,classification_3))

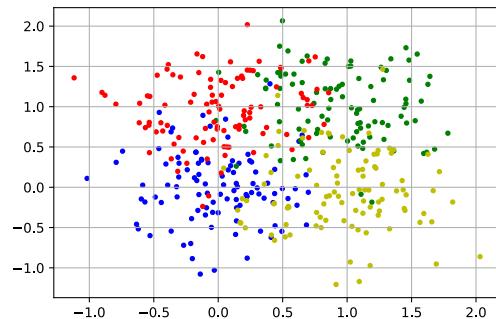
    plt.show()

    scatter = np.concatenate([G0, G1, G2, G3])
    classification_all_neighbours = np.concatenate([classification_0, classification_1, classification_2, classification_3])
    y_train = classification_all_neighbours
    x_train = scatter
    scatter.shape
    for k in (2,4,8):
        print(f"K={k} :")
        y_prediction = knn(y_train, x_train, test_samples, k=k)
```

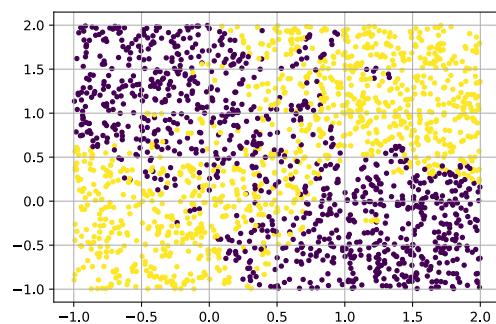
In [10]:

```
for s in (0.3, 0.5, 0.7):
    print(f"sigma={s}")
    generate_train_test(sigma=s)
    print('\n\n')
```

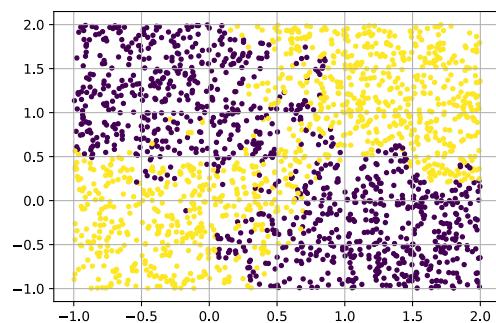
sigma=0.3



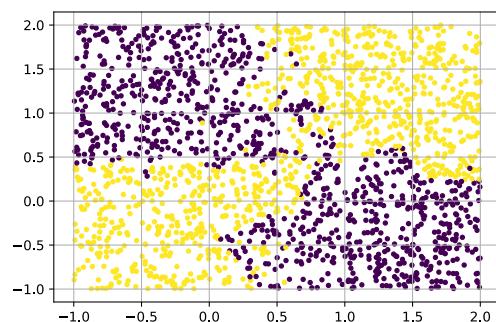
k=2 :



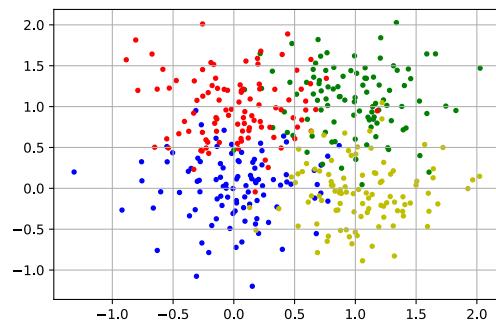
k=4 :



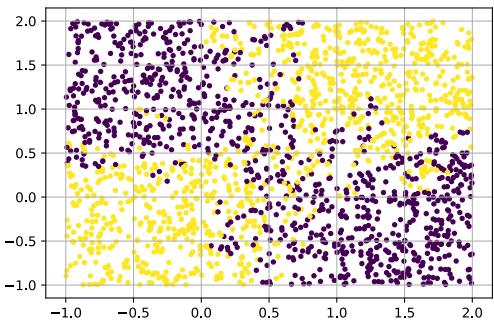
k=8 :



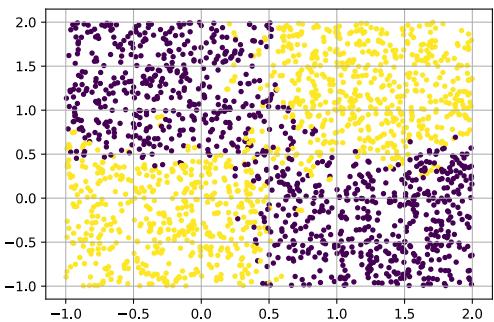
sigma=0.5



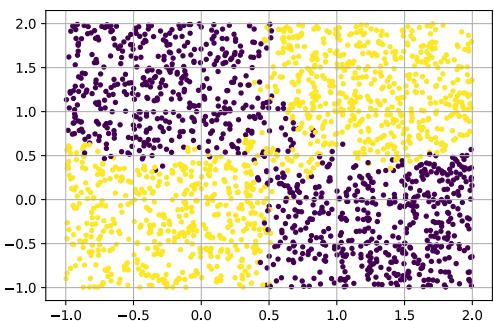
k=2 :



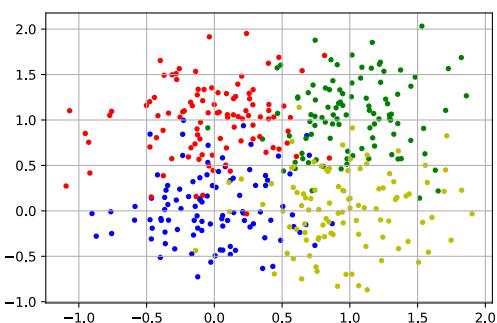
k=4 :



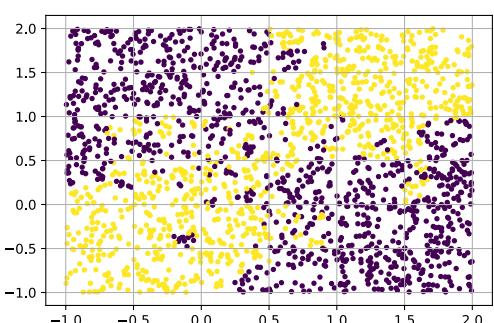
k=8 :



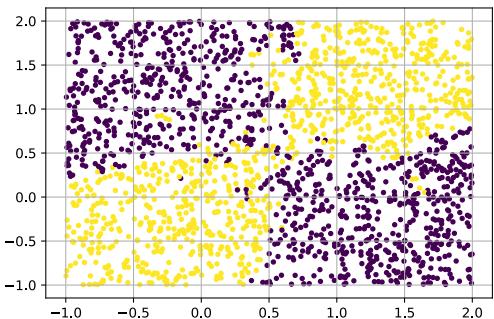
sigma=0.7



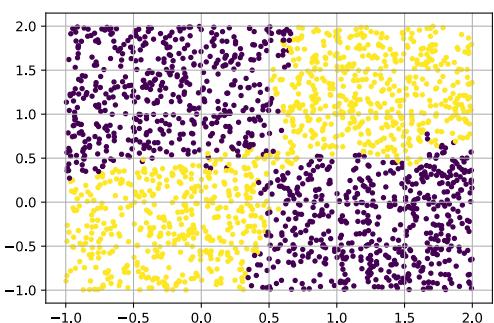
k=2 :



k=4 :



k=8 :



Conclusão

Neste trabalho foi possível realizar uma implementação do algoritmo de classificação K vizinhos mais próximos. O método, embora simples, classificou satisfatoriamente, dado o ajuste adequado do parâmetro K.