Análise de Dados Aplicada à Computação

APRENDIZAGEM NÃO SUPERVISIONADA

Prof. M.Sc. Howard Roatti

Tópicos

1. Introdução à aprendizagem não supervisionada
2. Clusterização
3. Redução de dimensionalidade
4. Análise de associação
5. Análise de outlier
6. Visualização de dados

Aprendizagem Não Supervisionada

- A aprendizagem não supervisionada é um tipo de aprendizagem de máquina que **visa encontrar padrões e estruturas nos dados sem a necessidade de um conjunto de dados rotulados ou uma variável alvo.** Dessa forma, a ideia é que o algoritmo aprenda sozinho a identificar relações entre as variáveis de entrada e agrupá-las em diferentes categorias ou clusters.

Aprendizagem Não Supervisionada

- Essa abordagem de aprendizagem é útil quando se deseja **explorar os dados para obter insights, descobrir relações ocultas ou encontrar agrupamentos naturais de observações.** Ela é aplicada em diversas áreas, como reconhecimento de padrões, análise de imagens e de texto, análise de agrupamentos e detecção de anomalias.

Aprendizagem Não Supervisionada

Algumas das **técnicas de aprendizagem não supervisionada** mais comuns são:

- **Análise de Componentes Principais** (PCA): utilizada para reduzir a dimensionalidade dos dados, identificando os componentes mais importantes.
- **Agrupamento** (Clustering): utilizada para identificar grupos ou clusters de observações semelhantes.
- **Redução de Dimensionalidade**: utilizada para transformar os dados em um espaço de menor dimensão mantendo as informações mais relevantes.

Aprendizagem Não Supervisionada

Exemplos de problemas que podem ser tratados com a aprendizagem não supervisionada incluem (1/3):

- **Identificação de perfis de clientes**: identificar grupos de clientes com características semelhantes para personalizar campanhas de marketing;
- **Análise de sentimento**: identificar tópicos ou sentimentos mais comuns em um conjunto de textos;

Aprendizagem Não Supervisionada

Exemplos de problemas que podem ser tratados com a aprendizagem não supervisionada incluem (2/3):

- **Segmentação de mercado**: identificar grupos de clientes com comportamentos de compra semelhantes para adaptar a oferta de produtos;
- **Detecção de anomalias**: identificar observações que sejam diferentes do restante do conjunto de dados;
- **Classificação de imagens**: agrupar imagens semelhantes para fins de organização ou categorização.

Aprendizagem Não Supervisionada

Exemplos de problemas que podem ser tratados com a aprendizagem não supervisionada incluem (3/3):

- **Classificação de genes**: agrupar genes com base em sua expressão em diferentes amostras para identificar padrões biológicos e possíveis marcadores de doenças.
- **Detecção de anomalias**: identificar observações incomuns ou discrepantes em um conjunto de dados para detectar fraudes em transações financeiras ou comportamentos incomuns em sistemas de monitoramento.
- **Agrupamento de documentos**: agrupar documentos semelhantes para criar taxonomias e facilitar a organização e recuperação de informações.

Clusterização

Clusterização é uma forma de identificar padrões em dados sem ter que rotulá-los previamente. Ao contrário da aprendizagem supervisionada, não há uma variável alvo conhecida, e o objetivo é encontrar estruturas e padrões ocultos nos dados.

Clusterização

Existem vários algoritmos de clusterização, como **K-Means** e **Hierarchical Clustering**.

- **O algoritmo K-Means** é um método iterativo que agrupa os dados em k clusters, onde k é o número de clusters definido pelo usuário. Cada cluster é representado por um centróide, que é atualizado a cada iteração do algoritmo. O objetivo é minimizar a soma dos quadrados das distâncias entre cada ponto e seu centróide mais próximo.

Clusterização: k-Means (1/5)

O K-means é um algoritmo de clusterização muito utilizado em aprendizado não supervisionado. O seu objetivo é agrupar os dados em k grupos distintos. O valor de k é definido pelo usuário e representa a quantidade de clusters que se deseja formar.

