АССОЦИАТИВНЫЕ ПРАВИЛА

Обучение на ассоциативных правилах (Associations rules learning — ARL) представляет из себя, с одной стороны, простой, с другой — довольно часто применимый в реальной жизни метод поиска взаимосвязей (ассоциаций) в датасетах, или, если точнее, айтемсетах (itemsests).

В общем виде ARL можно описать как «Кто купил x, также купил y». В основе лежит анализ транзакций, внутри каждой из которых лежит свой уникальный itemset из набора items. При помощи ARL алгоритмов находятся те самые «правила» совпадения items внутри одной транзакции, которые потом сортируются по их силе. Все, в общем, просто.

В 1992 году группа по консалтингу в области ритейла компании Teradata под руководством Томаса Блишока провела исследование 1.2 миллиона транзакций в 25 магазинах для ритейлера Osco Drug (Drug Store — формат разнокалиберных магазинов у дома). После анализа всех этих транзакций самым сильным правилом получилось «Между 17:00 и 19:00 чаще всего пиво и подгузники покупают вместе». К сожалению, такое правило показалось руководству Osco Drug настолько контринтуитивным, что ставить подгузники на полках рядом с пивом они не стали. Хотя объяснение паре пиво-подгузники вполне себе нашлось: когда оба члена молодой семьи возвращались с работы домой (как раз часам к 5 вечера), жены обычно отправляли мужей за подгузниками в ближайший магазин. И мужья, недолго думая, совмещали приятное с полезным — покупали подгузники по заданию жены и пиво для собственного вечернего времяпрепровождения.

Пусть у нас имеется некий датасет (или коллекция) *D*, такой, что $D = d_0 \dots d_j$, где *d* — уникальная транзакция-itemset (например, кассовый чек). Внутри каждой *d* представлен набор *items (i — item)*, причем в идеальном случае он представлен в бинарном виде:

d1 = [{Пиво: 1}, {Вода: 0}, {Кола: 1}, {...}],

d2 = [{Пиво: 0}, {Вода: 1}, {Кола: 1}, {...}].

Принято каждый itemset описывать через количество ненулевых значений (*k-itemset*), например, *[{Пиво: 1}, {Вода: 0}, {Кола: 1}]* является *2-itemset*.

Если изначально датасет в бинарном виде не представлен, можно при желании его преобразовать.

Таким образом, датасет представляет собой разреженную матрицу со значениями {1,0}. Это будет бинарный датасет. Существуют и другие виды записи — вертикальный датасет (показывает для каждого отдельного item вектор транзакций, где он присутствует) и транзакционный датасет (примерно, как в кассовом чеке).

Существует целый ряд базовых понятий в ARL:

Support (поддержка)

$$supp(X) = \frac{\{t \in T; X \in t\}}{|T|}$$

, где X — itemset, содержащий в себе i-items, а T — количество транзакций. Т.е. в общем виде это показатель «частотности» данного itemset во всех анализируемых транзакциях. Но это касается только X. Нам же интересен скорее вариант, когда у нас в одном itemset встречаются x_1 и x_2 (например). Пусть x_1 = *{Пиво}, а* x_2 = *{Подгузники}*, значит нам необходимо посчитать, во скольких транзакциях встречается эта пара.

$$supp(x_1 \cup x_2) = \frac{\sigma(x_1 \cup x_2)}{|T|}$$

где σ — количество транзакций, содержащих x_1 и x_2

Разберемся с этим понятием на небольшом примере.

Создадим датафрейм, где указаны номера транзакций, а также в бинарном виде представлено, что покупалось на каждой транзакции

```
import pandas as pd
```

```
data = {'Транзакция' : [1,2,3,4,5], 'Кола' : [1,0,1,1,0], 'Пиво' : [1,0,1,1,1], 'Подгузники' : [1,0,0,1,1] }
```

```
df = pd.DataFrame(data)
```

```
df
```

	Транзакция	Кола	Пиво	Подгузники
0	1	1	1	1
1	2	0	0	0
2	3	1	1	0
3	4	1	1	1
4	5	0	1	1

$$supp = \frac{\text{Транзакции с пивом и подгузниками}}{\text{Все транзакции}} = P(\text{Пиво} \cap \text{Подгузники})$$

$$supp = \frac{3}{5} = 60\%$$

Confidence (достоверность)

Следующее ключевое понятие — confidence. Это показатель того, как часто наше правило срабатывает для всего датасета.

$$conf(x_1 \cup x_2) = \frac{supp(x_1 \cup x_2)}{supp(x_1)}$$

Приведем пример: мы хотим посчитать confidence для правила «кто покупает пиво, тот покупает и подгузники».

