KNN (K - *Nearest Neighbors*)

- **Objetivos**

Assimilar os fundamentos da técnica de KNN (K - *Nearest Neighbors*) (teoria) por meio da execução de exemplos baseado em linguagem Python.

- **Revisão teórica**

KNN (K — *Nearest Neighbors*) é um dos muitos algoritmos (de aprendizagem supervisionada) usado no campo de *data mining* e *machine learning*, ele é um classificador onde o aprendizado é baseado “no quão similar” é um dado (um vetor) do outro.
O treinamento é formado por vetores de n dimensões.

O funcionamento do KNN é bem simples, imagine que você tem dados de:

- Imagens de bolinhas roxas;
- Imagens de bolinhas amarelas; e
- Uma imagem de uma bolinha que você não sabe se é roxa ou amarela, mas tem todos os dados sobre ela.

Dados: número RGB das cores de cada pixel.

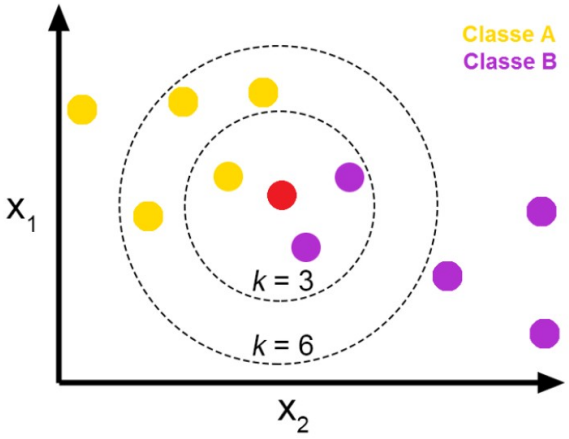
O KNN executa um cálculo matemático para medir a distância entre os dados para fazer sua classificação. Por exemplo: Euclidiana, Manhattan, Minkowski, Ponderada e etc.

- **METODOLOGIA da técnica KNN**

1. Recebe um dado não classificado;
2. Mede a distância (Euclidiana, Manhattan, Minkowski ou Ponderada) do novo; dado com todos os outros dados que já estão classificados;
3. Obtém as X (no caso essa variável X é o parâmetro K) menores distâncias;
4. Verifica a classe de cada da um dos dados que tiveram a menor distância e conta a quantidade de cada classe que aparece;
5. Toma como resultado a classe que mais apareceu dentre os dados que tiveram as menores distâncias; e

6. Classifica o novo dado com a classe tomada como resultado da classificação.

No exemplo da revisão, a aplicação da metodologia do KNN seria:



No exemplo há um dado não classificado (em vermelho) e todos os seus outros dados já classificados (amarelo e roxo) cada um com sua classe (A e B).

Então é calculado a distância do novo dado com todos os outros pra saber quais estão mais próximos (quais têm as menores distâncias), feito isso são pegados 3 (ou 6) dos dados mais próximos e verificado qual é a classe que mais aparece.

No caso da imagem acima, os dados mais próximos do novo dado são aqueles que estão dentro do primeiro círculo (de dentro pra fora), e ali dentro há 3 outros dados (já classificados), sendo que a classe predominante ali dentro é o roxo, pois há 2 bolinhas roxas e apenas 1 amarela.

O novo dado que antes não estava classificado, agora é classificado como roxo.

- **Vantagens e desvantagem da técnica KNN**

Pros:

- Sem suposições sobre os dados;
- Algoritmo simples - fácil de entender; e
- Pode ser usado para classificação e regressão.

Contras:

- Requisito de alta memória - Todos os dados de treinamento devem estar presentes na memória para calcular os K vizinhos mais próximos;
- Sensível a recursos irrelevantes; e
- Sensível à escala dos dados, pois estamos calculando a distância até os K pontos mais próximos.

ATIVIDADE 1

Vamos primeiro criar o nosso próprio conjunto de dados. Aqui você precisa de dois tipos de atributos ou colunas em seus dados: Características e alvo. A razão para dois tipos de coluna é a "natureza supervisionada do algoritmo KNN".

```
# Criar um dataset
# Primeira Característica
clima=['Ensolarado', 'Ensolarado', 'Nublado', 'Chuvoso', 'Chuvoso', 'Chuvoso', 'Nublado',
'Ensolarado', 'Ensolarado', 'Chuvoso', 'Ensolarado', 'Nublado', 'Nublado', 'Chuvoso']

# Segunda Característica
temp=['Quente', 'Quente', 'Quente', 'Suave', 'Legal', 'Legal', 'Legal', 'Suave',
'Legal', 'Suave', 'Suave', 'Suave', 'Quente', 'Suave']

# Variavel alvo
brincar=['Não','Não','Sim','Sim','Sim','Não','Sim','Não','Sim','Sim','Sim','Sim','Sim','Não']

# Importando LabelEncoder
from sklearn import preprocessing
#criando labelEncoder
le = preprocessing.LabelEncoder()

# Convertendo string labels para numeros.
clima_encoded = le.fit_transform(clima)
temp_encoded = le.fit_transform(temp)
print(clima_encoded)
print(temp_encoded)

# convertendo string labels para numeros
alvo = le.fit_transform(brincar)
print(alvo)

# Combinando clima e temp em um unica lista de tuplas
carac=list(zip(clima_encoded,temp_encoded))
carac
```

