

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА - Российский технологический университет» РТУ МИРЭА**

**Институт** Информационных Технологий

**Кафедра** Вычислительной техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**по дисциплине**

**«Проектирование интеллектуальных систем (Часть 1/2)»**

Студент группы ИКБО-04-22 Сариков Т.А.

*(Ф.И.О. студента)*

Руководитель работы Холмогоров В.В.

*(Ф.И.О. преподавателя)*

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc10014)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc10015)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc10016)

[2.1 Алгоритм Apriori 6](#_Toc10017)

[2.4 Генерация ассоциативных правил 8](#_Toc10020)

[3 ОПИСАНИЕ ДАННЫХ 9](#_Toc10021)

[3.1 Генерация данных 9](#_Toc10022)

[4 ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc10023)

[4.1 Реализация алгоритма Apriori 11](#_Toc10024)

[ЗАКЛЮЧЕНИЕ 17](#_Toc10027)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc10028)

[ПРИЛОЖЕНИЯ 19](#_Toc10029)

# 

# ВВЕДЕНИЕ

Задача поиска ассоциативных правил заключается в нахождении зависимостей в транзакциях - насколько часто один товар берут в паре с другим товаром, насколько часто одно явление происходит зависимо от второго.

В рассматриваемой предметной области «Анализ корзины покупателя» необходимо найти частые комбинации покупки товаров для системы рекомендации товаров, установки скидок.

# 1 ПОСТАНОВКА ЗАДАЧИ

Выбранная предметная область: «Анализ корзины покупателя».

Транзакция это множество продуктов купленных человеком

Цели:

* реализовать алгоритм поиска ассоциативных правил в рамках поставленной предметной области.

Поставлены следующие задачи:

* определить и изучить датасет, соотвествующий выбранной предметной области;
* изучить алгоритм поиска ассоциативных правил Apriori
* написать программный код для реализации указанного алгоритма;

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Для того, чтобы выявить ассоциативные правила, в первую очередь необходимо понимать, как определить качество этих правил. Для этого можно ввести следующие метрики (далее 𝐴 — множество объектов, составляющих условие, 𝐶 — множество объектов, составляющих следствие, 𝐴 → 𝐶 — рассматриваемся ассоциация, 𝑇 — множество всех транзакций):

### 1. Поддержка

Показывает вероятность встретить набор A и C в одной транзакции одновременно и высчитывается по Формуле 2.1.

|𝐴 ∪ 𝐶|

𝑠𝑢𝑝𝑝(𝐴 → 𝐶) = (2.1)

|𝑇|

### 2. Доверие

Показывает, насколько часто правило оказывается верным, т.е. по сути является условной вероятностью, что в наборе окажется множество C, если в нём есть множество A. Вычисляется по Формуле 2.2.

supp(𝐴 → 𝐶)

conf(𝐴 → 𝐶) = (2.2)

supp(𝐴)

### 3. Достоверность (убеждённость)

Показывает независимость A и C - чем больше значение, тем более зависимы. Вычисляется по Формуле 2.3.

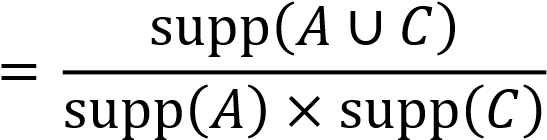
1 − sup(𝐶)

conv(𝐴 → 𝐶) = (2.3)

1 − conf(𝐴 → 𝐶)

### 4. Лифт

Также показывает, насколько независимы A и C друг от друга. Если лифт равен 1, то условие и следствие независимы, если больше 1, то зависимы, а если меньше 1, то наличие множества A в наборе имеет отрицательный эффект на присутствие второго объекта, и наоборот. Вычисляется по Формуле 2.4.

lift(𝐴 → 𝐶)  (2.4)

### 5. Рычаг

Также показывает, насколько независимы A и C друг от друга. Вычисляется как разность вероятности встретить A и C одновременно и мат.ожидания A и C, как если бы они были независимы друг от друга (Формула 2.5).

leve(𝐴 → 𝐶) = supp(𝐴 ∪ 𝐶) − supp(𝐴) × supp(𝐶) (2.5)

Алгоритмы поиска ассоциативных правил рассчитаны на ускорение перебора всех возможных наборов условий и следствий за счёт отбрасывания заведомо редких наборов. Поэтому наиболее применимой метрикой является поддержка, показывающая вероятность встретить набор во всём множестве транзакций.

