Data Mining

Name: Gurneet Chhabra

Date: 10-07-2021

Project Report

Abstract:

The problem is to implement two mining algorithms, Apriori and FP-Growth on the e Adult Census Dataset present in UCI ML repository. We also try to implement a variation of basic Apriori algorithm in order to improve its efficiency. We also generate the association rules in addition to the generation of frequent itemsets matching the confidence threshold mentioned.

Keywords

Pattern Mining, Apriori, FPGrowth, Association Rules, Naive Bayes, Adult Census Dataset

1.Introduction

Pattern mining is the process of extracting patterns from the datasets with the help of algorithms developed solely for this purpose. Frequent data mining involves extracting those patterns only which are frequent in the dataset that is which are above some threshold value.

The decision to when to consider a repeating pattern as frequent is based on a metric called "support". Support can be looked as a frequency of occurrence of a particular itemset in the data. Support is a relative quantity. Once you mine the frequent itemsets, you can generate rules based on minimum confidence from these frequent itemsets. Using these rules, which are called ass association rules will help us to derive important insights from the dataset.

I applied the pattern mining approach discussed to Adult Census data from UCI ML repository.

Following tasks were performed:

(1) Implemented Apriori algorithm and applied it on the dataset. Applied the algorithm on whole dataset that is 32561 rows.

(2) Implemented FPGrowth algorithm and applied it on the Adult Census dataset. Applied the algorithm on whole dataset that is 32561 rows.

(3) Implemented an improvement of the Apriori algorithm and apply it on the dataset to improve the execution time of the classic Apriori approach. Applied the algorithm on whole dataset that is 32561 rows.

(4) Generated Association rules by passing an input minimum confidence level

(5) Comparative study of the above algorithms.

2.Overview of Dataset:

Adult Census dataset is taken from UCI ML repository.

The dataset consists of 32561 rows and 15 attributes. The 15 attributes of the dataset along with the values they take are as follows:

1) age: continuous.
2) workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3) fnlwgt: continuous.
4) education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th,
5) 10th, Doctorate, 5th-6th, Preschool.
6) education-num: continuous.
7) marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
8) occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
9) relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
10) race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
11) sex: Female, Male.
12) capital-gain: continuous.
13) capital-loss: continuous.
14) hours-per-week: continuous.
15) native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
16) 16. class: >50K, <=50K

3.Exploratory Data Analysis

I have performed the EDA on Jupyter notebook and wrote the algorithm in Spyder IDE. Further created the plots in Jupyter notebook.

Preview of the dataset:

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	Salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Fig 1. Displays first five rows of the data.

