The solution description:

In order to evaluate the negative feedback for the apps, we use 2-parts algorithm and some manual process that was done. The description will refer to each of the parts.

**Level 1: Preprocessing the data.**

In the first part, we focus on the extracting the data and extracting some useful information from it. The data extraction was done by scrapping the web address that contains the applications' reviews. Later we use the data we extracted in order to create two files with the information we need for the next level:

1. reviews.txt - contains all the reviews. Ordered by a json format, each app has its own reviews applied to it.
2. info.txt – contains two lists. The first one has apps' names and the second has the popular subjects that appears in all the apps' reviews.

In the code -

We have a preprocessing module that is responsible for creating the mentioned files. It is done by running two other modules:

1. crawler.py – scrapping the web in order to extract the data.
2. topic\_extractor.py – given the data, it finds the popular tokens in it.

*All the code files for this level can be found in the 'preprocessing\_step' folder and the files that were created can be found in the 'files' folder.*

**Level 2: Manual Process.**

The work in this level was done manually. We received the popular tokens that appears in all the apps' reviews, after we removed stop words and the punctuations from the list. And later we kept only words that are nouns or adjectives. We chose to locate more tokens than needed so we could sort out what tokens would be useful for us.

We started with this list and tried to figure out what words would be useful for extracting info about the application's quality. We had word such as: "game", "play" and "coins" that couldn't help us figure out nothing about what the player liked or didn't liked in the game. In contrary, we had words like "time", "ads" and "money" that were informative.

We chose six main topics and for each topic we added some negative words that were related to it and saved them in a new file:

* info\_manual.txt – contains two lists. The first one has apps' names (untouched by us) and the other was built as a dictionary, each topic name was the key and its value was a list that contains the added word relating to the subject.

*The new file can be found in the 'files' folder.*

**Level 3: Analyzing the data.**

This part includes some parts inside of it. The user can choose one application from a given list. For the chosen application, we go over all its reviews and extract all the informative trigrams from it. For each trigram, we decide what its sentiment and if it's negative – we save it in a list.

In the next step, we go over the list of the negative trigrams and check for each topic (from the list we created in the previous level) how many trigrams are referring to this topic and creating a graph based on that info.

In the code –

We have a manager module that is responsible for creating the mentioned graph for the chosen application. It is done by running other modules:

1. reviews\_extraction.py – extracts the reviews for the application from the file that was created in the second level.
2. Info\_extraction.py – extracts the informative phrases as trigrams.
3. sentiment\_analyzer.py – it analyzes its phrases and determines its negative trigrams.
4. result\_analyzer.py – plot the analysis results in graph for the given app.

*All the code files for this level can be found in the 'analyzing\_step' folder and the graphs that were created can be found in the 'Graphs' folder.*

Important remarks:

* The popular subjects were taken from all the apps' reviews and not popular subjects for every app, so we have one list that includes the generally popular topics.
* Stop words are

To add

1. explanation - Why some words were informative and other weren't
2. packages we used, built-in algorithms