Estudo Dirigido KNN

Em resumo podemos dizer que o KNN:

KNN (K-Nearest Neighbors) é um algoritmo de classificação (ou regressão) que funciona assim:

1. **Dado um ponto novo**, ele calcula a **distância** dele para todos os pontos do conjunto de dados.

2. Encontra os **K vizinhos mais próximos** (por exemplo, os 3 mais próximos).

3. Verifica a **classe mais comum** entre esses vizinhos.

4. **Classifica** o novo ponto com essa classe.

Um exemplo de código simples:

```
# Dados de treino (features + classes)

treino = [

   ([1, 2], 0),

   ([2, 3], 0),

   ([3, 3], 0),

   ([6, 5], 1),

   ([7, 7], 1),

   ([8, 6], 1)

]

# Função para calcular a distância euclidiana

def distancia(p1, p2):

   return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2) ** 0.5
```

```python
# Função KNN
def knn(novo_ponto, treino, k=3):
    # Calcula distâncias para cada ponto do treino
    distancias = []
    for ponto, classe in treino:
        d = distancia(novo_ponto, ponto)
        distancias.append((d, classe))
    # Ordena pelas distâncias
    distancias.sort(key=lambda x: x[0])
    # Pega os k vizinhos mais próximos
    k_vizinhos = distancias[:k]
    # Conta as classes
    contagem = {}
    for _, classe in k_vizinhos:
        if classe not in contagem:
            contagem[classe] = 0
        contagem[classe] += 1
    # Retorna a classe com mais votos
    return max(contagem, key=contagem.get)
# Teste com um novo ponto
novo = [5, 5]
classe_prevista = knn(novo, treino, k=3)
print(f"O ponto {novo} foi classificado como classe {classe_prevista}")
```

Saída esperada:

```
O ponto [5, 5] foi classificado como classe 1
```

E como ficaria o uso do KNN com a base Titanic usando validação cruzada e a biblioteca pandas?

Importações

```
import pandas as pd
import numpy as np
```

Importa as bibliotecas `pandas` e `numpy` para manipulação de dados e cálculos numéricos.

```
import seaborn as sns

df = sns.load_dataset('titanic')
```

Usa o `seaborn` para carregar automaticamente o dataset **Titanic** em um DataFrame `df`.

Pré-processamento

```
df = df[['survived', 'pclass', 'sex', 'age', 'fare']]
```

#Seleciona apenas algumas colunas relevantes para o modelo:

- `survived`: variável alvo (0 = não sobreviveu, 1 = sobreviveu)
- `pclass`, `sex`, `age`, `fare`: variáveis preditoras (features).

```
df.dropna(inplace=True)
```

Remove linhas com valores ausentes (`NaN`) para evitar problemas nos cálculos.

```
df['sex'] = df['sex'].map({'female': 0, 'male': 1})
```

Codifica a variável categórica `sex` para valores numéricos:

female → 0
male → 1

Separando variáveis preditoras e alvo

```
X = df[['pclass', 'sex', 'age', 'fare']].values

y = df['survived'].values
```

#Cria duas variáveis:

- `X`: matriz de características (variáveis preditoras)
- `y`: vetor alvo (variável que queremos prever)

Normalização

```
X = (X - X.mean(axis=0)) / X.std(axis=0)
```

#Normaliza os dados para que cada coluna (feature) tenha média 0 e desvio padrão 1. Isso evita que variáveis com escalas diferentes distorçam a distância no KNN.

Função para calcular distância euclidiana

```
def euclidean_distance(x1, x2):

    return np.sqrt(np.sum((x1 - x2) ** 2))
```

#Calcula a **distância euclidiana** entre dois vetores (pontos) `x1` e `x2`.

Função para prever com KNN

```
def knn_predict(X_train, y_train, x_test, k=10):

    distances = [euclidean_distance(x_test, x_train) for x_train in
X_train]
```

#Calcula a distância entre o ponto de teste `x_test` e todos os pontos de treino `X_train`.

```
   k_indices = np.argsort(distances)[:k]
```

Ordena os índices pelas distâncias e seleciona os `k` mais próximos.

```
   k_nearest_labels = y_train[k_indices]
```

Pega as **classes** dos `k` vizinhos mais próximos.

```
   counts = np.bincount(k_nearest_labels)
```

Conta quantas vezes cada classe aparece entre os vizinhos.

```
   return np.argmax(counts)
```

Retorna a classe com mais votos (a predição final).

