CS 512 Assignment 1

Daniel Campos

March 5th,2021

1 Problem 1

1.1 List two differences between Frequent Pattern Growth algorithm and Apriori algorithm

Apriori requires K+1 scans of the database while FPGrowth requires 2. The apriori algorithm is a breath first search while FPGrowth is divide and conquer. Apriori requires a large amount of memory and FPGrowth does not.

1.2 Prove all nonempty subsets of a frequent itemset must also be frequent and prove the support of any nonempty subset s' of itemset s must be at least as large as the support of s.

1.2.1 Answer Prove all nonempty subsets of a frequent item list must be frequent

1. Let $S = i_0, i_1, ..., i_n$ be a set of items.
2. Let $R = r_0, r_1, ..., r_m$ be a set of records where each record r_j is a set of items where $\forall k \in r_j$ $r_{jk} \in S$.
3. Let $F = i_0, i_1, ..., i_k$ where each $\forall k \in F$ $F_k \in S$
4. A frequent item set contains items where their occurrence is $>= minsup$ where minsup represents a minimum support threshold where only items that occur at or more than this threshold are in F.
5. Let ss represent possible non empty subsets of F
6. Since every possible $ss \in F$ all items in ss occur $>= minsup$ and therefore all items $\in ss$ must all be frequent as well.
7. As a result, all non empty subsets of a frequent itemset must be frequent.

1.2.2 Prove any nonempty subset s must be at least as large as the support of s

1. Let $S = i_0, i_1, ..., i_n$ be a set of items.
2. Let $R = r_0, r_1, ..., r_m$ be a set of records where each record r_j is a set of items where $\forall k \in r_j$ $r_{jk} \in S$.

3. Given all non-empty subsets of S where $s' = i_1, i_k, ...$ where $s' \in S$.
4. Since the support of a itemset is driven by how often its items occur in R the support of S is that of its least common item.
5. Since all items $i_k \in ss$ are also in S the support for $i_k \in s'$ ¿ = support(S).
6. Since $\forall i_k \in s' > support(S)$ then we know $support(s') \geq support(s)$
7. Thus all non-empty subsets s' must be at least as large as the support of S

1.3 What problem might lift and χ^2 have? Explain why Jaccard function is null-invariant

Lift and X^2

1.3.1 What problems do $Lift$ and χ^2 have

Neither deal way with datasets that have large amounts of null transaction for the variables being compared. In other words if we are studying the correlation between variable A and B and both A and B do not have a balanced occurrence(in other words $\neg A > A$ and $\neg B > B$ both χ^2 and $Lift$ will not be good measures.

1.3.2 Why is Jaccard function null-invariant

The Jaccard function is null invariant because it is only looking at when variables occur and does not use information from where the variables do not occur. To build on our example from part a: Jaccard function looks at A and B and ignores $\neg A$ and $\neg B$. This makes Jaccard null-invariant.

1.4 Under what circumstances will PrefixSpan also have poor performance

Since the major cost in prefixspan is Constructing projected dbs and Suffixes are largely repeating, if the target records include a large quantity of unique items then storing the physical DB in memory can be prohibitive and require projections. In short when the transaction dataset is large and there are a large amount of prefixes(say items on amazon).

2 Problem 2: Apriori vs Frequent Pattern Growth

2.1 Present the intermediate results of the first three scans and the final derived frequent itemsets.

2.1.1 First Scan

See results in table 1

2.1.2 Second Scan

See results in table 2

Itemset	Support
A	4
B	3
C	5
F	4
M	3
P	3

Table 1: Apriori first scan

Itemset	Support
A,C	4
A,M	3
A,P	3
C, F	3
C, M	3
C, P	3

Table 2: Apriori second scan

2.1.3 Third Scan

See results in table 3

2.1.4 Final derived frequent itemsets

See results in table 4

2.2 Frequent Pattern

First off we go ahead and sort our items by frequency giving us the ordering: C(5), A(4), F(4), B(3), M(3), P(3). Using the updated frequency sorted ordering the updated transactional database is found in table 5. Using this updated transaction database we create the pattern tree found in figure 1.

