Entregable: Maximal/Closed frequent itemsets (R/Python)

Índice

Introducción

Este proyecto implementa algoritmos para descubrir **itemsets frecuentes, cerrados (closed)** y **maximal** a partir de datos transaccionales. La implementación es completamente propia, sin usar librerías externas para los algoritmos principales.

Objetivo

Dado un conjunto de transacciones (por ejemplo, compras en un supermercado), encontrar:

- **Itemsets frecuentes**: Conjuntos de productos que aparecen juntos con frecuencia
- **Closed itemsets**: Un conjunto de ítems X es cerrado si es frecuente y no existe ningún superconjunto de X que posea su mismo valor de soporte.
- **Maximal itemsets**: Un conjunto de ítems X es maximal si es frecuente y ninguno de sus superconjuntos inmediatos es frecuente.

Conceptos Fundamentales

Transacción

Una transacción es un conjunto de items comprados juntos. Por ejemplo:

```
Transacción 1: {Leche, Pan, Mantequilla}
Transacción 2: {Leche, Huevos}
Transacción 3: {Pan, Mantequilla, Huevos}
```

Soporte

El **soporte** de un itemset es la proporción de transacciones que lo contienen:

```
soporte(X) = |transacciones que contienen X| / |total de transacciones|
```

Por ejemplo, si `{Leche}` aparece en 15 de 20 transacciones:

```
soporte({Leche}) = 15/20 = 0.75 = 75%
```

Itemset Frecuente

Un itemset es **frecuente** si su soporte es mayor o igual al soporte mínimo definido.

Propiedad Anti Monótona

El soporte de un conjunto de ítems (itemset) es una función antimonótona: a medida que el tamaño del conjunto aumenta, su soporte disminuye o se mantiene igual, pero nunca aumenta. En consecuencia, si un itemset es frecuente, todos sus subconjuntos deben serlo también; por el contrario, si un itemset es infrecuente, cualquier superconjunto que lo contenga será inevitablemente infrecuente.

Esta propiedad es fundamental para la eficiencia del algoritmo Apriori, ya que permite realizar el "podado" (pruning) de ramas enteras en el espacio de búsqueda.

Algoritmo Apriori

Descripción

El algoritmo Apriori descubre todos los itemsets frecuentes de forma iterativa, empezando por itemsets de tamaño 1 y aumentando progresivamente.

Implementación: Función Principal (`apriori.py`)

La función `get_frequent_itemsets` implementa el ciclo principal del algoritmo:

```
def get_frequent_itemsets(
    transactions: List[frozenset], min_support: float
) -> Dict[frozenset, int]:
    n_transactions = len(transactions)
    min_support_count = int(min_support * n_transactions)

    frequent_itemsets: Dict[frozenset, int] = {}

    # PASO 1: Extraer todos los items únicos de las transacciones
    all_items = set()
    for transaction in transactions:
        all_items.update(transaction)

    # PASO 2: Crear itemsets de tamaño 1 y contar su soporte
    single_itemsets = {frozenset([item]) for item in all_items}
    support_counts = count_support(transactions, single_itemsets)

    # PASO 3: Filtrar por soporte mínimo
    current_frequent = {
        itemset
        for itemset, count in support_counts.items()
        if count >= min_support_count
    }

    for itemset in current_frequent:
        frequent_itemsets[itemset] = support_counts[itemset]

    # PASO 4: Iterar generando candidatos de tamaño creciente
    k = 2
    while current_frequent:
        candidates = generate_candidates(current_frequent, k)

        if not candidates:
            break

        support_counts = count_support(transactions, candidates)

        current_frequent = {
            itemset
            for itemset, count in support_counts.items()
            if count >= min_support_count
        }

        for itemset in current_frequent:
            frequent_itemsets[itemset] = support_counts[itemset]

        k += 1

    return frequent_itemsets
```

Implementación: Conteo de Soporte

La función `count_support` recorre todas las transacciones contando cuántas contienen cada itemset:

```
def count_support(
    transactions: List[frozenset], itemsets: Set[frozenset]
) -> Dict[frozenset, int]:
    support_counts = {itemset: 0 for itemset in itemsets}

    for transaction in transactions:
        for itemset in itemsets:
            # Un itemset está en la transacción si es subconjunto
            if itemset.issubset(transaction):
                support_counts[itemset] += 1

    return support_counts
```

