Using text analytics to build a recommender system for comedy

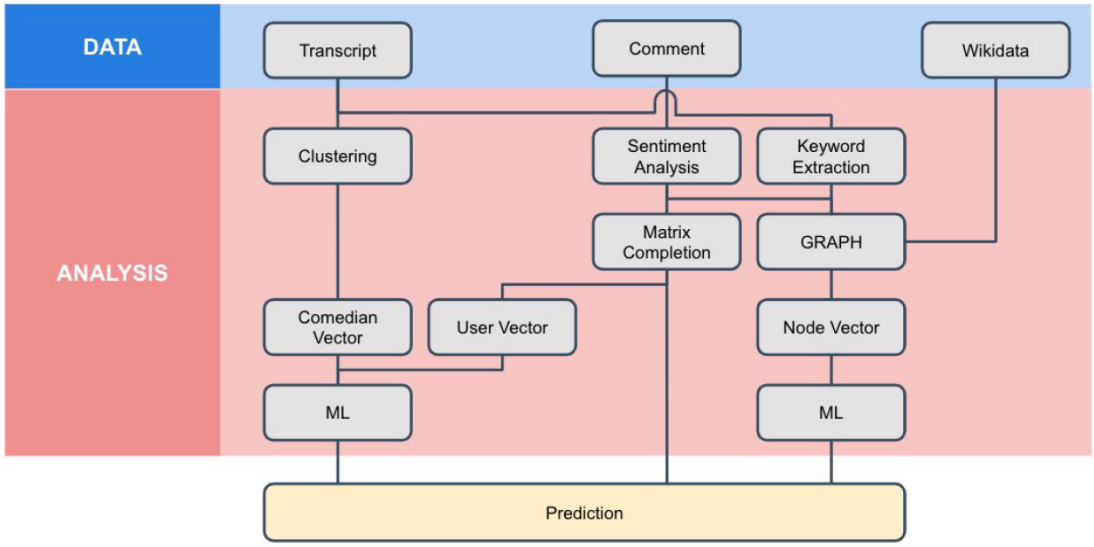
This case tries to build a recommender system using YouTube comments and video transcripts as input, and aiming to recommend better comedians to users. The unique aspect of this case is that it is specific to comedy, allowing us to study an important application of text analytics (NLP-based recommender systems) in a given Web-related context.

Recommender systems and current issues

Recommendation systems have been used widely in many applications, but ironically, in most of the cases, the performance of both Collaborative Filtering (CF) and Content-Based Filtering (CBF) are not ideal. The metric for properly evaluating the recommendation has long been neglected, constituting a bad experience on what can be observed in (for example) YouTube recommendations: the recommended videos fail to capture our tastes. To test and improve the established methods (CF and CBF), one option is to incorporate the transcripts and the YouTube comments into a recommender system, set up an objective evaluation metric, and come up with more variations of these classic methods to get better recommendations.

Proposed architecture

Here's a very high-level recommender system architecture. ML stands for machine learning. We will briefly describe some of the components below, but many others will be explored in the case questions.



Text preprocessing is probably one of the most important, and under-estimated, parts of the architecture. Here is a snippet of code that shows what kinds of text-preprocessing can be done in this domain:

```
# Lowercase every words
text = text.lower()
# Remove every words with [blah blah blah] format
text = re.sub('\[.*?\]', '', text)
# Remove every words with (blah blah blah) format
text = re.sub('\(.*?\)', '', text)
# Get rid of the punctuations
text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
# Get rid of all the numbers or words that contain numbers
text = re.sub('\w*\d\w*', '', text)
# Get rid of these specific punctuations
text = re.sub('[‘’“”…]', '', text)
# Get rid of '\n'
text = re.sub('\n', '', text)

# Tokenizes and lemmetizes (or stems) them
tokenized = word_tokenize(text)
stemmed = [porter_stemmer.stem(t) for t in tokenized]
```

Classic Collaborative Filtering

Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

Typically, the workflow of a collaborative filtering system is:

1. A user expresses his or her preferences by rating items (e.g. books, movies, or music recordings) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.

2. The system matches this user's ratings against other users' and finds the people with most "similar" tastes.

3. With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item)

A key problem of collaborative filtering is how to combine and weight the preferences of user neighbors. Sometimes, users can immediately rate the recommended items. As a result, the system gains an increasingly accurate representation of user preferences over time.

Case questions

1. Try to do some research on CF and CBF. What are the key differences and similarities?

2. Looking again at the proposed architecture, what role could Wikidata play in the analysis? Why is it linked to the 'graph' component in analysis?

3. Thinking back to class, describe the pros and cons of using NoSQL and graph databases for this kind of application. Should scale be a factor in making that decision? What if scale wasn't an issue?

4. Are there text preprocessing steps that we missed?

5. What techniques can we use to do sentiment analysis (i.e. name some tools)? Why do you think sentiment analysis is important for getting good recommendations?

6. What techniques can we use for clustering? How to convert the transcripts to feature vectors?

7. We described the classic collaborative filtering in the case. Try to map that to this specific case. Considering the 3-step workflow, how do we apply it to comedy recommendations?