2.3 Association rules

Top 3 association rules and their confidence.
$M \rightarrow A = 1.0$
$A \rightarrow C = 1.0$

Itemset	Support
A, C, M	3
A, C, P	3

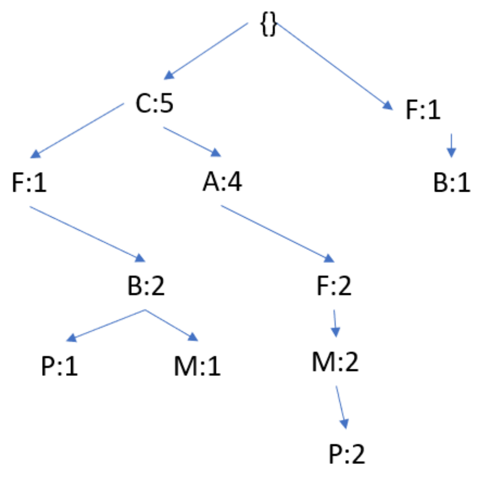
Table 3: Apriori third scan

Itemset	Support
A	4
B	3
C	5
F	4
M	3
P	3
A,C	4
A,M	3
A,P	3
C, F	3
C, M	3
C, P	3
A, C, M	3
A, C, P	3

Table 4: Apriori Derived frequent itemsets

TID	ITEMS
1	C, A, B, P
2	C, A, B, M
3	F, B
4	C, A, F, M, P
5	C, A, F, M, P
6	C, F

Table 5: FPGrowth updated and sorted transaction database



1

Figure 1: Frequent Pattern Tree

Product Types	169
Transactions	9835
Average Products per transactions	4.41
Top 5 most popular products	Whole milk(2513), other vegetables(1903), rolls/buns(1809), soda(1715), yogurt(1372)
Items with single purchase	baby food, sound storage medium

Table 6: Grocery dataset statistics

$P \to C = 1.0$

3 Problem 3: Frequent Pattern Mining Implementation

3.1 Data statistics

3.2 Results

Using the minimum support values of 0.05 and confidence of 0.25 the top 10 frequent item sets:

1. whole milk: 0.26
2. other vegetables:0.19
3. rolls/buns:0.184
4. soda:0.174
5. yogurt:0.140
6. bottled water: 0.11
7. root vegetables: 0.109
8. tropical fruit: 0.11
9. shopping bags: 0.10
10. sausage:0.09

Using the same thresholds the inferred rules are:

1. (whole milk) $\to$ (other vegetables) : 0.29
2. (rolls/buns) $\to$ (whole milk) : 0.29
3. (other vegetables) $\to$ (whole milk) : 0.29
4. (yogurt) $\to$ (whole milk) : 0.40

Item	Yogurt	¬ Yogurt	Total
Whole Milk	551	1962	2513
¬ Whole Milk	821	6501	7322
Total	1372	8463	9835

Table 7: Caption

Transactions	Apriori(ms)	FP-Growth(ms)
500	3.13e-5	1.58e-4
1000	6.37e-5	5.98e-5
2000	1.41e-4	7.90e-5
5000	1.74e-4	6.35e-5
Full	2.51e-4	1.09e-4

Table 8: FP-Growth vs Apriori execution time

3.3 Contingency Table: Yogurt and Whole milk

3.4 Interestingness using Lift

Let A represent Whole milk and B represent yogurt.

$$lift(A, B) = \frac{c(A \to B)}{s(B)} = \frac{s(A \cup B)}{s(A) * s(B)} = \frac{551/9836}{(1372/9835) * (2513/9835)} \quad (1)$$

$$lift(A, B) = \frac{0.05601871}{(0.13950178)(0.25551601)} = \frac{0.05601871}{0.0356449382134978} = 1.5715754552433814 = 1.57 \quad (2)$$

Since the Lift of A and B is over one they are positively correlated.

$$lift(A, \neg B) = \frac{c(A \to B)}{s(B)} = \frac{s(A \cup B)}{s(A) * s(B)} = \frac{1962/9836}{(8463/9835) * (2513/9835)} \quad (3)$$

$$lift(A, \neg B) = \frac{0.1994713298088654}{(0.8604982206405694)(0.25551601)} = \frac{0.1994713298088654}{0.21987107195017794} = 0.9072195266054142 = 0.91 \quad (4)$$

Since the lift of A and $\neg B$ is under one they are negatively correlated.

3.5 Run time

My run times comparing our implementation in of Apriori in python vs FP-Growth(via pyfpgrowth)

SID	Sequence
1	$\langle (_c f)(ac)b \rangle$
2	$\langle (_{)}c(bc)ab(ce) \rangle$
3	$\langle _{(}acf)acd \rangle$
4	$\langle (_c)(ae) \rangle$
5	$\langle (_d)df(ab) \rangle$
6	$\langle (_c)eab(cd)f(abd) \rangle$

Table 9: Projection for $\langle ab \rangle$

SID	Sequence
1	$\langle \rangle$
2	$\langle c(bc)ab(ce) \rangle$
3	$\langle \rangle$
4	$\langle (_c)(ae) \rangle$
5	$\langle (_d)df(ab) \rangle$
6	$\langle (_c)eab(cd)f(abd) \rangle$

