ASRO Lab8 ☆

File Edit View Insert Runtime Tools Help Last saved at 3:20 PM

Comment Share

+ Code + Text Connect

```
[ ] !pip install mlxtend --upgrade
```

```
[ ] !pip install spark
!pip install -U scikit-wine
```

Question 1

Test Drive ACLOSE algorithm to mine closed frequent patterns on a sample dataset of your choice. Test the same on a FIMI benchmark dataset which you have used for Apriori/FP-growth implementations.

```
[ ] import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth
from collections import defaultdict
import time

# load the dataset
data = fetch_mushroom()
data.keys()


def closed_itemsets(min_sup, min_conf):
    frequent_itemsets = list(fpgrowth(data, min_support=min_sup, min_confidence=min_conf))

    closed_freq_sets = []
    for i, itemset in enumerate(frequent_itemsets):
        is_closed = True
        support = itemset.support

        is_closed = True
        for j, other_itemset in enumerate(frequent_itemsets):
            if i != j:
                other_antecedent = other_itemset.item
                other_support = other_itemset.support

                if set(itemset.antecedent).issubset(set(other_antecedent)) and support == other_support:
                    is_closed = False
                    break

        if is_closed:
            closed_freq_sets.append(frozenset(itemset, support))

    return closed_freq_sets
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the resu
  and should_run_async(code)
```

```
[ ] start = time.process_time()
result = closed_itemsets(0.5, 0.8)
end = time.process_time()

# print the results
print("\n\n\n Time", end-start)

print("\n\n\n Closed Itemsets:")
result
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the resu
  and should_run_async(code)

Time taken: 0.4565380.700000


Closed Itemsets:
[(frozenset({0}), 1.0),
 (frozenset({1}), 0.7253236245954),
 (frozenset({4, 85}), 0.5284411120930276),
 (frozenset({4, 86}), 0.5703288940092753),
 (frozenset({4, 89}), 0.8230188679245283),
 (frozenset({4, 90}), 0.8230188679245283),
 (frozenset({4, 91}), 0.8230188679245283),
 (frozenset({4, 93}), 0.8570188679245283),
 (frozenset({4, 95}), 0.8570188679245283),
 (frozenset({4, 98}), 0.8230188679245283),
 (frozenset({4, 99}), 0.8570188679245283),
 (frozenset({4, 34, 85}), 0.5285918260696165),
 (frozenset({4, 34, 86}), 0.5704287571180665),
 (frozenset({4, 34, 89}), 0.8230188679245283),
 (frozenset({4, 34, 90}), 0.8230188679245283),
 (frozenset({4, 34, 91}), 0.8230188679245283),
 (frozenset({4, 34, 93}), 0.8570188679245283),
 (frozenset({4, 34, 95}), 0.8570188679245283),
 (frozenset({4, 34, 98}), 0.8230188679245283),
 (frozenset({4, 34, 99}), 0.8570188679245283),
 (frozenset({4, 36, 85}), 0.5285918260696165),
 (frozenset({4, 36, 86}), 0.5704287571180665),
 (frozenset({4, 36, 89}), 0.8230188679245283),
 (frozenset({4, 36, 90}), 0.8230188679245283),
 (frozenset({4, 36, 91}), 0.8230188679245283),
 (frozenset({4, 36, 93}), 0.8570188679245283),
 (frozenset({4, 36, 95}), 0.8570188679245283),
 (frozenset({4, 36, 98}), 0.8230188679245283),
 (frozenset({4, 36, 99}), 0.8570188679245283),
 (frozenset({4, 85, 86}), 0.4127061585988165),
 (frozenset({4, 85, 89}), 0.5284411120930276),
 (frozenset({4, 85, 90}), 0.5284411120930276),
 (frozenset({4, 85, 91}), 0.5284411120930276),
 (frozenset({4, 85, 93}), 0.5284411120930276),
 (frozenset({4, 85, 95}), 0.5284411120930276),
 (frozenset({4, 85, 98}), 0.5284411120930276),
 (frozenset({4, 85, 99}), 0.5284411120930276),
 (frozenset({4, 86, 89}), 0.5703288940092753),
 (frozenset({4, 86, 90}), 0.5703288940092753),
 (frozenset({4, 86, 91}), 0.5703288940092753),
 (frozenset({4, 86, 93}), 0.5704287571180665),
 (frozenset({4, 86, 95}), 0.5704287571180665),
 (frozenset({4, 86, 98}), 0.5703288940092753),
