Review Summary System
(Project Documentation)

Cloud Computing - Monsoon 2011

Team - 22

Sandhya S - *201107617*

Dharmesh Kakadia - *201107616*

Manoj Kumar M - *201107502*

Srinath R - *201107625*

IIIT Hyderabad

Review Summary System

Introduction

Many on-line shops and websites allow consumers to express their opinion on products and services they purchased. Although such information can be useful to other potential customers, reading and mentally processing hundreds of such reviews for a single product is tedious and time consuming.

This report discusses a Review summarization system, where customers of a product can read summarized reviews about that product, and thus make decisions in one glance rather than going through all reviews from various sites. The system provides you with the following features :-

1) Product Search

This feature allows users to search for a product, and the system lists all the features of that product, along with their average ratings. On clicking the feature, user can read the reviews which talk about that feature.

2) Feature Search

THis feature allows users to search for a feature and the system lists all the products with that feature along with their average ratings. On clicking the product, user can read the reviews which talk about that product.

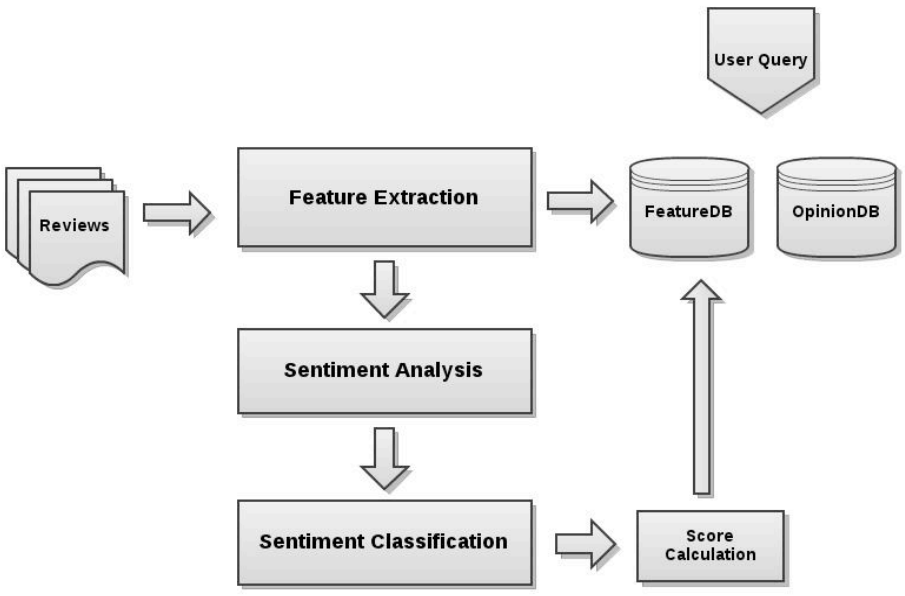
3) Product+Feature Search

This feature allows users to search for a particular feature of a product. The system lists the average rating of that feature for that product.

4) Compare two products

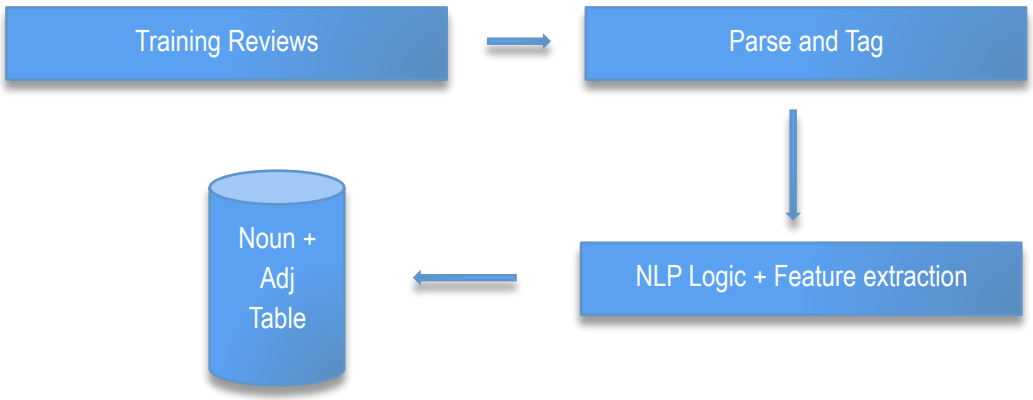
This feature allows users to get comparison of 2 products. The system lists all the common features of the products along with their average ratings in a table so as to make it easy for users to make a decision. THe system also lists all the other feature of the products. On clicking the feature, user can read the exact reviews which talk about that feature.

