FINDING FREQUENT ITEMSETS AND EXTRACTING ASSOCIATION RULES IN TEXTUAL DOCUMENTS: AN IMPLEMENTATION ON OLD NEWSPAPERS

Ozlu Dolma - 942512

October 27, 2022

1 Introduction

The purpose of this study to implement the market-basket analysis on a large amount of textual data (i.e. sentences or paragraphs) retrieved from an old newspaper corpus in order to find itemsets (i.e. words) that appear frequently in these documents. Association rules between the itemsets were also investigated. The SON algorithm (Savasere et al., 1995) in conjunction with a MapReduce program was applied to accomplish these tasks. The results obtained using this algorithm were compared against the results when the Parallel FP-growth algorithm is applied on the same dataset. The structure of the study is as follows: First, the theoretical framework is presented very briefly in order to describe the main mechanisms underlying the method adopted. Next, an overview of the dataset is provided and the steps of data preprocessing is explained. Then, the methodology and the experimental results are presented and discussed.

2 Theoretical Framework

The market-basket analysis, also called frequent-itemset mining, was originally proposed to determine the items that are commonly bought together by customers in shopping (Agrawal et al., 1993). In this context, the “items” are the different products that the store sells, and the “baskets” are the sets of items in a single market basket (physical shopping cart) (Rajaraman et al., 2014). If a set of items appears in many baskets, it is said to be “frequent.” More formally, if a set of items has a “support” value, which is the number of baskets for which that itemset is a subset, that is no less than a predetermined support threshold, then that itemset is said to be frequent. Frequent itemsets can be determined as singletons, doubletons, triples, quadruples or larger sets.

The A-Priori algorithm is one of the fundamental algorithms for finding frequent itemsets. It was introduced by Agrawal et al. (1993), and a year later, Agrawal and Srikant (1994) developed two algorithms (Apriori and AprioriTid, AprioriHybrid) to address the problem of finding frequent itemsets. The A-Priori algorithm performs two passes over the data. In its first pass, it simply counts the occurrences of itemsets of size k (e.g. k=1, i.e. all singleton itemsets) and collects those itemsets that satisfy the minimum support requirement in the set L_K of frequent itemsets. The next pass is composed of two phases. First, the frequent itemsets found in the previous pass are used to generate the candidate itemsets C_{k+1}, which is a superset of the sets of those frequent itemsets, that is, the candidate itemsets of size k+1 (e.g. candidate pairs generated using singletons). Then, in the second pass, the algorithm counts all the candidate itemsets C_{k+1} and determine the ones which appear no less than the minimum support threshold. These filtered itemsets form the itemsets L_{K+1}, which are the frequent itemsets of size k+1 (e.g. frequent pairs). This process can be repeated until no new large itemsets are found.

In their study, Agrawal and Srikant (1994) actually presented and elaborated the problem of discovering association rules between sets of items in a large database of transactions that have support and confidence greater than the user-specified minimum support and minimum confidence, respectively. They explained that such an association rule can be, for instance, a statement that "90 % of transactions that purchase bread and butter also purchase milk" and the "antecedent" of this rule consists of bread and butter and the "consequent" consists of milk alone. The number 90 % is the confidence factor of the rule (Agrawal et al., 1993).

In order to better understand the usefulness of an association rule, besides the confidence value, one also needs to take into account the "interest" of the rule. If an item such as a plastic bag that happens to be frequently bought in order to bring out the elements in the basket is a consequence of a rule, then the confidence of the rule will always be high since the plastic bag is purchased regardless of the rest of the content of the basket. In such a case, the confidence of the rule alone would not be very informative. An association rule is useful if it reflects a true relationship, where the item or items on the antecedent side somehow affect the item on the consequent side (Rajaraman et al., 2014). The interest of an association rule from the itemset I to the item j is defined as the difference between the confidence of the rule and the fraction of baskets that contain the item j. If an association rule has an interest of 0, then it means that the fraction of baskets including I that contain j would be exactly the same as the fraction of all baskets that contain j, that is, the itemset I has no influence on the item j. Furthermore, if a rule has a high interest value, it would mean that the presence of the itemset I in a basket somehow causes the presence of the item j. On the other hand, if a rule has a low interest value it would mean that the presence of the itemset I discourages the presence of the item j.