Clusterização: k-Means (2/5)

- Entrada: k é o número de clusters a ser gerado
- Cada cluster é representado por uma centroide
- Algoritmo:
  - particiona os registros em k clusters
    - cada registro é associado ao cluster mais próximo
  - recalcula centroides
  - repete o processo até que os clusters não modifiquem mais

Clusterização: k-Means (3/5)

Acesse o Link para Visualizar o Algoritmo em Funcionamento: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

Clusterização: k-Means (4/5)

Exemplo com Python

```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt

# criando um conjunto de dados de exemplo
data = pd.DataFrame({ 'x': [5, 3, 10, 12, 2, 4, 6, 8, 11, 15],
                      'y': [4, 6, 1, 2, 8, 7, 5, 3, 2, 1] })

# plotando os dados originais
plt.scatter(data['x'], data['y'])
plt.title('Dados originais')
plt.show()
```

Clusterização: k-Means (5/5)

Exemplo com Python

```
# criando o modelo K-means com 2 clusters
kmeans = KMeans(n_clusters=2)

# treinando o modelo com os dados
kmeans.fit(data)

# adicionando as previsões do modelo aos dados
data['cluster'] = kmeans.predict(data)

# plotando os dados com as cores dos clusters
colors = {0: 'red', 1: 'blue'}
data['color'] = data['cluster'].map(colors)
plt.scatter(data['x'], data['y'], color=data['color'])
plt.title('Dados clusterizados')
plt.show()
```

Clusterização

Já o **Hierarchical Clustering** é um algoritmo que constrói uma hierarquia de clusters, onde cada cluster é formado pela junção de dois clusters menores. Existem duas abordagens para o Hierarchical Clustering: aglomerativo e divisivo. Na abordagem aglomerativa, cada ponto é inicialmente considerado um cluster e, em seguida, os clusters são unidos iterativamente com base na distância entre eles. Na abordagem divisiva, todos os pontos são considerados como um único cluster e, em seguida, são divididos em clusters menores iterativamente.

Clusterização: Hierarchical Clustering (1/4)

O Hierarchical Clustering é um algoritmo de clusterização que parte de uma abordagem bottom-up ou top-down, onde as amostras são agrupadas de forma hierárquica. Esse algoritmo é baseado na construção de uma estrutura de árvore (dendrograma), que representa a relação de proximidade entre as amostras.

Clusterização: Hierarchical Clustering (2/4)

No Hierarchical Clustering, existem dois tipos principais de estratégias: o agglomerative (bottom-up) e o divisive (top-down). No primeiro, cada amostra é considerada um cluster, e a cada iteração, os clusters mais próximos são agrupados. No segundo, todas as amostras são consideradas um único cluster, e a cada iteração, o cluster é dividido em dois subclusters.

Clusterização: Hierarchical Clustering (3/4)

Para medir a distância entre os clusters, o algoritmo utiliza uma medida de dissimilaridade, que pode ser a distância euclidiana, a distância de Manhattan, a distância de correlação, entre outras.

Clusterização: Hierarchical Clustering (4/4)

Exemplo com Python

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Gerando um conjunto de dados sintético
X, y = make_blobs(n_samples=100, centers=3, random_state=42)
# Criando o objeto do algoritmo de clusterização
agg_clustering = AgglomerativeClustering(n_clusters=3)

# Ajustando o modelo aos dados
agg_clustering.fit(X)
# Obtendo as previsões dos clusters para cada amostra
cluster_labels = agg_clustering.labels_

# Plotando o resultado
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels)
plt.show()
```

Clusterização: Seleção de K

O método da silhueta e o método do cotovelo são técnicas comumente utilizadas para selecionar o valor adequado de "k" em algoritmos de clusterização, como o K-means.

Clusterização: Seleção de K

Método da Silhueta: O método da silhueta é uma medida de validação interna para avaliar a qualidade dos agrupamentos obtidos. Ele calcula um valor de silhueta para cada ponto de dados, que mede o quão bem ele se encaixa em seu próprio cluster em comparação com outros clusters próximos. O valor da silhueta varia de -1 a 1, onde valores mais próximos de 1 indicam que o ponto está bem ajustado ao seu cluster e distante dos outros clusters. Para determinar o valor de "k" ideal, você pode calcular a média dos valores de silhueta para diferentes valores de "k" e escolher aquele que resulta na maior média. Um valor mais alto de silhueta média indica uma estrutura de clusterização mais coerente.

Clusterização: Seleção de K

Método do Cotovelo: O método do cotovelo é uma técnica gráfica que busca identificar um "ponto de cotovelo" em um gráfico de soma das distâncias quadradas intra-cluster em relação ao número de clusters ("k"). A ideia é plotar os valores da soma das distâncias quadradas intra-cluster em função de diferentes valores de "k" e procurar o ponto onde a diminuição da soma das distâncias se torna menos acentuada. Esse ponto é chamado de "ponto de cotovelo" e é considerado um bom valor para "k". O nome "cotovelo" vem da forma do gráfico, que geralmente se assemelha a um braço dobrado no cotovelo. No entanto, é importante notar que nem sempre é fácil identificar claramente o ponto de cotovelo, e às vezes pode ser subjetivo determinar o valor ótimo de "k" com base apenas nessa técnica.

Clusterização: Seleção de K

- **Método da Silhueta** (*Silhouette Score*):
  - O gráfico mostrará os valores do score da silhueta para diferentes valores de "k". O valor de "k" com o score da silhueta mais alto indica a melhor quantidade de clusters.

- **Método do Cotovelo** (*Elbow Method*):
  - O gráfico mostrará os valores da inércia (soma das distâncias quadradas intra-cluster) para diferentes valores de "k". O valor de "k" onde ocorre uma mudança acentuada na inércia indica um possível ponto de cotovelo, que representa o número ideal de clusters.

