Data Science and Economics
Universit‘a degli Studi di Milano

Market Basket Analysis

(Ukraine Conflict)

Student: Vojimir Ranitovic 963780
Email: vojimir.ranitovic@studenti.unimi.it
Module: Algorithms for Massive Datasets

Abstract

Frequent Itemset Mining is a method for market basket analysis. In this project, the A-Priori algorithm is used to find all possible itemsets in textual tweets about the current Ukraine-Russian conflict. A-Priori managed to find 35 frequent itemsets in the sample of 2612 different tweets and threshold set at 5% of sample baskets. Also, after the implementation of the algorithm, specific Spark tunings were set, such as the number of partitions and usage of Kryo serializer instead of Java. Since this is all done on the random sample, possibly there are false positives and false negatives, so the future improvement will be the implementation of the SON algorithm to get the exact answer.

Dataset

For this project, the Ukraine Conflict Twitter[1] dataset is used. It is published on Kaggle and released under the CC-BY-SA 4.0 license. Dataset is updated daily and currently the size of the whole dataset is about 9 GB, and it is downloaded directly through the notebook. For further purposes of implementing an algorithm and finding frequent itemsets, 21th of April was randomly chosen to analyze. This chunk of the dataset (same as other chunks) is consisting of columns e.g., userid, username, followers, following, tweeted, text, language. For the purposes of this project, the only important column is text which contains 361,151 tweets. Of all the tweets, only ones written in the English language are chosen, therefore 261,202 English tweets (around 70% of all the tweets are in English). From these data, the random sample of 1% size is used for further pre-processing and testing algorithm, specifically 2612 rows/tweets.

Pre-processing phase

To find frequent items, each tweet is considered as basket and each distinctive word as an item, and before that, all unnecessary strings are cleaned. Below is an example of how the original tweet is processed into a suitable formulation:

Original tweet:

"⚡Fifth exchange of prisoners.We exchanged 60 servicemen, 10 of them officers. Also, 16 civilians.In general, 76 people are safe.#Ukraine never gives up on its people; we fight for everyone! #UkraineWillWin."

First, the punctuations, numbers, and URL links are removed. After that, the text is lowercased, tokenized (each word is one token), lemmatized and all stopwords are removed. Lemmatization is a technique that switches any kind of a word to its base root mode such as the word gives to give or exchanged to exchange. Also, all emojis are deleted since they are not benefit to analysis. After applying all these techniques, processed tweet looks like the one below.

[1]Dataset can be found on the following link: https://www.kaggle.com/datasets/bwandowando/ukraine-russian-crisis-twitter-dataset-1-2-m-rows

Processed tweet:

[*'officer', 'serviceman', 'safe', 'prisoner', 'civilian', 'give', 'fight', 'fifth', 'everyone', 'also', 'people', 'exchange', 'ukrainewillwin', 'general', 'never', 'ukraine'*]

Instead of a lemmatizer, stemming approach could be used. Stemming the words is a faster process than lemmatizing, but sometimes it can give unrecognizable words as output.

Implementing algorithm and results

For finding frequent itemsets plain A-Priori algorithm is chosen. After pre-processing data and representing it as Resilient Distributed Dataset (RDD), counting of all items is done. There are three steps before it becomes clearer how to make a function. Each step is implemented as it is represented in the picture[2] below.

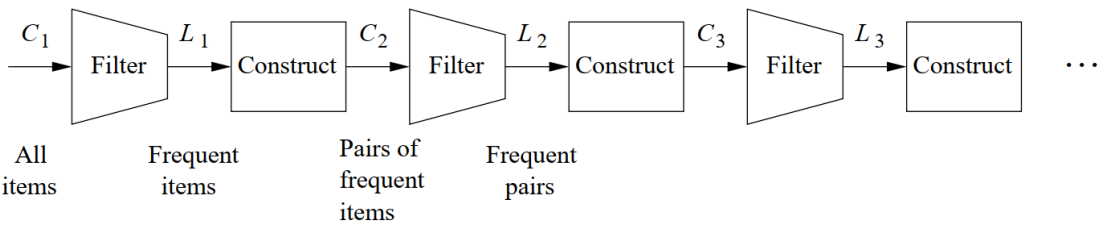

Figure 1: The A-Priori algorithm steps for making candidates and filtering

Steps are programmed using Map, Reduce and Filter functions, with the thought that the Filter function should be used as soon as possible due to avoid unnecessary calculations. Firstly all singletons are found, and the ones with frequencies lower than the threshold are filtered out. A threshold of 5% was set, so for chosen sample threshold was 131 baskets. All singletons that pass the threshold become members of the frequent itemset and also candidates for the next phase of counting pairs. Each candidate pair is then compared to each basket from the starting RDD. All pairs with frequencies that appear more than the threshold are joined to a frequent itemset, and become a candidate for making triples. This process will stop if there are no more items that can form superset. Accordingly, for the threshold of 5% and 2612 baskets, the generalized A-Priori function stopped after the third iteration, because there were no more than frequent pairs. Below is the output of A-Priori, there are 24 singletons and 11 doubletons.

Frequent singletons:

[*('standwithukraine',), ('humanitarian',), ('armukrainenow',), ('weapon',), ('mariupol',), ('ukrainian',), ('civilian',), ('world',), ('ethiopia',), ('support',), ('must',), ('putin',), ('force',), ('russian',), ('ukraine',), ('people',), ('u',), ('amp',), ('war',), ('potus',), ('secblinken',), ('tigray',), ('russia',), ('azovstal',)*]

[2]Picture is taken from the book *Mining of Massive Datasets*, written by A. Rajaraman and J. Ullman, chapter 6. Figure 6.4

Frequent doubletons:

[*('russian', 'ukraine'), ('russia', 'ukraine'), ('ukraine', 'war'), ('secblinken', 'tigray'), ('mariupol', 'russian'), ('ethiopia', 'tigray'), ('mariupol', 'ukraine'), ('people', 'ukraine'), ('amp', 'tigray'), ('russian', 'ukrainian'), ('ukraine', 'ukrainian')*]

As expected, the most frequent words were about Ukraine, Russia, and Mariupol battle. In singletons list, standwithukraine or armukrainenow can be seen, since they are hashtags for supporting Ukraine. Also, there are some frequent words such as Ethiopia and Tigray, which are not directly connected with the Ukraine-Russian war, but are important because there is also the war in the Tigray region, and people are writing about that usually as a comparison between these two wars in Europe and Africa.

Tuning the Spark

Many things can be tuned using Spark, such as the number of partitions and trying different serializers such as Java which is the default in Spark, and Kryo serializer. These tunings would not have a great impact on algorithm execution time, since Spark is running locally (in my case one CPU with two cores). When partitioning of RDD comes in place, suppose there are a cluster with two cores and that RDD is divided into two partitions. That means that one partition will be executed by one core and the second partition will be executed on the second core in parallel. If RDD is divided with three partitions, two cores will work in parallel on two partitions, but third one will be processed after the process of the first two is finished, which is not good practice because in the second iteration one core will remain idle. Also, if we have a whole RDD (one partition), one core will work on it, while the second one will remain idle. To understand if that is the correct approach, A-Priori is run, and after each time the number of partitions in RDD is changed from 2 to 10 in steps of 2. In Figure 2., it can be seen that the least execution time is for two partitions.

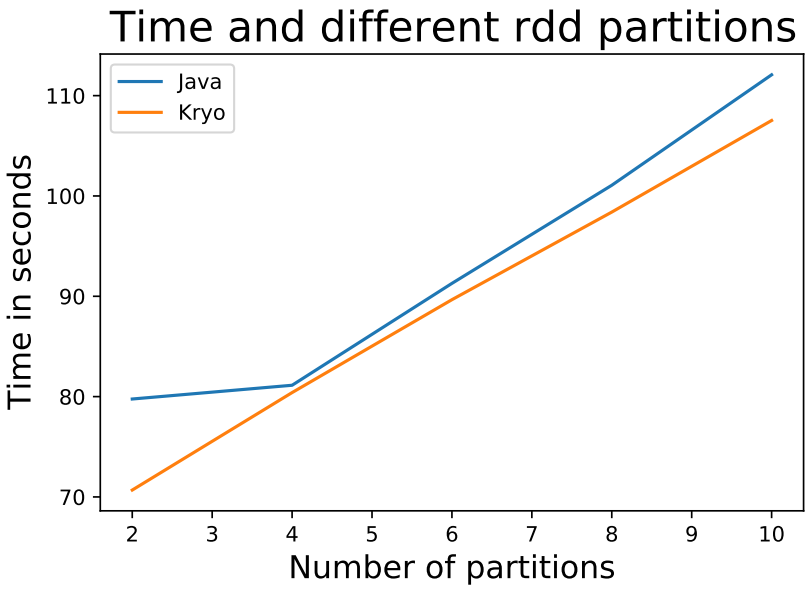

Figure 2: Java vs Kryo serialization for different numbers of partitions

Also, it was checked which serializer is better. The default one in Spark is a Java serializer, but the Kryo serializer is more efficient according to Spark documentation[3]. Serialization is the process of converting the object into bytes, for efficient transmitting into main or mass memory, and the most impact is for transmitting bytes over the network, since it is the slowest part of distributed system. Unfortunately, in case of local mode there is no transmition of serialized objects through the network, and also it is tried to avoid writing/reading from the hard disk, so the only benefit can be for transmitting data to the main memory. It can be seen in Figure 2., that the least executing time is for 2 partitions, but the Kryo sterilizer performs better.

When it comes to increasing the threshold and leaving the sample size fixed, A-Priori behaves as expected. In Figure 3., it can be seen that by increasing threshold, the algorithm needs less time to execute, since in the first passage more singletons would be filtered out, so there would be fewer doubletons, etc. There is no big difference between the two serializers in this task.

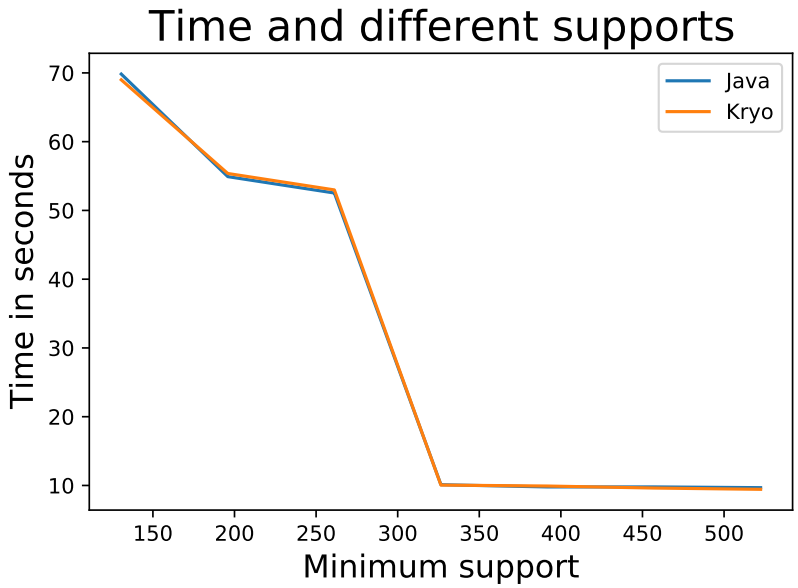

Figure 3: Java vs Kryo serialization for different thresholds and fixed sample

Since Kryo performed better than the Java serializer for the same number of partitions, Kryo is set to be the Spark serializer and RDD is partitioned into 2 parts. After that, it is checked how A-Priori behaves if sample size is increased, as Figure 4. shows. Time increases almost perfectly linear as sample size increases. In this part, threshold is changed from 5% to 50% .

[3]Documentation can be found at: https://spark.apache.org/docs/2.1.2/tuning

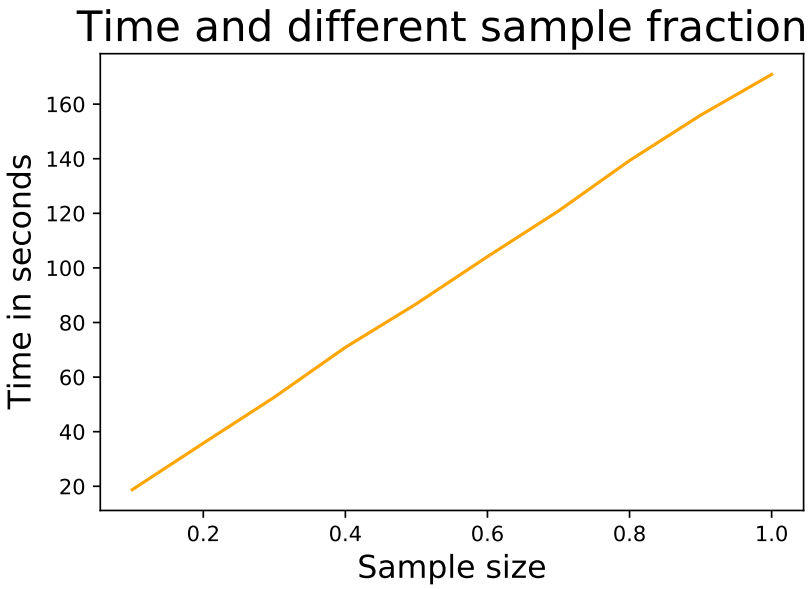

Figure 4: Kryo serialization and different sample size with fixed threshold

Comments and future improvements

Usually, marketing experts do not need more than frequent pairs or triples, because it could be hard to read and justify higher supersets. In this project, when words are used as items, it could be useful to find triplets, quadruplets, or a higher number of itemsets if it exists. The reason is that it can give more of a context about what is happening on a particular day, and an easier understanding of what people frequently write on Twitter, but the price for that will be a need for more memory. In this project, sample is used to test the algorithm, so there are probably false positives and false negatives. In the future, it will be great improvement to implement the SON algorithm and use whole dataset for the given day, and also get rid of false positives and false negatives.

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.