Frequent-itemset analysis is not only applied to mine market-basket data. It can be also applied in text-mining where the basket can be a textual document and the items can be the words or the tokens that compose that document. The purpose of frequent-itemset analysis, in this context, can be to discover the most frequent words in that set of documents which may reveal the topic that the documents can be associated to or to find word sets that appear frequently together and represent a common concept. In fact, in previous studies, frequent itemsets mining was applied to discover association rules for text categorization and to conduct text clustering (e.g. Fung et al., 2003, Zhang et al., 2010).

3 An Overview of the Dataset

In this study, the "baskets" are the newspaper texts (sentences or paragraphs from the news articles) and the "items" are the words. Accordingly, each newspaper text is composed of a set of tokens, which in general is called an itemset.

The dataset was retrieved from Kaggle datasets repository. It is titled by Kaggle dataset owners as "Old Newspapers" and released under the CC0 public domain license.[1] They published a cleaned version of the raw data from newspaper subset of the HC corpus[2], which is a resource that contains natural language text from various newspapers, social media posts and blog pages in multiple languages. The features of the dataset are: (1) the language of the text, (2) the source, which is the newspaper from which the text was retrieved, (3) the date of the article that contains the text, (4) the text, which is the sentence or the paragraph from the newspaper. Originally, the corpus contains 16,806,041 sentences or paragraphs in 67 languages.

4 Data Preprocessing

In the current study, only the text in English language were analyzed. There were 1,010,484 lines of text in English from various newspapers. In order to avoid the technical problems encountered in terms of the memory space and to have a reasonable running time, a random subset of it consisting of 505,242 lines of text (half of the total dataset) was analyzed. Before the analysis a sequence of data preprocessing was applied. First, all the punctuation was removed from the documents. Then, all the tokens were converted to lowercase. Next, each line of the text was split in a way that they are converted to lists of tokens. Then, all the numeric values were removed. To avoid the fact that the documents will be dominated by the stop words and may confound the analysis, the stop words were removed using NLTK (Natural Language Toolkit)[3]. Next, each line of text was purified by removing the duplicate tokens. Last, the final data frame called "baskets" was created by selecting the columns "Source", "Date", and "Text".

5 Methodology

The analysis was conducted using PySpark (Python on Spark). It is an interface for Apache Spark in Python. It allows writing Spark applications using Python APIs and also provides the PySpark shell for interactively analyzing the data in a distributed environment[4]. Once a dataset is created one can apply parallel operations on it. The central data abstraction of Spark is called the resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel[5]. Spark also allows any RDD to be divided into chunks,

[1] https://www.kaggle.com/datasets/alvations/old-newspapers
[2] https://web.archive.org/web/20161021044006/http://corpora.heliohost.org/
[3] https://www.nltk.org/index.html
[4] https://spark.apache.org/docs/latest/api/python/
[5] https://spark.apache.org/docs/latest/rdd-programming-guide.html

which it calls splits. Each split can be given to a different compute node, and the transformation on that RDD can be performed in parallel on each of the splits (Rajaraman et al., 2014).

There are two types of operations that RDDs support. These are (i) the transformations, which create a new dataset from an existing one, and (ii) the actions, which return a value to the driver program after running a computation on the dataset. For instance, “map” is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, “reduce” is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program. All transformations in Spark are ”lazy”, in that they do not compute their results immediately. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, a dataset created through map will be used in a reduce and only the result of the reduce to the driver will be returned, rather than the larger mapped dataset[6].

