Frequent itemsets in the Yelp reviews dataset

Nicola Rinaldi (student ID: 20161A)

Algorithms for Massive Datasets (A.Y. 2022/2023)

1 Introduction

The proposed project tries to extract the most common sets of words used in the Yelp datset.

This problem can be seen as the search of the frequent itemsets (or market-basket analysis) if we consider the text content of the reviews as baskets.

The code linked shows two possible approaches. The first uses the A-priori algorithm as it is on the dataset (or on a reduced version for presentation purposes) underlining its weaknesses when comes to manage bigger datasets. The second applies it in a randomized algorithm (the Toivonen's algorithm) proving that working on a well-chosen subset of the data it is able to achieve (most of the times) the same result.

2 Preprocessing

2.1 Extract words from reviews

Once we have access to the text of the reviews it is tokenized to extract the words. This is made in four steps:

- split the text on white-spaces
- remove all non-alphanumeric characters (such as punctuation) from the end and the beginning of each word
- remove stopwords (common words that don't carry any information)
- convert all in lowercase

The list of stopwords is obtained from the nlkt library. We remove them just because we don't want to find out that the most common words used in the reviews are the same used in almost all written text (like pronouns, conjunctions, articles and so on).

This process could be improved using some text normalization like stemming or lemming. This could bring to a less number of different items and then a less computational effort.

2.2 Words Encoding

It is easier to manage integer numbers instead of strings, so each item is mapped uniquely into an integer. This is made using an ad hoc built class called *EncodeItems* that allows us to get the integer value of a word and vice versa.

2.3 Basket representation

To improve the performance of the algorithm we apply some other strategies to store the baskets. First of all we can drop out all the duplicates of the items in the same basket because they don't give use any additional information. Next

we sort each basket because it will allow us to compute the candidates and the frequent itemsets in a more convenient way. At the end of this process we have a new json file containing a list of sorted integers for each line representing our baskets.

3 The A-priori Algorithm

3.1 Counting Pairs

The most memory consuming step is the counting of the frequent pairs. To count pairs we first compute the frequent sets of one item (frequent singletons). This is made reading basket by basket and update a python dictionary where the keys are the items (their integer encoding) and the values are the number of basket in which each item appears (its support value). We can consider only the ones which support value exceeds the support threshold.

Before computing the pairs we need to check if the support threshold is well-chosen. We want to exclude a big number of items and keep only the ones that are really frequent. A good option for this specific case is to consider frequents only the itemsets that appear in at least the 2% of the baskets. This choice is made looking at the data. For example using just the 1% becomes clear (plotting the counts of the singletons) that there are a lot of items that are slightly below or slightly above the threshold. Using the 2% we have a sharper distinction and it works well using different input sizes.

Once obtained the frequent singletons, we build the set C_2 of the candidates pairs. Thanks to the monotonicity property the candidates pairs are the ones composed by two frequent singletons. It is made in a double for loop We don't need further checks for guarantee that all subsets of size one are frequent (we will need that for the C_k with k greater than 2).

Before get the L_2 set we need to scan the file of baskets and count the frequences (support value). In the project it is made using the *triple-method* saving the count in a dictionaries just like in the singletons case but now the keys are couple of items. The same technique will be used for triples bigger sets. Finally we can filter out the ones that are below the threshold.

3.2 Counting frequent itemsets

The same approach is used for obtain the rest of the itemsets but the generation of the C_k set is slightly more complex.

We generate C_k combining pairs of items in L_{k-1} in two steps:

- **joining**: if two sets l_1, $l_2 \in L_{k-1}$ match on the first k-2 items then we can build a temporary candidate $\{l_{1,1},...,l_{1,k-1}, l_{2,k-1}\}$
- **pruning**: if all of the subset of size k-1 are frequent then we can add the set computed at the previous step to the C_k. This is made by generating all the combinations of size k-1 from the candidate and checking if are contained in L_{k-1}

Done that we compute L_k counting the frequences of the itemsets in C_k and extract only the ones that are above the threshold.