The basic statistics of the numerical column in the data is described below:

```
In [5]: adults.describe().T      #to get basic statistics of the data
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
age	32561.0	38.581647	13.640433	17.0	28.0	37.0	48.0	90.0
fnlwgt	32561.0	189778.366512	105549.977697	12285.0	117827.0	178356.0	237051.0	1484705.0
education-num	32561.0	10.080679	2.572720	1.0	9.0	10.0	12.0	16.0
capital-gain	32561.0	1077.648844	7385.292085	0.0	0.0	0.0	0.0	99999.0
capital-loss	32561.0	87.303830	402.960219	0.0	0.0	0.0	0.0	4356.0
hours-per-week	32561.0	40.437456	12.347429	1.0	40.0	40.0	45.0	99.0

Fig. Statistics of the data given.

The dataset given has no null values. So we can say that there are no missing values in the dataset given.

But looking at the distribution of data, we can say that it is not a balanced dataset. It is biased towards one target class. This is evident from the categorical plot shown below:

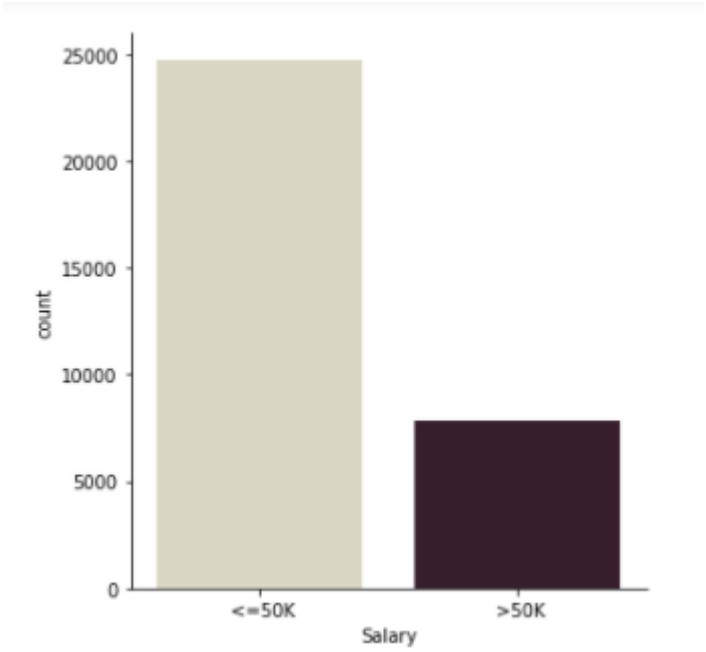


Fig: Value counts of each category

Existence of imbalance in the dataset.

- Percentage of rows with '>50K' : **24.1%** and
- Percentage of rows with label '<=50K' : **75.9%.**

The dataset also contains a mixture of continuous and discrete values.

The plots of "age", fnlwgt" , "education-num", "hours-per-week" are shown below.

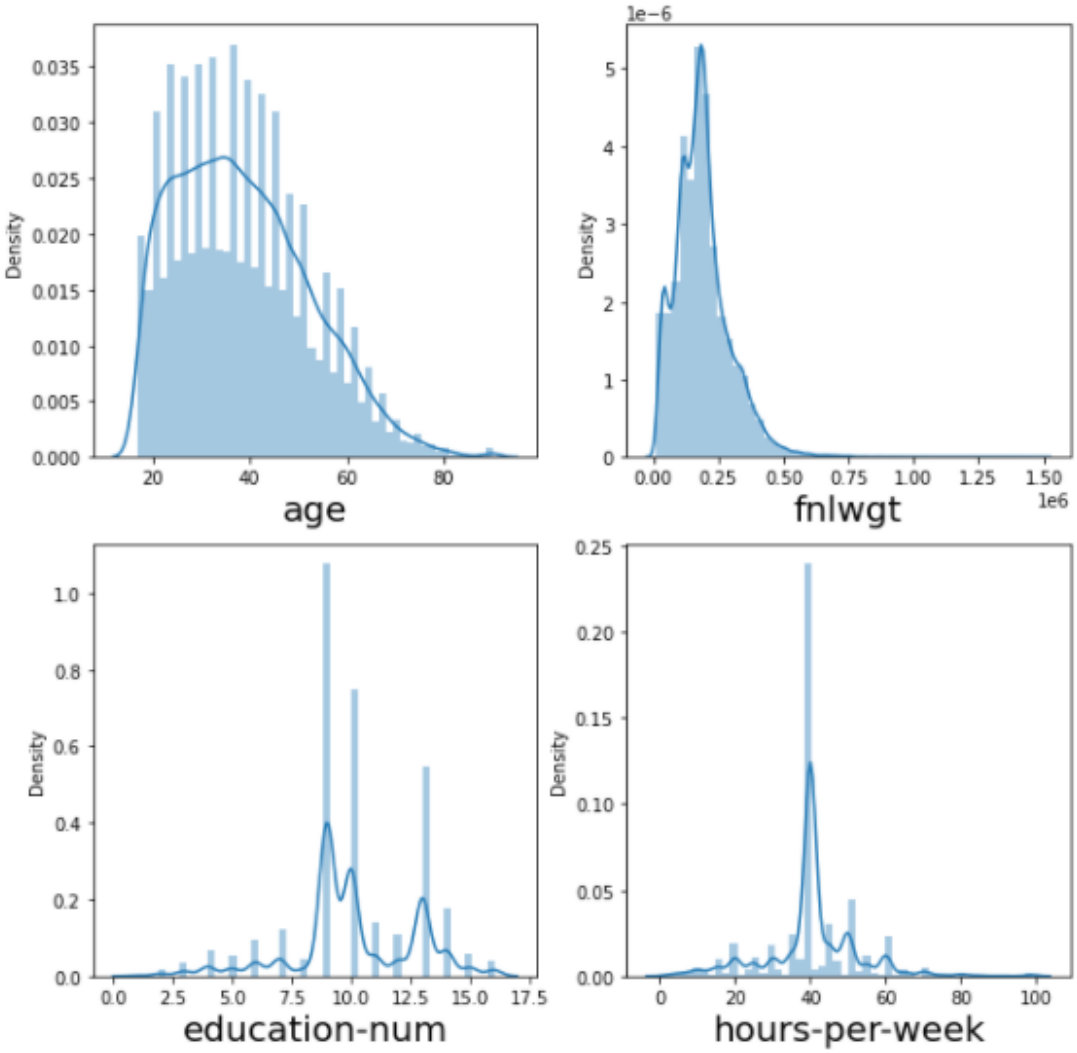


Fig. Distribution plots of various attributes.

Observations

i. The distribution of values of **age** are right skewed.
ii. The distribution of values of **fnlwgt** that is of final weight are also right skewed.
iii. We get a multimodel distributon for **education-num.**

4.Preprocessing the Data:

Filtering out the attributes:

The first task that I would do in pre-processing is remove the 'fnlwgt' attribute from the data.

The reason for removing the "fnlwgt" attribute is that it doesn't provide any information that would help us to determine if the person's income is greater than 50k. It only represents the weight that the row should have, which is defined by the censor takers. While we could have that weight in the dataset but it would complicate things, so I have decided to remove this attribute. Additionally, it is a numerical variable, so it would not make any sensne and we can just drop this attribute.

Similarly, other numerical attributes like "age", "hours-per-week" and "education-num" can also be dropped. Since these attributes are just numerical, we can drop them.

Handling Missing Values

