How PCY Algorithm works

Danilo Toapanta

2023-06-05

In this post I will go over the implementation of: An Effective Hash-Based Algorithm for Mining Association Rules[1].

Initializing the transaction baskets

```
import numpy as np
import pandas as pd
import itertools
```

```
# initialize lists
data = [[1,2,5], [2,3,5], [4,5], [1,6,7], [2,3,5,7], [1,2,7]]
pairs = [list(itertools.combinations(i, 2)) for i in data]
mydict = {'Items':data, 'Pairs': pairs}

# Create the pandas DataFrame with column name is provided explicitly
df = pd.DataFrame(mydict)

# Print dataframe
display(df)
```

	Items	Pairs
0	[1, 2, 5]	[(1, 2), (1, 5), (2, 5)]
1	[2, 3, 5]	[(2, 3), (2, 5), (3, 5)]
2	[4, 5]	[(4, 5)]
3	[1, 6, 7]	[(1, 6), (1, 7), (6, 7)]
4	[2, 3, 5, 7]	[(2, 3), (2, 5), (2, 7), (3, 5), (3, 7), (5, 7)]
5	[1, 2, 7]	[(1, 2), (1, 7), (2, 7)]

[1]Paper available at: DOI:10.1145/568271.223813

Items	Pairs

Calculating the supports

```
print(data)
```

```
[[1, 2, 5], [2, 3, 5], [4, 5], [1, 6, 7], [2, 3, 5, 7], [1, 2, 7]]
```

```
supports = {}
for row in data:
    for e in row:
        if e not in supports:
            supports[e] = 0

# Sorting dictionary
sorted_dict = supports.keys()
suppports = sorted(sorted_dict)
print(supports)
```

```
{1: 0, 2: 0, 5: 0, 3: 0, 4: 0, 6: 0, 7: 0}
```

```
for row in data:
    for e in row:
        supports[e] += 1

print(supports)
```

```
{1: 3, 2: 4, 5: 4, 3: 2, 4: 1, 6: 1, 7: 3}
```

```
item = [ key for key, value in supports.items()]
supp = [ value for key, value in supports.items()]
unique_items_count = {'Itemset': item, 'Sup': supp}

df_item_sup = pd.DataFrame(unique_items_count)
display(df_item_sup)
```

Table 2: Support

	Itemset	Sup
0	1	3
1	2	4
2	5	4
3	3	2
4	4	1
5	6	1
6	7	3

Pass 1

```
def hash_f_pair(i,j):
    return (i*j) % 7
```

```
unique_items = []
for row in pairs:
    for e in row:
        if e not in unique_items:
            unique_items.append(e)
print(unique_items)
```

```
[(1, 2), (1, 5), (2, 5), (2, 3), (3, 5), (4, 5), (1, 6), (1, 7), (6, 7), (2, 7), (3, 7), (5,
```

```
unique_items[0][1]
```

```
2
```

```
hash_value_pairs = []
for e in unique_items:
        hash_value_pairs.append(hash_f_pair(e[0],e[1]))
print(hash_value_pairs)
```

```
[2, 5, 3, 6, 1, 6, 6, 0, 0, 0, 0, 0]
```

```
my_dict2 = {}
for pair in unique_items:
    my_dict2[str(pair)] = 0
print(my_dict2)
```

```
{'(1, 2)': 0, '(1, 5)': 0, '(2, 5)': 0, '(2, 3)': 0, '(3, 5)': 0, '(4, 5)': 0, '(1, 6)': 0,
```

```
for row in pairs:
    for e in row:
        my_dict2[str(e)] +=1

print(my_dict2)
```

```
{'(1, 2)': 2, '(1, 5)': 1, '(2, 5)': 3, '(2, 3)': 2, '(3, 5)': 2, '(4, 5)': 1, '(1, 6)': 1,
```

```
counts = [ value for key, value in my_dict2.items()]
print(counts)
# df_counts = pd.DataFrame()
```

```
[2, 1, 3, 2, 2, 1, 1, 2, 1, 2, 1, 1]
```

```
unique_items_count = {'Pairs': unique_items, 'Count': counts, 'Hash': hash_value_pairs}
print(unique_items_count)
```

```
{'Pairs': [(1, 2), (1, 5), (2, 5), (2, 3), (3, 5), (4, 5), (1, 6), (1, 7), (6, 7), (2, 7), (
```

```
df_counts = pd.DataFrame(unique_items_count)
display(df_counts)
```

Table 3: Hash Pair Table

	Pairs	Count	Hash
0	(1, 2)	2	2
1	(1, 5)	1	5
2	(2, 5)	3	3
3	(2, 3)	2	6

	Pairs	Count	Hash
4	(3, 5)	2	1
5	(4, 5)	1	6
6	(1, 6)	1	6
7	(1, 7)	2	0
8	(6, 7)	1	0
9	(2, 7)	2	0
10	(3, 7)	1	0
11	(5, 7)	1	0

Minium support count is 2 so we eliminate from the Pair list the items that have less than 2 from Table 2. The resulting table is called the Candidate Pairs.

```
df_counts_after = df_counts.drop([5, 6, 8])
display(df_counts_after)
```

Table 4: Hash Pair Table after Pass 1

	Pairs	Count	Hash
0	(1, 2)	2	2
1	(1, 5)	1	5
2	(2, 5)	3	3
3	(2, 3)	2	6
4	(3, 5)	2	1
7	(1, 7)	2	0
9	(2, 7)	2	0
10	(3, 7)	1	0
11	(5, 7)	1	0

Pass 2

The final step is to build a table from Table 3 that counts the number of ocurrences per hash value.

```
hash_values = [x for x in range(max(hash_value_pairs)+1)]
my_dict3 = {}
for e in hash_values:
    my_dict3[e] = 0
```

```
for i, hash_value in enumerate(df_counts["Hash"]):
    my_dict3[hash_value] += df_counts["Count"][i]
print(my_dict3)

data_bucket = {"Bucket": hash_values, "Count": my_dict3.values()}
df_bucket_count = pd.DataFrame(data_bucket)
display(df_bucket_count)
```

```
{0: 7, 1: 2, 2: 2, 3: 3, 4: 0, 5: 1, 6: 4}
```

	Bucket	Count
0	0	7
1	1	2
2	2	2
3	3	3
4	4	0
5	5	1
6	6	4

With the result above we can look back to table Table 4 and see which hash values, have a value lower than the minimum support count. That is hash value 4 and 5. With this result we look at table Table 4 and delete those corresponding hash values. The final result is teh Final Candidates

```
df_counts_after_pass2 = df_counts_after.drop([1])
display(df_counts_after_pass2)
```

	Pairs	Count	Hash
0	(1, 2)	2	2
2	(2, 5)	3	3
3	(2, 3)	2	6
4	(3, 5)	2	1
7	(1, 7)	2	0
9	(2, 7)	2	0
10	(3, 7)	1	0
11	(5, 7)	1	0