KNN (*K-Nearest Neighbors*)

Conceito

O KNN (*K-nearest neighbors* ou “K-vizinhos mais próximos”) é um dos algoritmos mais utilizados em *machine learning* e possui um conceito bastante simplista assumindo que coisas semelhantes existem nas proximidades ou que coisas semelhantes estão próximas umas das outras.

Ele é um algoritmo não paramétrico que significa que a estrutura do modelo será determinada pelo conjunto de dados a ser utilizado e conhecido como um algoritmo de aprendizado lento ou *lazy*.

Neste tipo de algoritmo, não existe a necessidade de utilizar dados de treinamento para produzir o modelo, os dados obtidos no conjunto de dados já são adotados na fase de teste oferecendo maior agilidade ao modelo, porém com um processo de validação e apuração dos resultados mais lentos.

O KNN pode ser utilizado para tratar problemas tanto de classificação como de regressão. Ele é mais explorado em problemas de classificação, no entanto, existem abordagens do seu uso em tarefas de regressão. Na classificação, este algoritmo é utilizado para informar a qual grupo um determinado registro irá pertencer. Na regressão, ele irá nos fornecer um valor numérico.

Sobre o tipo de aprendizado que este algoritmo se encaixa, podemos dizer que ele pode atender a problemas relacionados ao aprendizado supervisionado (classificação e regressão) como também para problemas de aprendizado não supervisionado (clusterização).

Diferentes Nomeações para KNN

Como informado acima, o KNN é um algoritmo bastante utilizado e aplicado a diversas disciplinas. Abaixo são compartilhadas diferentes nomeações atreladas ao algoritmo KNN:

- Aprendizado Baseado em Instância: Cada linha do conjunto de dados é denominada com uma instância e são utilizadas no seu formato mais puro (bruto) para geração das previsões.
- Aprendizado preguiçoso: Devido nenhum aprendizado do modelo ser necessário e todo o trabalho ser produzido quando uma previsão é solicitada. Além do adjetivo preguiçoso, comumente também vemos o KNN ser chamado de aprendizado lento.
- Não paramétrico: Esta nomeação ocorre por não realizar suposições sobre a forma funcional do problema

Como funciona

A variável K pertencente ao nome do modelo é também o principal parâmetro do algoritmo. Através deste parâmetro K é definida a quantidade de vizinhos (neighbor) que o algoritmo irá considerar em seus processos que veremos adiante.

De maneira sucinta, em situações em que lidamos com modelos binários obtendo duas classes (sim ou não; verdadeiro ou falso etc.), geralmente inicia-se K com um valor ímpar. A fim de fornecer um maior entendimento, na Figura 1 – Nova Classe no Conjunto de Dados, é exemplificada uma situação em que queremos prever uma nova classe entre dois grupos denominados como classe A e B.

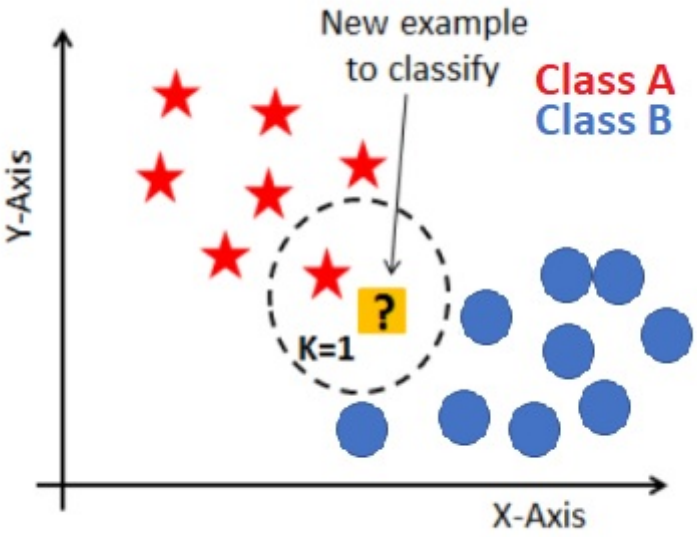

Figura 1 – Nova Classe no Conjunto de Dados

Inicialmente o algoritmo KNN recebe a nova classe ainda não classificada e utiliza uma unidade de distância para medir a distância da nova classe em relação a cada uma das classes já classificadas. Neste momento, o valor definido para K entra no processo.