```
adults.isnull().sum() #Checking for null values
```

```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
Salary            0
dtype: int64
```

It seems that there are no missing values in our data.

It seems that there are no missing values in the dataset.

Further I have also added prefix in front of the data values, that will enable to make distiction between the column value while generating frequent itemsets. The prefix is name of the attribute to which the data value belongs.

For example: values of "capital-gain" becomes "capital-gain : 0", “white ”Value in race attribute becomes “Race: White”, and similarly modified allt eh values in the dataset. This would help us to easiy identify to which attribute the value belongs.

5.Implementation:

5.1 Apriori:

Apriori algorithm is used in data mining to generate frequent itemsets. It is one of the most basic Frequent Pattern Mining algorithm. Implemantation of Apriori is done by following the below algorithm which is mentioned in the textbook:

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧ ... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Fig. Apriori Algorithm

The key idea in Apriori is

- All subsets of a frequent itemset must be frequent Similarly, for any infrequent itemset
- all its supersets must be infrequent too.

The frequent itemsets generated by running the Apriori algorithm on 32561 rows resulted in generation of 63 frequent itemsets when minimum support is 0.5.

The itemsets generated are as follows:

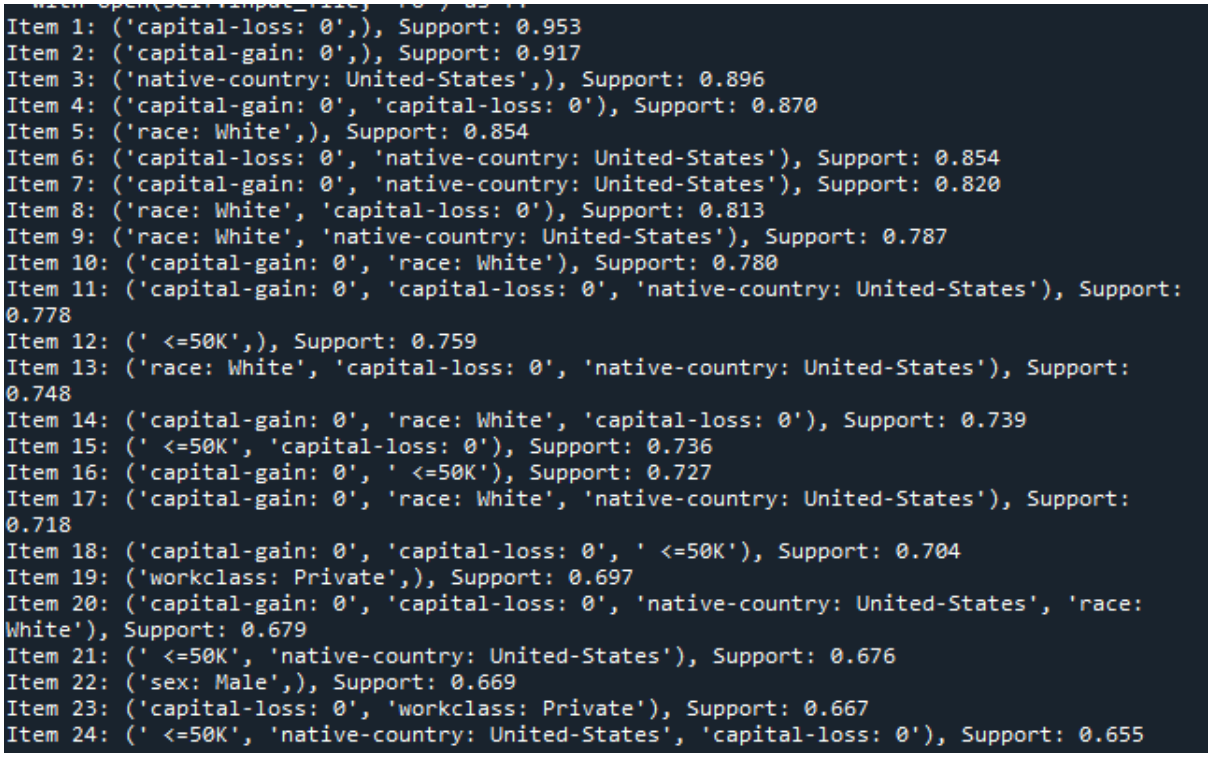


Fig: Number of frequent itemsets generated with min_support = 0.5

```
Item 24: (' <=50K', 'native-country: United-States', 'capital-loss: 0'), Support: 0.655
Item 25: ('capital-gain: 0', ' <=50K', 'native-country: United-States'), Support: 0.647
Item 26: ('capital-gain: 0', 'workclass: Private'), Support: 0.644
Item 27: ('race: White', ' <=50K'), Support: 0.636
Item 28: ('sex: Male', 'capital-loss: 0'), Support: 0.634
Item 29: ('capital-gain: 0', 'capital-loss: 0', 'native-country: United-States', ' <=50K'),
Support: 0.626
Item 30: ('workclass: Private', 'native-country: United-States'), Support: 0.618
Item 31: ('race: White', ' <=50K', 'capital-loss: 0'), Support: 0.616
Item 32: ('capital-gain: 0', 'capital-loss: 0', 'workclass: Private'), Support: 0.614
Item 33: ('capital-gain: 0', 'race: White', ' <=50K'), Support: 0.608
Item 34: ('capital-gain: 0', 'sex: Male'), Support: 0.605
Item 35: ('sex: Male', 'native-country: United-States'), Support: 0.599
Item 36: ('race: White', 'workclass: Private'), Support: 0.596
Item 37: ('workclass: Private', 'capital-loss: 0', 'native-country: United-States'),
Support: 0.591
Item 38: ('sex: Male', 'race: White'), Support: 0.589
Item 39: ('capital-gain: 0', 'capital-loss: 0', 'race: White', ' <=50K'), Support: 0.588
Item 40: ('race: White', ' <=50K', 'native-country: United-States'), Support: 0.581
Item 41: ('capital-gain: 0', 'workclass: Private', 'native-country: United-States'),
Support: 0.570
Item 42: ('sex: Male', 'capital-gain: 0', 'capital-loss: 0'), Support: 0.570
Item 43: ('race: White', 'capital-loss: 0', 'workclass: Private'), Support: 0.569
Item 44: ('sex: Male', 'capital-loss: 0', 'native-country: United-States'), Support: 0.567
Item 45: ('capital-loss: 0', 'native-country: United-States', 'race: White', ' <=50K'),
Support: 0.562
Item 46: ('sex: Male', 'race: White', 'capital-loss: 0'), Support: 0.557
Item 47: ('capital-gain: 0', 'native-country: United-States', 'race: White', ' <=50K'),
Support: 0.556
Item 48: ('capital-gain: 0', 'race: White', 'workclass: Private'), Support: 0.549
Item 49: (' <=50K', 'workclass: Private'), Support: 0.545
```

Fig: Number of frequent itemsets generated with min_support = 0.5