Validação Cruzada (K-Fold)

```
def k_fold_cross_validation(X, y, k_folds=10, k_neighbors=5):

    fold_size = len(X) // k_folds
```

```
    indices = np.arange(len(X))
    np.random.shuffle(indices)
```

- Divide os dados em `k_folds` partes.
- Embaralha os índices dos dados.

```
    scores = []
```

Lista para guardar as **acurácias** de cada "fold".

```
    for fold in range(k_folds):
        start = fold * fold_size
        end = start + fold_size
        val_indices = indices[start:end]
        train_indices = np.concatenate((indices[:start],
indices[end:]))
```

Separa os dados de validação (`val_indices`) e treino (`train_indices`) para o fold atual.

```
        X_train, y_train = X[train_indices], y[train_indices]
        X_val, y_val = X[val_indices], y[val_indices]
```

Cria os conjuntos de treino e validação para este fold.

```
        correct = 0
        for i in range(len(X_val)):
            pred = knn_predict(X_train, y_train, X_val[i],
k=k_neighbors)
            if pred == y_val[i]:
                correct += 1
```

#Faz predições para todos os exemplos do conjunto de validação e conta quantos foram corretos.

```
        accuracy = correct / len(X_val)
```

```
        scores.append(accuracy)
        print(f"Fold {fold+1}: Accuracy = {accuracy:.4f}")
```

#Calcula a **acurácia** do fold atual e imprime.

```
    print("Average accuracy:", np.mean(scores))
```

Calcula e imprime a **acurácia média final** entre todos os folds.

Executa a validação

```
k_fold_cross_validation(X, y, k_folds=10, k_neighbors=5)
```

#Executa o KNN com validação cruzada (10 folds, k=5 vizinhos).

Código fonte completo

```
import pandas as pd
import numpy as np
# Fazendo a leitura do dataset/ Load Titanic dataset
import seaborn as sns
df = sns.load_dataset('titanic')
# Escolhendo as Features mais relevantes/ Use a few relevant columns
df = df[['survived', 'pclass', 'sex', 'age', 'fare']]
df.dropna(inplace=True)
# Alterando Feature Categórica /Encode 'sex' (female=0, male=1)
df['sex'] = df['sex'].map({'female': 0, 'male': 1})
# Separando o conjunto de Features de Rótulos/ Get features and labels
X = df[['pclass', 'sex', 'age', 'fare']].values
y = df['survived'].values
# Normalizando os dados /Normalize features (important for KNN)
X = (X - X.mean(axis=0)) / X.std(axis=0)
```

```
# Função que calcula a distância euclidiana

def euclidean_distance(x1, x2):

   return np.sqrt(np.sum((x1 - x2) ** 2))

# Função que chama o algoritmo KNN da biblioteca pandas

def knn_predict(X_train, y_train, x_test, k=10):

   distances = [euclidean_distance(x_test, x_train) for x_train in
X_train]

   k_indices = np.argsort(distances)[:k]

   k_nearest_labels = y_train[k_indices]

   counts = np.bincount(k_nearest_labels)

   return np.argmax(counts)

# Função que gera os n-folds para validação cruzada

def k_fold_cross_validation(X, y, k_folds=10, k_neighbors=5):

   fold_size = len(X) // k_folds

   indices = np.arange(len(X))

   np.random.shuffle(indices)

   scores = []

   for fold in range(k_folds):

       start = fold * fold_size

       end = start + fold_size

       val_indices = indices[start:end]

       train_indices = np.concatenate((indices[:start], indices[end:]))

       X_train, y_train = X[train_indices], y[train_indices]

       X_val, y_val = X[val_indices], y[val_indices]

       correct = 0
```

```
        for i in range(len(X_val)):
            pred = knn_predict(X_train, y_train, X_val[i],
k=k_neighbors)
            if pred == y_val[i]:
                correct += 1
# Calculando a acurácia do modelo
        accuracy = correct / len(X_val)
        scores.append(accuracy)
        print(f"Fold {fold+1}: Accuracy = {accuracy:.4f}")
    print("Average accuracy:", np.mean(scores))
k_fold_cross_validation(X, y, k_folds=10, k_neighbors=5)
```