Neste exemplo, consideramos um valor ímpar onde $K = 1$, ou seja, estamos estabelecendo apenas 1 vizinho mais próximo a nova classe. O algoritmo KNN faz uma verificação das classes já classificadas que tiveram as K menores distâncias e contabiliza a quantidade de vezes que cada classe apareceu. Neste exemplo, como $K = 1$, então o único vizinho e de menor distância sendo considerado para a lógica do algoritmo permite classificar a nova classe na Classe A.

Se tivéssemos definido o valor de $K = 5$, o algoritmo iria considerar as 5 classes de menor distância com relação a nova classe, ou seja, considerando os vizinhos mais próximos e no *label* (classe A ou B) contabilizado dos vizinhos mais próximos, a nova classe seria classificada. Por exemplo, se dos 5 vizinhos identificados 3 deles pertencessem à classe A sendo a maioria dos 5 vizinhos definidos através de $K = 5$, então a nova classe pertenceria à classe A.

Em resumo, o processo realizado pelo algoritmo KNN é:

1. Receber um dado não classificado e medir a distância do novo dado em relação a cada um dos outros dados que já estão classificados;
2. Selecionar as K menores distâncias de acordo com o valor de K definido;

3. Checar a(s) classe(s) dos dados que tiveram as K menores distâncias e contabilizar a quantidade de vezes que cada classe apareceu;
4. Por fim, classificar o novo dado como pertencente à classe que mais apareceu.

Principais Unidades de Distância Utilizadas no KNN

Existem diversas técnicas para realização do cálculo da distância. Abaixo, compartilho alguns exemplos:

- **Distância Euclidiana (Euclidean):** Esta é a distância mais utilizada e representa a distância mais curta entre dois pontos. Ela é calculada como a raiz quadrada da soma das diferenças quadráticas entre um novo ponto (x) e um ponto existente (y):

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

- **Distância Manhattan:** Esta é a distância entre vetores reais usando a soma de sua diferença absoluta ou a soma das diferenças absolutas entre os pontos em todas as dimensões:

$$\sqrt{\sum_{i=1}^{k}|x_i - y_i|}$$

- **Distância de Hamming:** Ela calcula a distância entre dois vetores binários. Essa técnica acaba sendo uma boa opção para casos de variáveis categóricas (*one-hot*).

- **Distância de Mahalanobis:** É a distância entre dois pontos no espaço multivariado. Em caso de variáveis não correlacionadas a distância Euclidiana acaba resultando a mesma coisa que a distância Mahalanobis. Porém quando temos mais de duas variáveis correlacionadas, pode se tornar impossível traçar a distância entre eles com uma única reta:

$$D^2 = (x - m)^T . C^{-1} . (x - m)$$

x: vetor da variável observação

m: vetor da média das variáveis independentes

C: covariância das variáveis independentes

Performance e Dimensionalidade

De maneira geral, o algoritmo KNN produz melhores resultados quando lida com baixa volumetria de dados. Em casos que existe um aumento significativo de variáveis no modelo, se faz necessária a realimentação do modelo com quantidade cada vez maiores de registros, ou seja, quanto mais colunas tivermos mais dados serão necessários.

O grande risco com esta tendência de crescimento é nos depararmos com um *overfitting*, que é quando o algoritmo aprende muito sobre os dados e memoriza os relacionamentos. Por consequência, não gera previsões lógicas e corretas. Como complemento e analogamente ao *overfitting*, o *underfitting* é quando o algoritmo não é capaz de aprender com os dados e não entende o relacionamento entre eles.