Implementación: Generación de Candidatos

La función `generate_candidates` genera candidatos de tamaño k a partir de frecuentes de tamaño k-1, aplicando la propiedad de clausura descendente:

```
def generate_candidates(frequent_itemsets: Set[frozenset], k: int) -> Set[frozenset]:
    candidates = set()
    itemsets_list = list(frequent_itemsets)

    # Combinar pares de itemsets frecuentes
    for i in range(len(itemsets_list)):
        for j in range(i + 1, len(itemsets_list)):
            union = itemsets_list[i] | itemsets_list[j]

            # Solo considerar si la unión tiene tamaño k
            if len(union) == k:
                # PODA: Verificar que todos los subconjuntos de tamaño k-1 sean frecuentes
                all_subsets_frequent = True
                for subset in combinations(union, k - 1):
                    if frozenset(subset) not in frequent_itemsets:
                        all_subsets_frequent = False
                        break

                if all_subsets_frequent:
                    candidates.add(union)

    return candidates
```

Complejidad

- **Peor caso**: O(2^n) donde n es el número de items únicos.
- **Caso práctico**: Mucho menor gracias a la poda por la propiedad anti monótona.

Detección de Closed Itemsets

Definición

Un itemset X es **cerrado (closed)** si es frecuente y no existe ningún superconjunto Y ⊃ X que tenga el mismo soporte.

```
X es closed ⟺ ∀Y ⊃ X : soporte(Y) < soporte(X)
```

Intuición

Los closed itemsets son aquellos donde añadir cualquier item adicional reduciría el soporte. Representan la "frontera" donde el soporte cambia.

Ejemplo

Itemset	Soporte

Itemset	Soporte
{A}	10
{B}	10
{A, B}	10
{A, B, C}	3

- `{A}` NO es closed → `{A, B}` tiene el mismo soporte (10)
- `{B}` NO es closed → `{A, B}` tiene el mismo soporte (10)
- `{A, B}` SÍ es closed → ningún superconjunto tiene soporte 10
- `{A, B, C}` SÍ es closed → no hay superconjunto frecuente

Implementación Completa (`closed_itemsets.py`)

```
def find_closed_itemsets(
    frequent_itemsets: Dict[frozenset, int]
) -> Dict[frozenset, int]:
    if not frequent_itemsets:
        return {}

    # Agrupar itemsets por tamaño para búsqueda eficiente de superconjuntos
    itemsets_by_size: Dict[int, List[frozenset]] = defaultdict(list)
    for itemset in frequent_itemsets:
        itemsets_by_size[len(itemset)].append(itemset)

    sizes = sorted(itemsets_by_size.keys(), reverse=True)
    closed_itemsets = {}

    for size in sizes:
        for itemset in itemsets_by_size[size]:
            support = frequent_itemsets[itemset]
            is_closed = True

            # Solo buscar en itemsets de mayor tamaño (posibles superconjuntos)
            for larger_size in sizes:
                if larger_size <= size:
                    break  # sizes ordenado desc, no hay más tamaños mayores

                for other_itemset in itemsets_by_size[larger_size]:
                    if itemset < other_itemset:  # es subconjunto propio
                        if frequent_itemsets[other_itemset] == support:
                            is_closed = False
                            break

                if not is_closed:
                    break

            if is_closed:
                closed_itemsets[itemset] = support

    return closed_itemsets
```

Explicación del Algoritmo

1. **Agrupación por tamaño**: Usando `defaultdict(list)` agrupamos los itemsets por su longitud
2. **Iteración ordenada**: Recorremos los tamaños de mayor a menor
3. **Búsqueda selectiva**: Solo comparamos con itemsets de tamaño estrictamente mayor
4. **Verificación**: Si existe un superconjunto con el mismo soporte, X no es closed
5. **Optimización**: Evitamos O(n²) comparaciones innecesarias

Propiedad Importante

Los closed itemsets son una **compresión sin pérdida** de los itemsets frecuentes. A partir de ellos se puede reconstruir el soporte de cualquier itemset frecuente.

Detección de Maximal Itemsets

Definición

Un itemset X es **maximal** si no existe ningún superconjunto Y ⊃ X que también sea frecuente.

```
X es maximal ⟺ ∀Y ⊃ X : Y no es frecuente
```

Intuición

Los maximal itemsets son los itemsets frecuentes "más grandes" - no se pueden extender sin perder la propiedad de frecuente.

Ejemplo

Itemset	Frecuente
{A}	Sí
{B}	Sí
{C}	Sí
{A, B}	Sí
{A, C}	Sí
{A, B, C}	Sí

Solo `{A, B, C}` es maximal porque todos los demás tienen un superconjunto frecuente.