In the current study, the algorithm of Savasere, Omiecinski, and Navathe (1995) together with Map and Reduce operations was used to determine the frequent itemsets. The name of the algorithm is abbreviated as the SON Algorithm. It enables the implementation of Map and Reduce, thus the input data file can be divided into chunks of a resilient distributed dataset. The support threshold, however, needs to be adjusted accordingly. Thus, for each chunk, the support threshold is computed as the global support threshold divided by the number of chunks. Then, an A-Priori type algorithm used to find frequent itemsets is run on each chunk in parallel. Two Map and Reduce steps were performed. These sequence of steps completed in order to find frequent doubleton itemsets can be summarized as follows:

In the first MapReduce step, separately for each chunk of the dataset, candidate itemsets are found. In the beginning of the Map phase, all singleton items, that is, the items themselves are considered as candidate items to be frequent. During the first pass over the chunk, the counts of each singleton (word or token) are compared against the adjusted support threshold and if the count of a singleton is no less than this threshold then that singleton is saved. Then, a second pass is performed to determine frequent doubletons (i.e. word pairs). During the second pass over the chunk, all the pairs that are composed of two frequent singletons found in the first pass, are counted. In particular, all possible pairs of frequent words were generated and then, for each pair, the frequency in the chunk was counted and stored. The adjusted support threshold as in the first step was applied and if the count of a doubleton is equal to or greater than that support threshold then the doubleton itemset is filtered as a candidate frequent pair of words. Once all the chunks are processed in this way, the union of all the itemsets that have been found frequent for one or more chunks is recorded. The output of this step is a set of key-value pairs (F, V), where F is a frequent itemset from a chunk and the value is irrelevant. In the Reduce phase, then, these key-value pairs are processed in a way that the values are

[6]https://spark.apache.org/docs/latest/rdd-programming-guide.html

totally ignored and only the keys (itemsets) are retained. This is output of this first MapReduce step and it can be called the candidate itemsets.

An itemset that is frequent in the whole but not in the chunk is a false negative, while an itemset that is frequent in the chunk but not in the whole is a false positive (Rajaraman et al., 2014). There are no false negatives but there can be false positives which however will be eliminated in the next Map and Reduce step. There cannot be false negatives because an itemset that is frequent in the whole dataset is frequent in at least one chunk and thus no truly frequent itemset is missed. Since, in the end, there are neither false positives nor false negatives, it can be concluded that the SON algorithm is an exact algorithm.

In the Map phase of the second MapReduce step, all the candidate itemsets that were determined in the output of the first Map and Reduce step are taken as input and again, separately for each chunk, the number of occurrences of each of the candidate itemsets are counted. The output is a set of key-value pairs, where the keys are the candidate itemsets and the values are the frequencies of those itemsets in the chunks. In the Reduce phase, then, for each candidate itemset, the counts of occurrences in all the chunks are summed and the itemsets for which the total support value is equal to or greater than the global support threshold are output as the truly frequent itemsets.

Since in this study, the PySpark code to run the SON algorithm was written from scratch, the veracity of the results obtained using this algorithm was checked against the results obtained using the parallel FP-Growth algorithm already provided by Apache Spark's in the Machine Learning Library (MLlib) in the Frequent Pattern Mining section. The FP-growth is a frequent-pattern-tree-based mining method developed by Han et al. (2000). In their study, the developers of this method explain that the FP-growth is an efficient method, in which the costly generation of a large number of candidate sets is avoided. The large database is compressed into a condensed, smaller data structure and a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks (Han et al., 2000). In spark.mllib, a parallel FP-growth algorithm, called PFP, is provided, which is described in Li et al., (2008). In their study, Li et al., (2008) summarize the FP-growth algorithm as follows:

> "FP-Growth works in a divide and conquer way. It requires two scans on the database. FP-Growth first computes a list of frequent items sorted by frequency in descending order(F-List) during its first database scan. In its second scan, the database is compressed into a FP-tree. Then FP-Growth starts to mine the FP-tree for each item whose support is larger than the support threshold by recursively building its conditional FP-tree. The algorithm performs mining recursively on FP-tree. The problem of finding frequent itemsets is converted to constructing and searching trees recursively."