Для этого сначала посчитаем, какой support у правила «покупает пиво», потом посчитаем support у правила «покупает пиво и подгузники», и просто поделим одно на другое. Т.е. мы посчитаем в скольких случаях (транзакциях) срабатывает правило «купил пиво» *supp*(X), «купил подгузники и пиво».

$$supp(X \cup Y)$$

Проверим на нашем примере:

$$conf(\text{Пиво} \cap \text{Подгузники}) = \frac{supp(\text{Пиво} \cap \text{Подгузники})}{supp(\text{Пиво})} = P(\text{Подгузники} \mid \text{Пиво})$$

$$conf = \frac{3}{4} = 75\%$$

Lift (поддержка)

Следующее понятие в нашем списке — lift. Грубо говоря, lift — это отношение «зависимости» items к их «независимости». Lift показывает, насколько items зависят друг от друга. Это очевидно из формулы:

$$lift(x_1 \cup x_2) = \frac{supp(x_1 \cup x_2)}{supp(x_1) \times supp(x_2)}$$

Например, мы хотим понять зависимость покупки пива и покупки подгузников. Для этого считаем support правила «купил пиво и подгузники» и делим его на произведение правил «купил пиво» и «купил подгузники». В случае, если *lift = 1*, мы говорим, что items независимы и правил совместной покупки тут нет. Если же *lift > 1*, то величина, на которую lift, собственно, больше этой самой единицы, и покажет нам «силу» правила. Чем больше единицы, тем лучше. Если *lift* < 1, то это покажет, что правило основания x_2 негативно влияет на правило x_1. По-другому lift можно определить как отношение confidence к expected confidence, т.е. отношение достоверности правила, когда оба (или больше) элемента покупаются вместе к достоверности правила, когда один из элементов покупался (неважно, со вторым или без).

$$lift = \frac{Confidence}{Expected\ confidence} = \frac{P(\text{Подгузники} \mid \text{Пиво})}{P(\text{Подгузники})}$$

$$lift = \frac{3/4}{3/5} = 1.25$$

Т.е. наше правило, что пиво покупают с подгузниками, на 25% мощнее правила, что подгузники просто покупают.

Conviction (убедительность)

В общем виде Conviction — это «частотность ошибок» нашего правила. Т.е., например, как часто покупали пиво без подгузников и наоборот.

$$conv(x_1 \cup x_2) = \frac{1 - supp(x_2)}{1 - conf(x_1 \cup x_2)}$$

$$conv(\text{Пиво} \cup \text{Подгузники}) = \frac{1 - supp(\text{Подгузники})}{1 - conf(\text{Пиво} \cup \text{Подгузники})} = \frac{1 - 0.6}{1 - 0.75} = 1.6$$

Чем результат по формуле выше 1, тем лучше. Например, если conviction покупки пива и подгузников вместе был бы равен 1.2, это значит, что правило «купил пиво и подгузники» было бы в 1.2 раза (на 20%) более верным, чем если бы совпадение этих items в одной транзакции было бы чисто случайным. Немного не интуитивное понятие, но оно и используется не так часто, как предыдущие три.

Существует ряд часто используемых алгоритмов, позволяющих находить правила в itemsets согласно перечисленным выше понятиям. Рассмотрим некоторые из них: Apriori и FP-Growth алгоритм

Apriori Алгоритм

Используемые понятия:

- Множество объектов (itemset):

$$X \subseteq I = \{x_1, x_2, \ldots, x_n\}$$

- Множество идентификаторов транзакций (tidset):