```
import warnings
warnings.filterwarnings('ignore')

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

data = pd.DataFrame({'x': [5, 3, 10, 12, 2, 4, 6, 8, 11, 15],
                     'y': [4, 6, 1, 2, 8, 7, 5, 3, 2, 1]})

silhouette_scores = []
k_values = range(2, 10)  # Testando valores de k de 2 a 9

for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init='auto', random_state=1)
    kmeans.fit(data)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(data, labels)
    silhouette_scores.append(silhouette_avg)

# Plotando o gráfico de silhuetas
plt.plot(k_values, silhouette_scores, 'bo-')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Score da silhueta')
plt.title('Método da Silhueta')
plt.show()
```

- Método da Silhueta (*Silhouette Score*):

```
import warnings
warnings.filterwarnings('ignore')

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

data = pd.DataFrame({'x': [5, 3, 10, 12, 2, 4, 6, 8, 11, 15],
                     'y': [4, 6, 1, 2, 8, 7, 5, 3, 2, 1]})

inertia = []
k_values = range(2, 11)  # Testando valores de k de 2 a 9

for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init='auto', random_state=1)
    kmeans.fit(data)
    inertia.append(kmeans.inertia_)

# Plotando o gráfico de cotovelo
plt.plot(k_values, inertia, 'bo-')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Inertia')
plt.title('Método do Cotovelo')
plt.show()
```

- **Método do Cotovelo** (*Elbow Method*):

Redução de Dimensionalidade

A redução de dimensionalidade é uma técnica utilizada na análise de dados para diminuir a quantidade de variáveis ou características que descrevem um conjunto de dados, com o objetivo de tornar a análise mais simples, rápida e eficiente.

Basicamente, a redução de dimensionalidade consiste em transformar um conjunto de dados com várias dimensões em um conjunto com menos dimensões, mantendo as informações mais importantes dos dados originais. Isso é feito pela remoção de características irrelevantes ou redundantes, ou pela combinação de características similares.

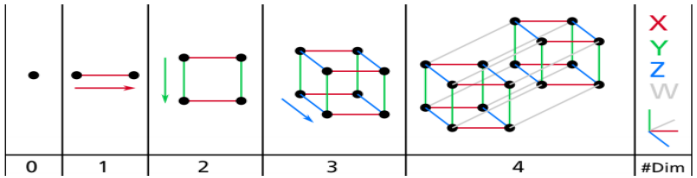

Figure 8-1. Point, segment, square, cube, and tesseract (0D to 4D hypercubes)[2]

Redução de Dimensionalidade

Existem várias técnicas de redução de dimensionalidade disponíveis, mas duas das mais populares são a **Análise de Componentes Principais** (PCA) e o **t-Distributed Stochastic Neighbor Embedding** (t-SNE). A PCA é uma técnica linear que procura projetar os dados em um novo conjunto de dimensões com menos variáveis, preservando a maior parte da variação presente nos dados originais. Já o t-SNE é uma técnica não-linear que busca preservar as relações de vizinhança entre os pontos dos dados em um espaço de menor dimensão.

Redução de Dimensionalidade: PCA (1/6)

O PCA (Principal Component Analysis) é uma técnica de redução de dimensionalidade que busca identificar os componentes principais de um conjunto de dados, ou seja, as direções ao longo das quais os dados variam mais. Esses componentes principais são utilizados para projetar os dados em um espaço de menor dimensionalidade, sem perda significativa de informação.

Redução de Dimensionalidade: PCA (2/6)

O PCA funciona encontrando os autovetores e autovalores da matriz de covariância dos dados. Os autovetores representam as direções principais dos dados, enquanto que os autovalores representam a importância relativa dessas direções. Os autovetores correspondentes aos autovalores mais altos são os componentes principais.

Redução de Dimensionalidade: PCA (3/6)

O processo do PCA pode ser resumido em quatro etapas:

1. Normalização dos dados para que as variáveis tenham média zero e desvio padrão igual a um.
2. Cálculo da matriz de covariância dos dados normalizados.
3. Cálculo dos autovetores e autovalores da matriz de covariância.
4. Projeção dos dados nos componentes principais.

Redução de Dimensionalidade: PCA (4/6)

Vamos utilizar o conjunto de dados de flores Iris para exemplificar o uso do PCA. Primeiro, importamos as bibliotecas necessárias e carregamos os dados:

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target
```