Li et al., (2008) developed a parallelized version of the FP-growth (PFP) in order to address the problems of memory usage and computational cost when large datasets are processed. They explain that the PFP partitions computation in such a way that each machine executes an independent group of mining tasks and that such partitioning eliminates computational dependencies between machines, and thereby communication between them.

When Apache Spark's PFP-Growth algorithm is run, besides the minimum support, it is possible to specify the minimum confidence for generating the association rules from frequent itemsets. Once association rules are extracted as antecedents and consequences, along with the support and the confidence values, it also provides a "lift" value, which is defined as a measure of how well the antecedent predicts the consequent. It is computed as the ratio between the support of the itemset composed of the antecedent and the consequent and the support of the antecedent multiplied by the support of the consequent. It can be also defined as the ratio of the confidence of a rule and the support of the consequent[7]. It quantifies the predictive power of the association rule and the rules that have a lift value greater than 1 are significant (Ordonez, 2006).

6 Experimental Results

In this study, the support threshold was set as 1% of the number of baskets. The baskets were partitioned into 5 chunks for parallel computing. In the search for frequent singletons, the first MapReduce phase resulted in 227 frequent singleton itemsets, but in the second Map Reduce phase, the number of frequent singleton itemsets reduced to 216. This means that there were 11 false positives, which were eliminated in the second MapReduce phase. These false positive items were "working", "thing", "others", "number', "news", "north", "Washington", "trying", "Sunday", "tax", "information".

Frequent singletons and their associated support values are presented in Table 1. One item which was not a token per se ("—") but was among the frequent singletons was not included in this list of frequent items. As presented in Table 1, the most frequent singleton is the token "said". Since the texts were taken from old newspapers, it is not surprising to discover a word describing a reporting action as a frequently appearing item. Other frequent singletons are the tokens "one", "new", "would", "also", "two", and "year", which have a support value higher than 5%. In the borderline, where the occurrence frequency is barely above the threshold, there are the tokens "schools", "care", "line", "support", "far", "service", and "Cleveland", just to name some.

An overall view of the most frequent singletons was created using a word cloud. A word cloud is a visualization technique for text data in which the most frequent words are shown in a bigger font size. The word cloud was generated using the Python's built-in library called "WordCloud". As seen in Figure 1, "new", "two", "one", "year", "last", "said", "say", "time", "state", "first", "three", "county", "people", "school", "way", and "game" are some of the most frequently occurring words in the analyzed newspapers texts.

The search for frequent pairs resulted in 18 truly frequent pairs. As it was the case for the discovery of frequent singletons, there was one pair ("said", "two") output by the