System Architecture



Feature Extraction/Training Phase

Our system is based on a train and test model. The system is first initially trained to identify features in a particular product category by parsing training reviews of that category. It builds up enough data within the database which it then uses during actual identification of features in review sentences. With this model, it is possible to extend the system to identify and summarize reviews about any product. The only necessity is that the system has to be trained initially with any product category and not a particular product. The entire training architecture can be given by the following figure.



The system's training phase is carried out by the module present in the 'FeatureTrainer' class that encapsulates all the necessary functions and data required for training. The feature trainer's actual working can be given by the following list of steps.

Algorithm for Feature Training

1. The feature trainer is initially given a set of training files for a particular product.

2. The feature trainer loads the 'Stanford parser' module initially.

3. It then reads each line of the input and tags the Parts of Speech in the line.

4. It then performs a complex NLP logic and identifies nouns and associated affecting adjectives from the dependency relation tree constructed by the parser.

5. It then enters all the nouns and corresponding adjectives into the Noun + Adjective table with their associated frequencies within the training file.

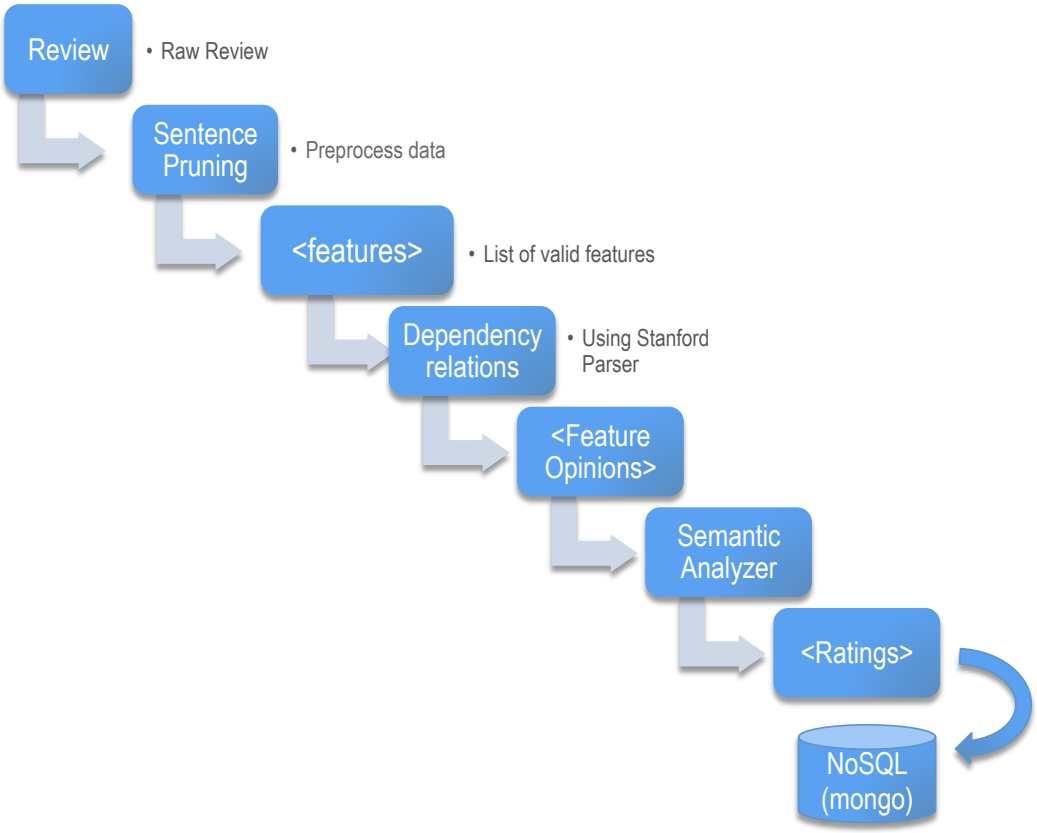
The actual logic behind the storing of the all nouns and adjectival modifiers is that, nouns that occur frequently in user reviews and those that are modified heavily by adjectives have a high probability of being actual features about a particular product. Hence with increase in number of actual training reviews parsed by the system, the system learns a lot about possible features and hence can perform exceedingly well during testing phase. Further, the system is designed to be continuously learning in the sense that, it learns new features while testing also. This makes the system more adaptive to new features added to products in the future because, as a new product is rolled out with a new feature in the future, users are prone to comment about the new feature more and hence the system will learn this quickly and start rating the new features also correctly in the future. This makes our system future proof for all types of products.