Table 10: Projection for $\langle (ab) \rangle$

4 Problem 4: Sequential and Graph Pattern Mining

4.1 Sequential database

4.1.1 Projections

for $\langle ab \rangle$ see table 9, for $\langle (ab) \rangle$ see table 10, for $\langle de \rangle$ see table 11, for $\langle abc \rangle$ see table 12, for $\langle acb \rangle$ see table 13,for $\langle abca \rangle$ see table 14

4.1.2 Sequential Patterns

Sequential patterns where support = 3 and length ≥ 3 aca, abc, acd, dea, eab, bca, abd, afa, bce, add

SID	Sequence
1	$\langle \rangle$
2	$\langle \rangle$
3	$\langle _{(}af)acb(acf)acd \rangle$
4	$\langle _{(}abc)(ae) \rangle$
5	$\langle \rangle$
6	$\langle \rangle$

Table 11: Projection for $\langle de \rangle$

SID	Sequence
1	$\langle (_f)(ac)b \rangle$
2	$\langle _(bc)ab(ce) \rangle$
3	$\langle (_f)acd \rangle$
4	$\langle (_)(ae) \rangle$
5	$\langle \rangle$
6	$\langle eab(cd)f(abd) \rangle$

Table 12: Projection for $\langle abc \rangle$

SID	Sequence
1	$\langle \rangle$
2	$\langle (_c)ab(ce) \rangle$
3	$\langle (acf)acd \rangle$
4	$\langle \rangle$
5	$\langle (_d)df(ab) \rangle$
6	$\langle (cd)f(abd) \rangle$

Table 13: Projection for $\langle acb \rangle$

SID	Sequence
1	$\langle_\rangle$
2	$\langle _b(ce) \rangle$
3	$\langle (_cf)acd \rangle$
4	$\langle \rangle$
5	$\langle (_b) \rangle$
6	$\langle (_bd) \rangle$

Table 14: Projection for $\langle acba \rangle$

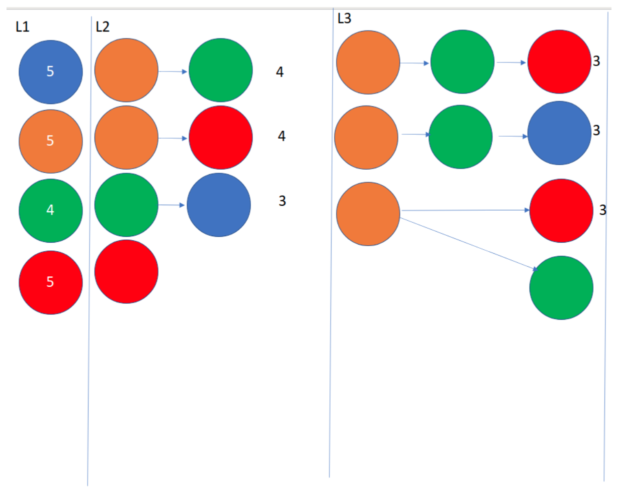


Figure 2: Graph sequence apriori mining

4.2 Graph mining

4.2.1 Support

P1 support is $\frac{4}{5}$ and is supported by G2, G3, G4, and G5.
P2 support is $\frac{3}{5}$ and is supported by G1, G2, and G5.
P3 support is $\frac{3}{5}$ and is supported by G1, G3 and G5.

4.2.2 Apriori-based

see figure 2

5 Problem 5: SVM, RBF Kernel

5.1 Prove all eigenvalues are non negative

We know from other work that every semidefinite matrix only has non negative eigenvalues. For a matrix to be semi-definite $x^T Ax \geq 0$ for all $x \in \mathbf{R}^N$. Since we know that $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$ to ensure semi-definite we ensure A ¿ 0. Since A is the transform its always ¿ 0.

5.2 Prove RBF Kernel is the sum of infinite polynomial kernels

A kernel is any function in the form $K(x, y) = \langle \psi(x), \psi(y) \rangle$ where ψ projects x into a new vector space.
We are trying to prove that $\psi_{RBF} : \mathbf{R}^N \to \mathbf{R}^\infty$.