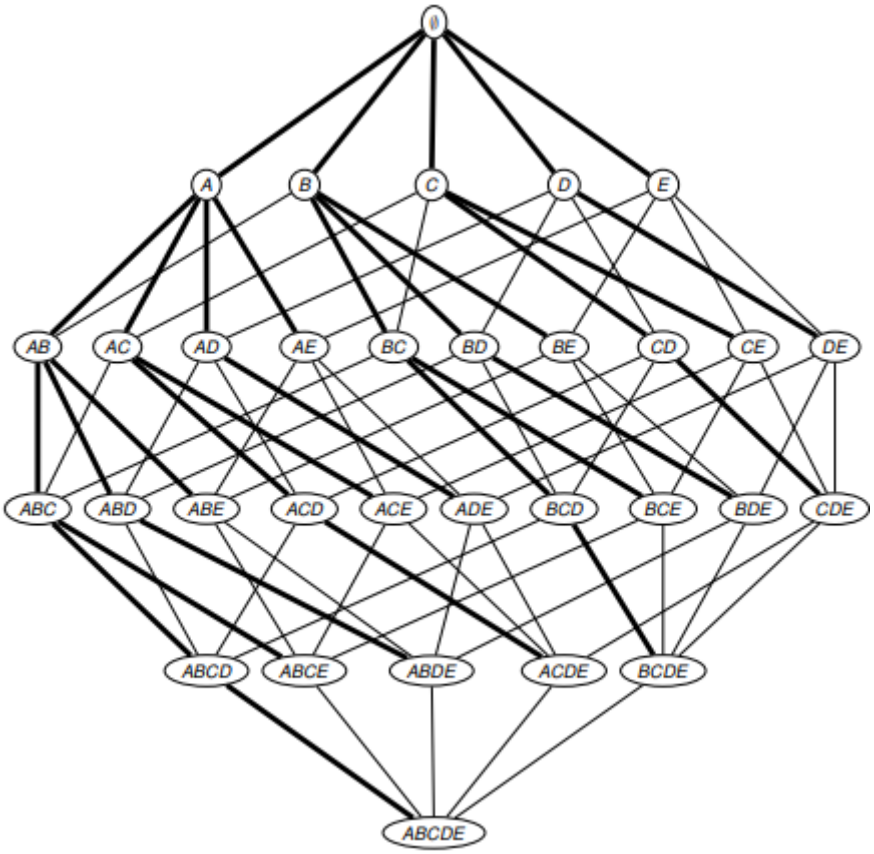
$$T = \{t_1, t_2, \ldots, t_m\}$$

- Множество транзакций (transactions):

$$\{(t, X): t \in T, X \in I\}$$

Введем дополнительно еще несколько понятий.

Будем рассматривать дерево префиксов (prefix tree), где 2 элемента X и Y соединены, если X является прямым подмножеством Y. Таким образом мы можем пронумеровать все подмножества множества I. Рисунок приведен ниже:



Рассмотрим алгоритм Apriori.

Apriori использует следующее утверждение: если $X \subseteq Y$, то $supp(X) \geq supp(Y)$. Отсюда следуют следующие 2 свойства:

- если Y встречается часто, то любое подмножество

$$X : X \subseteq Y$$

  так же встречается часто

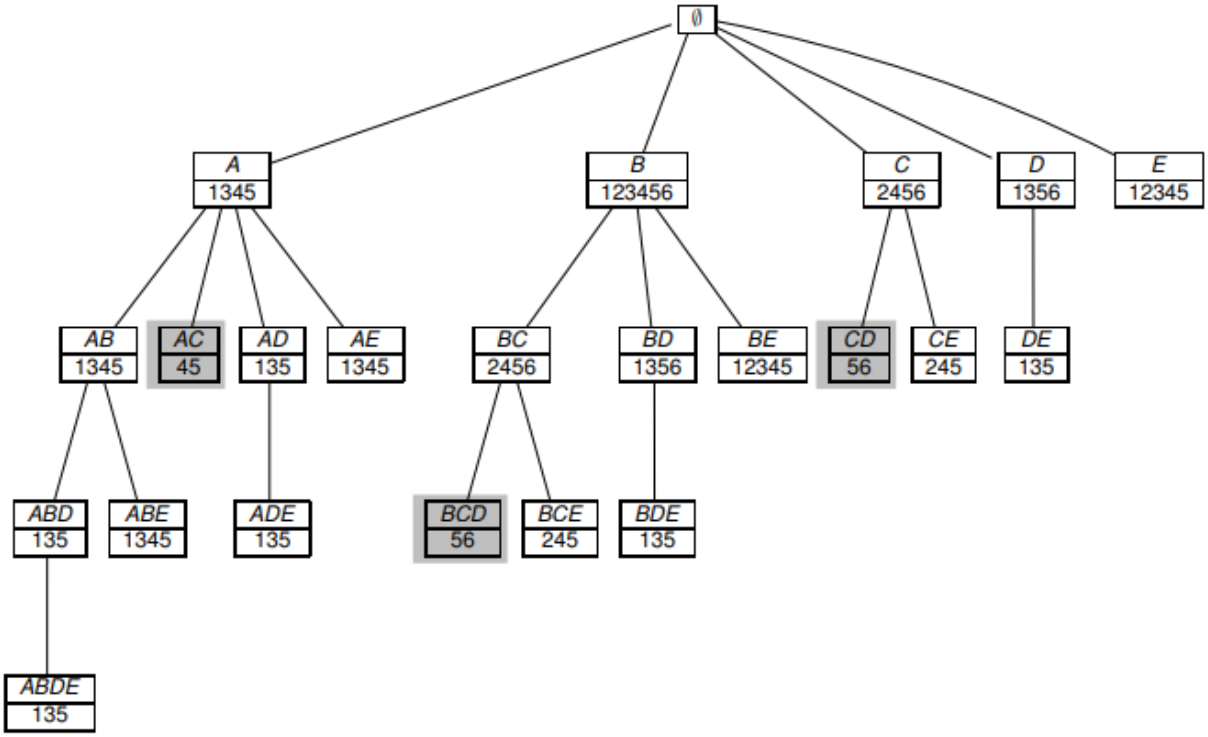
- если X встречается редко, то любое супермножество

$$Y : Y \supseteq X$$

  так же встречается редко

Apriori алгоритм по-уровнево проходит по префиксному дереву и рассчитывает частоту встречаемости подмножеств X в D. Таким образом, в соответствии с алгоритмом:

- исключаются редкие подмножества и все их супермножества
- рассчитывается *supp(X)* для каждого подходящего кандидата X размера k на уровне k



Таким образом, Apriori сначала ищет все единичные (содержащие 1 элемент) itemsets, удовлетворяющие заданному пользователем *supp*, затем составляет из них пары по принципу иерархической монотонности, т.е. если x_1 встречается часто и x_2 встречается часто, то и $[x_1, x_2]$ встречается часто.

Явным минусом такого подхода является то, что необходимо «просканировать» весь датасет, посчитать все *supp* на всех уровнях. Это также может сильно нагрузить оперативную память на больших датасетах, хотя алгоритм в плане скорости все равно намного эффективнее брутфорса.

FP-Growth Algorithm

FP-Growth (Frequent Pattern Growth) более новый алгоритм, впервые он описан в 2000 году.

FP-Growth предлагает радикальную вещь — отказаться от генерации кандидатов (генерация кандидатов лежит в основе Apriori). Теоретически, такой подход позволит еще больше увеличить скорость алгоритма и использовать еще меньше памяти.

Это достигается за счет хранения в памяти префиксного дерева (trie) не из комбинаций кандидатов, а из самих транзакций.

При этом FP-Growth генерирует таблицу заголовков для каждого item, чей *supp* выше заданного пользователем. Эта таблица заголовков хранит связанный список всех однотипных узлов префиксного дерева.

В качестве примера возьмем данные по покупкам лекарств в аптеке. Датасет небольшой, содержит всего 30 транзакций (чеков).

```
df.head()
```

	0	1	2	3	4
0	Афобазол	Пустырник	Элькар	Валериана	Гематоген
1	Пустырник	Анальгин	Панангин	Элькар	Супрастин
2	Анальгин	Валериана	Супрастин	Пустырник	Элькар
3	Аскорбинка	Афобазол	Рибоксин	Пустырник	Панангин
4	Рибоксин	Панангин	Гематоген	Витамины	NaN

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       30 non-null     object
 1   1       30 non-null     object
 2   2       27 non-null     object
 3   3       14 non-null     object
 4   4       8 non-null      object
dtypes: object(5)
memory usage: 1.3+ KB
```

Визуализируем данные для лучшего представления. Посчитаем количество покупок конкретных лекарств во всех чеках. Также выполним нормализацию, чтобы вывести относительную частоту встречаемости.

```
df.stack().value_counts()
```

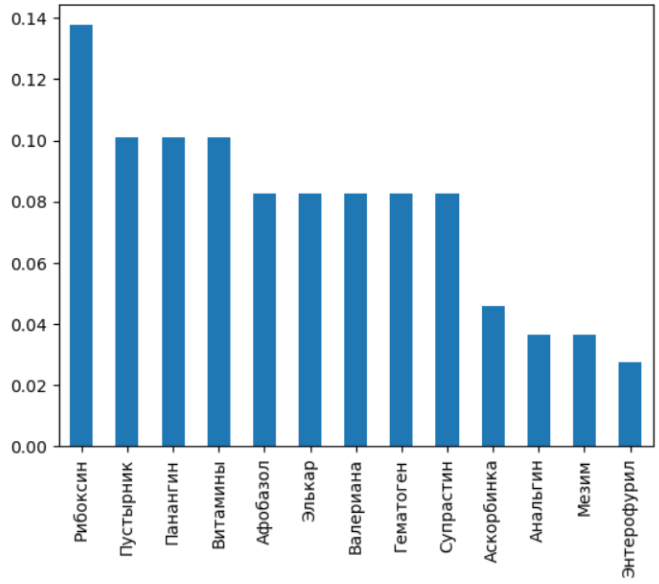
```
Рибоксин       15
Пустырник      11
Панангин       11
Витамины       11
Афобазол        9
Элькар          9
Валериана       9
Гематоген       9
Супрастин       9
Аскорбинка      5
Анальгин        4
Мезим           4
Энтерофурил     3
dtype: int64
```

