Отчёт по практической работе «Поиск частых наборов»

Корсун КЭ-401

Задание 1. Разработайте программу, которая выполняет поиск частых наборов объектов в заданном наборе данных с помощью алгоритма Apriori (или одной из его модификаций). Список результирующих наборов должен содержать как наборы, так и значение поддержки для каждого набора. Параметрами программы являются набор, порог поддержки и способ упорядочивания результирующего списка наборов (по убыванию значения поддержки или лексикографическое).

Для выполнения задания была разработана функция чтения транзакций из файла read_file. Реализация данной функции с необходимыми комментариями отображена в листинге 1.

Листинг 1 – Реализация функции read_file

```
def read_file(filename): # Функция чтения транзакций из файла
  with open(filename, 'r') as temp_f:
    col_count = [ len(l.split(" ")) for l in temp_f.readlines() ]
# Подсчет максимального количества объектов в транзакции
  column_names = [i for i in range(0, max(col_count))]
    df = pd.read_csv(filename, sep = ' ', header=None,
names=column_names)
    df_out = df.apply(lambda x: list(x.dropna().values),
axis=1).tolist() # Создание массива транзакций
   transactionEncoder = TransactionEncoder() # Трансформация
массива к нужному для алгоритма формату
  dataset = transactionEncoder.fit(df_out).transform(df_out)
  dataset = dataset.astype('int')
  df = pd.DataFrame(dataset, columns=transactionEncoder.columns_)
  return df # Возврат готового DataFrame для алгоритма Apriori
```

Работа алгоритма Apriori для набора данных представлена в листинге 2.

Листинг 2 – Работа алгоритма Apriori

```
df_retail = read_file('retail.dat') # Чтение файла
frequent_itemsets = apriori(df_retail, min_support=0.05) # Поиск
частых наборов с помощью алгоритма Apriori
sorted_frequent_itemsets = frequent_itemsets.sort_values(by =
'support', ascending=False) # Сортировка результатов по порогу
поддержки
sorted_frequent_itemsets
```

Задание 2. Проведите эксперименты на двух наборах из различных предметных областей. Наборы данных должны существенно отличаться друг от друга по количеству транзакций и/или типичной длине транзакции (количеству объектов). В экспериментах варьируйте пороговое значение поддержки (например: 1%, 3%, 5%, 10%, 15%).

Задание 3. Выполните визуализацию результатов экспериментов в виде следующих диаграмм:

- сравнение быстродействия на фиксированном наборе данных при изменяемом пороге поддержки;
- количество частых наборов объектов различной длины на фиксированном наборе данных при изменяемом пороге поддержки.

Для выполнения заданий были разработаны функции find_result для проведения экспериментов над наборами данных и нахождения результата этих экспериментов и visualize_results для визуализации результатов экспериментов. Реализации функции представлены в листингах 3 и 4.

Листинг 3 – Реализация функции find_result

```
def find_result(df): # Функция экспериментирования над наборами
данных
  min_supports = [] # Порог поддержки для каждого эксперимента
  items_count = [] # Количество наборов
  process_time = [] # Время работы
  for i in range(0,5):
    start = time.time()
    if len(df) <= 1000:
      min_support = i*0.05+0.05
      frequent_itemsets = apriori(df, min_support = min_support)
# Запуск алгоритма Apriori
      min_supports.append(min_support*100)
    else:
      min_support = i*0.1+0.5
      frequent_itemsets = apriori(df, min_support = min_support)
      min_supports.append(min_support*100)
    end = time.time()
    items_count.append(frequent_itemsets['itemsets'].count())
    process_time.append(end-start)
  result = pd.DataFrame(dict(min_supports = min_supports,
items_count = items_count, process_time = process_time))
```

```
  return result # Возврат результатов эксперимента над набором
данных
```

Листинг 4 – Реализация функции visualize_results

```
def visualize_results(result): # Функция визуализации
результатов с помощью библиотеки Plotly
  fig = px.bar(result, x = 'min_supports', y = 'process_time',
labels = {'min_supports': 'Порог поддержки, %', 'process_time':
'Время выполнения, с'})
  fig.show()
  fig = px.bar(result, x = 'min_supports', y = 'items_count',
labels = {'min_supports': 'Порог поддержки, %', 'items_count':
'Количество наборов'})
  fig.show()
```

В эксперименте над набором данных ритейла использовались пороговые значения поддержки от 5 до 25% с шагом 5%, для данных ДТП использовались значения от 50 до 90% с шагом 10%. Для визуализации результатов использовалась библиотека Plotly.