```
Item 49: (' <=50K', 'workclass: Private'), Support: 0.545
Item 50: ('workclass: Private', 'race: White', 'native-country: United-States'), Support:
0.544
Item 51: ('capital-gain: 0', 'capital-loss: 0', 'native-country: United-States',
'workclass: Private'), Support: 0.542
Item 52: ('sex: Male', 'race: White', 'native-country: United-States'), Support: 0.542
Item 53: ('capital-gain: 0', 'sex: Male', 'native-country: United-States'), Support: 0.540
Item 54: (' <=50K', 'capital-gain: 0', 'capital-loss: 0', 'native-country: United-States',
'race: White'), Support: 0.537
Item 55: ('sex: Male', 'capital-gain: 0', 'race: White'), Support: 0.531
Item 56: (' <=50K', 'workclass: Private', 'capital-loss: 0'), Support: 0.529
Item 57: ('capital-gain: 0', ' <=50K', 'workclass: Private'), Support: 0.523
Item 58: ('capital-gain: 0', 'capital-loss: 0', 'workclass: Private', 'race: White'),
Support: 0.522
Item 59: ('capital-loss: 0', 'workclass: Private', 'native-country: United-States', 'race:
White'), Support: 0.519
Item 60: ('sex: Male', 'capital-loss: 0', 'native-country: United-States', 'race: White'),
Support: 0.512
Item 61: ('capital-gain: 0', 'sex: Male', 'capital-loss: 0', 'native-country: United-
States'), Support: 0.508
Item 62: ('capital-gain: 0', 'capital-loss: 0', 'workclass: Private', ' <=50K'), Support:
0.508
Item 63: ('capital-gain: 0', 'workclass: Private', 'native-country: United-States', 'race:
White'), Support: 0.500
```

Fig: Number of frequent itemsets generated with min_support = 0.5

Generating Association Rules

Also I have implemented the code to generate the association rules. The number of association rules generated for minimum support of 0.5 and minimum confidence of 0.8 by running the algorithm on whole data is 177. The algorithm is executed oin whole daatset.

Some of the association rules generated are as follows:

```
Rule: 1  (' <=50K',) ==> ('race: White', 'capital-gain: 0') , 0.801
Rule: 2  ('native-country: United-States',) ==> ('race: White', 'capital-gain: 0') , 0.801
Rule: 3  ('capital-loss: 0', 'sex: Male') ==> ('native-country: United-States', 'capital-gain: 0') , 0.802
Rule: 4  ('capital-loss: 0', 'native-country: United-States', 'capital-gain: 0') ==> (' <=50K',) , 0.805
Rule: 5  ('capital-gain: 0',) ==> ('capital-loss: 0', 'race: White') , 0.806
Rule: 6  ('sex: Male',) ==> ('native-country: United-States', 'capital-gain: 0') , 0.807
Rule: 7  ('capital-loss: 0', 'sex: Male') ==> ('native-country: United-States', 'race: White') , 0.808
Rule: 8  (' <=50K', 'capital-gain: 0') ==> ('capital-loss: 0', 'race: White') , 0.809
Rule: 9  ('workclass: Private', 'native-country: United-States') ==> ('race: White', 'capital-gain: 0') , 0.809
Rule: 10  ('capital-loss: 0', 'capital-gain: 0') ==> (' <=50K',) , 0.810
Rule: 11  ('sex: Male',) ==> ('native-country: United-States', 'race: White') , 0.810
Rule: 12  ('workclass: Private', 'capital-gain: 0') ==> ('capital-loss: 0', 'race: White') , 0.810
Rule: 13  (' <=50K',) ==> ('capital-loss: 0', 'race: White') , 0.811
Rule: 14  ('workclass: Private', 'capital-gain: 0') ==> (' <=50K',) , 0.813
Rule: 15  ('workclass: Private', 'capital-loss: 0') ==> ('native-country: United-States', 'capital-gain: 0') , 0.813
Rule: 16  ('capital-loss: 0',) ==> ('native-country: United-States', 'capital-gain: 0') , 0.816
Rule: 17  ('workclass: Private',) ==> ('capital-loss: 0', 'race: White') , 0.816
Rule: 18  ('workclass: Private',) ==> ('native-country: United-States', 'capital-gain: 0') , 0.818
Rule: 19  ('capital-loss: 0', ' <=50K', 'native-country: United-States') ==> ('race: White', 'capital-gain: 0') , 0.820
Rule: 20  ('native-country: United-States', ' <=50K') ==> ('race: White', 'capital-gain: 0') , 0.823
```

Fig. – Association rules generated.