[7]https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html

Table 1: **Frequent Singletons and Their Support Values**

Item	Support	Item	Support	Item	Support	Item	Support	Item	Support
said	113950	police	11615	need	7992	set	6567	call	5738
one	37839	season	11598	never	7986	though	6530	enough	5729
new	31160	go	11575	five	7983	went	6510	always	5726
would	30210	since	11501	former	7969	used	6482	plan	5717
also	28148	say	11122	end	7934	coach	6453	april	5683
two	26465	think	10988	left	7796	early	6437	third	5669
year	25689	another	10706	night	7767	program	6404	making	5648
last	24415	know	10705	place	7698	ago	6395	theres	5628
first	24084	week	10356	great	7674	point	6394	yearold	5599
years	23824	see	10343	officials	7567	government	6392	ohio	5598
time	23802	public	10224	show	7505	friday	6380	york	5593
like	22345	next	10218	better	7410	times	6375	asked	5591
state	21202	st	10201	im	7384	job	6332	bill	5585
people	21083	right	10187	today	7366	health	6291	wednesday	5583
get	19613	thats	10145	group	7286	board	6288	street	5545
us	18918	four	10029	business	7282	department	6264	road	5539
could	18558	including	9868	found	7241	past	6262	find	5534
city	15950	high	9803	put	7223	month	6245	whether	5525
back	15673	want	9756	came	7198	university	6230	free	5480
three	15593	pm	9750	states	7148	look	6219	saturday	5459
make	15207	got	9653	court	7136	tuesday	6177	park	5446
says	14894	around	9428	world	7110	district	6153	give	5423
even	14480	play	9222	days	7088	director	6127	san	5418
many	14250	second	9201	life	7065	students	6109	less	5409
school	14095	center	9158	games	7052	area	6064	young	5381
going	13698	big	9103	something	7031	away	6057	along	5365
game	13683	president	9079	called	7020	months	6044	getting	5364
good	13465	told	9046	office	7014	among	6039	small	5363
home	13464	part	9033	several	6974	thursday	6028	john	5359
way	13396	best	9027	law	6942	local	5980	doesnt	5346
made	13278	according	8808	hes	6905	keep	5979	hit	5311
may	13185	house	8666	national	6883	six	5965	players	5279
day	13139	company	8630	took	6879	open	5964	already	5270
county	12709	help	8590	might	6848	win	5935	report	5262
much	12661	really	8577	run	6810	monday	5906	points	5256
still	12533	lot	8557	use	6794	least	5893	country	5228
work	12304	little	8540	things	6767	american	5877	cleveland	5185
well	12245	family	8527	federal	6671	start	5870	service	5124
team	12161	long	8463	top	6663	pay	5860	far	5089
percent	11828	come	8460	later	6650	th	5859	support	5088
take	11752	every	8367	without	6605	members	5846	line	5088
dont	11727	money	8214	man	6579	children	5838	care	5068
million	11681	didnt	8031	case	6569	community	5811	schools	5054

algorithm in the first MapReduce phase, which was a false positive pair. It was eliminated in the second MapReduce phase. Frequent pairs are displayed in Table 2. Most frequent pair was ("said", "would") with a support of 10275.

In determining the association rules between frequent pairs, no minimum confidence value was specified. Confidence and interest values were computed[8], which are presented in Table 3. As mentioned, a high confidence value per se is not sufficient to label an association rule as "useful". To be able to define a true causal relationship between an antecedent and consequent, the interest value needs to be high, as well. In Table 3, two possible relationships between frequent pairs and their associated confidence and interest values are presented as "1. Rule" and "2. Rule", respectively. As seen, in general, both values for associations were quite low. Only for the rule "york $\rightarrow$ new" both values were high. For this rule, the confidence and interest values were 0.97 and 0.91, respectively. This indicates that the existence of the word "york"

[8]The confidence and interest values were computed in Microsoft Excel.

