Hotel Comparison Documentation

Compile and Execute **2**

Solution Idea **3**

Requirements **3**

Solution / Approach. **3**

Code Explanation **4**

Classes **4**

objects.CountNValue **4**

objects.TopicSentimentResults **4**

bjects.ReviewInfo **5**

object.HotelInfo **5**

object.HotelObj **5**

operations.ReadHotelFiles **5**

operations.TopicSearchEvaluate **5**

operations.CompareTopic **6**

utils.Commons **6**

run.Demo **6**

Possible Extensions **7**

Stemming / Distance toleration **7**

Lemmatization **7**

Inverted Index **7**

Compile and Execute

To compile the code go inside the projects directory, inside **HotelCompare** and write "`mvn clean install`" and after that "`mvn clean compile assembly:single`", this probably will require some packages to download and and after that will create a .jar inside the target directory which will be called to execute the program.

Prior to executing this command the .json files need to be in the same directory, also inside that same directory should be another directory named "Semantic" (capital S) that contains the semantic.json file.

SO:

```
Dir*
|____reviw1.json, review2.json …. ,  , , , (.json files)
|
|______Semantic (directory)
|        |
|        |___semantic.json    (file)
```

"`java -cp target/Hotel-0.0.1-SNAPSHOT-jar-with-dependencies.jar run.Demo "<the directory where the files are>" "<the topic which we want to search the hotels for>" "` ….(*without the <> chars*)

The run.Demo is the main class, the first argument is the <Dir*> directory path, and the second argument is the topic that we want to search for.

The result will be displayed in descending order of value and along with the
Hotel: points expected couple of results additional information for the search will be displayed.

Statistical data like the total mentions of the topic in the reviews, the total, positive and negative and average sentiment points as estimated by the program.

Additionally there will be displayed 3 sentences to the user to let him better understand the sentiment that the hotel visitor meant.

Another way of running the program is by using maven with the command
'`mvn exec:java -Dexec.mainClass=run.Demo -Dexec.args= "<path_argument> <topic_argument>"`'

This will display the results along some maven run information.

Solution Idea

Requirements

We have a list of reviews about each hotel in our database. We would like you to build a tool that lets us compare different hotels.
The tool receives as input:
* A list of reviews for each hotel.
* The topic we are interested in.

The tool provides as output:
* An overview of which hotel scores best on the given topic.
* Useful statistics about the processed reviews.

You should score hotels based on whether the reviews speak positively or negatively about the given topic.
A topic is something like *room*, or *breakfast*, or *spa*. Take into account that multiple words
can indicate the same topic, like *staff* and *personnel*.

Solution / Approach.

Create an application that can read the reviews, in this case specifically to read the .json data files and from them gather useful information about the hotels regarding a specific topic but also additional information, if available.

Functional requirements:

- The system requires a list of reviews for the program to consider (Specifically in this occasion is the path of the directory where the review files are stored) AND a topic of interest to estimate its sentiment on the reviews.
- The system should search through the reviews for the search-topic and estimate the sentiment expressed about it in the review.
- The system should return the results ordered by the topic value that estimated.

Non Functional requirements

- The system can provide additional information about the hotel or the review as to help the user make a more informed decision.

We will leverage the semantic.json file with its semantic content to determine/evaluate numerically the sentiment in the comment. The evaluation is done by defining the sentences inside the comment paragraph and for each sentence all its words are considered. In the first run it is determined if the topic is present in the sentence. If it is found, then the second run is done in the same sentence to evaluate numerically the sentiment the user has expressed. This is done for all the comments of the hotel reviews and a sum of the positive and negative sentiments is presented as a total for the hotel's perception for that topic.

At the end of the run when the results are shown to the user, we have at our disposal a lot more data that can be shown however their representation in the terminal would be confusing and ineffective so I decided not to show them.

Code Explanation

Classes

Google's Gson libraries are used to help with java to json conversion and vice versa.

The reviews in the files have a similar structure however is not exactly the same in all the files that's why we can use the direct json to java conversion that gson library supports, instead we have to take a more iterative approach, that is iterate over the json key:value attributes to get the ones we want.

objects.CountNValue

This class is used to store the actual numerical ratings that the users have given to the hotel regarding some attributes. All that is kept here is an attributes count and sum of rating from all the reviews of a hotel. This class helps with the secondary, additional information that we could display for a hotel.

objects.TopicSentimentResults

This class is used to store the primary type of information required by the system, that is information about the topic the user is searching for. Specifically it keeps the topic, the number of times a topic is mentioned throughout the reviews, the positive and negative sum of the sentiments as evaluated by the system and a max of 3 representative sentences from different comments where the topic was mentioned. Along with the representative sentences there is additional information for the review in which the sentences are taken from, like: author, authorsLocation, date, title and reviewid. These are data kept in the ReviweInfo class.