```
Rule: 158  ('workclass: Private', 'native-country: United-States') ==> ('capital-loss: 0',) , 0.955
Rule: 159  ('capital-loss: 0', ' <=50K', 'native-country: United-States') ==> ('capital-gain: 0',) , 0.956
Rule: 160  (' <=50K', 'race: White') ==> ('capital-gain: 0',) , 0.957
Rule: 161  ('native-country: United-States', ' <=50K', 'race: White') ==> ('capital-gain: 0',) , 0.957
Rule: 162  ('workclass: Private',) ==> ('capital-loss: 0',) , 0.957
Rule: 163  ('capital-loss: 0', ' <=50K') ==> ('capital-gain: 0',) , 0.957
Rule: 164  ('native-country: United-States', ' <=50K') ==> ('capital-gain: 0',) , 0.958
Rule: 165  (' <=50K',) ==> ('capital-gain: 0',) , 0.958
Rule: 166  ('workclass: Private', 'capital-loss: 0', ' <=50K') ==> ('capital-gain: 0',) , 0.960
Rule: 167  ('workclass: Private', ' <=50K') ==> ('capital-gain: 0',) , 0.961
Rule: 168  ('native-country: United-States', ' <=50K', 'race: White', 'capital-gain: 0') ==> ('capital-loss: 0',) , 0.967
Rule: 169  (' <=50K', 'race: White', 'capital-gain: 0') ==> ('capital-loss: 0',) , 0.968
Rule: 170  ('native-country: United-States', ' <=50K', 'capital-gain: 0') ==> ('capital-loss: 0',) , 0.968
Rule: 171  ('native-country: United-States', ' <=50K', 'race: White') ==> ('capital-loss: 0',) , 0.968
Rule: 172  (' <=50K', 'capital-gain: 0') ==> ('capital-loss: 0',) , 0.969
Rule: 173  (' <=50K', 'race: White') ==> ('capital-loss: 0',) , 0.969
Rule: 174  ('native-country: United-States', ' <=50K') ==> ('capital-loss: 0',) , 0.969
Rule: 175  (' <=50K',) ==> ('capital-loss: 0',) , 0.970
Rule: 176  ('workclass: Private', ' <=50K', 'capital-gain: 0') ==> ('capital-loss: 0',) , 0.970
Rule: 177  ('workclass: Private', ' <=50K') ==> ('capital-loss: 0',) , 0.971