## 2.1 Алгоритм Apriori

Для данного алгоритма данные представляются в нормализованном виде,

т.е. если в i-ой транзакции элемент *Item* встречается, то в датасете в соответствующей ячейке будет стоять 1, и 0 в обратном случае.

Далее строится префиксное дерево - родителем каждого узла является набор, содержащий префикс данного набора без одного элемента. Для первого уровня просто отбираются те элементы, которые встречаются чаще заданного порога, далее каждый уровень заполняется, используя наборы из предыдущего.

Для генерации наборов k-го уровня объединяются попарно все наборы k - 1 уровня, которые имеют одинаковый префикс. Для полученного набора проверяется множество его подмножеств, состоящих из k - 1 элемента - если хотя бы одно такое подмножество встречается реже заданного порога, то набор отбрасывается.

Алгоритм выполняется до тех пор, пока есть возможность составлять набор более высоких уровней.

## 2.4 Генерация ассоциативных правил

Все рассмотренные выше алгоритмы предназначены для ускорения поиска часто встречающихся наборов, но генерация самих правил для них общая.

Из каждого множества размером k рассматриваются все подмножества размером k - 1 — эти подмножества становятся условием в правиле, а оставшийся элемент — следствием. Уже для составленных правил выполняется отбор на основе, например, метрики доверия.

# 3 ОПИСАНИЕ ДАННЫХ

## 3.1 Анализ данных

Датасет был взят с платформы Kaggle

Всего 9835 наборов и максимальная по количеству товаров корзинка состоит из 32 продуктов

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 3.1 — Количественные характеристики распределения корзины**

# 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

Для всех алгоритмов задан одинаковый минимальный порог поддержки: 4.

## 4.1 Реализация алгоритма Apriori

После выполнения самого алгоритма получено (Рисунок 4.1).

Изображение выглядит как текст, меню, снимок экрана, черно-белый

Контент, сгенерированный ИИ, может содержать ошибки.

#### Рисунок 4.1 — Правила, полученные алгоритмом Apriori

# ЗАКЛЮЧЕНИЕ

В результате реализации алгоритма Apriori выяснено, что данный алгоритм требует значительного времени на нормализацию и подготовку данных. Алгоритм использует массивы и множества для хранения промежуточных результатов в памяти, что требует существенных ресурсов памяти при работе с большими наборами данных. Apriori показал умеренные результаты по количеству найденных ассоциативных правил..

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Azevedo, P. J., Jorge, A. M. Comparing Rule Measures for Predictive Association Rules 2007.

# ПРИЛОЖЕНИЯ

Приложение А — Реализация алгоритма Apriori.

**Приложение А**

Реализация алгоритма Apriori

*Листинг А.1 — Генерация данных*

print('Hello world')

import pandas as pd

import numpy as np

from itertools import combinations

def custom\_apriori(transactions, min\_support):

    all\_items = set()

    for transaction in transactions:

        all\_items.update(transaction)

    all\_items = list(all\_items)

    print(f"Всего уникальных товаров: {len(all\_items)}")

    item\_counts = {}

    for item in all\_items:

        count = sum(1 for transaction in transactions if item in transaction)

        support = count / len(transactions)

        if support >= min\_support:

            item\_counts[frozenset([item])] = support

    print(f"Частые 1-элементные наборы: {len(item\_counts)}")

    frequent\_itemsets = {1: item\_counts}

    k = 2

    while frequent\_itemsets[k-1]:

        print(f"Поиск {k}-элементных наборов...")

*Продолжение листинга A.1*

        candidates = generate\_candidates(list(frequent\_itemsets[k-1].keys()), k)

        candidate\_counts = {}

        for candidate in candidates:

            count = sum(1 for transaction in transactions

                       if candidate.issubset(set(transaction)))

            support = count / len(transactions)

            if support >= min\_support:

                candidate\_counts[candidate] = support

        if candidate\_counts:

            frequent\_itemsets[k] = candidate\_counts

            print(f"Найдено {k}-элементных наборов: {len(candidate\_counts)}")

        else:

            print(f"Нет частых {k}-элементных наборов")

            break

        k += 1

    all\_frequent = {}

    for level\_itemsets in frequent\_itemsets.values():

        all\_frequent.update(level\_itemsets)

    print(f"Всего найдено частых наборов: {len(all\_frequent)}")

    return all\_frequent

def generate\_candidates(frequent\_itemsets, k):

    candidates = []

    n = len(frequent\_itemsets)