Como é um algoritmo que se baseia na distância para classificar ou realizar a previsão de valores é necessário padronizar as escalas dos dados. Por exemplo, se houver valores em uma determinada variável variando entre 1.000 e 1.000.000 e uma outra variável que possui valores que variam entre 0 e 10, o algoritmo também gerará resultados imprecisos ou incorretos.

A fim de mitigarmos os pontos citados acima, é recomendado realizar um pré-processamento dos dados comumente chamado de redução de dimensionalidade (*Dimensionality reduction*), o mais conhecido é o PCA (*Principal Component Analysis*). O PCA aloca a representatividade dos valores das colunas em um mesmo range ou escala, como por exemplo, entre os valores 0 e 1. Como os números são infinitos e são passíveis deste pré-processamento, resultados mais precisos e seguros são produzidos pelo KNN.

Como estimar o melhor valor para K no KNN

O número de vizinhos (neighbor = K) no algoritmo KNN é um hiperparâmetro no qual se deve definir ao construir um modelo, assim o valor de K é um valor de controle para as previsões que o algoritmo irá produzir. Não existe um número mágico, pois cada conjunto de dados possui suas próprias características, cada um com um tipo de padrão. Valores atribuídos a K que sejam muito baixos serão influenciados por ruídos nos dados gerando assim influências no resultado obtido, em contrapartida valores muito altos serão computacionalmente custosos.

Alguns estudos com valores K baixos, apresentaram que o processo de treinamento (*fit*) se torna mais flexível, gerando resultados menos enviesados (bias), mas com uma alta variância. Já os resultados com valores altos atribuídos a K geraram consequentemente vários vizinhos, produzindo uma fronteira de decisão (*decision boundary*) mais sensível e gerando uma baixa variância dos resultados, mas um risco altíssimo de enviesamento.

Em geral, cientistas testam valores ímpares para K, pois foram identificados melhores resultados com valores não pares. No entanto, a melhor maneira de se obter o melhor K é testando diversos valores, podemos tirar a raiz quadrada do valor total de pontos existentes no conjunto de dados ou aplicar técnicas de *tuning* para identificação do valor que irá performar melhor, como por exemplo o *k-fold cross validation*.

Campos de Aplicação

Este algoritmo, o KNN, pode ser aplicado em diversos segmentos de negócio sendo explorado em campos como: finanças, saúde, ciência política, reconhecimento de imagem, reconhecimento de vídeos, análise de churn de clientes e outros.

Também, o uso do KNN, como foi explanado ao longo deste material, pode ser aplicado em problemas de aprendizado supervisionado (classificações e regressões), bem como problemas de aprendizado não supervisionado (clusterização). Sua utilidade é bastante híbrida, cabe fazer o uso de acordo como problema ou necessidade de negócio.

Principais Vantagens e Desvantagens

As principais vantagens do algoritmo KNN, podem ser consideradas as seguintes:

- É um dos mais simples algoritmos a se implementar;
- Boa precisão na maioria dos casos aplicados;
- O KNN pode ser utilizado em dados não lineares bem como para problemas de regressão;

- Por ser um algoritmo de aprendizado lento não necessita da etapa de treinamento antes de fazer previsões fazendo o KNN muito mais rápido para ser aplicado quando comparado com outros algoritmos;
- Fácil inserção de novos dados na série de maneira instantânea, já que não é necessário treinar;
- Existem apenas dois parâmetros necessários para implementar KNN, ou seja, o valor de K e a função de distância (Euclidiana, Manhattan e outras).

Assim, oferece facilidade para se efetuar *tuning* dos poucos parâmetros existentes.

Por outro lado, abaixo, destaco a visíveis desvantagens da sua utilização:

- KNN não é indicado para ser aplicado com dados de grandes dimensões, porque ele precisa calcular a distância de todos os pontos de dados entre si. Desta maneira, quanto mais dados e mais dimensões desses registros, maior será o tempo para processar os cálculos. Desta maneira, gera alto custo de previsão para grandes conjuntos de dados;
- Uma demora excessiva na fase de teste e o alto consumo de memória para realizar esta atividade de teste, uma vez que ele armazena todo conjunto de dados em memória;
- Como utiliza medidas de distância necessita sempre de ter atenção nas escalas dos valores utilizados nas variáveis, para evitar a produção ou geração de resultados sem sentido;
- Baixa performance com variáveis categóricas explicativas.

Aplicação Prática

Através do link abaixo, pode ser feito o download do(s) script(s) python criado(s) para exemplificar uma abordagem do uso prático do KNN (*K-Nearest Neighbors*) como também o(s) respectivo(s) conjunto(s) de dados utilizados.

Script Python
Dataset 1

```
## KNN - K Nearest Neighbor

### Importando as bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy as scp
import warnings
warnings.filterwarnings("ignore")

### Importando o dataset
df = pd.read_csv('mushrooms.csv',engine='python', sep=',')

### Explorando o dataset
df.head()
```

```
# Informações gerais sobre o dataset
df.info()

# Produção de estatísticas sobre os dados
df.describe()

### Verificação de registros nulos no dataset
def distribuicao (data):
    '''
    Esta função exibirá a quantidade de registros únicos para cada coluna
    existente no dataset

    dataframe -> Histogram
    '''
    # Calculando valores únicos para cada label: num_unique_labels
    num_unique_labels = data.apply(pd.Series.nunique)

    # plotando valores
    num_unique_labels.plot( kind='bar')

    # Nomeando os eixos
    plt.xlabel('Campos')
    plt.ylabel('Número de Registros únicos')
    plt.title('Distribuição de dados únicos do DataSet')

    # Exibindo gráfico
    plt.show()

distribuicao(df)

### Análise de distribuição dos dados da classe Y (Venenoso = p, Comestível =
e)
e = pd.value_counts(df['class']) [0]
p = pd.value_counts(df['class']) [1]

tam = len(df)

print('Cogumelos Comestiveis: ',e)
print('Cogumelos Venenosos: ',p )

pie = pd.DataFrame([['Comestivel',e],['Venenoso',p]],columns=['Tipo' ,
'Quantidade'])

def pie_chart(data,col1,col2,title):
    labels = {'Comestivel':0,'Venenoso':1}
    sizes = data[col2]
```

```
    colors = ['#e5ffcc', '#ffb266']

    plt.pie(sizes, labels=labels, colors=colors,
                autopct='%1.1f%%', shadow=True, startangle=140, labeldistance
=1.2)
    plt.title( title )

    plt.axis('equal')
    plt.show()

pie_chart(pie,'Tipo' , 'Quantidade','Distribuição Percentual Classes de
Cogumelos')

plt.bar(pie.Tipo,pie.Quantidade, color = ['#e5ffcc', '#ffb266'])
plt.title("Distribuição das Classes de Cogumelos")
plt.xlabel("Tipo de Cogumelo")
plt.ylabel('Quantidade de Registros')
plt.show()

### Split do conjunto de dados
# X = colunas de informação, variáveis independentes
X = df.drop('class', axis=1)

# y = Variável dependente, a qual será utilizada para classificar os dados
y = df['class']

# Verificando se X está com a coluna class
X.head()

### Aplicação da técnica One Hot Encoder

#### Transformar as labels em números.
#### O OneHotEncoder gera novas colunas com valor 0 ou 1, onde 1 representa a
ocorrência daquela característica e 0 a não ocorrência.

#Importando o encoder para transformar as labels em chaves numéricas
from sklearn.preprocessing import OneHotEncoder
Oht_enc = OneHotEncoder()
X = pd.DataFrame(Oht_enc.fit_transform(X).A)
X.shape

### Train Test Split

#### Nesta fase separamos o conjunto de dados em Treinamento e Teste,
definindo o percentual que utilizaremos para teste e para treino do modelo
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3)

### Feature Scaling

#### Etapa importante que irá reduzir a escala numérica das colunas, para que todas estejam dentro de uma mesma escala de valor, lembrando que na matemática os números são infinitos dentro de suas escalas, podendo serem representados então em diversas escalas diferentes. Se houver medidas com escalas de valor muito diferentes, a distância calculada pelo algoritmo será influenciada podendo gerar resultados errôneos.

#Importing librarie
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

### Creating KNN Model

#### Agora iremos aplicar nossos dados ao algoritmo KNN

#Importando o modelo KNN
from sklearn.neighbors import KNeighborsClassifier

# Definindo o valor de vizinhos
classifier = KNeighborsClassifier(n_neighbors=5)

#Treinando o modelo, com dados de treinamento
classifier.fit(X_train, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_pred = classifier.predict(X_test)

### Avaliando o Algoritmo
##### Analisando e validando os resultados obtidos

# Importando métricas para validação do modelo
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Imprimindo a matriz confusa
print("Matriz Confusa: ")
```

```
print(confusion_matrix(y_test, y_pred), "\n")

# Imprimindo o relatório de classificação
print("Relatório de classificação: \n", classification_report(y_test,
y_pred))

# Imprimindo o quão acurado foi o modelo
print('Acurácia do modelo: ' , accuracy_score(y_test, y_pred))

### Loop para gerar testes com diferentes valores de vizinho (K)
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

### Comparando o Error Rate gerado de valores K diferentes
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

### Aplicando melhor parâmetro para K encontrado
# Treinando o modelo KNN com o melhor parâmetro para K

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)

# Aplicando os valores de teste novamente
y_pred = classifier.predict(X_test)

# Importando métricas para validação do modelo
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Imprimindo a matriz confusa
print("Matriz Confusa: ")
print(confusion_matrix(y_test, y_pred), "\n")

# Imprimindo o relatório de classificação
```

```
print("Relatório de classificação: \n", classification_report(y_test,
y_pred))

# Imprimindo o quão acurado foi o modelo
print('Acurácia do modelo: ' , accuracy_score(y_test, y_pred))
```