 (frozenset({4, 86, 99}), 0.5704287571180665),
 (frozenset({4, 89, 90}), 0.8230188679245283),
 (frozenset({4, 89, 91}), 0.8230188679245283),
 (frozenset({4, 89, 93}), 0.8570188679245283),
 (frozenset({4, 89, 95}), 0.8570188679245283),
 (frozenset({4, 89, 98}), 0.8230188679245283),
 (frozenset({4, 89, 99}), 0.8570188679245283),
 (frozenset({4, 90, 91}), 0.8230188679245283),
 (frozenset({4, 90, 93}), 0.8570188679245283),
 (frozenset({4, 90, 95}), 0.8570188679245283),
 (frozenset({4, 90, 98}), 0.8230188679245283),
 (frozenset({4, 90, 99}), 0.8570188679245283),
 (frozenset({4, 91, 93}), 0.8570188679245283),
 (frozenset({4, 91, 95}), 0.8570188679245283),
 (frozenset({4, 91, 98}), 0.8230188679245283),
 (frozenset({4, 91, 99}), 0.8570188679245283),
 (frozenset({4, 93, 95}), 0.8570188679245283),
 (frozenset({4, 93, 98}), 0.8570188679245283),
 (frozenset({4, 93, 99}), 0.8570188679245283),
 (frozenset({4, 95, 98}), 0.8570188679245283),
 (frozenset({4, 95, 99}), 0.8570188679245283),
 (frozenset({4, 98, 99}), 0.8570188679245283),
 (frozenset({4, 34, 85, 86}), 0.4127061585988165),
 (frozenset({4, 34, 85, 89}), 0.5285918260696165),
 (frozenset({4, 34, 85, 90}), 0.5285918260696165),
 (frozenset({4, 34, 85, 91}), 0.5285918260696165),
 (frozenset({4, 34, 85, 93}), 0.5285918260696165),
 (frozenset({4, 34, 85, 95}), 0.5285918260696165),
 (frozenset({4, 34, 85, 98}), 0.5285918260696165),
 (frozenset({4, 34, 85, 99}), 0.5285918260696165),
 (frozenset({4, 34, 86, 89}), 0.5704287571180665),
 (frozenset({4, 34, 86, 90}), 0.5704287571180665),
 (frozenset({4, 34, 86, 91}), 0.5704287571180665),
 (frozenset({4, 34, 86, 93}), 0.5704287571180665),
 (frozenset({4, 34, 86, 95}), 0.5704287571180665),
 (frozenset({4, 34, 86, 98}), 0.5704287571180665),
 (frozenset({4, 34, 86, 99}), 0.5704287571180665),
 (frozenset({4, 36, 85, 86}), 0.4127061585988165),
 (frozenset({4, 36, 85, 89}), 0.5285918260696165),
 (frozenset({4, 36, 85, 90}), 0.5285918260696165),
 (frozenset({4, 36, 85, 91}), 0.5285918260696165),
 (frozenset({4, 36, 85, 93}), 0.5285918260696165),
 (frozenset({4, 36, 85, 95}), 0.5285918260696165),
 (frozenset({4, 36, 85, 98}), 0.5285918260696165),
 (frozenset({4, 36, 85, 99}), 0.5285918260696165),
 (frozenset({4, 36, 86, 89}), 0.5704287571180665),
 (frozenset({4, 36, 86, 90}), 0.5704287571180665),
 (frozenset({4, 36, 86, 91}), 0.5704287571180665),
 (frozenset({4, 36, 86, 93}), 0.5704287571180665),
 (frozenset({4, 36, 86, 95}), 0.5704287571180665),
 (frozenset({4, 36, 86, 98}), 0.5704287571180665),
 (frozenset({4, 36, 86, 99}), 0.5704287571180665),
 (frozenset({4, 85, 86, 89}), 0.5284411120930276),
 (frozenset({4, 85, 86, 90}), 0.5284411120930276),
 (frozenset({4, 85, 86, 91}), 0.5284411120930276),
 (frozenset({4, 85, 86, 93}), 0.5284411120930276),
 (frozenset({4, 85, 86, 95}), 0.5284411120930276),
 (frozenset({4, 85, 86, 98}), 0.5284411120930276),
 (frozenset({4, 85, 86, 99}), 0.5284411120930276),
 (frozenset({4, 34, 85, 86, 89}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 90}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 91}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 93}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 95}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 98}), 0.5285918260696165),
 (frozenset({4, 34, 85, 86, 99}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 89}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 90}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 91}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 93}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 95}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 98}), 0.5285918260696165),
 (frozenset({4, 36, 85, 86, 99}), 0.5285918260696165),
```