bjects.ReviewInfo

This class is used to keep additional information about the reviews where the search-topic has been found.

object.HotelInfo

Is used to keep information about the hotel of which reviews are currently being examined. Contains information like the name of the hotel, the address, site and image URL etc. It is always filled with information regardless weather the topic has been mentioned in the reviews or not.

object.HotelObj

It represents the virtual object of Hotel in our program. It is composed of HotelInfo attribute, TopicSentimentResults which are the main informations that the user requires from the system and a list of rated attributes to provide additional information to the users.

operations.ReadHotelFiles

Is The class that provides the main functionality of the system by reading the files and ultimately transforming them into java objects with useful information. Since to retrieve specific parts of the json object the entire file has to be read (and loaded into memory) we use this class to provide all the functionality necessary for the system so that it won't be required to read again. This class provides methods to read from json the hotelinformations obj, the ratings of the reviews and provides the reviews as an array to be examined in another class more carefully.

It implements the methods of the **ReadData** interface which would be the main methods required for the functionality of system even if the access to the data would be of a different nature and implemented in different manner. It orients the classes to implement methods to return hotel informations, a map of the user rated attributes and their values and the topic search results.

operations.TopicSearchEvaluate

Is the class that extracts the actual topic sentiment from the comments in the reviews. Here we utilize "BreakIterator" to find which are the sentences in the text paragraph. And in those sentences we utilize BreakIterator again to find the words in the sentence. The algorithm goes through every word to find the topic. If it is not found the algorithm moves on to the next sentence, but if it is found then the algorithm goes through all the words again to find semantically valuable words according to the semantic.json file. The way this is done is by finding intensifiers (if there are) and multiply them with the first sentiment valuable word and continue to do so for every possible word in the sentence that has semantic value. In the end the produced positive and negative evaluations that may have been found are added to the entire comments positive and negative sentiment evaluation. From a syntactical point of view this approach is correct as it gives more value to sentiments that are emphasized in the sentence by intensifiers and also considers appropriately the ones that are not emphasized. However this approach doesn't measure correctly the sentiment of the topic but rather the sentiment of the sentence in which the topic is in and attributes it all as the topics sentiment. Most of the time the sentence and topic sentiment coincide, but we can't be always sure about that.

This approach is taken for every review in the file. So it turns out we analyze every word of every sentence of every comment and produce overall sentiment evaluations for the hotel regarding a topic.

operations.CompareTopic

Is just a class that only helps with the sorting of the hotels based on the topic sentiment.

utils.Commons

Is a class containing data structures loaded with the semantic file data to be used quickly and enyware they are needed.

run.Demo

Is the class with the main method that initializes the functionality of the system. It checks the validity of the review files directory and prints the results of the search and evaluation that the system has done.

Possible Extensions

Stemming / Distance toleration

These are two techniques that can help our algorithms return better results for the sentiment evaluation of the search-topic.

Stemming is the removal of suffixes and return of the word to a simple basic form. This would be useful for finding the topic in a sentence but even more useful for the finding semantically valuable words considering that the way a sentiment is expressed can vary a lot mainly by adding suffixes to the base, stem of the word or by using colorful intensifiers. In all this variation the same or similar sentiment can be expressed in multiple ways but they would not be found by our systems lexical evaluation methods as they would not be equal strings.

A different method to better approach this problem is by allowing a certain degree of error between the two words that we are trying to pair. We could consider similar words that are almost equal to each other but not entirely equal. If we suppose that a word might have changed slightly to better express an emotion we could accept it being semantically similar to its original form and consider the pairing as acceptable.

These techniques have their drawbacks as well. Stemming would be the more acceptable of the two methods as the pairing of two words that might result after its usage would be with a higher degree of confidence compared to the usage of string distance toleration. However stemming is "heavier" from the aspect of processes and time required that most string similarity methods are.

Lemmatization

One of the requirements for this task was to help users by considering in the search that the same topic could have synonyms or some other word of similar meaning. My Firs approach to this was to look online for hotel ontologies and utilize their predefined same/similar words for the topic considering that writing my own hotel topic synonyms would be inefficient and possibly with mistakes.

Practically lemmatization was the required approach for this problem, that is lexicographical stemming and usage of a dictionary and other transformation methods to produce similar or synonymous words for the search-topic and possibly for the semantic words.

Inverted Index

Since the completion of this task requires heavily computation, by reading from the disk and traversing the in-memory json file to find the parts that are needed, a good idea would be to store the results that are of interest in an index where the topic would be the key and the documents and the statistically important information would be the values corresponding to it. This way a huge amount of processing and time would be eliminated.