Redução de Dimensionalidade: PCA (5/6)

Em seguida, normalizamos os dados:

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

Depois, criamos um objeto PCA e ajustamos aos dados normalizados:

```
pca = PCA(n_components=2)
pca.fit(X_norm)
```

Podemos agora projetar os dados nos dois componentes principais:

```
X_pca = pca.transform(X_norm)
```

Redução de Dimensionalidade: PCA (6/6)

Por fim, podemos plotar os dados projetados nos dois componentes principais:

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.show()
```

Redução de Dimensionalidade: t-SNE (1/4)

O t-SNE (t-distributed stochastic neighbor embedding) é uma técnica de redução de dimensionalidade que foi desenvolvida para visualização de dados em baixas dimensões (geralmente 2D ou 3D). Ao contrário do PCA, que preserva as relações lineares entre as variáveis, o t-SNE preserva as relações não lineares entre as variáveis, o que é importante para a visualização de dados complexos.

Redução de Dimensionalidade: t-SNE (2/4)

O t-SNE funciona reduzindo a dimensionalidade dos dados enquanto tenta manter as relações entre as amostras. Ele faz isso mapeando cada amostra em um ponto em um espaço de baixa dimensão (por exemplo, um espaço de duas dimensões) de tal forma que as amostras que são semelhantes na alta dimensão sejam mapeadas próximas umas das outras no espaço de baixa dimensão. O t-SNE usa uma distribuição t-student para medir a semelhança entre as amostras na alta dimensão e outra distribuição t-student para medir a semelhança entre as amostras no espaço de baixa dimensão.

Redução de Dimensionalidade: t-SNE (3/4)

O t-SNE é frequentemente usado para visualizar dados em conjunto com outras técnicas de análise de dados, como clustering. Ele é particularmente útil quando se deseja visualizar dados em alta dimensão em um espaço de baixa dimensão, a fim de detectar padrões ou estruturas que não seriam evidentes em dimensões mais altas.

Redução de Dimensionalidade: t-SNE (4/4)

Exemplo com Python

```
from sklearn.datasets import load_iris
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Carregando o dataset de dígitos
iris = load_iris()
X, y = iris.data, iris.target
X_norm = (X - X.mean(axis=0)) / X.std(axis=0)

# Definindo o modelo e ajustando-o aos dados
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_norm)

# Plotando os resultados
plt.scatter(X_tsne[:, 0], X_tsne[:, 1] , c=y)
plt.show()
```

Análise de Associação (1/14)

A análise de associação é uma técnica de aprendizagem não supervisionada usada para descobrir associações ou relações entre diferentes variáveis em um conjunto de dados. O objetivo é identificar regras que possam explicar a relação entre as variáveis.

Análise de Associação (2/14)

O algoritmo mais comum utilizado na análise de associação é o Apriori. Ele é baseado em encontrar conjuntos de itens frequentes em um conjunto de transações, ou seja, encontrar grupos de itens que são frequentemente comprados juntos.

Análise de Associação (3/14)

O Apriori funciona em três etapas: suporte, confiança e lift. O **suporte** é a frequência com que um conjunto de itens aparece em todas as transações, a **confiança** é a probabilidade condicional de um item aparecer em uma transação, dado que outro item também apareceu nessa transação, e o **lift** é uma medida da força da associação entre os itens.

Análise de Associação (4/14)

Um exemplo de aplicação da análise de associação é em uma loja de varejo, onde a loja pode querer descobrir quais produtos são frequentemente comprados juntos para ajudar a determinar a organização das prateleiras da loja ou para criar pacotes de produtos. Outro exemplo é em um site de comércio eletrônico, onde a análise de associação pode ser usada para recomendar produtos para os clientes com base no histórico de compras.

Análise de Associação (5/14)

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Dados de exemplo
dataset = [['Leite', 'Cerveja', 'Pão'],
           ['Leite', 'Cerveja'],
           ['Leite', 'Pão'],
           ['Leite', 'Cerveja', 'Pão'],
           ['Leite', 'Cerveja', 'Pão']]

# Transforma os dados em um formato aceitável pelo Apriori
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Aplica o Apriori com suporte mínimo de 0.6
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)

# Imprime os conjuntos de itens frequentes
print(frequent_itemsets)
```

Análise de Associação (6/14)

Os resultados fornecidos consistem em duas colunas: "support" e "itemsets". Vamos analisar cada uma delas:

Support:

- O suporte é uma métrica que indica a frequência absoluta ou relativa de um itemset no conjunto de dados. Ele representa a proporção de transações que contêm o itemset em relação ao número total de transações.
- Por exemplo, na primeira linha, o valor de suporte é 0.8, o que significa que o item "Cerveja" aparece em 80% das transações.

