CSCI 4144/6405

Data Mining and Data Warehousing

A Tutorial for Apriori Algorithm

Winter 2017

Revised by Virlla Devi Soothar
vr265712@dal.ca

Outline

1. **Association Rule Mining and Apriori**
2. **Implementation Steps and Hints**
3. **Program Demo**

What is Association Mining?

> *"Finding **frequent patterns**, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories."*
>
> *-- Han, Kamber*

- For example:
  - What products are often purchased together?

    If a customer buys a vegetable, there is a good chance that he will also buy a snack food.

    If a customer buys a phone, there is a good chance that she will also buy a case.

Notation for Association Rules

- Association rules have the form:

  { [var]=[value], … } → { [var]=[value], … } [Supp, Conf]

- For example (In this assignment):

  {[PlayTennis]=[Y],[Humidity]=[normal]}
  ---> {[Windy]=[false]} [(Support=28.6%, Confidence=66.7%)]

Support and Confidence

- **Support** is an indication that how frequent itemsets appear in the database.
- Rules that have a support rate greater than a user-specified support is said to have minimum support.

- **Confidence** is the indication of how often rule has been found true.
- Rules that have a confidence greater than a user-specified confidence is said to have minimum confidence

Notation: Support and Confidence Measures

Support rate for {A}→{B} = $\frac{\text{Count rows where \{A\} and \{B\} occur}}{\text{Total Row Count}}$

(*percentage of transactions (rows) that contain* ***both A and B***)

Confidence rate for {A}→{B} = $\frac{\text{Support Count for \{A\}} \rightarrow \text{\{B\}}}{\text{Support Count for \{A\}}}$

(*total transactions (rows) that contain both A and B, divided by the number of transactions that* ***only*** *contain A*)

Confidence rate for {B}→{A} = $\frac{\text{Support Count for \{B\}} \rightarrow \text{\{A\}}}{\text{Support count for \{B\}}}$

(*stating the same rule in reverse: this rule would have the same support value, but different confidence!*)

Example

- Data

TID	Itemsets
T1	A,B,C
T2	A,B,D
T3	B,C
T4	A,C
T5	B,C,D

- Find Support A-> B

$$\text{Support}(A \rightarrow B) = \frac{\text{Count rows where } \{A\} \text{ and } \{B\} \text{ occur}}{\text{Total Row Count}}$$

$$=2/5$$

$$=0.4$$

Example

$$\text{Support}(B \rightarrow A) = \frac{\text{Count rows where } \{A\} \text{ and } \{B\} \text{ occur}}{\text{Total Row Count}}$$

$=2/5$

$=0.4$

$$\text{Confidence}(A \rightarrow B) = \frac{\text{Support Count for } \{A\} \rightarrow \{B\}}{\text{Support Count for } \{A\}}$$

$=2/3$

$=0.667$

$$\text{Confidence}(B \rightarrow A) = \frac{\text{Support Count for } \{B\} \rightarrow \{A\}}{\text{Support Count for } \{A\}}$$