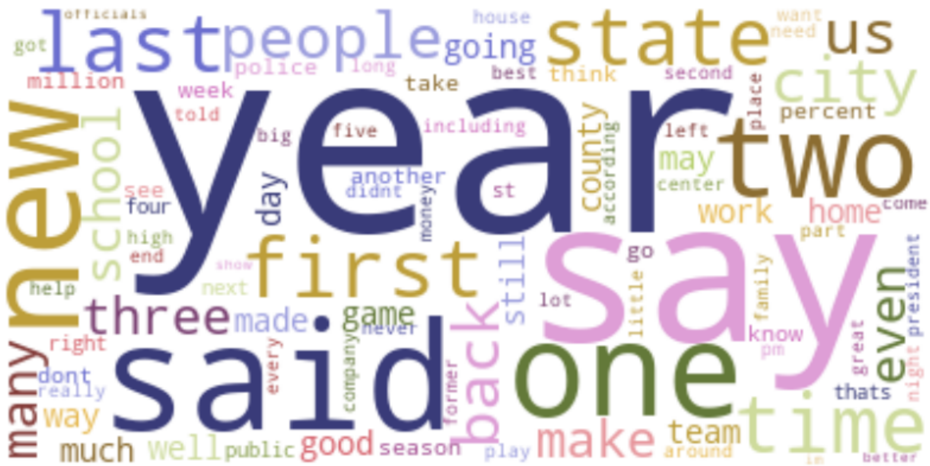

Figure 1: **Word Cloud of The Most Frequent 100 Singletons**

has a significant impact on the presence of the word "new". In other words, if a text contains the word "york" it is highly probable to observe the word "new" in the same text. But, the opposite rule, "new → york" does not indicate such a causal relationship. For this rule, the confidence and interest values were 0.17 and 0.16, respectively. This shows that the existence of the word "new" in a text does not necessarily lead to the existence of the word "york" in the same text. There were two other association rules, "think → said" and "going → said", the confidence values of which were 0.50 and 0.51, respectively. However, for these rules the interest values were low, 0.27 and 0.28.

Table 2: **Frequent Item Pairs and Their Support Values**

	Item	Support		Item	Support
1	said, would	10275	10	said, us	5993
2	one, said	8636	11	said, year	5743
3	people, said	7852	12	also, said	5678
4	last, year	7329	13	could, said	5628
5	get, said	7112	14	said, think	5495
6	going, said	6967	15	new, york	5437
7	like, said	6874	16	last, said	5247
8	said, time	6458	17	said, years	5097
9	new, said	6358	18	said, state	5078

Table 3: **Association Rules between Frequent Pairs**

	Pair	Pair Freq.	1. Sing	1. Sing. Freq.	2. Sing.	2. Sing. Freq.	1. Rule	1. Rule Conf.	1. Rule Int.	2. Rule	2. Rule Conf.	2. Rule Int.
1	{would, said}	10275	would	30210	said	113950	would → said	0.34	0.11	said → would	0.09	0.03
2	{said, one}	8636	said	113950	one	37839	said → one	0.08	0.00	one → said	0.23	0.00
3	{said, people}	7852	said	113950	people	21083	said → people	0.07	0.03	people → said	0.37	0.15
4	{year, last}	7329	year	25689	last	24415	year → last	0.29	0.24	last → year	0.30	0.25
5	{said, get}	7112	said	113950	get	19613	said → get	0.06	0.02	get → said	0.36	0.14
6	{said, going}	6967	said	113950	going	13698	said → going	0.06	0.03	going → said	0.51	0.28
7	{said, like}	6874	said	113950	like	22345	said → like	0.06	0.02	like → said	0.31	0.08
8	{time, said}	6458	time	23802	said	113950	time → said	0.27	0.05	said → time	0.06	0.01
9	{said, new}	6358	said	113950	new	31160	said → new	0.06	-0.01	new → said	0.20	-0.02
10	{us, said}	5993	us	18918	said	113950	us → said	0.32	0.09	said → us	0.05	0.02
11	{year, said}	5743	year	25689	said	113950	year → said	0.22	0.00	said → year	0.05	0.00
12	{said, also}	5678	said	113950	also	28148	said → also	0.05	-0.01	also → said	0.20	-0.02
13	{said, could}	5628	said	113950	could	18558	said → could	0.05	0.01	could → said	0.30	0.08
14	{think, said}	5495	think	10988	said	113950	think → said	0.50	0.27	said → think	0.05	0.03
15	{york, new}	5437	york	5593	new	31160	york → new	0.97	0.91	new → york	0.17	0.16
16	{said, last}	5247	said	113950	last	24415	said → last	0.05	0.00	last → said	0.21	-0.01
17	{years, said}	5097	years	23824	said	113950	years → said	0.21	-0.01	said → years	0.04	0.00
18	{state, said}	5078	state	21202	said	113950	state → said	0.24	0.01	said → state	0.04	0.00