Itemsets:

- Essa coluna lista os conjuntos de itens que foram descobertos pelo algoritmo Apriori.
- Por exemplo, a segunda linha indica que o itemset consiste apenas no item "Leite".
- A última linha indica que o itemset contém três itens: "Cerveja", "Pão" e "Leite".

Análise de Associação (7/14)

A métrica de confiança é especificada como "confidence" e um valor mínimo de 0.7 é definido usando o parâmetro min_threshold.

```
# Gera as regras de associação com métricas de confiança e lift
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Imprime as regras de associação com métricas de confiança e lift
display(rules)
```

Análise de Associação (8/14)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Cerveja)	(Leite)	0.8	1.0	0.8	1.00	1.0000	0.00	inf	0.00
1	(Leite)	(Cerveja)	1.0	0.8	0.8	0.80	1.0000	0.00	1.0	0.00
2	(Cerveja)	(Pão)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25
3	(Pão)	(Cerveja)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25
4	(Pão)	(Leite)	0.8	1.0	0.8	1.00	1.0000	0.00	inf	0.00
5	(Leite)	(Pão)	1.0	0.8	0.8	0.80	1.0000	0.00	1.0	0.00
6	(Cerveja, Pão)	(Leite)	0.6	1.0	0.6	1.00	1.0000	0.00	inf	0.00
7	(Cerveja, Leite)	(Pão)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25
8	(Pão, Leite)	(Cerveja)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25
9	(Cerveja)	(Pão, Leite)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25
10	(Pão)	(Cerveja, Leite)	0.8	0.8	0.6	0.75	0.9375	-0.04	0.8	-0.25

Análise de Associação (9/14)

A tabela apresenta as regras de associação descobertas a partir dos conjuntos de itens frequentes. Vamos interpretar as principais colunas (1/4):

- **antecedents e consequents**: Indicam os conjuntos de itens que formam as regras. Por exemplo, a regra (Cerveja) -> (Leite) significa que a presença de Cerveja é um indicativo para a presença de Leite.
- **antecedent support e consequent support**: Representam a frequência de ocorrência dos conjuntos de itens antecedentes e consequentes, respectivamente.

Análise de Associação (10/14)

A tabela apresenta as regras de associação descobertas a partir dos conjuntos de itens frequentes. Vamos interpretar as principais colunas (2/4):

- **support**: Refere-se à frequência de ocorrência conjunta dos conjuntos de itens antecedentes e consequentes.
- **confidence**: Mede a probabilidade condicional de ocorrência do conjunto consequente dado o conjunto antecedente. Por exemplo, uma confiança de 0.8 indica que, quando o antecedente ocorre, o consequente também ocorre com uma probabilidade de 0.8.

Análise de Associação (11/14)

A tabela apresenta as regras de associação descobertas a partir dos conjuntos de itens frequentes. Vamos interpretar as principais colunas (3/4):

- **lift**: Indica a medida pela qual a ocorrência conjunta dos conjuntos antecedentes e consequentes é maior do que a ocorrência esperada se eles fossem independentes. Valores de lift acima de 1 indicam associações positivas, onde a ocorrência de um conjunto aumenta a probabilidade de ocorrência do outro conjunto.
- **leverage**: Mede a diferença entre a ocorrência conjunta observada e a ocorrência esperada se os conjuntos antecedentes e consequentes fossem independentes. Valores de leverage acima de 0 indicam associações positivas.

Análise de Associação (12/14)

A tabela apresenta as regras de associação descobertas a partir dos conjuntos de itens frequentes. Vamos interpretar as principais colunas (4/4):

- **conviction**: Avalia o grau de dependência do consequente em relação ao antecedente. Valores de conviction acima de 1 indicam uma forte dependência.
- **zhangs_metric**: É uma métrica alternativa que combina confiança e lift para avaliar as regras de associação. Valores negativos indicam associações negativas.

Análise de Associação (13/14)

Com base nos resultados fornecidos, podemos fazer as seguintes observações (**1/2**):

- As regras (Cerveja) -> (Leite) e (Pão) -> (Leite) possuem uma confiança de 1.0, o que significa que a ocorrência de Cerveja ou Pão garante a presença de Leite. Isso sugere uma forte associação entre esses itens.
- As regras (Leite) -> (Cerveja) e (Leite) -> (Pão) possuem uma confiança de 0.8, indicando uma associação ligeiramente menos forte. Ainda assim, é possível inferir que a ocorrência de Leite aumenta a probabilidade de Cerveja ou Pão serem adquiridos.