Agora vamos importar a técnica KNN para o nosso projeto em Python e treinar o algoritmo com o *dataset* previamente criado.

```
from sklearn.neighbors import KNeighborsClassifier
modelo = KNeighborsClassifier(n_neighbors=3)

# Treinando o modelo usando os ajustes de treinamento.
modelo.fit(carac,alvo)

#Predito
predito = modelo.predict([[0,2]]) # 0:Nublado, 2:Suave
print(predito)
```

ATIVIDADE 2

Vamos dar uma olhada em como podemos classificar os dados usando o algoritmo K-*Nearest Neighbours* em Python.

Para este exemplo, usaremos novamente o conjunto de dados de câncer de mama do módulo sklearn.datasets.

Precisamos começar importando as bibliotecas de procedimentos.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()

from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

O conjunto de dados classifica os tumores em duas categorias (malignos e benignos) e contém cerca de 30 características.

Devemos codificar dados categóricos para que sejam interpretados pelo modelo (ou seja, maligno = 0 e benigno = 1).

```
cancer_mama = load_breast_cancer()
X = pd.DataFrame(cancer_mama.data, columns=cancer_mama.feature_names)
X = X[['mean area', 'mean compactness']]
X
y = pd.Categorical.from_codes(cancer_mama.target, cancer_mama.target_names)
y = pd.get_dummies(y, drop_first=True)
y
```

Agora vamos treinar o algoritmo de KNN com o *dataset* de câncer de mama.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
y_pred
```

E por fim plotar os gráficos com os resultados obtidos pelo algoritmo de KNN treinado com o *dataset* de câncer de mama.

```
sns.scatterplot(
    x='mean area',
    y='mean compactness',
    hue='benign',
    data=X_test.join(y_test, how='outer')
)
plt.scatter(
    X_test['mean area'],
    X_test['mean compactness'],
    c=y_pred,
    cmap='coolwarm',
    alpha=0.7
)
confusion_matrix(y_test, y_pred)
```

ATIVIDADE 3

Vamos dar uma olhada em como garantir a confiabilidade da classificação realizada pelo algoritmo K-NN.

Para este exemplo, usaremos novamente o conjunto de dados Iris do módulo sklearn.datasets.

Precisamos começar pelas etapas anteriormente descritas para a implementação do algoritmo K-NN e as suas métricas.

```
# bibliotecas
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# dataset
iris = load_iris()

# caracteristicas e alvo
X = iris.data
y = iris.target

# treino e teste
(X_train, X_test, y_train, y_test) = train_test_split(X,y)

# instanciando o modelo
modelo = KNeighborsClassifier()

# treinando o modelo utilizando o conjunto de treino
modelo.fit(X_train,y_train)

# validando o modelo utilizando o conjunto de teste
precisao = str(round(modelo.score(X_test,y_test) * 100, 2))+"%"

# imprimindo o resultado
print("A acurácia do modelo k-NN foi",precisao)

# predizendo o teste
```

```
y_pred = modelo.predict(X_test)

# comparando predição com o real
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Na matriz de confusão é apresentado que sempre acertamos 100% das setosas. Provavelmente por elas estarem geometricamente mais separadas das outras classes quando suas medidas são vistas como coordenadas de vetores no espaço. Podemos visualizar graficamente essa hipótese utilizando o exemplo abaixo:

```
# bibliotecas
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# dataset
iris = load_iris()

# dataset para pandas dataframe
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['Species'] = iris.target

# mostra graficos
sns.pairplot(df, hue='Species', vars=iris.feature_names)
plt.show()
```

Podemos comprovar a confiabilidade da classificação realizada pelo algoritmo K-NN por meio de um histograma contendo as médias e o desvio padrão para milhares de treinamento. Abaixo um exemplo:

```
# bibliotecas
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import Normalizer
import numpy as np

# remover warnings
import warnings
warnings.filterwarnings("ignore")

# dataset
iris = load_iris()

# caracteristicas e alvo
X = iris.data
y = iris.target

# normalizando
scaler = Normalizer()
scaler.fit(X)
X = scaler.transform(X)

scores = []
for i in range(2000):
    X_train, X_test, y_train, y_test = train_test_split(X,y)
    model = KNeighborsClassifier()
    model.fit(X_train,y_train)
    precisao = model.score(X_test,y_test)
    scores.append(precisao)

print("Média: {:.2f}%".format(np.mean(scores)*100))
print("Desvio padrão: {:.2f}%".format(np.std(scores)*100))

import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(scores)
plt.yticks([])
plt.title("Acurácias do k-NN")
plt.show()
```