LISTA 5

ALUNO: Caio Gomes Alcântara Glória

MATRICULA: 763989

PROFESSORA: Cristiane Neri

Questão 1)

Calcular o suporte dos itemsets de 1 item:

Contar quantas vezes cada item aparece nas transações e verificar se o suporte é maior ou igual a 0.3 (30%).

Gerar os itemsets de 2 itens e calcular o suporte:

Verificar combinações de dois itens que aparecem juntos e calcular seu suporte.Apenas os itemsets que satisfazem o suporte mínimo são considerados.

Suporte dos itemsets de 1 item:

Leite: 3 vezes (0.3)

Café: 5 vezes (0.5)

Cerveja: 3 vezes (0.3)

Pão: 5 vezes (0.5)

Manteiga: 5 vezes (0.5)

Arroz: 4 vezes (0.4)

Feijão: 4 vezes (0.4

Suporte dos itemsets de 2 itens:

(Leite, Café): 2 vezes (0.2) → Não atende.

(Leite, Cerveja): 1 vez (0.1) → Não atende.

(Leite, Pão): 2 vezes (0.2) → Não atende.

(Leite, Manteiga): 3 vezes (0.3) → Atende.

(Leite, Arroz): 2 vezes (0.2) → Não atende.

(Leite, Feijão): 1 vez (0.1) → Não atende.

Questão 2)

Questão 3)

```
import pandas as pd
Antecedente = []
Consequente = []
suporte = []
confianca = []
lift = []
RegrasFinais = []
for resultado in saida:
    s = resultado[1]
    result_rules = resultado[2]

    # Imprime o itemset e seu suporte
    print(f"Itemset: {resultado[0]}, Suporte: {s:.2f}")

    for result_rule in result_rules:
        a = list(result_rule[0])
        b = list(result_rule[1])
        c = result_rule[2]
        l = result_rule[3]

        if 'nan' in a or 'nan' in b:
            continue
        if len(a) == 0 or len(b) == 0:
            continue

        Antecedente.append(a)
        Consequente.append(b)
        suporte.append(s)
        confianca.append(c)
        lift.append(l)
RegrasFinais = pd.DataFrame({
    'Antecedente': Antecedente,
    'Consequente': Consequente,
    'suporte': suporte,
    'confianca': confianca,
    'lift': lift
})
print(RegrasFinais)
```

Questão 4)

```
import pandas as pd

Antecedente = []
Consequente = []
suporte = []
confianca = []
lift = []
RegrasFinais = []

for resultado in saida:
    s = resultado[1]
    result_rules = resultado[2]


    print(f"Itemset: {resultado[0]}, Suporte: {s:.2f}")

    for result_rule in result_rules:
        a = list(result_rule[0])
        b = list(result_rule[1])
        c = result_rule[2]
        l = result_rule[3]

        # Modifica para considerar a ausência de produtos
        a_neg = [f"não {item}" for item in a]
        b_neg = [f"não {item}" for item in b]


        if 'nan' in a_neg or 'nan' in b_neg:
            continue
        if len(a_neg) == 0 or len(b_neg) == 0:
            continue

        Antecedente.append(a_neg)
        Consequente.append(b_neg)
        suporte.append(s)
        confianca.append(c)
        lift.append(l)
```

```
# Cria o DataFrame com as regras finais
RegrasFinais = pd.DataFrame({
    'Antecedente': Antecedente,
    'Consequente': Consequente,
    'suporte': suporte,
    'confianca': confianca,
    'lift': lift
})
print(RegrasFinais)
```

Questão 5)

A mlxtend é uma biblioteca do Python projetada para facilitar a implementação de algoritmos de machine learning e tarefas de pré-processamento de dados. No contexto da geração de regras de associação, ela oferece módulos eficientes para aplicar algoritmos de mineração de padrões como Apriori e FP-Growth, com o objetivo de encontrar associações frequentes em conjuntos de dados transacionais.