Question 2

Test Drive Prism search to mine maximal frequent patterns on a sample dataset of your choice. Test the same on a FIMI benchmark dataset which you have used for Apriori/FP-growth implementations.

```
[ ] from IPython.display import transcript
display(transcript('google.colab.output.setIframeHeight(0, true, {maxHeight: 5000})'))
```

```
def mkv_gen(ab,mfs):
    mfs = mfs.copy()

    for infrequent_itemset in ab:
        for mfs_itemset in mfs.copy():
            if set(_item in mfs_itemset for _item in infrequent_itemset):  # if infrequent itemset is a subset of mfs itemset
                mfs.remove(mfs_itemset)

                for item in infrequent_itemset:
                    updated_mfs_itemset = mfs_itemset.copy()
                    updated_mfs_itemset.remove(item)
                    if not any(ui[0]==tu for ui,tu in updated_mfs_itemset for mfs_itemset in mfs):
                        mfs.append(updated_mfs_itemset)

    return mfs




def gen_next_ck(k,k-1):
    num_freq=len(k)
    cmsets=[]
    for i in range(num_freq):
        j=i+1
        print(f"comm freq: {k[i][:k-1]==k[j][:k-1]}")
        new_itemset=k[i][:k-1]+k[i][k-1]+k[j][k-1]
        insert_is_new=True
        # Ktest
        for _item in new_itemset:
        if len(_item) == k-1:
            insert_is_new=True
        else:
            for s in combinations(k[i:j],k-1):
                if set(_item in s for _item in s):
                    insert_is_new=True
        if insert_is_new:
            cmsets.append(new_itemset)
        j+=1

    return cmsets




def mfs_prune(ck,k,curr_mfs):
    new_ck=ck.copy()
    for itemset in ck:
        for item in curr_mfs:
            if set(i).issubset(set(itemset)):
                new_ck.remove(i)

    return new_ck




def mfcs_prune(ck,k,infreq_itemsets,curr_mfcs):
    new_cksets=[]
    for i in ck:cksets
        for infreq_itemset in infrs:
            if set(i).issubset(set(itemset)):
                new_cksets.append(i)

    return new_cksets




def pincerSearch(tran,min_sup):
    item_counts = {}
    for transaction in tran:
        item_counts.update(transaction)
    level_k=1
    cand_freq_itemsets=[[item] for item in item]
    level_freq_itemsets=[]
    level_infreq_itemsets=[]

    mfcs=[item.copy()]
    mfs=[]

    print("MFCS:{mfcs}")
    print("MFS:{mfs}")

    while len(cand_freq_itemsets)!=0:
        print("Level {level_k}")
        print("{level_k} : {cand_freq_itemsets}")

        cand_freq_itemsets_cnt={frozenset(set_freq_itemset):
        mfcs_itemsets_set={frozenset(mfcs)}

        # step 1: read pass from db and get support for ck and mfcs
        for each in tran:
            for k_itemset in enumerate(cand_freq_itemsets):
                if set(k_itemset).issubset(each):
                    cand_freq_itemsets_cnt[i]+=1
            for i,itemset in enumerate(mfcs):
                if set(itemset).issubset(each):
                    mfcs_itemsets_set[i]+=1

        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            print(f"{itemset}  {supp}")
        print("\n")
        for itemset,supp in zip(mfcs,mfcs_itemsets_set):
            print(f"{itemset}  {supp}")
        print("\n")

        # step 2: infreq itemsets from step 1a ck
        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            if itemset/min_sup:
                infreq_itemsets.append(itemset)

        print("MFS : {mfs}")
        print("infreq_itemsets:{infreq_itemsets}")
        # step 3: infreq itemsets in mfcs
        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            if itemset/min_sup:
                mfcs_itemsets.append(itemset)
