Lista 5 - Inteligência Artificial

Henrique Oliveira da Cunha Franco

Questão 1

Código usado para elaboração das questões: LinkCodigo
Código em Latex desse arquivo no Overleaf: LinkLatex

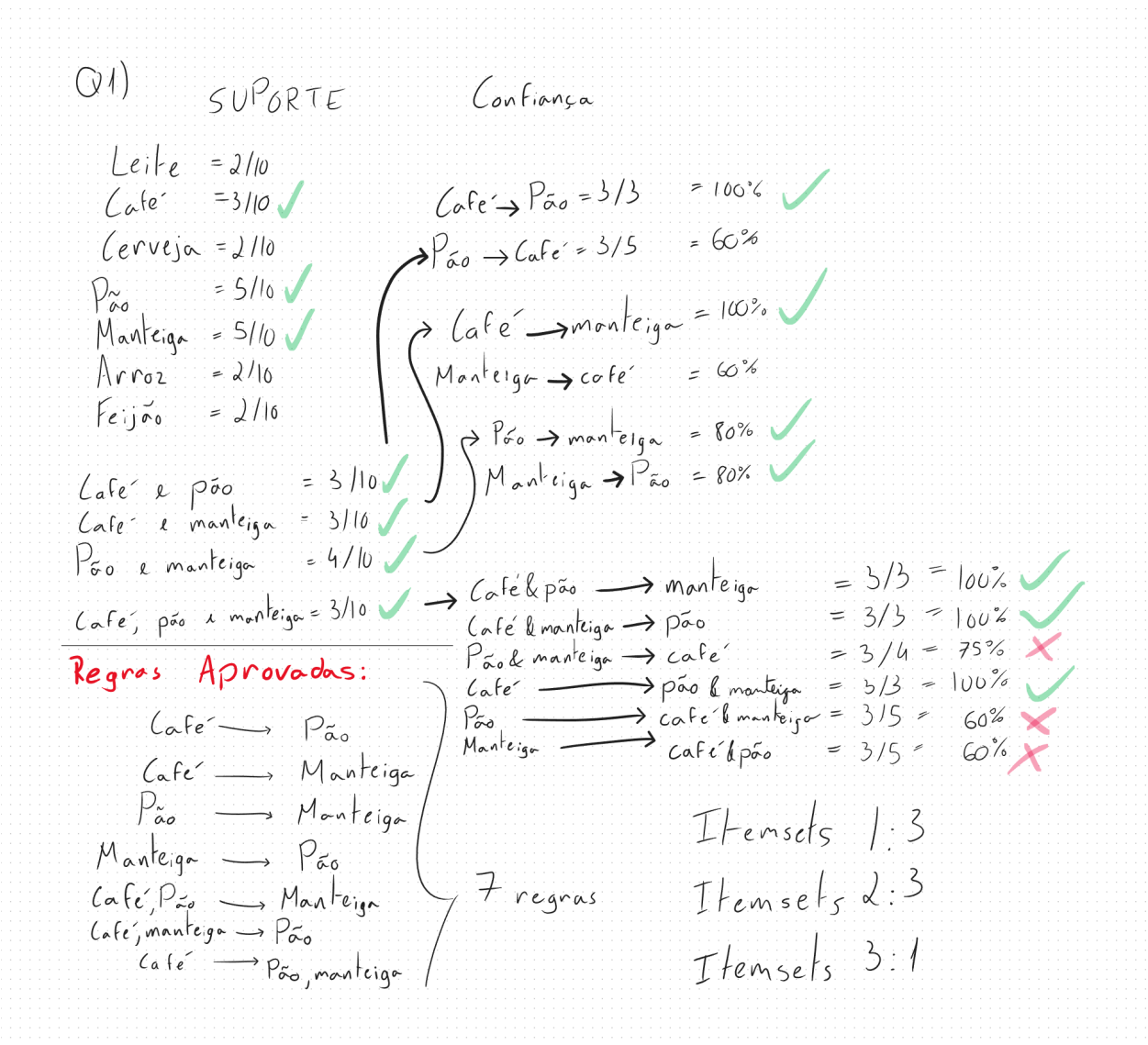

Figure 1: Questão 1

Questão 2

Analisando os resultados obtidos a partir da execução do algoritmo, é possível afirmar que os resultados foram idênticos, gerando 7 regras.

```
1  RegrasFinais.sort_values(by='lift', ascending =False)
```