$=2/4$

$=0.5$

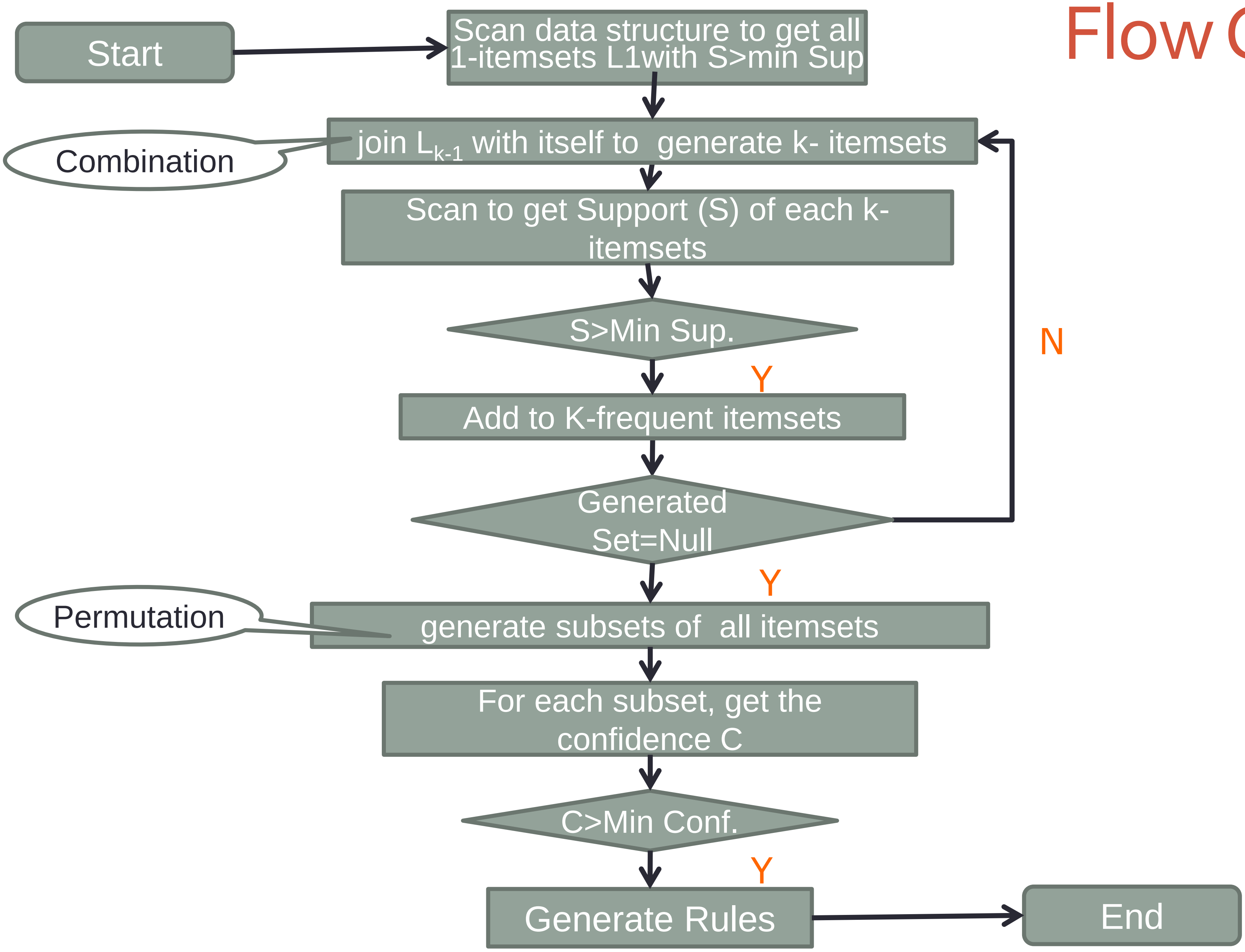
Apriori: Some Definitions

- Item: attribute-value pair < attribute: value>
- Itemset: A group of items is referred to as an itemset.
- An itemset that contains k items is a k-itemset.
- If an itemset satisfies minimum support, then it is a frequent itemset.
- A set of frequent k-itemsets is commonly denoted by L_k.
- To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k.

Apriori Algorithm Steps

- Two main steps:
  1. Find frequent itemsets from source data
     Frequent itemsets must have support rate >= minimum support rate
  2. Generate rules from frequent itemsets
     Rules must have confidence >= minimum confidence

Flow Chart


Start
Scan data structure to get all 1-itemsets L1with S>min Sup
Combination
join L_{k-1} with itself to generate k- itemsets
Scan to get Support (S) of each k-itemsets
S>Min Sup.
Y
N
Add to K-frequent itemsets
Generated Set=Null
Y
Permutation
generate subsets of all itemsets
For each subset, get the confidence C
C>Min Conf.
Y
Generate Rules
End

Apriori Algorithm: Our Context

- Get parameters from user (source file, min.sup, min.conf, etc.)
- Read source data into data structures

- Scan data structure to find all 1-itemsets
- Count occurrences of 1-itemsets
  - reject those with support < minimum support
- Join 1-itemsets to generate list of candidate 2-itemsets
- Count occurrences of 2-itemsets
  - reject those with support < minimum support
- Join 2-itemsets to generate list of candidate 3-itemsets
- … *(until no more joins are possible)*

- For each frequent itemsets where k>1, generate all candidate rules
- Calculate confidence for each candidate rule
  - reject those with confidence < minimum confidence

- Output list of frequent rules to a file

Apriori Principle: Finding all frequent itemsets

- **Principle**:
  - If an itemset is infrequent, its superset is infrequent too
  - Any subset of a frequent itemset must be frequent

  Example:

  Let say A,B→C is frequent then A→ B , A→C , B→C, C→B should also be frequent.

  Let say A→D are not frequent then A,B→ D , A,D→C etc should also be infrequent.