Результаты экспериментов изображены на рисунках 1 и 2.

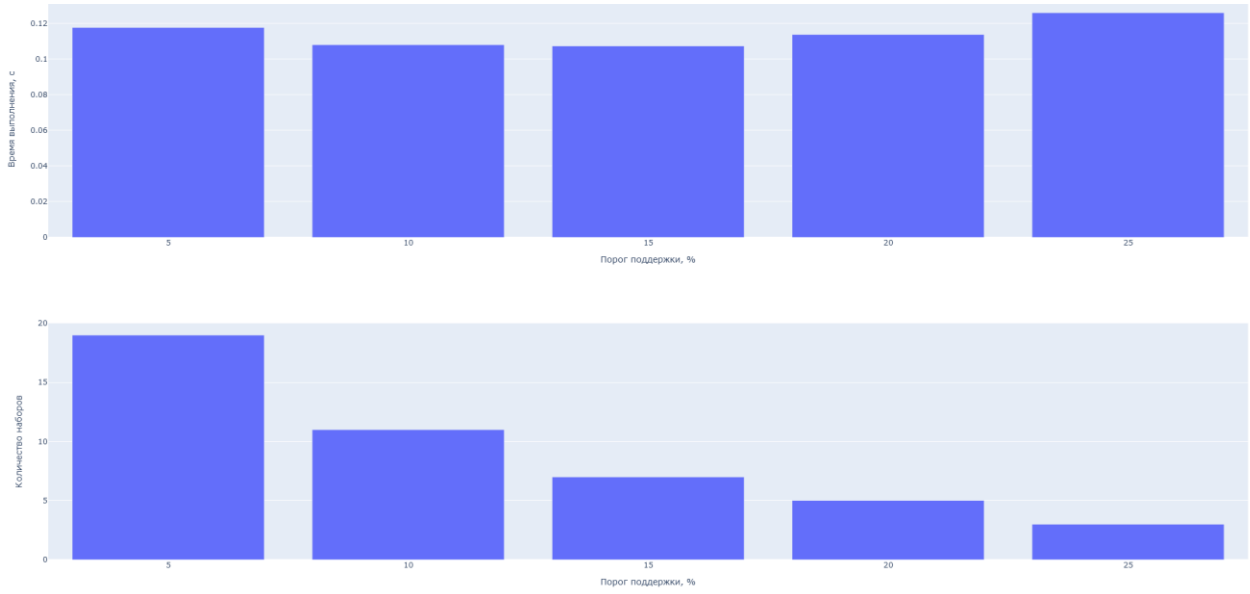

Рисунок 1 – Результаты эксперимента над набором данных ритейла

Рисунок 2 – Результаты эксперимента над набором данных ДТП

По графикам для первого набора данных можно сделать вывод, что даже для малого порога поддержки в 5% находится мало частых наборов, что связано с тем, что многие товары практически не повторяются в различных транзакциях и редко пересекаются с другими товарами несколько раз. Время работы программы практически не отличается для разных значений порога поддержки.

Исходя из графиков для набора данных о ДТП можно сделать вывод, что даже для больших значений порога поддержки (>50%) находится большое количество частых наборов, что говорит о частых пересечениях факторов ДТП и повторении некоторых из них в большом количестве инцидентов. Время работы программы существенно зависит от значения порога поддержки, чем ниже значение, тем дольше работает алгоритм.

Ссылка на репозиторий с исходными кодами: https://github.com/Vlater1/TAIP