```
df.stack().value_counts(normalize=True)
```

```
Рибоксин       0.137615
Пустырник      0.100917
Панангин       0.100917
Витамины       0.100917
Афобазол       0.082569
Элькар         0.082569
Валериана      0.082569
Гематоген      0.082569
Супрастин      0.082569
Аскорбинка     0.045872
Анальгин       0.036697
Мезим          0.036697
Энтерофурил    0.027523
dtype: float64
```

```
df.stack().value_counts(normalize=True).plot(kind='bar') # Относительная частота
```

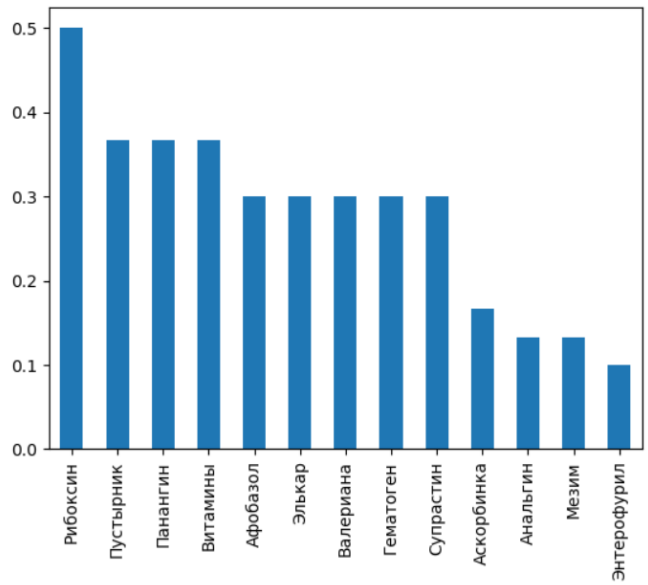
```
<AxesSubplot:>
```



Выведем также и фактическую частоту. То есть, глядя на полученную гистограмму можно сказать, что в 50% корзин есть Рибоксин, в 37% Пустырник и т.д.

```
df.stack().value_counts().apply(lambda item: item / df.shape[0]).plot(kind='bar') # Фактическая частота
```

```
<AxesSubplot:>
```



В большинстве библиотек, которые реализуют алгоритмы поиска ассоциативных правил в качестве входных значений необходимо подавать список транзакций, то есть список списков, поэтому нам необходимо преобразовать наш датасет в такой формат. При этом необходимо

контролировать пустые значения в данных (NaN), их добавлять в список не надо.

```
transactions = []
for i in range(df.shape[0]):
    row = df.iloc[i].dropna().tolist()
    transactions.append(row)
```

```
transactions[0][0]
```

```
'Афобазол'
```

```
transactions[0]
```

```
['Афобазол', 'Пустырник', 'Элькар', 'Валериана', 'Гематоген']
```

Также выведем первый элемент и первый список, и сравним с нашим датасетом, чтобы удостоверится, что преобразование прошло удачно.

Далее воспользуемся первой реализацией алгоритма Aprori из библиотеки apriori_python.

```
from apriori_python import apriori
```

```
t=[]
start=time.perf_counter()

t1, rules = apriori(transactions,minSup = 0.2, minConf = 0.46)
time1=(time.perf_counter()-start)
t.append(time1)
```

```
rules
```

```
[[{'Рибоксин'}, {'Афобазол'}, 0.4666666666666667],
 [{'Рибоксин'}, {'Панангин'}, 0.4666666666666667],
 [{'Пустырник'}, {'Афобазол'}, 0.5454545454545454],
 [{'Пустырник'}, {'Валериана'}, 0.6363636363636364],
 [{'Витамины'}, {'Гематоген'}, 0.6363636363636364],
 [{'Панангин'}, {'Рибоксин'}, 0.6363636363636364],
 [{'Афобазол'}, {'Пустырник'}, 0.6666666666666666],
 [{'Супрастин'}, {'Элькар'}, 0.6666666666666666],
 [{'Элькар'}, {'Супрастин'}, 0.6666666666666666],
 [{'Валериана'}, {'Пустырник'}, 0.7777777777777778],
 [{'Афобазол'}, {'Рибоксин'}, 0.7777777777777778],
 [{'Гематоген'}, {'Витамины'}, 0.7777777777777778]]
```

В качестве аргументов принимается наш список списков с покупками, и настраиваются два параметра minSup и minConf. Для каждых данных они будут **свои**! Необходимо их определить так, чтобы количество правил было не слишком малым и не слишком большим. Если эти параметры сделать слишком малыми, то может существенно увеличится время выполнения алгоритма и найдется слишком много ненадежных правил. А если параметры будут слишком большие, то правил может вообще не найтись. Поэтому при первой прогонке рекомендуется сделать параметры поменьше, чтобы вывести как можно больше правил, а затем постепенно их уменьшать, пока количество правил не станет разумным.

minSup – это минимальная поддержка. Значение поддержки меняется от 0 (когда условие и следствие не встречаются вместе ни в одной транзакции) до 1 (когда условие и следствие во всех транзакциях появляются совместно).

В общем случае поддержка является мерой надежности, с которой ассоциативное правило выражает ассоциативную связь между условием и следствием. Если поддержка S > 0,8, то связь сильная, а само правило заслуживает доверия. В случае, когда 0,5 < S < 0,8, ассоциативная связь средняя, а правило следует использовать с осторожностью. При S < 0,5 связь слабая, а ассоциативное правило является сомнительным.

minConf – минимальная достоверность. Это показатель, характеризующий уверенность в том, что ассоциация A → B является ассоциативным правилом. То есть предположение о том, что появление события A влечёт за собой появление события B, является достаточно достоверным.

Воспользуемся другой библиотекой для применения алгоритма Apriori.