Funcionamento da Geração de Regras de Associação com mlxtend:

1. Algoritmo Apriori: O mlxtend implementa o algoritmo Apriori que é amplamente usado para identificar itemsets frequentes em transações. Ele funciona gerando todos os conjuntos de itens possíveis e filtrando aqueles que atendem a um limite mínimo de suporte. Isso ajuda a revelar combinações de itens que frequentemente aparecem juntas em um conjunto de dados, como compras em supermercados.

2. Geração de Regras: Com os itemsets frequentes identificados, a mlxtend pode gerar regras de associação, onde a biblioteca calcula métricas como confiança, lift e support para avaliar a relevância das regras geradas. As regras são do tipo "Se o item A for comprado, então o item B também será", sendo úteis para decisões estratégicas em marketing e vendas.

3. Facilidade de Uso: A mlxtend é conhecida por sua interface amigável e integração direta com o pandas, permitindo que as transações sejam fornecidas como dataframes. A função apriori() processa o dataframe para encontrar os itemsets frequentes, enquanto a função association_rules() gera as regras com base nas métricas desejadas.

Ex código em pytohn:

```
from mlxtend.frequent_patterns import apriori, association_rules

transactions = pd.DataFrame([

   [1, 0, 1, 0],

   [1, 1, 0, 0],

   [0, 1, 1, 1],

], columns=['Leite', 'Pão', 'Manteiga', 'Café'])

itemsets = apriori(transactions, min_support=0.5, use_colnames=True)
```

regras = association_rules(itemsets, metric="confidence", min_threshold=0.7)

print(regras)

Vantagens:

- Simplicidade: A mlxtend é de fácil aprendizado e implementação.

- Flexibilidade: Suporte para diversas métricas e ajustes, como o limite de suporte mínimo e o - uso de colunas nomeadas.

- Integração: Compatível com bibliotecas populares como pandas e numpy.

Questão 6)

O artigo “A comprehensive review of visualization methods for association rule mining: Taxonomy, challenges, open problems and future ideas” apresenta uma análise abrangente das técnicas de visualização utilizadas na mineração de regras de associação. Os autores destacam a importância de métodos de visualização para facilitar a interpretação e compreensão dos resultados da mineração de regras, especialmente em grandes volumes de dados.

Principais Pontos do Artigo:

Motivação e Relevância: A mineração de regras de associação (ARM) é uma técnica vital na descoberta de relações entre atributos em bancos de dados transacionais. A visualização das regras é um componente crucial para tornar os resultados mais compreensíveis e úteis para os usuários.

Histórico e Evolução: O artigo traça a evolução dos métodos de visualização de ARM, começando com os gráficos de dispersão e avançando para representações mais complexas, como diagramas de Sankey e mapas de metrô.

Desafios e Limitações: São discutidos os desafios associados a essas visualizações, como a sobrecarga visual em representações com muitos itens e a dificuldade de representar múltiplas medidas de interesse (como suporte e confiança) de forma eficaz.

Classificação dos Métodos:

Métodos Tradicionais: Incluem gráficos de dispersão, gráficos baseados em matriz e diagramas de mosaico.

Métodos Modernos: Como diagramas de Sankey, gráficos de mapa de metrô e visualizações moleculares, que oferecem representações mais intuitivas e detalhadas.

Propostas Futuras e Open Problems: Os autores destacam a necessidade de métodos que combinem clareza visual com interatividade e suporte a dados de alta dimensionalidade. A integração com técnicas de inteligência artificial explicável (XAI) é vista como uma direção promissora para tornar as visualizações mais transparentes e compreensíveis.

Aplicações Práticas: O artigo cita a aplicação dessas visualizações em áreas como análise de mercado, diagnósticos médicos e mineração de padrões em séries temporais.

Conclusão:

O artigo conclui que, embora muitos métodos tenham sido desenvolvidos, há uma lacuna na integração de soluções que sejam simultaneamente interativas, escaláveis e eficazes em representar grandes volumes de regras. Os autores sugerem um foco contínuo em inovações que combinem simplificação visual e técnicas de XAI.

Essa revisão fornece uma visão abrangente das técnicas de visualização de ARM e incentiva pesquisas futuras em métodos que combinem eficiência visual com acessibilidade e aplicabilidade prática.