Visualize

	Antecedente obj...	Consequente obj...	suporte float64	confianca float64	lift float64
4	['Café']	['Manteiga', 'Pão']	0.3	1	2.5
0	['Café']	['Manteiga']	0.3	1	2
1	['Café']	['Pão']	0.3	1	2
5	['Manteiga', 'Café']	['Pão']	0.3	1	2
6	['Pão', 'Café']	['Manteiga']	0.3	1	2
2	['Manteiga']	['Pão']	0.4	0.8	1.6
3	['Pão']	['Manteiga']	0.4	0.8	1.6

7 rows, 5 cols 10 / page « ‹ Page 1 of 1 › » Format

Figure 2: Questão 2

Questão 3

Para realizar a impressão de todos os ItemSet's, um novo de trecho foi criado, que por sua vez passa pelos sets de itens gerados que apresentam suporte acima do mínimo estabelecido e imprime suas informações na tela.

```
1  i = 0
2  print("ItemSets:")
3  for result in saida:
4      if "nan" in result.items:
5          continue
6      i += 1
7      print(f"ItemSet {i}:", list(result.items), "\t| Suporte:", result.support)
```

```
ItemSets:
ItemSet 1: ['Manteiga', 'Café']    | Suporte: 0.3
ItemSet 2: ['Pão', 'Café']         | Suporte: 0.3
ItemSet 3: ['Manteiga', 'Pão']     | Suporte: 0.4
ItemSet 4: ['Manteiga', 'Pão', 'Café']      | Suporte: 0.3
```

Figure 3: Questão 3

Questão 4

Para realizar o que foi pedido, é possível alterar o estado da variável `criarRegrasNegativas`, que por sua vez age como um controle das regras geradas. Se estiver em seu estado **False**, ela gera somente regras positivas (comportamento padrão, apresentado na imagem 4).

Inicialização da leitura de arquivo e estabelecimento da base:

```
base = pd.read_csv(
    "/work/lista5/PãoeManteiga Sim.csv", sep=";", encoding="cp1252", header=None
)
criarRegrasNegativas = False
for col in base.columns:
    primeiroElemento = base[col].iloc[0]

    if criarRegrasNegativas:
        base[col] = base[col].apply(
            lambda x: primeiroElemento if x != "?" else f"nao {primeiroElemento}"
        )
    else:
        base[col] = base[col].apply(lambda x: primeiroElemento if x != "?" else "nan")


base = base.drop(base.index[0]).reset_index(drop=True)
base
```

Visualize

	0 object	1 object	2 object	3 object	4 object	5 object	6 object
	nan 80% Leite 20%	nan 70% Café 30%	nan 80% Cerveja 20%	Pão 50% nan 50%	Manteiga 50% nan 50%	nan 80% Arroz 20%	nan 80% Feijão 20%
0	nan	Café	nan	Pão	Manteiga	nan	nan
1	Leite	nan	Cerveja	Pão	Manteiga	nan	nan
2	nan	Café	nan	Pão	Manteiga	nan	nan
3	Leite	Café	nan	Pão	Manteiga	nan	nan
4	nan	nan	Cerveja	nan	nan	nan	nan
5	nan	nan	nan	nan	Manteiga	nan	nan
6	nan	nan	nan	Pão	nan	nan	nan
7	nan	nan	nan	nan	nan	nan	Feijão
8	nan	nan	nan	nan	nan	Arroz	Feijão
9	nan	nan	nan	nan	nan	Arroz	nan

10 rows, 7 cols 10 / page « ‹ Page 1 of 1 › » Format

Figure 4: Geração de regras positivas

Entretanto, como foi solicitado, deseja-se obter as regras de associação quando não há presença do produto. Logo, logicamente, alterando o estado da variável `criarRegrasNegativas` para **True**, somente regras negativas são geradas, resultando em um alto número de regras elaboradas. Isso é possível de ser observado a seguir, na imagem 5:

```
1 RegrasFinais.sort_values(by='lift', ascending =False)
```

