Sequential Pattern Discovery using Equivalence Classes Data Mining (CSF415)

S.Snehanjali Reddy I.D.No.: -2016A8PS0360H

Email:-f20160360@hyderabad.bits-pilani.ac.in

***Abstract* — Today, a strong online presence is proven to be the key for the success of any organisation. Having a good website helps in interacting with potential new customers and old customers alike. So it is important to have a website and optimise it according to user preferences to make the most out of it. Web data analytics is a widely used technique to optimise content of the website and improve the performance. Sequential Pattern mining is a widely used technique to understand the patterns of browsing among users of a website. Understanding the patterns further helps in optimizing the website. In this paper, we analyse the browsing data of news website MSNBC. The algorithm Sequential Pattern Mining using Equivalence Classes is used to find frequent patterns and rules in the websites browsing data. Gaining insights on this will help the website to concentrate more on the content in frequently visited pages and also make some important business decisions.At the end , results obtained after applying the algorithm and the shortcomings are also discussed.**

Keywords - Sequential Pattern Mining, frequent patterns, web analytics.

I. INTRODUCTION

In the age of the digital world, for any company or business it is important to have a website and a strong online presence to have a lasting effect on its customers and also on the market. It also helps greatly in leveraging the business as most of the decisions regarding customer purchases are taken after a search engine search. Moreover, having a website exhibits professionalism and builds a level of trust with new customers.

While having a website is a positive start, constantly analysing user data and drawing meaningful interpretations from it can help businesses make important decisions regarding policies and change/adapt goals accordingly. Most of the information today comes from the web and the amount of information coming is growing exponentially. So it is also important to use efficient data analysis/mining techniques.

Web data mining is the process of employing techniques which automatically discover and extract useful information from the web. It is mainly used to get useful information from the World Wide Web and its usage patterns. There are three main types of web data mining:

- Web Content Mining : It is the extraction of useful insights from the web documents and content. Web content has text, images, vedios ,audio files etc. These methods provide useful and interesting insights into the user's needs.
- Web Structure Mining : Used to extract structure information from the web. In web graphs , web pages are represented as nodes and hyperlinks as the edges connecting different nodes.This type of mining techniques show the structure summary of any particular website. They come in very handy when analysing the relationships between any two commercial websites.
- Web Usage Mining : This application identifies interesting usage patterns from large data sets which in turn help us understand user behavior. It is also called log mining as data is usually collected in the form of logs.

In this paper, we wish to analyse the page visits of users of a particular website. This comes under Web Usage Mining technique.

Data used for this is the page visit data of users on the news section of msn.com website. It was obtained from the Internet Information Server (IIS) logs for msnbc.com and news-related portions of msn.com for the entire day of September, 28, 1999. Each sequence in the dataset corresponds to page views of a user during that twenty-four hour period. Each event in the sequence corresponds to a user's request for a page. Requests are not recorded at the finest level of detail---that is, at the level of URL, but rather, they are recorded at the level of page category. The categories are "frontpage", "news", "tech", "local", "opinion", "on-air", "misc", "weather", "health", "living", "business", "sports", "summary",”bulletin board service”, "travel", "msn-news", and "msn-sports".

A total of 989818 sequences corresponding to the user's request were recorded in a single day. Each category is associated--in order--with an integer starting with “1” and going till “17” with the order of categories being : ‘frontpage’ ‘news’ ‘tech’ ‘local’ ‘opinion’ ‘on-air’ ‘misc’ ‘weather’ ‘msn-news’ ‘health’ ‘living’ ‘business’ ‘msn-sports’ ‘sports’ ‘summary’ ‘bbs’ ‘travel’. For example if a sequence in the data set is as follows:

3 4 6

It means that the user first visited the ‘tech’ page then the ‘local’ page and then the ‘on-air’ page.

In order to identify frequent subsequences and frequent rules out of them we employ the Sequential Pattern Discovery using Equivalence Classes(SPADE) algorithm. Sequential pattern mining is essentially finding time-related or event based frequent patterns (frequent subsequences). A sequence is an ordered list of elements. Each element is a collection of one or more events. A sequence can be characterized by its length and number of occurring events. Length of sequence corresponds to the number of elements present in the sequence. k-sequence is a sequence that contains k events. The formal definition of a subsequence is as follows: If a sequence <a1 a2 … an> is contained in another sequence <b1 b2 …bm> (m ≥ n) if there exist integers i1 < i2 < … < in such that a1 ⊆ bi1 , a2 ⊆ bi2, …, an ⊆ bin. Then <a1 a2 … an> is a subsequence of <b1 b2 …bm>. Say D is a dataset that contains one or more data sequences. Data sequence refers to an ordered list of events associated with a single data object. The support of a subsequence w is defined as the fraction of data sequences that contain . A sequential pattern is a frequent subsequence i.e., a subsequence whose support is ≥ minsup. The goal of sequential pattern discovery is to find all the subsequences with minsup>mentioned threshold. SPADE adopts a candidate generate- and-test approach using vertical data format where the data is represented as ⟨itemset : (sequence ID, event ID)⟩).

I. DATA PREPROCESSING

Data used for processing is in a file called "msnbc990928.seq". This file is taken as the input file and each line is read individually. Each line in this dataset represents a sequence of pages visited by every user. These sequences are converted into a list of tuples according to the vertical Data format which is required for the SPADE algorithm.