177 Association Rules have been generated from the frequent itemsets with minimum confidence 0.8.
```

Fig. Association Rules Generated.

Key Results for Apriori

Here are the key results of running Apriori on the Adult Census Dataset with a minimum support of 0.5:

1) The total number of transactions in Adult.data = 32561
2) Time taken to generate the itemsets = 156.2239 seconds
3) Total number of itemsets generated = 63
4) Maximum itemset size = 5

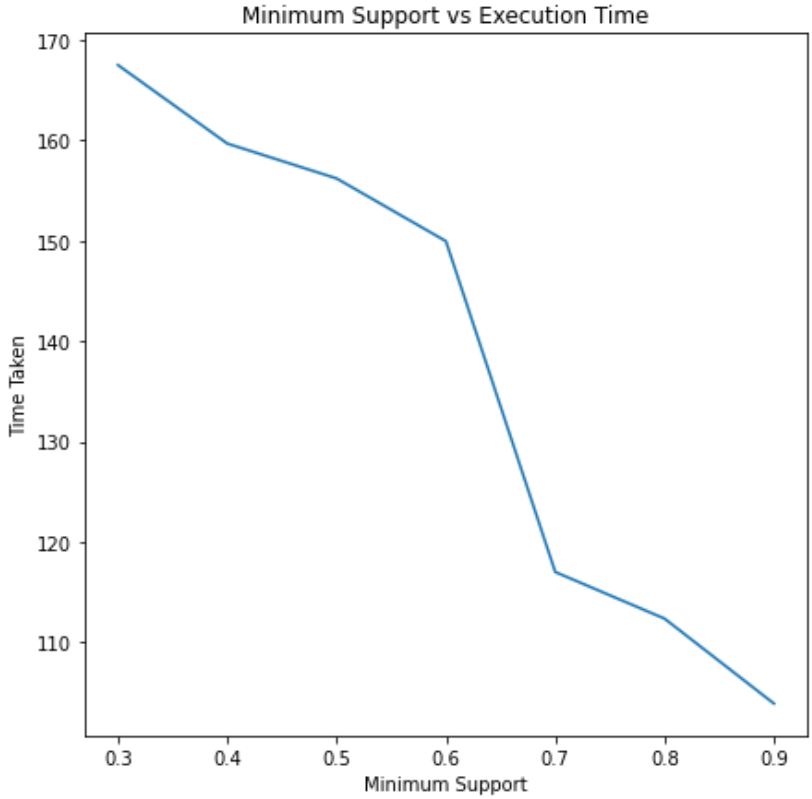
5) Total number of association rules generated = 177

Other Results:

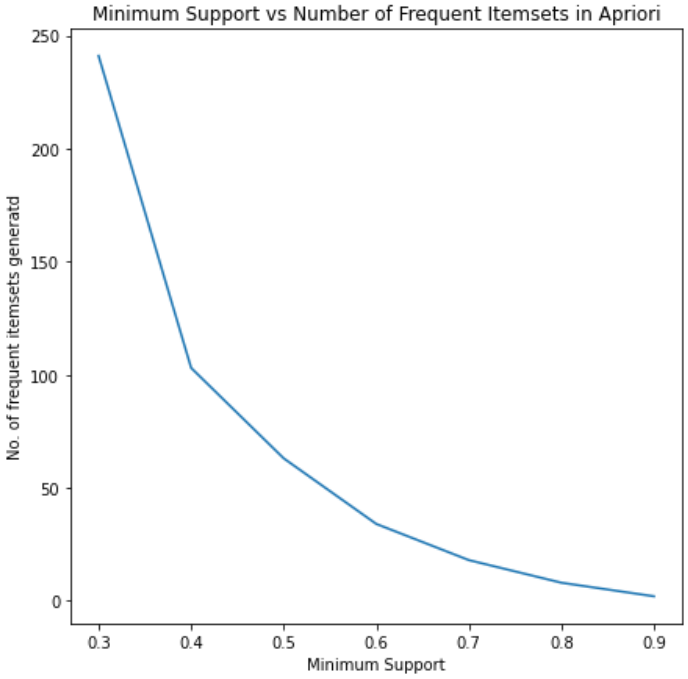
Further I ran the code for different values of minimum support and recorded the number of frequent itemsets generated and also the time taken to execute the code when run on whole dataset.

Support	No. of Freq Itemsets	Execution Time
0.3	241	167.53
0.4	103	159.7026
0.5	63	156.2239
0.6	34	149.9687
0.7	18	116.9856
0.8	8	112.3324
0.9	2	103.8683

The same result can also be viewed graphically as:



It is evident from the graph that as minimum support increases the execution time decreases.



It is evident from the graph that as minimum support increases the the number of frequent itemsets generated decreases.

5.2 FP-Growth:

It is an scalable and efficient way of generating frequent itemsets.

This can be implemented by following steps:

i. Scan DB once, find frequent 1-itemset (single item pattern)
ii. Sort frequent items in frequency descending order, f-list
iii. Scan DB again, construct FP-tree
iv. Construct the conditional FP tree in the sequence of reverse order of F - List –
v. Generate frequent item set

In FP- Growth the database is scanned only twice unlike Apriori. This reduces its execution time.

Output:

```
Item 44: capital-loss: 0, native-country: United-States, sex: Male, Support: 0.567
Item 45: capital-loss: 0, native-country: United-States, race: White,  <=50K, Support:
0.562
Item 46: capital-loss: 0, race: White, sex: Male, Support: 0.557
Item 47: capital-gain: 0, native-country: United-States, race: White,  <=50K, Support:
0.556
Item 48: capital-gain: 0, race: White, workclass: Private, Support: 0.549
Item 49:  <=50K, workclass: Private, Support: 0.545
Item 50: native-country: United-States, race: White, workclass: Private, Support: 0.544
Item 51: capital-loss: 0, capital-gain: 0, native-country: United-States, workclass:
Private, Support: 0.542
Item 52: native-country: United-States, race: White, sex: Male, Support: 0.542
Item 53: capital-gain: 0, native-country: United-States, sex: Male, Support: 0.540
Item 54: capital-loss: 0, capital-gain: 0, native-country: United-States, race: White,
<=50K, Support: 0.537
Item 55: capital-gain: 0, race: White, sex: Male, Support: 0.531
Item 56: capital-loss: 0,  <=50K, workclass: Private, Support: 0.529
Item 57: capital-gain: 0,  <=50K, workclass: Private, Support: 0.523
Item 58: capital-loss: 0, capital-gain: 0, race: White, workclass: Private, Support: 0.522
Item 59: capital-loss: 0, native-country: United-States, race: White, workclass: Private,
Support: 0.519
Item 60: capital-loss: 0, native-country: United-States, race: White, sex: Male, Support:
0.512
Item 61: capital-loss: 0, capital-gain: 0, native-country: United-States, sex: Male,
Support: 0.508
Item 62: capital-loss: 0, capital-gain: 0,  <=50K, workclass: Private, Support: 0.508
Item 63: capital-gain: 0, native-country: United-States, race: White, workclass: Private,
Support: 0.500
fp takes 0.871415376663208
```

Fig: Time taken to generate freq itemsets for min support= 0.5

For this min_support= 0.5, 63 itemsets were generated same as Apriori.

Key Results

The key observations of the output are:

(1) Total number of transactions in Adult.data = 32561
(2) Time taken to generate the itemsets = 0.87 seconds This is much faster than Apriori for generating the itemsets on Adult.data.
(3) Total number of itemsets generated = 63 (Same as Apriori)
(4) Maximum itemset size = 5 (Same as Apriori)
(5) Total number of association rules genereated will be same as that of Apriori as the number of frequent itemset sets are same.

The only two differences between Apriori and FPGrowth are that :

(i) it generates the itemsets much quickly and
(ii) it scans the dataset only two times.

Other Results:

Further I ran the code for different values of minimum support and recorded the number of frequent itemsets generated and also the time taken to execute the code when run on whole dataset.

Support	No. of Freq Itemsets	Execution Time (in seconds)
0.3	241	1.18
0.4	103	0.91
0.5	63	0.87
0.6	34	0.86
0.7	18	0.85
0.8	8	0.845
0.9	2	0.841

The same result can also be viewed graphically as:

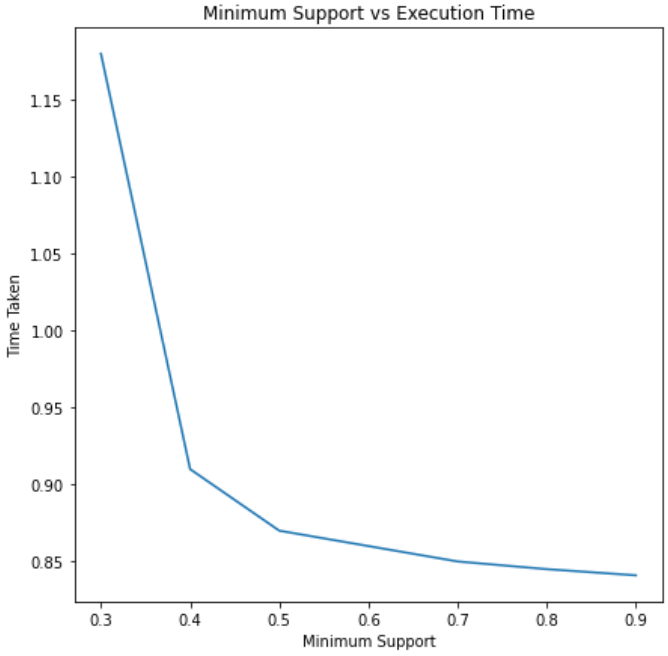


Fig. Minimum Support vs Execution Time FP-Growth

It is evident from the graph that as minimum support increases the execution time decreases.

5.3 Improved Apriori

Even though Apriori performed the desired task, but we still want to improve its efficiency, so that it takes less time to execute.

There are various ways to improve efficiency of Apriori. Some of them are lsited below:

1. Parallelization of itemset frequency check

2. Reduce the number of transactions scanned.

3. Hash-based technique

Here I am going to apply reduced transaction method.

Each transaction is associated with a Transaction ID. :

A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k + 1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent database scans for j-itemsets, where j > k, will not need to consider such a transaction.