Visualize

	Antecedente obj...	Consequente obj...	suporte float64	confianca float64	lift float64
	['Café'] 7% ['nao Feijão', ... 3.4% 116 others 89.7%	['nao Cervej... 10.1% ['nao Arroz'] 9.2% 47 others 80.7%	0.3 - 0.7	0.8 - 1.0	1.0 - 3.333333333...
444	['Pão', 'Manteiga', ...	['Café']	0.3	1	3.333333333
354	['Pão', 'Manteiga', ...	['nao Feijão', 'Café']	0.3	1	3.333333333
343	['Café']	['nao Feijão', 'Pão'...	0.3	1	3.333333333
410	['Café']	['Pão', 'Manteiga', ...	0.3	1	3.333333333
415	['nao Feijão', 'Café']	['Pão', 'Manteiga', ...	0.3	1	3.333333333
158	['Pão', 'Manteiga', ...	['Café']	0.3	1	3.333333333
347	['nao Feijão', 'Café']	['Pão', 'Manteiga', ...	0.3	1	3.333333333
327	['Pão', 'Manteiga', ...	['Café']	0.3	1	3.333333333
322	['Pão', 'Manteiga', ...	['Café', 'nao Arroz']	0.3	1	3.333333333
424	['nao Feijão', 'Caf...	['Pão', 'Manteiga', ...	0.3	1	3.333333333

445 rows, 5 cols 10 / page Page 1 of 45 Format

Figure 5: Resultado das regras incluindo negativas

Questão 5

A biblioteca **mlxtend** oferece funcionalidades para mineração de padrões frequentes e regras de associação, incluindo algoritmos como **Apriori** e **FP-Growth**. O módulo `frequent_patterns` permite identificar itens frequentes e extrair regras de associação úteis em dados transacionais.

O método **Apriori** da mlxtend identifica conjuntos de itens frequentes com base no suporte mínimo especificado. Para gerar regras de associação, utiliza-se o método `association_rules`, que calcula métricas como **confiança**, **lift**, e **convicção** entre pares de itens, possibilitando análises sobre a probabilidade de co-ocorrência entre eles.

A função `apriori` possui parâmetros importantes:

- `df`: um DataFrame binário representando itens transacionais.
- `min_support`: define o limite de frequência mínima.
- `use_colnames`: se `True`, retorna nomes dos itens em vez de índices.

Após identificar conjuntos frequentes, `association_rules` gera as regras usando um DataFrame com colunas `antecedents`, `consequents`, `support`, `confidence`, e `lift`. Isso permite identificar padrões relevantes para recomendações ou marketing.

Exemplo em código:

```
from mlxtend.frequent_patterns import apriori, association_rules
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
```

O **mlxtend** facilita a aplicação de mineração de dados de associação em Python de forma eficiente e com implementações robustas.

Questão 6

O artigo *A Comprehensive Review of Visualization Methods for Association Rule Mining: Taxonomy, Challenges, Open Problems and Future Ideas* analisa métodos de visualização para **mineração de regras de associação** (ARM), com foco em organizar e avaliar essas técnicas. A ARM é uma ferramenta de mineração de dados que identifica relações em grandes conjuntos de dados, historicamente aplicada em contextos como análises de mercado, diagnóstico médico e padrões de comportamento. A **visualização** é crucial para interpretar os resultados de ARM, permitindo que usuários compreendam melhor as associações descobertas.

A pesquisa revisa métodos tradicionais, como **gráficos de dispersão** e **diagramas de grafos**, e novos métodos, como **mapas de metrô** e **diagramas de Sankey**, que exploram diferentes abordagens visuais e métricas para facilitar a compreensão dos dados. Cada método é avaliado com base em critérios como número de medidas de interesse, interatividade e aplicabilidade a tipos específicos de dados. Por exemplo, gráficos de dispersão ajudam a visualizar *suporte* e *confiança* em regras, enquanto diagramas de grafos e Sankey representam as conexões entre itens de forma mais interativa.

Os autores também destacam desafios como a dificuldade de representar conjuntos de regras complexas e a limitação dos métodos atuais para dados não binários. Concluem apontando a necessidade de novas pesquisas para desenvolver ferramentas visuais que combinem a simplicidade com a capacidade de interpretar dados complexos, além de destacar a importância de integrar ARM com **inteligência artificial explicável** para melhorar a confiança dos usuários nas decisões baseadas em ARM.