Example:

Original sequences- 3 4 6

6 2

Vertical Data Format-

SequenceID	EventID	Sequence
1	1	3
1	2	4
1	3	6
2	1	6
2	2	2

The list of tuples generated is further used for analysis and is shown below:

```
In [14]: # This is the list of tuples in the order: sequenceID,EventID ,sequence
         sequences

Out[14]: [(1, 1, '1'),
          (1, 2, '1'),
          (2, 1, '2'),
          (3, 1, '3'),
          (3, 2, '2'),
          (3, 3, '2'),
          (3, 4, '4'),
          (3, 5, '2'),
```

II. CODE

After preparing the data for analysis, the list of tuples having all the sequences is loaded into an object of default dictionary from the collections module. The advantage of using this default dictionary is that it handles missing key values effectively and does not throw a KeyError.

```
import collections

class _KeyDefaultDict(collections.defaultdict):
    def __missing__(self,key):
        if self.default_factory is None:
            raise KeyError(key)
        else:
            ret = self[key] = self.default_factory(key)
            return ret
```

```
elements = _KeyDefaultDict(Element)
for sid,eid,itemset in sequences:
    for item in itemset:
        elements[tuple(item)] |= Element(tuple(item),Event(sid=sid,eid=eid))
```

After converting, this default dictionary is passed into a function which returns all the frequent on element subsequences. Also, the threshold is given as the number of times a sequence has to occur in order for it to be frequent.Here, the threshold value is given as 494909 which is 50% of the total sequences. Hence the support here is 0.5.

```
def subset(elements,support_threshold):
    subset_dict = _KeyDefaultDict(Element)
    for element_name,element in elements.items():
        support = len(set([event.sid for event in element.events]))
        if support >= support_threshold:
            subset_dict[element_name] = element
    return subset_dict
```

```
#returns all single element subsequences which meet the threshold
#threshold is given in terms of number of occurances(eg: 494909 is 50% of 989818) so supp = 0.5
elements = subset(elements,494909)
```

After getting the single element frequent sub sequences we proceed on to get the frequent subsequences of length two. For this we pass the obtained one element subsequences to a function which returns a dictionary of two sequences as keys with frequency of each two-sequence as value.

```
def frequent_two_seq(elements,support_threshold):

    '''First converting the  dictionary of elements into horizontal form to be able to count the
    frequency of two-item sequences'''

    horizontal_db = {}
    for element_name,element in elements.items():
        for event in element.events:
            if event.sid not in horizontal_db:
                horizontal_db[event.sid] = []
            horizontal_db[event.sid].append((element_name,event.eid))

    # create counts using horizontal_db
    count = collections.defaultdict(int)

    for sid,seq in horizontal_db.items():
        for event_index_i,event_i in enumerate(seq):
            for event_index_j,event_j in enumerate(seq[event_index_i+1:]):
                if event_i[1] <= event_j[1]:
                    two_seq = event_i[0]+event_j[0]
                else:
                    two_seq = event_j[0]+event_i[0]

                counts[two_seq] += 1

    return {tuple(sorted(two_seq)) for two_seq,c in count.items() if c  >= support_threshold}
```

```
#returns all the two-element subsequences which meet the threshold

freq_elements_len_eq_2 = frequent_two_seq(elements,494909)
```

For sup =0.5 there was only one element in the dictionary.

In order to find candidate 2-sequences, all pairs of single items are joined if they are frequent, share the same sequenceID and their eventIDs follow a sequential ordering.The first item in the pair must occur as an event before the second item, where both occur in the same sequence. For this purpose a function is written.(temporal_join)

```
for two_seq in freq_elements_len_eq_2:
    R = temporal_join(elements[tuple(two_seq[0])],elements[tuple(two_seq[1])])
    for seq,element in R.items():
        support = len(set([event.sid for event in element.events]))
        if support >= support_threshold:
            elements_len_eq_2[seq] |= element
```

III. Results and Shortcomings

For the code written, I was able to generate two element length sub sequences, but I could not join them in accordance with their sequenceID and EventID. This is mainly because of the huge size of the data which in turn made the code run for a very long time and yet not come out of the function written to join the elements. Also, for sup = 0.5 only one frequent subsequence having two elements was obtained.

```
freq_elements_len_eq_2
```

```
{('1', '1')}
```

One of the major issues faced while implementing the algorithm was the lack of proper resources online/offline to read about it and the no resources were available in any programming language to implement it. As a result of which, it was difficult to implement the complex pseudo code in the original research paper in python. As of now, only one library is available to implement it in python(pycspade) which is difficult to run on conventional systems as it requires other tools. Also, due to the large dataset(close to one million sequences), I found it computationally difficult to implement the algorithm in a feasible time.

IV. Conclusion

Though Sequential Pattern Mining is an effective method to analyse web usage data, it is computationally expensive due to the multiple scans of the huge dataset which have to be conducted to find the frequency and to check and join the obtained frequent subsequences. Also, the given algorithm SPADE is a relatively new one and hence there was a difficulty in obtaining good resources to understand and implement it completely.

V. References

https://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data

https://pypi.org/project/pycspade/

Class Slides