It only scan those transaction which have their ID in the transaction ID list and leaves the other transactions.

For a min_support= 0., it produces same number of frequrnt itesets as FP-growth and Apriori that is 63, when run on whole data of 32561 rows. The execution time is slightly better than Apriori.

Key Results

The key observations of the output are:

(1) Total number of transactions in Adult.data = 32561
(2) Time taken to generate the itemsets = 120.891 seconds This is much faster than Apriori for generating the itemsets on Adult.data.
(3) Total number of itemsets generated = 63 (Same as Apriori)
(4) Maximum itemset size = 5 (Same as Apriori)
(5) Total number of association rules genereated will be same as that of Apriori as the number of frequent itemset sets are same.

Other Results:

Further I ran the code for different values of minimum support and recorded the number of frequent itemsets generated and also the time taken to execute the code when run on whole dataset.

Support	No. of Freq Itemsets	Execution Time
0.3	241	151.95
0.4	103	137.37
0.5	63	120.891
0.6	34	110.63
0.7	18	105.211
0.8	8	102.36
0.9	2	96.36

The same result can also be viewed graphically as:

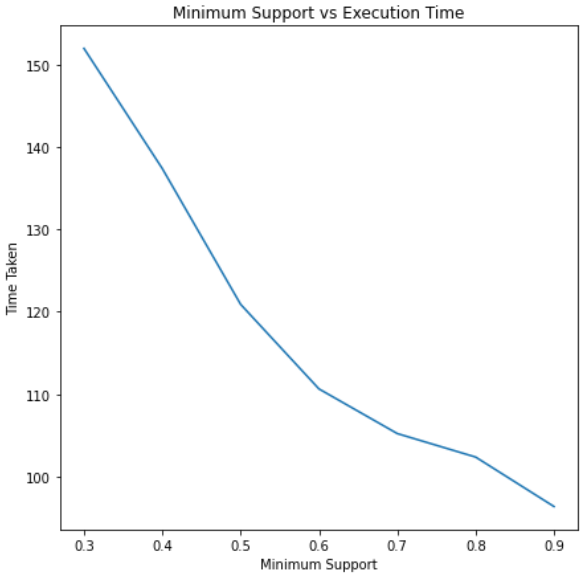


Fig. Minimum Support vs Execution Time FP-Growth

It is evident from the graph that as minimum support increases the execution time decreases.

Comparative Study

For a minimum support equal to 0.5:

- Apriori takes 156.2239 seconds
- Fp-growth takes 0.87 seconds
- Improved Apriori takes 120.89197 seconds.
- Clearly the execution time of FP-growth is much less than that of Apriori.
- The number of frequent itemsets generated by all three algorithms are equal .
- Also the association rules generated by all the algorithms are same.

Execution time for different minimum support whem the algorithm is executed on whole data, 32561 rows.

Support	Apriori	Improved Apriori	FP-Growth
0.3	167.53 seconds	151.95 seconds	1.18 seconds
0.4	159.7026 seconds	137.37 seconds	0.91 seconds
0.5	156.2239 seconds	120.891 seconds	0.87 seconds
0.6	149.9687 seconds	110.63 seconds	0.86 seconds
0.7	116.9856 seconds	105.211 seconds	0.85 seconds
0.8	112.3324 seconds	102.36 seconds	0.845 seconds
0.9	103.8683 seconds	96.36 seconds	0.841 seconds

We can visualise the same result graphically as:

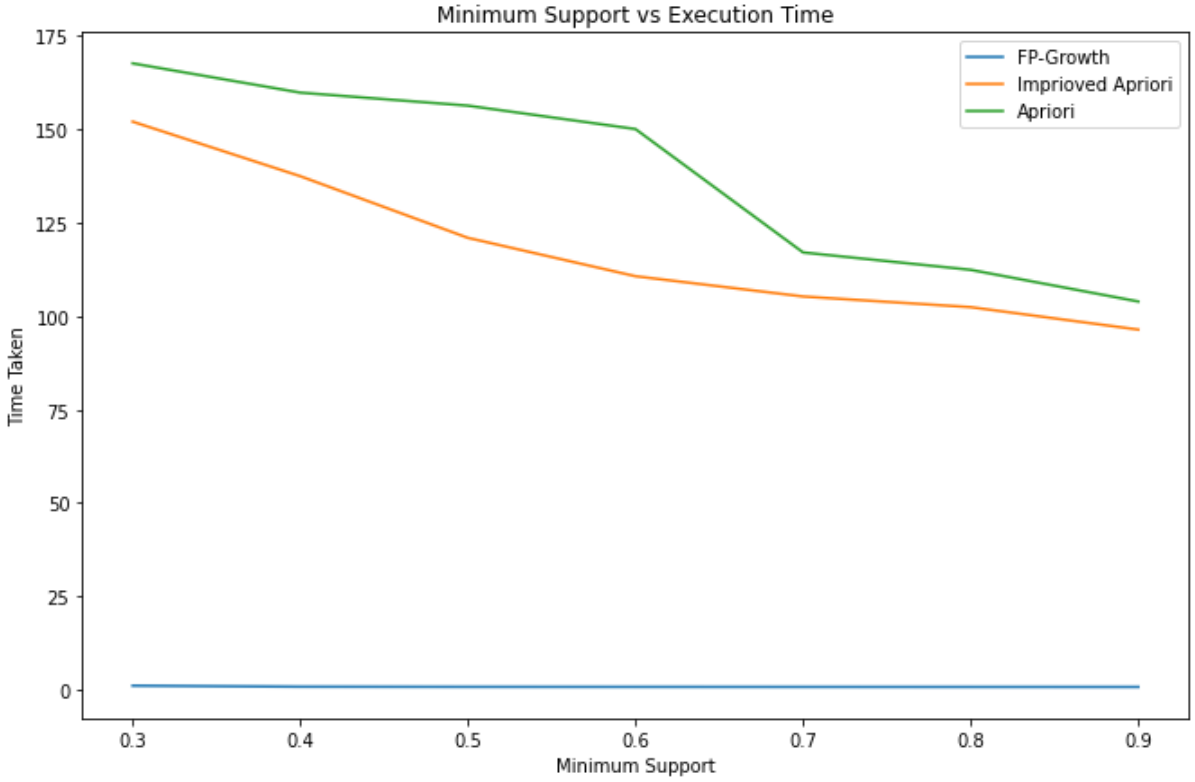


Fig. Comparative study.

In the above plot fp- growth shows a straight line as the the time taken by fp-growth alforithm is very much less than the time taken by the other two algorithms.

FP-growth algorithm is more efficient than Apriori. Following are the reasons to justify why FP-growth works better than Apriori:

- In FP-growth the database is scanned only twice, while Apriori scan the transactions for each iteration.
- It does not involve pairing of itemsets like in Apriori, so works faster than Apriori.
- FP- growth allows frequent itemset without candidate generation.
- FP- growth extracts itemsets directly from the FP tree and traverses through it.
- Execution time in FP-growth is lesser than Apriori due to absence of candidates.

Conclusion

We can conclude by the above results that FP-Growth works better way than Apriori. The number of frequent itemsets generated for a particular value of minimum support is same for both the algorithm. The execution time for FP-Growth is very much less than that of Apriori. Even though Apriori can be improved by changing its algorithm a bit, it cannot outperform FP-Growth. So, I conclude that FP- Growth is the fastest algorithm for generating frequent-itemsets.

References

[1] Jiawei Han, Jian Pei, Micheline Kamber Data Mining: Concepts and Techniques, 3rd Edition, Morgan Kaufmann