Análise de Associação (14/14)

Com base nos resultados fornecidos, podemos fazer as seguintes observações (**2/2**):

- As regras envolvendo três itens, como (Cerveja, Pão) -> (Leite) e suas combinações correspondentes, também apresentam uma confiança de 1.0, indicando uma associação forte. Isso sugere que a ocorrência conjunta de Cerveja e Pão é altamente indicativa da presença de Leite.
- Os valores de lift próximos a 1 indicam que as associações descobertas não são muito mais fortes do que seria esperado ao acaso.
- Os valores negativos da métrica de Zhang indicam associações negativas, o que sugere que a ocorrência conjunta dos itens é menor do que o esperado.

Análise de Outliers (1/6)

A análise de outlier é uma técnica utilizada para detectar observações que se desviam significativamente do comportamento esperado em um conjunto de dados. Essas observações, também conhecidas como "outliers", podem ser causadas por erros de medição, erros de entrada de dados ou podem ser indicativos de eventos raros ou incomuns.

Análise de Outliers (2/6)

O objetivo da análise de outlier é identificar esses pontos de dados que são diferentes do restante e determinar se são uma fonte de erro ou se contêm informações valiosas. A análise de outlier pode ser realizada usando várias técnicas, como abordagens baseadas em distância, abordagens baseadas em modelo e abordagens baseadas em densidade.

Análise de Outliers (3/6)

Um algoritmo comum usado para análise de outlier é o Local Outlier Factor (LOF). O LOF é um algoritmo de detecção de outlier que calcula o grau de anomalia de cada ponto de dados com base em sua densidade local em relação aos seus vizinhos mais próximos.

Análise de Outliers (4/6)

O algoritmo LOF funciona da seguinte maneira: para cada ponto de dados, o algoritmo identifica os seus k vizinhos mais próximos e calcula a densidade local do ponto de dados com base na distância média entre ele e seus vizinhos. Em seguida, o LOF calcula a razão entre a densidade local do ponto de dados e a densidade local dos seus vizinhos. Esse valor é chamado de Local Outlier Factor e indica o grau de anomalia do ponto de dados.

Análise de Outliers (5/6)

O LOF pode ser aplicado em vários domínios, como detecção de fraude em transações financeiras, detecção de anomalias em sensores de monitoramento e detecção de falhas em sistemas de manufatura.

Análise de Outliers (6/6)

Exemplo em Python

```
from sklearn.neighbors import LocalOutlierFactor
import numpy as np

# Criação de um conjunto de dados de exemplo
X = np.array([[1.1, 1.2], [1.3, 1.4], [10, 12], [10.5, 11.8]])

# Criação de uma instância do algoritmo LOF
lof = LocalOutlierFactor(n_neighbors=2)

# Ajuste do modelo aos dados e previsão dos rótulos dos outliers
y_pred = lof.fit_predict(X)

# Impressão dos rótulos dos outliers (-1 indica um outlier)
print(y_pred)
```

Visualização de Dados (1/17)

No contexto de aprendizagem não supervisionada, a visualização de dados pode ser utilizada para representar a distribuição dos dados em um espaço de menor dimensão, gerado por técnicas de redução de dimensionalidade, como PCA e t-SNE. Além disso, existem técnicas de visualização específicas para representar clusters, como a plotagem de dendrogramas para Hierarchical Clustering e a plotagem de pontos para o K-Means.

Visualização de Dados (2/17)

Algumas ferramentas úteis para visualização de dados em aprendizagem não supervisionada incluem:

- **Matplotlib**: biblioteca de plotagem em Python, utilizada para criar gráficos simples e personalizados
- **Seaborn**: biblioteca de visualização estatística em Python, utilizada para criar gráficos mais complexos e esteticamente agradáveis
- **Plotly**: biblioteca de visualização interativa em Python, utilizada para criar gráficos dinâmicos e interativos
- **Tableau, Power BI, Metabase**: ferramenta de visualização de dados em geral, utilizada para criar gráficos e dashboards personalizados

Visualização de Dados (3/17)

Exemplo 1: Scatter plot em 2D gerado a partir de uma redução de dimensionalidade com PCA, colorido de acordo com os clusters gerados por K-Means

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Gerando um dataset aleatório com 4 clusters e 8 features
X, y = make_blobs(n_samples=500, centers=4, n_features=8, random_state=42)

# Reduzindo a dimensionalidade para 2D com PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

Visualização de Dados (4/17)

```
# Realizando a clusterização com K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Plotando o scatter plot com cores definidas pelos clusters gerados pelo K-Means
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans)
plt.title('Scatter plot em 2D gerado a partir de uma redução de dimensionalidade com PCA,
colorido de acordo com os clusters gerados por K-Means')
plt.show()
```

