Hotel Comparison Documentation

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. Also open nlp model files for sentence tokenizer and POS tagger are needed for this project to work. This files (the models) path need to be specified in the "config" file, in the "src/main/resources" directory of the project.

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:

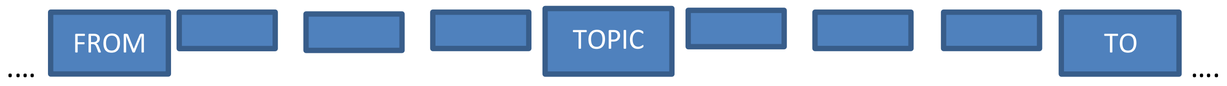
- The system requires a list of reviews for the program to consider (Specifically in this occasion is the path of the directory where the review files are stored) AND a topic of interest to estimate its sentiment on the reviews.
- The system should search through the reviews for the search-topic and 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 then we search within a certain distance of words from the topic +- "TOPIC_RANGE" (arbitrary range of words in the sentence) to find semantically valuable expressions and assign its value to the topic.

The approach that is taken for this task is to split the comments into sentences and the sentences into words and iterate through them looking to sing the search-topic. If it is found then it is assumed that any sentiment expression near the topic (*near the topic being ~4 words of distance from it) it could actually be referring to the topic the user is looking for. So we look for semantic expressions TOPIC_RANGE in front of the topic and TOPIC_RANGE after the topic.

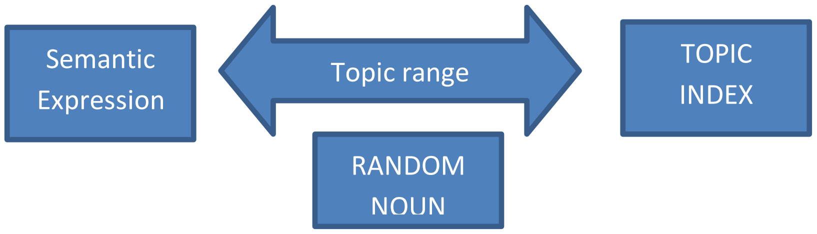


the blue parallelograms are words of the sentence around the topic that is found.

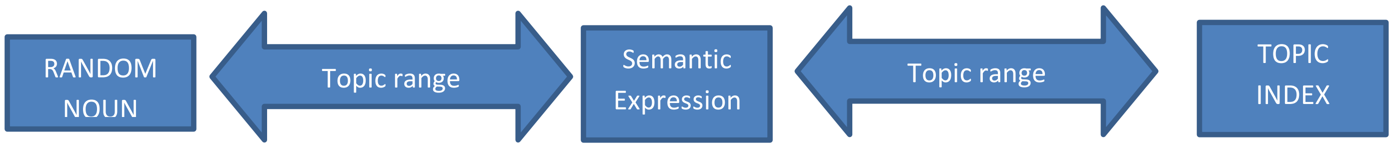
If a semantic expression is found inside this range then it has to be considered whether it is referring to the topic the user is looking for or some other subject. To help ourselves with this task we utilize the POS tagger library from the open nlp packet. This library helps us to determine the role of the words in the sentence (whether it is a noun, pronoun, adjective, verb etc.). By parsing the sentence through the tagger we are looking for the nouns of the sentence, since they are other possible subjects to which the semantic expression could be referring to. After the POS tagging we have a list of the noun indexes in the sentence in hand.

This list is utilized by looking for another random noun from the list in proximity of the semantic expression. There are 2 cases that we pay attention to:

1) There is a random noun between the semantic expression and the noun



2) There is a random noun which is also in TOPIC_RANGE distance from the semantic expression but not between the expression and the topic.



In the first case it is assumed that the entire value of the expression is directed toward the random noun and not the topic. So no points are assigned to it (the topic). And in the second case since it is impossible to determine, without lexicographical evaluation, where is the sentiment directed at, the value of the expression is divided between the two nouns. That means that the topic in which the user is interested in takes only half the value of the semantic expression. And Of curse in case there is no other noun in topic_range of the expression then the topic gets all its value.

*To be noted the images above show only the case when the expression and the random noun happens to be before the topic but they can also occur after the topic as well (from the right side of the topic in the image).

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. The action that happen here are stemming of the words so that the root of the words is captured. Tokenization of the sentences and the words of the sentence which allows us to better handle the words of interest we encounter. Part Of Speech tagging from which we can determine the nouns in the sentence. And iterations, all the sentence to find the topic, the range check for the possible sentiment inside the range and the check for other nouns that can acquire the sentiment of the expression instead of the topic.

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.

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 everywhere they are needed. It also holds and initializes the sentence and pos-tagg model of the opennlp package so that it doesn’t need to be initialized every time.

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

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.

Improvements from last version

Stemming

A stemming algorithm "Porter Stemmer" has been applied to the word tokens of the sentence which has increased noticeably the topic & semantic hits in the comments.

Search

The search has changed by allowing for a more efficient search. Instead of 2 passages to the sentence, one for the topic and the other for the semantic expression value. Now the semantic search is done in a confined range near the topic which is more efficient but also more precise regarding the topic sentiment expressed. So instead of assigning to the topic the semantic value of the entire sentence, now we assign to the topic what with a high probability belongs to the topic.

Apach open nlp.

With the utilization of this package the efficiency of the application has increased. However the cost, specifically time requirements has increased as well even though these packets libraries are not fully utilized as the open nlp package offers a lot more tools than what are being used in this task. We utilize the stemmer, sentence and word tokenizer and the POS tagger.