WLOG we set $\gamma = 1$

$$\begin{aligned} K_{RBF}(x,y) &= exp[-1|x-y|^2] \\ &= exp[-1\langle x-y, x-y\rangle] \\ &= exp[-1(\langle x, x-y\rangle - \langle y, x-y\rangle)] \\ &= exp[-1(\langle x,x\rangle - \langle x,y\rangle - \langle y,x\rangle + \langle y,y\rangle)] \\ &= exp[-1(|x|^2 + |y|^2 - 2\langle x,y\rangle)] \\ &= exp[-1|x|^2 - 1|y|^2]exp[-1 - 2\langle x,y\rangle] \end{aligned}$$

$$c := exp[-1|x|^2 - 1|y|^2] \tag{5}$$

$$\begin{aligned} K_{RBF}(x,y) &= c * e^{x,y\rangle} \\ &= c\sum_{n=0}^{\infty} \frac{(\langle x,y\rangle)^n}{n!} \text{via taylor expansion of } e^x \\ &= c\sum_{n=0}^{\infty} \frac{K_{poly(n)}(x,y)}{n!} \end{aligned}$$

Thus we see that RBF is an infinite sum over poly kernels.

5.3 Decision Boundary

If we were to use a large γ we would likely be able to separate the training data nearly perfectly. We would be over fitting to training but could likely produce results seen in figure 3

5.4 Effect of large γ

If we are to make the γ infinitely large we will completely fit the SVM to the training data and overfit the model. Our training accuracy may reach 100% but our transfer accuracy will suffer. This is because larger RBF kernel bandwidths(which are smaller γ values produce a smoother feature space mapping. As we scale our γ value we produce a less smooth kernel which will allow us to learn harder decision boundaries but will overfit to the decision boundary of the training corpus.

6 Problem 6:Feature Selection

6.1 Classification Results

On the test portion accuracy:0.68, precision:0.76, recall:0.38, F1 score:0.51. More detailed breakdown can be found in table 15

6.2 Fisher Score Feature Selection

Using Fisher Score calculation and a threshold of ≥ 1 we find that only columns [8, 11, 18, 26] pass the threshold. We then remove all the other indexes and retrain our svm. Performance on

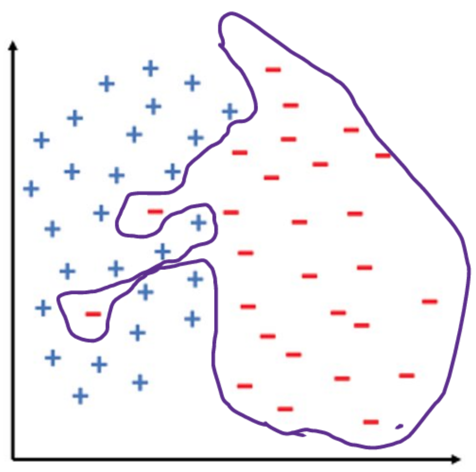


Figure 2: Distribution of Training Examples.

Figure 3: Decision Boundary using high γ

Reference Label	Predicted 0	Predict 1
0	60	6
1	31	19

Table 15: SVM

Reference Label	Predicted 0	Predict 1
0	61	5
1	36	14

Table 16: Fisher Score Augmented SVM

Reference Label	Predicted 0	Predict 1
0	60	6
1	29	21

Table 17: Self-Training Augmented SVM

train goes from 0.81 to 0.94 and performance on test goes from 0.68 to 0.84! On the test portion accuracy:0.84, precision:0.88, recall:0.72, F1 score:0.79 . More detailed breakdown can be found in table 16

7 Problem 7:Self Training

Using self training we first train an svm based on the regular training corpus(non fisher score adjusted). Then, we label the unlabeled corpus and only keep data where confidence is ≥ 0.6. By that we mean there is over a 0.6 difference between the confidence scores and we set the winning label to the self-trained label. Using this threshold of training we include 322 inferred label samples in our training data.
Using this amplified training data we find performance improvements on non Fisher-score optimized models and fisher score optimized models. Without using fisher-score optimization the introduction of self-trained data improves F1 score from 0.51 to 0.55, recall goes from 0.38 to 0.42, precision goes from 0.76 to 0.78. More details available on self training model performance are available in table 17. Using fisher score optimization the amplified training data improves F1 from 0.79 to 0.80, recall goes from 0.72 to 0.74, precision goes from 0.88 to 0.86. More details about fisher+self training model performance are available in table 18

8 Random Walk with Restart

8.1 Capture multiple weighted relationships

Starting with the formula $rc * A * r + (1 - c) * e$ from the textbook(chapter 8 page 449) we can see that the RWR leverages information in the adjacency matrix but since the e term is kept constant this modifier allows consistent steering of r. In doing so we can ensure that we keep tweaking r to have the initial value for the query seed which allows for multiple path explorations.

Reference Label	Predicted 0	Predict 1
0	60	6
1	13	37

Table 18: Fisher Score and Self Training Augmented SVM

8.2 RWR implemented

Top 10 nodes sorted by importance starting with a seed node of 42: 160, 121, 82, 107, 62, 434, 249, 183, 86, 166