```
from apyori import apriori
```

```
start=time.perf_counter()
rules = apriori(transactions=transactions,
                min_support=0.2, # как часто элемент(ы) встречается в наборе данных
                min_confidence=0.46, # как часто наше правило будет срабатывать
                min_lift=1.0001) # насколько лучше по сравнению с чистой случайностью

results = list(rules)
time2=(time.perf_counter()-start)
t.append(time2)
```

Здесь принимаются те же аргументы, что и в прошлой библиотеке, но необходимо указать минимальный лифт чуть больше 1, чтобы исключить вывод независимых правил.

```
results
```

```
[RelationRecord(items=frozenset({'Афобазол', 'Пустырник'}), support=0.2, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Афобазо
л'}), items_add=frozenset({'Пустырник'}), confidence=0.6666666666666667, lift=1.8181818181818186), OrderedStatistic(items_base=frozenset({'Пусты
рник'}), items_add=frozenset({'Афобазол'}), confidence=0.5454545454545455, lift=1.8181818181818186)]),
 RelationRecord(items=frozenset({'Рибоксин', 'Афобазол'}), support=0.23333333333333334, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'Афобазол'}), items_add=frozenset({'Рибоксин'}), confidence=0.7777777777777778, lift=1.5555555555555556), OrderedStatistic(items_base=frozens
et({'Рибоксин'}), items_add=frozenset({'Афобазол'}), confidence=0.4666666666666667, lift=1.5555555555555556)]),
 RelationRecord(items=frozenset({'Валериана', 'Пустырник'}), support=0.23333333333333334, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'Валериана'}), items_add=frozenset({'Пустырник'}), confidence=0.7777777777777778, lift=2.1212121212121215), OrderedStatistic(items_base=fro
zenset({'Пустырник'}), items_add=frozenset({'Валериана'}), confidence=0.6363636363636365, lift=2.1212121212121215)]),
 RelationRecord(items=frozenset({'Витамины', 'Гематоген'}), support=0.23333333333333334, ordered_statistics=[OrderedStatistic(items_base=frozens
et({'Витамины'}), items_add=frozenset({'Гематоген'}), confidence=0.6363636363636365, lift=2.1212121212121215), OrderedStatistic(items_base=froze
nset({'Гематоген'}), items_add=frozenset({'Витамины'}), confidence=0.7777777777777778, lift=2.1212121212121215)]),
 RelationRecord(items=frozenset({'Панангин', 'Рибоксин'}), support=0.23333333333333334, ordered_statistics=[OrderedStatistic(items_base=frozense
t({'Панангин'}), items_add=frozenset({'Рибоксин'}), confidence=0.6363636363636365, lift=1.272727272727273), OrderedStatistic(items_base=frozense
t({'Рибоксин'}), items_add=frozenset({'Панангин'}), confidence=0.4666666666666667, lift=1.272727272727273)]),
 RelationRecord(items=frozenset({'Супрастин', 'Элькар'}), support=0.2, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Супрастин'}),
items_add=frozenset({'Элькар'}), confidence=0.6666666666666667, lift=2.2222222222222228), OrderedStatistic(items_base=frozenset({'Элькар'}), ite
ms_add=frozenset({'Супрастин'}), confidence=0.6666666666666667, lift=2.2222222222222228)])]
```

Для более удобной интерпретации результатов отформатируем вывод:

```
for result in results:
       for subset in result[2]:
           print(subset[0],subset[1])
           print("Support: {0}; Confidence: {1}; Lift: {2};".format(result[1],subset[2],subset[3]))
           print()
```