- **Method**:
  - Generate L_{k+1} itemsets from L_k frequent itemsets only (JOIN)
  - Verify the L_{k+1} itemsets by checking min. sup (PRUNE)

Frequent Itemsets (minsup=2)

From L2 to C3: {1,2} is not frequent but still in C3 we have {1,2,3}

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{4}~~	~~1~~
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
~~{1 2}~~	~~1~~
{1 3}	2
~~{1 5}~~	~~1~~
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset	Sup
{2 3 5}	2
~~{1 2 3}~~	~~1~~
~~{1,3,5}~~	~~1~~
~~{1,2,5}~~	~~1~~

Scan D

L_3

itemset	sup
{2 3 5}	2

Apriori Algorithm: Join itemsets

- Evaluate all possible pairs of itemsets from previous level: similar to *combination*
- Only join items that have the same first item (to create k-itemsets (k>2), the first k - 2 items must match)
- For example, if we have two 4-itemsets:

  { 1 2 3 4 }

  { 1 2 3 5 }
- First: can these 4-itemsets be joined?
  - Yes, because 3 of the items match (k-1 = 3)
- Next: generate the 5-itemset:

  { 1 2 3 4 5 }

Apriori Algorithm: Candidate Rules

- Evaluate all frequent itemsets: similar to *permutation*
- For example, if we have a frequent itemset:

  { 1 2 3 }

- … then we should test these candidate rules:

  { 1 } → { 2 3 }
  { 2 } → { 1 3 }
  { 3 } → { 1 2 }

  { 1 2 } → { 3 }
  { 1 3 } → { 2 }
  { 2 3 } → { 1 }

- But how to implement this efficiently?
  - *Hint: work through a larger example by hand, on paper, and think how you would do this in an ordered way.*

Item and Itemset

- An item in relational database is an attribute-value pair, which can be encoded using attribute-value indexing scheme.
  - e.g.

  Outlook =overcast can be encoded as <1.2>

  temperature = hot can be encodes as <2.1>

- An itemset is a set of attribute-value pairs.
  - e.g. (outlook=rain) and (temperature=hot) is represented as {<1,3>,<2,1>}.

Item Encoding

Outlook	Temperature
Sunny	Hot
Overcast	Mild
Rain	Cool
Sunny	Mild

Outlook (attribute No.=1)

- Sunny (value No.=1)
- Overcast (value No.=2)
- Rain (value No.=3)

Temperature (attribute No.=2)

- Hot (value No.=1)
- Mild (value No.=2)
- Cool (value No.=3)

e.g.

outlook=rain can be encoded as "1.3"

temperature=hot can be encoded as "2.1".

Hint:

Here String has been used as a representation for the item.

And we use dot "." to denote the entry for convenience. The left number is the attribute id while the right is the value id

Itemset Encoding

Outlook	Temperature
Sunny	Hot
Overcast	Mild
Rain	Cool
Sunny	Mild

- Now we can convert the relational db to transactions based on the previous scheme:
- E.g.
  - <Outlook=Sunny, Temperature=Hot> ➔ <1.1, 2.1>
  - <Outlook=Rain, Temperature=Cool> ➔ <1.3, 2.3>
  - <Outlook=Sunny, Temperature=Mild> ➔ <1.1, 2.2>

Itemset Encoding

- An Itemset is a set of items, so we can denote one Itemset using Set structure.
  - You can make use of the functions (intersection, union etc.) provided by the data structure.

- E.g. **HashSet<String>**

Outlook	Temperature
Sunny	Hot

➔ HashSet<“1.1”, “2.1”>

Itemset Encoding With its count

Map<Set<String>, Integer> can be used to store the itemsets with their counts

- E.g.
  - **<<“1.1”>, 4>** : Itemset of <Outlook=Sunny> appears 4 times in the database
  - **<<“1.1”, “2.3”>, 300>** : Itemset of <Outlook=Sunny, Temperature=Cool> appears 300 times in the database

Itemsets Join

- **Note:**

  An itemset can not have two/more items sharing a same attribute name.
- The above constraint is a major difference between association rule mining from relational databases and transactional databases.
- It imposes a constraint on itemset join:
  - Two k-itemset p and q are joinable iff p and q have k-1 identical attribute-value pairs and different attributes in one attribute-value pair.
  - E.g.
    - <“1.2”, “2.1”> and <“1.2”, “3.2”> can be joined to create <“1.2”, “2.1”, “3.2”>.
    - But {<“1.2”, **“2.1”**>} and {<“1.2”, **“2.2”**>} is not joinable.