We stop to check for bigger frequent itemsets when for some k, L_k is empty.

3.3 Limitations of the A-priori Algorithm

The main limitation of the A-priori algorithm is that we need to read the entire basket file k times and this could be a bottleneck for the computation.

Executing the proposed code on a single machine with a file of some hundred of thousands of baskets we can notice that it is able to compute the result in a reasonable amount of time ($\sim$ 20 minutes).

If we try to run the code on the entire dataset ($\sim$ 7 millions of baskets) the execution time increase a lot (some hours).

In the next section is proposed a solution based on the Tovoinen's algorithm that most of the times can give the same result but applying the algorithm on a sample of the original baskets file.

4 The Toivonen's algorithm

The solution proposed in the project is to use a sample of the data that can fit in the main memory of the machine and compute the algorithm on this sample. We clearly need to choose a representative sample that reflect the properties of the whole file. To do so we use a randomized approach.

4.1 Choosing the sample

We set a parameter p that represent the fraction of the dataset that we extract. For each basket we add it to the sample with a probability p.

The threshold s must be scaled as well, in this case we set it to αps (with alpha ¡ 1) because we don't want to miss any frequent itemset (called a False Negative). In the code it is set to 0.8 or a bit lower if we consider a smaller input.

Clearly it could be changed in function of the choice of p.

For example reducing the original file from some hundred of thousands of basket with $p = 0.1$, α can be set to 0.8. If the original file contains tens of thousands of baskets 0.75 is a better choice.

Now we can run the algorithm (slightly modified because of the nature of the algorithm explained in the next section) on the sample hoping to find a good approximation.

4.2 False Positives and False Negatives

The result of the randomized algorithm is clearly an approximation, just consider that we use a threshold that is less than the proportional value in the original dataset. This is sufficient to have at least some false positive.

But we can delete them with a single scan of the original baskets file. This is made by counting the frequences of all L_k in the sample for each k and filter out the ones that are below the original threshold s.

Done that we can notice that the result is a very good approximation (if not the exact result). This is understandable because we are managing baskets in the domain of business reviews and we expect that the frequent set of words used is still the same if we consider an enough representative sample.

To be sure that the result is not only a good approximation but it is the exact one we can follow the Toivonen's algorithm main feature: if no itemset of the negative border of the sample is frequent in the whole dataset then we don't have any false negative.

To do so we have to compute the negative border (the itemsets of size k that are not frequent in the sample but all their subsets are). This is easy to get if we look back at how we compute C_k and L_k. If an element is a member a C_k then all its subsets are in L_{k-1}. This meets the first condition of being part of the negative border. The second is that the itemset must not be included in L_k. So we just need to keep track of those itemsets in C_k which count is below the threshold.

When all the L_ks are computed we can count the frequences of the itemsets in the negative border in the same pass over the entire dataset when we check that the L_ks in the sample are frequent in the whole dataset.

If none of the itemset of the negative border is found frequent in the whole the algorithm gives the exact answer.

5 Conclusions

Once computed the frequent itemsets of words we can define the relative association function. If I is a frequent itemset then for each $j \in I$ we can define it as:

$$I \setminus \{j\} \to j$$

In the attached code we show a further check of the correctness of the process. Working with a not huge file we ca compute the result for both the whole file (used as ground truth) and the sample proving that the outcome is the same.

Of course this check is made only for presentation purposes and it isn't feasible in the general case specially when the input becomes huge. In that case we can only rely upon the Toivonen's algorithm that could not give us a response in some case (or just an approximate result). If so, we can retry using a different sample.

In the end, we have shown that we can obtain the same result (or at least a good approximation) of the frequents set of words of the Yelp reviews reducing the input size.

This give us good chance of use this strategy even on larger dataset of reviews.

6 Statements of authenticity

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.