Implementación Completa (`maximal_itemsets.py`)

```
def find_maximal_itemsets(
    frequent_itemsets: Dict[frozenset, int]
) -> List[frozenset]:
    if not frequent_itemsets:
        return []

    # Agrupar itemsets por tamaño para búsqueda eficiente de superconjuntos
    itemsets_by_size: Dict[int, List[frozenset]] = defaultdict(list)
    for itemset in frequent_itemsets:
        itemsets_by_size[len(itemset)].append(itemset)

    sizes = sorted(itemsets_by_size.keys(), reverse=True)
    maximal_itemsets = []

    for size in sizes:
        for itemset in itemsets_by_size[size]:
            is_maximal = True

            # Solo buscar en itemsets de mayor tamaño (posibles superconjuntos)
            for larger_size in sizes:
                if larger_size <= size:
                    break  # sizes ordenado desc, no hay más tamaños mayores

                for other_itemset in itemsets_by_size[larger_size]:
                    if itemset < other_itemset:  # es subconjunto propio
                        is_maximal = False
                        break

                if not is_maximal:
                    break

            if is_maximal:
                maximal_itemsets.append(itemset)

    return maximal_itemsets
```

Explicación del Algoritmo

1. **Agrupación por tamaño**: Usando `defaultdict(list)` agrupamos los itemsets por su longitud
2. **Iteración ordenada**: Recorremos los tamaños de mayor a menor
3. **Búsqueda selectiva**: Solo comparamos con itemsets de tamaño estrictamente mayor
4. **Decisión**: Si existe cualquier superconjunto frecuente, X no es maximal
5. **Resultado**: Solo añadimos los itemsets que no tienen superconjuntos frecuentes

Diferencia con Closed Itemsets

Aspecto	Closed	Maximal
Condición	Sin superconjunto de**igual soporte**	Sin **ningún** superconjunto frecuente
Información	Preserva soportes (lossless)	Pierde soportes (lossy)
Uso típico	Reglas de asociación	Identificar patrones máximos

Propiedad Importante

Los maximal itemsets son una **compresión con pérdida** - sabemos qué itemsets son frecuentes, pero perdemos información sobre los soportes individuales.

Relación entre Tipos de Itemsets

```
Maximal ⊆ Closed ⊆ Frequent
```

Tipo	Descripción	Información Preservada
Frequent	Todos los itemsets con soporte ≥ mínimo	Completa
Closed	Frequent sin superconjuntos de igual soporte	Completa
Maximal	Frequent sin superconjuntos frecuentes	Parcial

Guía de Ejecución

1. Preparar el Entorno

```
# Descargar el proyecto
cd maximal-closed-frequent-itemsets

# Crear entorno virtual (Opcional ya que no hay dependencias externas)
python -m venv .venv
source .venv/bin/activate  # Linux/Mac
# .venv\Scripts\activate   # Windows
```

2. Ejecutar el Programa

```
# Ejecución básica con dataset por defecto (groceries.csv)
python main.py

# Especificar dataset y parámetros
python main.py --data data/groceries.csv --min-support 0.05

# Limitar número de transacciones (para pruebas)
python main.py --data data/groceries.csv --min-support 0.05 --limit 200

# Dataset con cabecera
python main.py --data data/market.csv --min-support 0.05 --header true
```

3. Parámetros Disponibles

Parámetro	Descripción	Ejemplo
`--data`	Ruta al archivo CSV	`--data data/groceries.csv`
`--min-support`	Soporte mínimo (0-1)	`--min-support 0.05`
`--limit`	Máximo de transacciones	`--limit 500`
`--top`	Resultados a mostrar	`--top 10`
`--header`	Si CSV tiene cabecera	`--header true`

Interpretación de Resultados

Salida del Programa

```
============================================================
MAXIMAL AND CLOSED FREQUENT ITEMSETS MINING
============================================================

Loading transactions from: data/validation.csv
  - Transactions loaded: 20      ← Número de transacciones cargadas
  - Unique items: 5              ← Items únicos encontrados
  - Min support: 0.15 (3 transactions)  ← Umbral de soporte

Finding frequent itemsets (Apriori algorithm)...
  - Frequent itemsets found: 13  ← Total de itemsets frecuentes

Finding closed itemsets...
  - Closed itemsets found: 8     ← Subconjunto: closed itemsets

Finding maximal itemsets...
  - Maximal itemsets found: 2    ← Subconjunto: maximal itemsets
```

Verificación de Propiedades

```
Property verification: |maximal| ≤ |closed| ≤ |frequent|
2 ≤ 8 ≤ 13: ✓
```

Esta verificación confirma que la relación teórica se cumple: los maximal son un subconjunto de los closed, que a su vez son un subconjunto de los frequent.

Interpretación de Itemsets

Frequent Itemsets: Combinaciones de productos que aparecen juntos frecuentemente.

- Útil para: Análisis de cesta de compra, recomendaciones

Closed Itemsets: Representación compacta que preserva toda la información de soporte.

- Útil para: Almacenamiento eficiente, reglas de asociación

Maximal Itemsets: Los patrones más largos/completos.

- Útil para: Identificar las asociaciones más fuertes y completas

---

Autor

Miguel de la Fuente Muñoz
Máster en Investigación en Inteligencia Artificial - UIMP