Example of Candidate Pruning(Bonus)

- L_3={*abc, abd, acd, ace, bcd*}
- Self-joining: $L_3 * L_3$
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*
- Pruning:
- *acde* is removed because *ade* is not in L_3
- C_4={*abcd*}

- Hint: To make this process efficiently, try to use the set operations provided in the APIs. One example is shown in page 26.

Frequent Itemsets(With candidate Pruning, minsup=2)

From L2 to C3: {1,2} {1,5} is not frequent so we don't need to generate {1,2,3} , {1,3,5} , {1,2,5}

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

{1}	2
{2}	3
{3}	3
~~{4}~~	~~1~~
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
~~{1 2}~~	~~1~~
{1 3}	2
~~{1 5}~~	~~1~~
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset	Sup
{2 3 5}	2

Scan D

L_3

itemset	sup
{2 3 5}	2

A General Implementation Architecture

General Code Structure (main functions):

- loadDB()
- buildFirstItemset()
- pruneItemset()
- genFrequentKItemset()
  - generateCandidates() //do not forget to use apriori rules while // generating candidates
  - countAndPruneItemsInDB()
- createRules()

Constructor

Association_rule_mining{

1. loadDB(database);
   //read in the data file

2. buildFirstItemset(all_Tuples, minsup);
   //create the candidate 1-itemset ,and then frequent 1-itemset

3. genFrequentKItemset(all_Tuples, all_1_itemsets, minsup);
   //using apriori algorithm to find all frequent K-itemSets (K>=2)

4. createRules(all_Frequent_itemsets, minsup, minconf);
   //Mine all strong rules from frequent K-itemsets

5. outputRules(all_Rules);
   // Output all mined rules into the external file, say "Rules"

}

Pseudocode

```
generateCandidate1ItemSet(all_Tuples) {
    Map<Set<String>, Integer> result <- Null;
    for tuple in all_Tuples
        for item in tuple
                Set<String> key = new HashSet<String>();
                key.add(item)
                if result.containsKey(item)
                    countTmp = result.get(key)
                    result.put(key, countTmp+1)
                else
                    result.put(key,1)
    return result
}
```

Note: *all_Tuples* is a set of itemsets. Set<Set<String>>

tuple is one itemset (one row)

Pseudocode

```
generateFrequent1ItemSet(allCandidate1Itemsets, support_count) {
        Map<Set<String>, Integer> result <- Null;
        for itemset in allCandidate1Itemsets
                count = allCandidate1Itemset.get(itemset)
                if count>=support_count
                        result.put(itemset, count)
        return result
}
```

Pseudocode

```
//generate candidate k-itemsets from frequent k-1 (M==k-1) itemset
generateCandidateKItemsets(frequentMItemset, m) {
    Set<Set<String>> = null;
    for itemset1, itemset2 in frequentMItemset
        candidateKItemsets <- generateCandiateItemSetFromTwoSubsets(
            itemset1, itemset2) //function pseudocode is shown in next slide
        // below is using apriori rules to prune the candidates
        if candidateKItemset has infrequent subset //it is your work to finish this function, you can consider if all its k-1 subsets are in frequentMItemset
            //do nothing
        else
            result.add(candidateKItemsets)
            return result
}
```

Pseudocode

```
//Set<String> A = {"1.1", "2.2", "3.1"}
//Set<String> B = {"1.1", "2.2", "4.1"}
//Use A and B to generate candidate 4-Itemset {"1.1", "2.2", "3.1", "4.1"}
generateCandiateItemSetFromTwoSubsets(A, B) {
        result <- null              // initialize the result set
        size <- A.size()            // the size of A or B, here it is 3
        A' <- A                     // copy A to A'
        A.retainAll(B)              // intersection of A and B
        if(A'.size() == size-1) then
          result.addAll(A)          //result = {"1.1", "2.2", "3.1"}
          result.removeAll(B)       //result = {"3.1"}
          result.addAll(B)          //result={"1.1", "2.2", "3.1", "4.1"}
        return result
}
```

Pseudocode

```
generateFrequentKItemSet(allFrequent1Itemsets, sup_count) {
   Map<Integer, Set<Set<String>>> result = NULL
   Set<Set<String>> frequentMItemset = allFrequent1Itemsets.keySet() ;
   m = 1 // initialize m as 1, that is we start from 1-frequent itemsets
   //iteratively execute the program to generate all >1 frequent itemset
   while(!frequentMItemset.isEmpty()) {
       candKItemsets<- generateCandidateKItemsets(frequentMItemset, m)
       frequentMItemset <- NULL     //empty the frequentMItemset
       m <- m+1                     //update m to m+1
       for itemset in candKItemsets
           c<- count the times of itemset which appears in the database
           if c>=sup_count
               result.add(candKItemsets)
               frequentMItemset.add(candKItemsets)
   }
  return result
}
```

Data structures

- You can use data structure of your own choice.
- Some suggestions for data structure for storing data of each transaction and itemsets are:
- HashMap
- HashSet
- KeyValuePair
- List<String>(Collections)
- You can also construct your own class/structure to store data in your desired way.

