Aldea-Ionescu Alina Elena, group 1077

Homework Design Patterns

I will talk here about my bachelor thesis, where I created a social media analysis platform. This application performs sentiment analysis on Tweets, using 2 distinct aproaches:

- Using a lexicon, which is more of a rule-based approach. This lexicon is similar to a dictionary which is comprised of words describing sentiments and associated scores. The sentiment of a message/Tweet is computed iterating over the message and summing up all the scores, thus at the end rendering a sole number (1 denotes an extremly positive sentiment, -1 extremely negative, and values close to 0 neutral).
- Using Machine Learning, where I used as a training data 1500 messages retrieved from Twitter and adnotated them manually (as positive, negative & neutral). With this input data, I trained my model, and using the trained model, I can make predictions by feeding an input message to it.

The overall system is composed of 3 main components:

1. A Twitter Collector: which creats a stream using the Twitter API and retrievs messages based on filters (eg. Keywords and the romanian language). The messages are placed in a cloud queue.
2. A Sentiment Analyzer: is where the actual sentiment analysis is performed. Here, the messages are retrieved from the queue, the analysis is carried out and the results are stored in a cloud database.
3. An Online Portal: this is the only interaction with the user. Here, you can create keywords organized in distinct projects, and this keywords will server as a filter for the stream in the first app. On this portal, the statistics and trends are shown to the user, for the messages containing those chosen keywords.

Note: Thus far, I use the 2 approaches separatly: the lexicon approach is employed in the overall application, while the ML approach is used quite separately and I test it with Postman. The aim was not only to create an application which performs sentiment analysis, but also to make a comparison between the 2 approaches and see which one renders better results (and the ML won).

Design patterns that I thought it might make sense in the context above:

1. **Creational**
    a) **Singleton:** this can be employed when connecting to the database, for example in the third app (online portal). Frankly, I initialized the db context multiple times in this app, and a singleton would do much better here.
    b) **Factory Method:** in the third application, I have several classes (a family of objects), such as Project, Keyword, Tweet, User. Perhaps it would make sense to create a factory to give me this objects when I need them, instead of instantiating each object using the constructor.
2. **Structural**
    a) **Adapter**: A company is buying this application but it wants to integrate in it its own module which works with a Facebook API. The current application (the one described in the paper) only supports a Twitter API for extracting the messages. It's not desired to change any extisting code, but rather to integrate the the 2 applications in order to make it also work with the Facebook retrieval of messages.
    b) **Flyweight**: A user needs to create hundreds of projects, but most of them share the same analysis variables and it creats redundancy within the application, saving the same information multiple times. Or maybe the same keywords are even shared between different users, not only projects of a user. It is desired to find a way to minimize the large memory footprint that this situation creats.
    c) **Proxy**: Multiple projects involve rendering graphics and analytics for each one of them. It is desirable to have a mechanism which creats objects on demand, using the cache memory, such that, in the case the user accesses it very frequently, the charts could be displayed from cache instead of retrieving the data all the time from the database.
3. **Behavioral**
    a) **Strategy**: For the machine learning approach, I have used 6 distinct classifiers just to compare which one would be most suitable. Assuming that the training data volume increases in time, some classifiers would render better results than previous. We want to be able to choose at run-time which kind of classifier would be most suitable to train the model.
    b) **Observer**: On the online platform, the users can register to receive notifications in case a certain trend from a project exhibits extreme values. For example, if users chose to follow the trend of messages containing the keyword "Samsung", and if the public opinion in social media is lousy, and the graphs would show a downward trend, the users may want to get notified that something wrong is happening with that product/service.

c) **Command**: Because the buffer from the Twitter API is overflowing in case there is a burst of messages on social media, the analysis cannot be carried out in real time, because it takes time and we would lose incoming messages. Thus, we want to use a mechanism that allows the Tweets to be analyzed later.