AULA PRÁTICA DE KNN (K - *Nearest Neighbors*)

- **Objetivos**

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

- **Revisão teórica**

KNN (K - *Nearest Neighbors*) é um dos muitos algoritmos de aprendizagem supervisionada utilizados no campo de *machine learning*. É um algoritmo que pode ser utilizado tanto para problemas de classificação quanto para problemas de regressão, onde o aprendizado é baseado "no quão similar" é um dado de outro. Como dado, considera-se um vetor formado por todas as variáveis preditoras, representando uma linha da tabela.

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

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

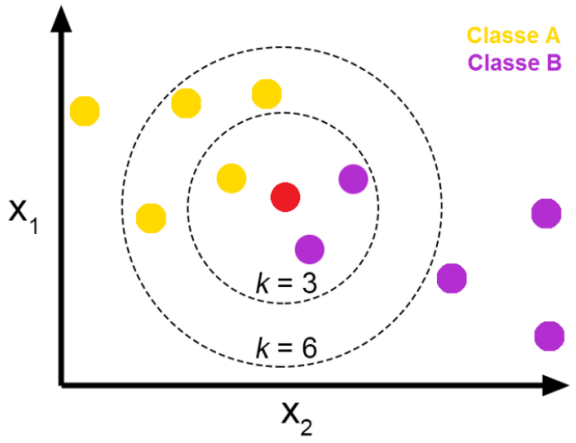
O KNN executa um cálculo matemático para medir a distância entre os dados para fazer sua classificação. Por exemplo: Euclidiana, Manhattan, Minkowski, Ponderada e etc.

Para uma leitura mais completa sobre o cálculo das possíveis distâncias tem-se o link abaixo:
https://mineracaodedados.wordpress.com/tag/distancia-euclidiana/#:~:text=Dist%C3%A2ncia%20Euclideana%3A%20%E2%88%9A((x,%E2%80%93%20y2)%C2%B2).

- **METODOLOGIA da técnica KNN**

1. Recebe um dado não classificado;
2. Calcula a distância do novo dado com todos os outros dados que já estão classificados;
3. Obtém as K menores distâncias (essa é a variável que dá nome ao método KNN);
4. Verifica a classe de cada um dos K dados que tiveram a menor distância e conta a quantidade de cada classe que aparece;
5. Atribui a classe que mais aparece como a classificação do dado não classificado.

No exemplo das bolinhas, 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 para saber quais estão mais próximos (quais têm as menores distâncias), feito isso são considerados 3 (ou 6) dos dados mais próximos e é verificado qual é a classe que mais aparece.

No caso da imagem acima, considerando K = 3, os dados mais próximos do novo dado são aqueles que estão dentro do primeiro círculo (de dentro pra fora), e ali dentro existem 3 outros dados (já classificados), sendo que a classe predominante é a classe B (roxo), pois existem 2 bolinhas roxas e apenas 1 amarela. O novo dado que antes não estava classificado, agora é classificado como roxo.

Entretanto, se considerarmos K = 6 (o segundo círculo apresentado no gráfico), a classe predominante seria a classe A (amarelo) pois neste caso existem 2 bolinhas roxas contra 4 bolinhas amarelas e o novo dado, que antes não estava classificado, agora seria classificado como amarelo.

Exemplo 1 - Classificação com KNN em Python

A seguir temos um exemplo de um problema de **classificação** usando o algoritmo K-*Nearest Neighbours* em Python. Para este exemplo, usaremos o conjunto de dados de câncer de mama do módulo sklearn.datasets. Precisamos começar importando as bibliotecas e métodos necessários.

```
import pandas as pd

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

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

Devemos carregar a base de dados e fazer a divisão entre variáveis preditoras (X) e variável target (y). A base de dados load_breast_cancer possui cerca de 30 variáveis preditoras médicas, ou seja, 30 colunas.

```
cancer_mama = load_breast_cancer()
X = pd.DataFrame(cancer_mama.data, columns=cancer_mama.feature_names)
print(X.shape)
print(X.head())
```

O conjunto de dados classifica os tumores em duas categorias (malignos e benignos). Devemos transformar dados categóricos em numéricos para que sejam interpretados pelo modelo (ou seja, maligno = 0 e benigno = 1).

```
y = pd.Categorical.from_codes(cancer_mama.target,
cancer_mama.target_names)
print(y)
# A função get_dummies() substitui as possíveis classificações (classes)
# por um valor numérico. Neste caso, malignant = 0 e benign = 1
y = pd.get_dummies(y, drop_first=True)
print(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 = Previsões realizadas pelo algoritmo
y_pred = knn.predict(X_test)
print(y_pred)
```

E por fim, apresentamos os resultados obtidos pelo algoritmo de KNN treinado com o *dataset* de câncer de mama. Utilizaremos uma matriz de confusão e um relatório de classificação (também chamado de relatório de qualidade).

```
# Retorna a matriz de confusão mostrando os acertos e erros da nossa
previsão
print(confusion_matrix(y_test, y_pred))

# Retorna o relatório de classificação, apontando:
# acurácia, precisão, revocação e F1.
print(classification_report(y_test, y_pred, target_names=['Maligno',
'Benigno']))
```

Exemplo 2 - Classificação com KNN em Python

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 KNN 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()

# preditoras 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
acuracia = str(round(modelo.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo KNN foi",acuracia)

# aplicando as previsões do algoritmo para o conjunto de 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))
```

No relatório de classificação é apresentado que sempre acertamos 100% das setosas (classe 0). 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 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()
```