Interface

- The interface of the system prompts users to specify
  - The dataset to be operated on;
  - The minimum support (*minsup*);
  - The minimum confidence (*minconf*).
  - When mining process finishes, a message should appear to indicate this.
  - You should incorporate checking mechanism for illegal file name and support and confidence values.

  - **For your convenience, you can convert minimum support to minimum support count.**

Some Hints

1. You can find all candidate itemsets and then remove these ones that are not frequent

generateCandidateItemsets → generateFrequentItemsets

1. Generating 1-itemset candidates and 2-itemset candidates (to some extent) are different from K-itemsets
So, you can write these steps in different ways (using IF-THEN or SWITCH-CASE)

A Demo Example: Interface

```
bluenose: ~/demo$ ./Apriori

What is the name of the file containing your data?
 data1

Please select the minimum support rate(0.00-1.00):.25

Please select the minimum confidence rate(0.00-1.00):.5

The result is in the file Rules.
 *** Algorithm Finished ***
```

A Demo Example: Input file format

```
outlook temperature Humidity  Windy  PlayTennis
sunny      hot         high     false    N
sunny      hot         high     true     N
overcast   hot         high     false    P
rain       mild        high     false    P
rain       cool        normal   false    P
rain       cool        normal   true     N
overcast   cool        normal   true     P
sunny      mild        high     false    N
sunny      cool        normal   false    P
rain       mild        normal   false    P
sunny      mild        normal   true     P
overcast   mild        high     true     P
overcast   hot         normal   false    P
rain       mild        high     true     N
```

Hint: These are space-separated, not fixed width!

A Demo Example: Output file format

```
Summary:
Total rows in the original set: 14
Total rules discovered: 236
The selected measures: Support=0.10 Confidence=0.50
----------------------------------------------------------

Rules:

Rule#1: (Support=0.14, Confidence=0.50)
{ temperature=hot }
----> { outlook=sunny }

Rule#2: (Support=0.21, Confidence=0.60)
{ outlook=sunny }
----> { Humidity=high }

…

Rule#236: (Support=0.14, Confidence=0.50)
{ Humidity=normal  Windy=false  PlayTennis=P }
----> { temperature=cool }
```

Hints: Some final advice

- **Start early!**
  - **This is a fairly intensive assignment, even for experienced and confident programmers, so you really don' t want to wait until the night before.**
  - **If you have any doubt/query/problem ,please feel free to ask.**

- Keep the interface simple, such as command-line only, like in the demo program
  - Fancy GUI interfaces won't get you any extra points, so concentrate on the algorithm

- You don't have to use Bluenose to *develop* your program, but it would be a good idea to occasionally *check* that everything you do also works there
  - It must run on Bluenose eventually … you don't want to have to make last minute changes just to get it to compile

- Documentation
  - Make sure you have a complete README file, since this is part of the evaluation

Running Your Code on Bluenose

- In order to run your code for testing on bluenose you can use following commands.
- C family:
- gcc program.cs
- Java

  javac program.java
- Python

  python program.py

Evaluation Components:

- README Documentation
  - Introduction to the code structure / architecture
  - Other instructions / comments to the user
  - Brief Description about bonus part (if attempt)
  - Specify limitations of the program(if any)

- Program Execution
  - User Interface
  - Frequent Itemset generation & performance
  - Rule generation & performance

- Code Design
  - Modularity / Functionality
  - Code readability and comments

- Bonus
  - Search pruning for rule generation and explanation

Action Plan & Participation

- **Ass3 Due:** Feb 22
- **Ass3 Tutorial**: (Feb 16, 6:00-7:30PM, CS 127)
- **Ass3 Help Hours:**

- 13th Feb Mon, 5:30-6:30PM, in CS 233
- 14th Feb Tue, 1:00-2:30PM, CS 233
- 15th Feb Wed, 5:30-6:30PM, in CS 233
- 17th Feb Fri, 1:00-2:00PM, in CS 233

For any queries contact:

Virlla Devi Soothar

vr265712@dal.ca

Good Luck