Materiais complementares

- Programação Dinâmica at Youtube: CLASSIFICAÇÃO com k-vizinhos mais próximos (K-NN) | Machine Learning #07
- Let's Code channel at Youtube: Machine Learning além das previsões
- S. Zhang, "Challenges in KNN Classification" in IEEE Transactions on Knowledge & Data Engineering, vol. , no. 01, pp. 1-1, 5555. doi: 10.1109/TKDE.2021.3049250 keywords: {training;nearest neighbor methods;data mining;prediction algorithms;training data;partitioning algorithms;licenses}
- ZHANG, Shichao et al. Learning k for knn classification. ACM Transactions on Intelligent Systems and Technology (TIST), v. 8, n. 3, p. 1-19, 2017.
- GUO, Gongde et al. KNN model-based approach in classification. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer, Berlin, Heidelberg, 2003. p. 986-996.

Referências

- S. Zhang, "Challenges in KNN Classification" in IEEE Transactions on Knowledge & Data Engineering, vol., no. 01, pp. 1-1, 5555. doi: 10.1109/TKDE.2021.3049250 keywords: {training;nearest neighbor methods;data mining;prediction algorithms;training data;partitioning algorithms;licenses}
- ZHANG, Shichao et al. Learning k for knn classification. ACM Transactions on Intelligent Systems and Technology (TIST), v. 8, n. 3, p. 1-19, 2017.
- GUO, Gongde et al. KNN model-based approach in classification. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer, Berlin, Heidelberg, 2003. p. 986-996.

Tópico anterior Próximo Tópico