```
frozenset({'Афобазол'}) frozenset({'Пустырник'})
Support: 0.2; Confidence: 0.6666666666666667; Lift: 1.8181818181818186;

frozenset({'Пустырник'}) frozenset({'Афобазол'})
Support: 0.2; Confidence: 0.5454545454545455; Lift: 1.8181818181818186;

frozenset({'Афобазол'}) frozenset({'Рибоксин'})
Support: 0.23333333333333334; Confidence: 0.7777777777777778; Lift: 1.5555555555555556;

frozenset({'Рибоксин'}) frozenset({'Афобазол'})
Support: 0.23333333333333334; Confidence: 0.4666666666666667; Lift: 1.5555555555555556;

frozenset({'Валериана'}) frozenset({'Пустырник'})
Support: 0.23333333333333334; Confidence: 0.7777777777777778; Lift: 2.1212121212121215;

frozenset({'Пустырник'}) frozenset({'Валериана'})
Support: 0.23333333333333334; Confidence: 0.6363636363636365; Lift: 2.1212121212121215;

frozenset({'Витамины'}) frozenset({'Гематоген'})
Support: 0.23333333333333334; Confidence: 0.6363636363636365; Lift: 2.1212121212121215;

frozenset({'Гематоген'}) frozenset({'Витамины'})
Support: 0.23333333333333334; Confidence: 0.7777777777777778; Lift: 2.1212121212121215;

frozenset({'Панангин'}) frozenset({'Рибоксин'})
Support: 0.23333333333333334; Confidence: 0.6363636363636365; Lift: 1.272727272727273;

frozenset({'Рибоксин'}) frozenset({'Панангин'})
Support: 0.23333333333333334; Confidence: 0.4666666666666667; Lift: 1.272727272727273;

frozenset({'Супрастин'}) frozenset({'Элькар'})
Support: 0.2; Confidence: 0.6666666666666667; Lift: 2.2222222222222228;

frozenset({'Элькар'}) frozenset({'Супрастин'})
Support: 0.2; Confidence: 0.6666666666666667; Lift: 2.2222222222222228;
```

Воспользуемся третьей реализацией с помощью библиотеки efficient_apriori. Аналогично зададим параметры минимальной поддержки и минимальной достоверности.

```
from efficient_apriori import apriori
```

```
start=time.perf_counter()
itemsets, rules = apriori(transactions,min_support = 0.2, min_confidence = 0.46)
time3=(time.perf_counter()-start)
t.append(time3)
```

```
for i in range(len(rules)):
    print(rules[i])
```

```
{Пустырник} -> {Афобазол} (conf: 0.545, supp: 0.200, lift: 1.818, conv: 1.540)
{Афобазол} -> {Пустырник} (conf: 0.667, supp: 0.200, lift: 1.818, conv: 1.900)
{Рибоксин} -> {Афобазол} (conf: 0.467, supp: 0.233, lift: 1.556, conv: 1.312)
{Афобазол} -> {Рибоксин} (conf: 0.778, supp: 0.233, lift: 1.556, conv: 2.250)
{Пустырник} -> {Валериана} (conf: 0.636, supp: 0.233, lift: 2.121, conv: 1.925)
{Валериана} -> {Пустырник} (conf: 0.778, supp: 0.233, lift: 2.121, conv: 2.850)
{Гематоген} -> {Витамины} (conf: 0.778, supp: 0.233, lift: 2.121, conv: 2.850)
{Витамины} -> {Гематоген} (conf: 0.636, supp: 0.233, lift: 2.121, conv: 1.925)
{Рибоксин} -> {Панангин} (conf: 0.467, supp: 0.233, lift: 1.273, conv: 1.187)
{Панангин} -> {Рибоксин} (conf: 0.636, supp: 0.233, lift: 1.273, conv: 1.375)
{Элькар} -> {Супрастин} (conf: 0.667, supp: 0.200, lift: 2.222, conv: 2.100)
{Супрастин} -> {Элькар} (conf: 0.667, supp: 0.200, lift: 2.222, conv: 2.100)
```

Применим алгоритмом FP-Growth из библиотеки fpgrowth_py. Аналогично прошлым алгоритмам, необходимо задать минимальные поддержку и достоверность.