            if itemset/min_sup:
                level_infreq_itemsets.append(itemset)

        print("{level_k} : {level_freq_itemsets}")
        print("{level_k} : {level_infreq_itemsets}")

        # step 4: mfcs_prune if ck is non empty
        mfcs=mfcs_prune(level_infreq_itemsets,mfcs)
        print("MFCS : {mfcs}")

        # step 5 : pruning cand using mfs
        print("{level_k} mfs : {level_freq_itemsets}")
        level_freq_itemsets=mfs_prune(level_freq_itemsets,mfs)
        print("After pruning,{level_k} : {level_freq_itemsets}")

        # step 6 : gen next ck from level k-1
        cand_freq_itemsets=gen_next_ck(level_freq_itemsets,level_k)

        # step 7 : prune cand k with mfcs
        cand_freq_itemsets=mfcs_prune(cand_freq_itemsets,mfcs)

        level_k+=1

    return mfs



transactions = [{1, 2, 3, 4},
                {1, 2, 4},
                {1, 2, 5},
                {1, 3, 4},
                {1, 3, 5},
                {1, 4, 5},
                {2, 3, 4},
                {2, 3, 5},
                {2, 4, 5},
                {3, 4, 5},
                {1, 2, 3},
                {1, 2, 4},
                {1, 2, 5},
                {1, 3, 4},
                {1, 3, 5},
                {1, 4, 5},
                {2, 3, 4},
                {2, 3, 5},
                {2, 4, 5},
                {3, 4, 5}]

min_support_count = 3
mfs = pincerSearch(transactions, min_support_count)
print("MFS:{}".format(mfs))
```

```
MFS:[{1, 2, 3, 4, 5, 6, 7, 8, 9}]

mfs[]

level 1
{1 : [{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}]
{1} 4
{2} 4
{3} 4
{4} 4
{5} 4
{6} 3
{7} 3
{8} 3
{9} 3

{1, 2, 3, 4, 5, 6, 7, 8, 9} -0


mfs : []
C2 : [{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}]
mfs : [{1, 2, 3, 4, 5, 6, 7, 8, 9}]
After pruning,2 : [{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}]
other pruning,2 : [{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}]
{1, 2} 4
{1, 3} 4
{1, 4} 4
{1, 5} 4
{1, 6} 3
{1, 7} 3
{1, 8} 3
{1, 9} 3
{2, 3} 4
{2, 4} 4
{2, 5} 4
{2, 6} 3
{2, 7} 3
{2, 8} 3
{2, 9} 3
{3, 4} 4
{3, 5} 4
{3, 6} 3
{3, 7} 3
{3, 8} 3
{3, 9} 3
{4, 5} 4
{4, 6} 3
{4, 7} 3
{4, 8} 3
{4, 9} 3
{5, 6} 3
{5, 7} 3
{5, 8} 3
{5, 9} 3
{6, 7} 3
{6, 8} 3
{6, 9} 3

{1, 2, 4, 5, 6, 7, 8, 9} - 0


mfs : []
C3 : [{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 2, 6}, {1, 2, 7}, {1, 2, 8}, {1, 2, 9}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 3, 8}, {1, 3, 9}, {1, 4, 5}, {1, 4, 6}, {1, 4, 7}, {1, 4, 8}, {1, 4, 9}, {1, 5, 6}, {1, 5, 7}, {1, 5, 8}, {1, 5, 9}, {1, 6, 7}, {1, 6, 8}, {1, 6, 9}]
mfs : [{1, 2, 4}, {1, 2, 5}, {1, 2, 6}, {1, 2, 7}, {1, 2, 8}, {1, 2, 9}]
After pruning,3 : [{1, 2, 3}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 3, 8}, {1, 3, 9}, {1, 4, 5}, {1, 4, 6}, {1, 4, 7}, {1, 4, 8}, {1, 4, 9}, {1, 5, 6}, {1, 5, 7}, {1, 5, 8}, {1, 5, 9}, {1, 6, 7}, {1, 6, 8}, {1, 6, 9}]
other pruning,3 : [{1, 2, 3}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 3, 8}, {1, 3, 9}]
{1, 2, 3} 4
{1, 3, 4} 4
{1, 3, 5} 4
{1, 3, 6} 3
{1, 3, 7} 3
{1, 3, 8} 3
{1, 3, 9} 3

{1, 3, 4, 5, 6, 7, 8, 9} - 0


mfs : []
C4 : [{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 3, 6}, {1, 2, 3, 7}, {1, 2, 3, 8}, {1, 2, 3, 9}, {1, 3, 4, 5}, {1, 3, 4, 6}, {1, 3, 4, 7}, {1, 3, 4, 8}, {1, 3, 4, 9}, {1, 3, 5, 6}, {1, 3, 5, 7}, {1, 3, 5, 8}, {1, 3, 5, 9}, {1, 3, 6, 7}, {1, 3, 6, 8}, {1, 3, 6, 9}]
mfs : [{1, 2, 3}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 3, 8}, {1, 3, 9}]
After pruning,4 : []
other pruning,4 : []
MFS : [{1, 2, 3}, {1, 3, 4}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 3, 8}, {1, 3, 9}]
```

Colab paid products - Cancel contracts here