Visualização de Dados (5/17)

Exemplo 2: Gráfico de barras horizontal gerado a partir de técnicas de análise de associação

```
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Carregando os dados
data = [['Leite', 'Cebola', 'Batatas'],
        ['Cerveja', 'Cebola'],
        ['Leite', 'Cerveja', 'Bolachas', 'Cebola'],
        ['Leite', 'Bolachas', 'Cebola'],
        ['Leite', 'Cerveja', 'Cebola'],
        ['Leite', 'Bolachas', 'Cebola'],
        ['Leite', 'Bolachas', 'Cerveja', 'Cebola']]
```

Visualização de Dados (6/17)

```
# Convertendo os dados em um formato adequado
te = TransactionEncoder()
te_ary = te.fit(data).transform(data)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Aplicando o algoritmo Apriori
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)

# Gerando as regras de associação
rules = association_rules(frequent_itemsets, metric='support', min_threshold=0.3)

# Ordenando as regras de associação pelo suporte
rules = rules.sort_values(by='support', ascending=False)

# Gerando o gráfico de barras
plt.barh(range(len(rules)), rules['support'], align='center')
plt.yticks(range(len(rules)), [', '.join(items) for items in rules['antecedents']])
plt.xlabel('Support')
plt.title('Regras de Associação e seus Suportes')
plt.show()
```

Visualização de Dados (7/17)

Exemplo 3 - Gráfico de Área com sobreposição das características gerais de cada cluster para comparação a partir do Hierarchical Clustering:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import linkage, fcluster

# Carregando o dataset Iris
iris = load_iris()

# Realizando o Hierarchical Clustering
Z = linkage(iris.data, method='ward')

# Gerando os clusters
k = 3
clusters = fcluster(Z, k, criterion='maxclust')
```

Visualização de Dados (8/17)

```
# Criando um dicionário para guardar os dados de cada cluster
cluster_data = {i+1: [] for i in range(k)}

for i in range(len(iris.data)):
    cluster_data[clusters[i]].append(iris.data[i])

# Obtendo as médias de cada característica para cada cluster
cluster_means = {}
for i in range(k):
    cluster_points = np.array(cluster_data[i+1])
    cluster_means[i+1] = np.mean(cluster_points, axis=0)

# Configurando as características e os rótulos para o radar chart
features = iris.feature_names
num_features = len(features)

# Criando um array de ângulos para as características
angles = np.linspace(0, 2 * np.pi, num_features, endpoint=False).tolist()
angles += angles[:1]
```

Visualização de Dados (9/17)

```
# Criando um array de valores para cada cluster
cluster_values = []
for i in range(k):
    values = cluster_means[i+1].tolist()
    values += values[:1]
    cluster_values.append(values)

# Plotando o radar chart para cada cluster
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw={'polar': True})
for i, values in enumerate(cluster_values):
    ax.plot(angles, values, linewidth=1, linestyle='solid', label=f'Cluster {i+1}')
    ax.fill(angles, values, alpha=0.25)

# Configurando os rótulos das características
ax.set_xticks(angles[:-1])
ax.set_xticklabels(features)

# Definindo o título do gráfico
ax.set_title('Clusters gerados pelo Hierarchical Clustering')
```

Visualização de Dados (10/17)

```
# Adicionando uma legenda
ax.legend()

# Exibindo o gráfico
plt.show()
```

Visualização de Dados (11/17)

Exemplo 4 - Gráfico de barras representando a frequência de ocorrência dos outliers detectados por Local Outlier Factor em cada grupo

```
from sklearn.datasets import load_breast_cancer
from sklearn.neighbors import LocalOutlierFactor
import numpy as np
import matplotlib.pyplot as plt

# Carregando o dataset
breast_cancer = load_breast_cancer()

# Realizando a detecção de outliers
clf = LocalOutlierFactor(n_neighbors=20)
y_pred = clf.fit_predict(breast_cancer.data)
```

Visualização de Dados (12/17)

```
# Criando um dicionário para guardar os dados de cada grupo (outlier ou não outlier)
groups_data = {1: [], -1: []}
for i in range(len(breast_cancer.data)):
    groups_data[y_pred[i]].append(breast_cancer.target[i])

# Contando a frequência de ocorrência dos outliers em cada grupo
outliers_freq = [0, 0]
for i in groups_data[-1]:
    outliers_freq[i] += 1