```
from fpgrowth_py import fpgrowth
```

```
start=time.perf_counter()
itemsets, rules = fpgrowth(transactions,minSupRatio = 0.2, minConf = 0.46)

time4=(time.perf_counter()-start)
t.append(time4)
```

```
for i in range(len(rules)):
    print(rules[i])
```

```
[{'Пустырник'}, {'Афобазол'}, 0.5454545454545454]
[{'Афобазол'}, {'Пустырник'}, 0.6666666666666666]
[{'Рибоксин'}, {'Афобазол'}, 0.4666666666666667]
[{'Афобазол'}, {'Рибоксин'}, 0.7777777777777778]
[{'Пустырник'}, {'Валериана'}, 0.6363636363636364]
[{'Валериана'}, {'Пустырник'}, 0.7777777777777778]
[{'Витамины'}, {'Гематоген'}, 0.6363636363636364]
[{'Гематоген'}, {'Витамины'}, 0.7777777777777778]
[{'Панангин'}, {'Рибоксин'}, 0.6363636363636364]
[{'Рибоксин'}, {'Панангин'}, 0.4666666666666667]
```

Сравним время выполнения наших алгоритмов:

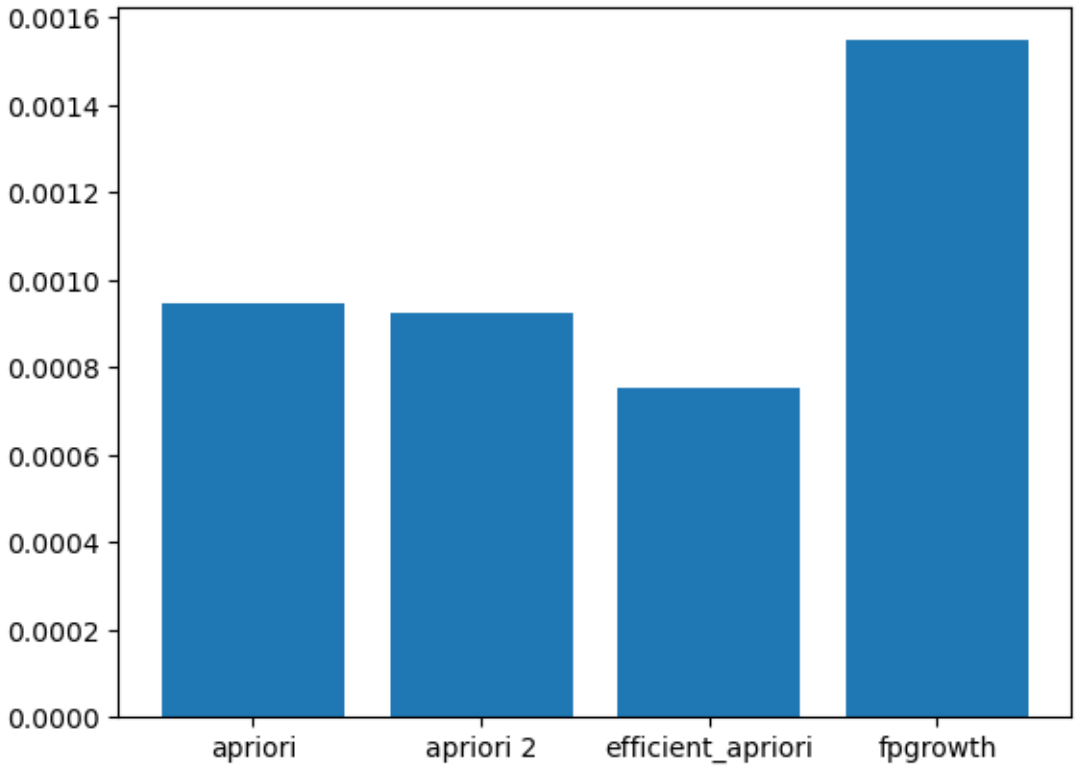
```
print('Время выполнения apriori: ', t[0], '\n')
print('Время выполнения apriori 2: ', t[1], '\n')
print('Время выполнения efficient_apriori: ', t[2], '\n')
print('Время выполнения fpgrowth: ', t[3], '\n')
plt.bar(['apriori','apriori 2','efficient_apriori','fpgrowth'],t)
plt.show()
```

```
Время выполнения apriori:  0.0009458000000002187

Время выполнения apriori 2:  0.0009239999999994808

Время выполнения efficient_apriori:  0.000753600000000354

Время выполнения fpgrowth:  0.0015472999999999474
```



Как видим на небольших данных время выполнения почти не различается, а также преимущества алгоритма FPGrowth нивелируется достаточно хорошей оптимизации алгоритма Apriori в используемых библиотеках.

Практическая работа

1. Загрузить данные Market_Basket_Optimisation.csv.
2. Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для 20 наиболее популярных товаров).
3. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori).
4. Применить алгоритм FP-Growth из библиотеки fpgrowth_py.
5. Сравнить время выполнения всех алгоритмов и построить гистограмму.
6. Загрузить данные data.csv.
7. Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для 20 наиболее популярных товаров).
8. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori).
9. Применить алгоритм FP-Growth из библиотеки fpgrowth_py.
10. Сравнить время выполнения всех алгоритмов и построить гистограмму.
11. Сформулировать выводы и сделать отчет.