*Продолжение листинга A.1*

    for i in range(n):

        for j in range(i+1, n):

            union\_set = frequent\_itemsets[i].union(frequent\_itemsets[j])

            if len(union\_set) == k:

                candidates.append(union\_set)

    return candidates

def custom\_generate\_rules(frequent\_itemsets, min\_confidence):

    print(f"\n=== ГЕНЕРАЦИЯ ПРАВИЛ (min\_confidence={min\_confidence}) ===")

    rules = []

    for itemset, support in frequent\_itemsets.items():

        if len(itemset) < 2:

            continue

        items = list(itemset)

        for i in range(1, len(items)):

            for antecedent in combinations(items, i):

                antecedent = frozenset(antecedent)

                consequent = itemset - antecedent

                antecedent\_support = frequent\_itemsets.get(antecedent, 0)

                if antecedent\_support > 0:

                    confidence = support / antecedent\_support

*Продолжение листинга A.1*

                    if confidence >= min\_confidence:

                        consequent\_support = frequent\_itemsets.get(consequent, 0)

                        lift = confidence / consequent\_support if consequent\_support > 0 else 0

                        leverage = support - (antecedent\_support \* consequent\_support)

                        conviction = (1 - consequent\_support) / (1 - confidence) if confidence < 1 else float('inf')

                        rules.append({

                            'antecedent': antecedent,

                            'consequent': consequent,

                            'support': support,

                            'confidence': confidence,

                            'lift': lift,

                            'leverage': leverage,

                            'conviction': conviction

                        })

    print(f"Сгенерировано правил: {len(rules)}")

    return rules

def analyze\_rules(rules):

    print("\n=== АНАЛИЗ МЕТРИК ===")

    if not rules:

        print("Нет правил для анализа")

        return

*Продолжение листинга A.1*

    rules\_sorted = sorted(rules, key=lambda x: x['confidence'], reverse=True)

    print("\n=== ТОП-5 ПРАВИЛ ПО УВЕРЕННОСТИ ===")

    for i, rule in enumerate(rules\_sorted[:5], 1):

        print(f"{i}. {set(rule['antecedent'])} → {set(rule['consequent'])}")

        print(f"   Support: {rule['support']:.3f}")

        print(f"   Confidence: {rule['confidence']:.3f}")

        print(f"   Lift: {rule['lift']:.3f}")

        print(f"   Leverage: {rule['leverage']:.3f}")

        print(f"   Conviction: {rule['conviction']:.3f}\n")

    print("=== ТОП-3 ПРАВИЛА ПО ЛИФТУ ===")

    rules\_by\_lift = sorted(rules, key=lambda x: x['lift'], reverse=True)

    for i, rule in enumerate(rules\_by\_lift[:3], 1):

        print(f"{i}. {set(rule['antecedent'])} → {set(rule['consequent'])}")

        print(f"   Lift: {rule['lift']:.3f}, Confidence: {rule['confidence']:.3f}\n")

def load\_data(file\_path, sample\_size=1000):

    try:

        print("Загрузка данных...")

        df = pd.read\_csv(file\_path, nrows=sample\_size)

        print(f"Размер данных: {df.shape}")

        print(f"Столбцы: {df.columns.tolist()}")

        print(f"Первые 5 строк:\n{df.head()}")

        transactions = []

        for \_, row in df.iterrows():

            transaction = []

            for col in df.columns[1:]:

*Продолжение листинга A.1*

                if pd.notna(row[col]) and row[col] != '':

                    transaction.append(row[col])

            transactions.append(transaction)

        print(f"Количество транзакций: {len(transactions)}")

        print(f"Пример транзакции: {transactions[0]}")

        return transactions

    except Exception as e:

        print(f"Ошибка при загрузке данных: {e}")

        import traceback

        traceback.print\_exc()

        return []

def main():

    try:

        file\_path = "groc.csv"

        transactions = load\_data(file\_path, sample\_size=1000)

        if not transactions:

            print("Не удалось загрузить данные")

            return

        print("\n" + "="\*50)

        print("АНАЛИЗ АССОЦИАТИВНЫХ ПРАВИЛ АЛГОРИТМОМ APRIORI")

        print("="\*50)

        frequent\_itemsets = custom\_apriori(transactions, min\_support=0.01)

        rules = custom\_generate\_rules(frequent\_itemsets, min\_confidence=0.5)

        analyze\_rules(rules)

            except Exception as e: