Projet II

Task 1:

The Spark library has a very handy function to load the data. All that was left to do was calling `SparkContext.textFile` and then mapping the correct columns for both the titles and ratings. As for persisting the data, I used the function `persist().`

Task 2:

The hard part was probably having the right data structure to enable the incremental update. In the `init()` I have a first data structure that contains the left outer join of the titles and the ratings, where the title id is the key. Then the join is aggregated using `aggregateByKey()`, which gives for each movie the sum of the ratings and the count (e.g number of users who rated it). If a user already rated the movie, then the count is not updated, its old rating is subtracted from the sum and the new rating is added.
The functions `getResults()` and `getKeywordQueryResult()` are then pretty straightforward to implement. For the `updateResult()` function, I first try to find if a user already rated the movie and then update the initial state. Then the aggregate is computed only on these new ratings, and added to the previous aggregate.

Task 3:

It actually took me some time to understand how the `getBuckets()` worked. Once I found what to give as input to the `hash()` function, the implementation of the near-neighbor lookups was simply a matter of following the instructions of the pdf… As for the cache, I found that the doc for the function was a bit confusing (I think there's a mistake in the doc: "The second RDD corresponds to queries that result in cache hits and need to be directed to LSH", it probably should be "misses"). To build the cache, I have a RDD which counts the number of times a signature occurs when `cacheLookup()` is called, as mentioned is the pdf, signatures that occurs in more than 1% of the queries are looked up in the `lshindex` and kept in the cache, finally its values are broadcast to all the workers.