Same results were obtained when the Parallel FP-growth algorithm was implemented. The association rules output of the Parallel FP-growth algorithm includes besides the confidence values, the support values for the frequent pairs and the lift values. These results are presented in Table 4. As seen, the confidence and the support values are exactly the same as the values that were obtained using the SON algorithm. The additional lift values that are computed by the Parallel FP-growth algorithm are symmetrical, that is, the lift value for a rule (e.g. year → last) is the same as the lift value for the opposite rule (e.g. last → year). In line with the previous results, the lift value for the association rule "york → new" was high compared to the lift values for other associations rules, which also indicates that the occurrence of the word "york" has a positive effect on the occurrence of the word "new".

Table 4: **Association Rules Output by the PFP-Growth Algorithm**

Antecedent	Consequent	Confidence	Lift	Support
one	said	0.228	1.012	0.017
year	said	0.224	0.991	0.011
year	last	0.285	5.904	0.015
new	said	0.204	0.905	0.013
new	york	0.174	15.762	0.011
also	said	0.202	0.894	0.011
like	said	0.308	1.364	0.014
years	said	0.214	0.949	0.010
would	said	0.340	1.508	0.020
time	said	0.271	1.203	0.013
people	said	0.372	1.651	0.016
state	said	0.240	1.062	0.010
going	said	0.509	2.255	0.014
last	year	0.300	5.904	0.015
last	said	0.215	0.953	0.010
get	said	0.363	1.608	0.014
said	one	0.076	1.012	0.017
said	new	0.056	0.905	0.013
said	would	0.090	1.508	0.020
said	think	0.048	2.217	0.011
said	also	0.050	0.894	0.011
said	year	0.050	0.991	0.011
said	last	0.046	0.953	0.010
said	years	0.045	0.949	0.010
said	time	0.057	1.203	0.013
said	like	0.060	1.364	0.014
said	state	0.045	1.062	0.010
said	people	0.069	1.651	0.016
said	get	0.062	1.608	0.014
said	us	0.053	1.405	0.012
said	could	0.049	1.345	0.011
said	going	0.061	2.255	0.014
us	said	0.317	1.405	0.012
think	said	0.500	2.217	0.011
york	new	0.972	15.762	0.011
could	said	0.303	1.345	0.011

7 References

Agrawal, R., Imielinski, T., Swami, A. (1993). Mining Association Rules between Sets of Items in Large Databases, in: ACM SIGMOD International Conference on Management of Data. Washington DC, USA.

Agrawal, R., and Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487-499.

Fung, B.C.M., Wang, K., Ester, M. (2003). Hierarchical Document Clustering Using Frequent Itemsets. In Proceedings of the SIAM International Conference on Data Mining, Volume 30, 59-70.

Han, J., Pei, J., Yin, Y. (2000). Mining Frequent Patterns without Candidate Generation. SIGMOD Rec. 29, 2, 1-12.

Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E. (2008). PFP: Parallel FP-Growth for Query Recommendation. In Proceedings of the 2008 ACM conference on Recommender systems (RecSys '08). Association for Computing Machinery, New York, NY, USA, 107-114.

Ordonez, C. (2006). Association Rule Discovery with The Train and Test Approach for Heart Disease Prediction. IEEE Transactions on Information Technology in Biomedicine, Vol. 10, No. 2, 334-343.

Rajaraman, A., Rajaraman, J.,, Ullman, J. D. (2014). Mining Massive Datasets.

Savasere, A., Omiecinski, E., Navathe, S. B. (1995). An Efficient Algorithm for Mining Association Rules in Large Databases. In Proceedings of the 21th International Conference on Very Large Data Bases (VLDB '95). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 432-444.

Zhang, W., Yoshida, T., Tang, X., Wang, Q. (2010). Text Clustering Using Frequent Itemsets. Knowledge-Based Systems, 23(5), 379-388.

8 My Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.