# Gerando o gráfico de barras
fig, ax = plt.subplots()
ax.bar(np.arange(2), outliers_freq)
ax.set_xticks(np.arange(2))
ax.set_xticklabels(['malignant', 'benign'])
ax.set_ylabel('Frequência')
ax.set_title('Frequência de ocorrência dos outliers')
plt.show()
```

Visualização de Dados (13/17)

Exemplo 5 - Gráfico de dispersão representando a ocorrência de outliers

```
from sklearn.datasets import load_breast_cancer
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt

# Carregando o dataset
breast_cancer = load_breast_cancer()

# Realizando a detecção de outliers
clf = LocalOutlierFactor(n_neighbors=20)
y_pred = clf.fit_predict(breast_cancer.data)

# Separando os pontos de dados em outliers e não outliers
outliers = breast_cancer.data[y_pred == -1]
non_outliers = breast_cancer.data[y_pred == 1]
```

Visualização de Dados (14/17)

```
# Gerando o gráfico de dispersão
plt.scatter(non_outliers[:, 0], non_outliers[:, 1], c='blue', label='Não Outlier')
plt.scatter(outliers[:, 0], outliers[:, 1], c='red', label='Outlier')
plt.xlabel('Primeiro Atributo')
plt.ylabel('Segundo Atributo')
plt.title('Gráfico de Dispersão dos Outliers')
plt.legend()
plt.show()
```

Visualização de Dados (15/17)

Exemplo 6 - Gráfico de linha representando os pontos de uma série temporal com a ocorrência de outliers

```
import pandas as pd
import numpy as np
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt

# Carregando o conjunto de dados de séries temporais
data = pd.read_csv('./DataSets/Time_Series/MLTempDataset.csv')

# Convertendo a coluna de datas para o formato datetime
data['Datetime'] = pd.to_datetime(data['Datetime'])

# Definindo as datas de início e fim para o filtro
start_date = '2022-01-04 00:00:00'
end_date = '2022-01-06 00:00:00'
```

Visualização de Dados (16/17)

```
# Filtrando o dataframe com base nas datas
filtered_data = data.loc[(data['Datetime'] >= start_date) & (data['Datetime'] <= end_date)]

# Separando as colunas relevantes
datetime = filtered_data['Datetime']
values = filtered_data['DAYTON_MW']

# Transformando os dados em formato adequado para o algoritmo
X = values.values.reshape(-1, 1)

# Realizando a detecção de outliers
clf = LocalOutlierFactor(n_neighbors=20)
y_pred = clf.fit_predict(X)
```

Visualização de Dados (17/17)

```
# Gerando o gráfico de linha com destaque para outliers
plt.figure(figsize=(12, 6))
plt.plot(datetime, values, color='blue', label='Série Temporal')
plt.scatter(datetime[y_pred == -1], values[y_pred == -1], color='red', label='Outlier')
plt.xlabel('Data e Hora')
plt.ylabel('DAYTON_MW')
plt.title('Visualização de Outliers em uma Série Temporal (Período Filtrado)')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Resumo da Aula (1/4)

Nesta aula, exploramos os principais conceitos e técnicas da aprendizagem não supervisionada, que é uma forma de análise de dados que não requer um conjunto de treinamento rotulado. Exploramos os diferentes tipos de problemas que podem ser resolvidos com aprendizagem não supervisionada, como a clusterização, redução de dimensionalidade, análise de associação e análise de outliers.

Resumo da Aula (2/4)

Na clusterização, aprendemos sobre diferentes algoritmos, como K-Means e Hierarchical Clustering, e vimos como eles podem ser usados para agrupar dados sem rótulos em clusters. Na redução de dimensionalidade, vimos técnicas como PCA e t-SNE, que podem ser usadas para reduzir a dimensionalidade dos dados, mantendo a maior quantidade possível de informações.

Resumo da Aula (3/4)

Na análise de associação, exploramos algoritmos como Apriori, que podem ser usados para descobrir padrões em conjuntos de dados, e na análise de outliers, aprendemos sobre algoritmos como Local Outlier Factor, que podem ser usados para detectar pontos de dados incomuns ou anômalos.

Resumo da Aula (4/4)

Por fim, abordamos a importância da visualização de dados e como diferentes técnicas e ferramentas podem ser usadas para visualizar dados em alta dimensão e facilitar a interpretação de resultados obtidos com algoritmos de aprendizagem não supervisionada. A aprendizagem não supervisionada oferece uma ampla gama de ferramentas para ajudar a explorar e entender conjuntos de dados não rotulados e pode ser aplicada a uma variedade de problemas em diversas áreas, como marketing, finanças, medicina, entre outras.

FAESA
CENTRO UNIVERSITÁRIO

Análise de Dados Aplicada à Computação

PROF. M.SC HOWARD ROATTI