Feature Tagging

Now that the valid set of features have been extracted for a particular product, these are used to identify and tag the features present in a particular review sentence.

We have used Stanford parser to tag POS of a sentence as well as output the typed dependancies for each sentence. These typed dependancies are stored for each sentence.

Work Flow



The nouns extracted from the tagged sentence are extracted and validated against the existing features. If they are valid nouns (NN/NNS) then the corresponding opinion is extracted using the typed dependancies. If the noun(NN/NNS) is not a valid feature it is stored in the database and the corresponding frequency is incremented and threshold is adjusted accordingly. This way the system learns and identifies and new feature if the frequency exceeds the threshold.

Step 1:

Tag the sentence and extract the nouns.

Example sentence: *"It has amazing picture quality."*

Tagging:

It/PRP has/VBZ amazing/JJ picture/NN quality/NN ./.

Here the nouns are {picture, quality}

Valid nouns are {picture}

Step 2:

Typed dependancies are extracted.

Typed dependencies:

nsubj(has-2, It-1)
root(ROOT-0, has-2)
amod(quality-5, amazing-3)
nn(quality-5, picture-4)
dobj(has-2, quality-5)

A custom NLP logic is applied and the complete feature and corresponding opinion is extracted from the set of dependancies.

NLP Dependancies

Below are a few typed dependancies used for feature and opinion extraction:

nsubj : nominal subject
A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb, which can be an adjective or noun.

“Clinton defeated Dole” nsubj(defeated, Clinton) “The baby is cute” nsubj(cute, baby)

nn: noun compound modifier
A noun compound modifier of an NP is any noun that serves to modify the head noun. (Note that in the current system for dependency extraction, all nouns modify the rightmost noun of the NP – there is no intelligent noun compound analysis. This is likely to be fixed once the Penn Treebank represents the branching structure of NPs.)

“Oil price futures” nn(futures, oil) nn(futures, price)

amod: adjectival modifier
An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.

“Sam eats red meat” amod(meat, red)

advmod: adverbial modifier
An adverbial modifier of a word is a (non-clausal) adverb or adverbial phrase (ADVP) that serves to modify the meaning of the word.

“Genetically modified food” advmod(modified, genetically) “less often” advmod(often, less)

conj: conjunct
A conjunct is the relation between two elements connected by a coordinating conjunction, such as “and”, “or”, etc. We treat conjunctions asymmetrically: The head of the relation is the first conjunct and other conjunctions depend on it via the conj relation.

“Bill is big and honest” conj(big, honest) “They either ski or snowboard” conj(ski, snowboard)

neg: negation modifier

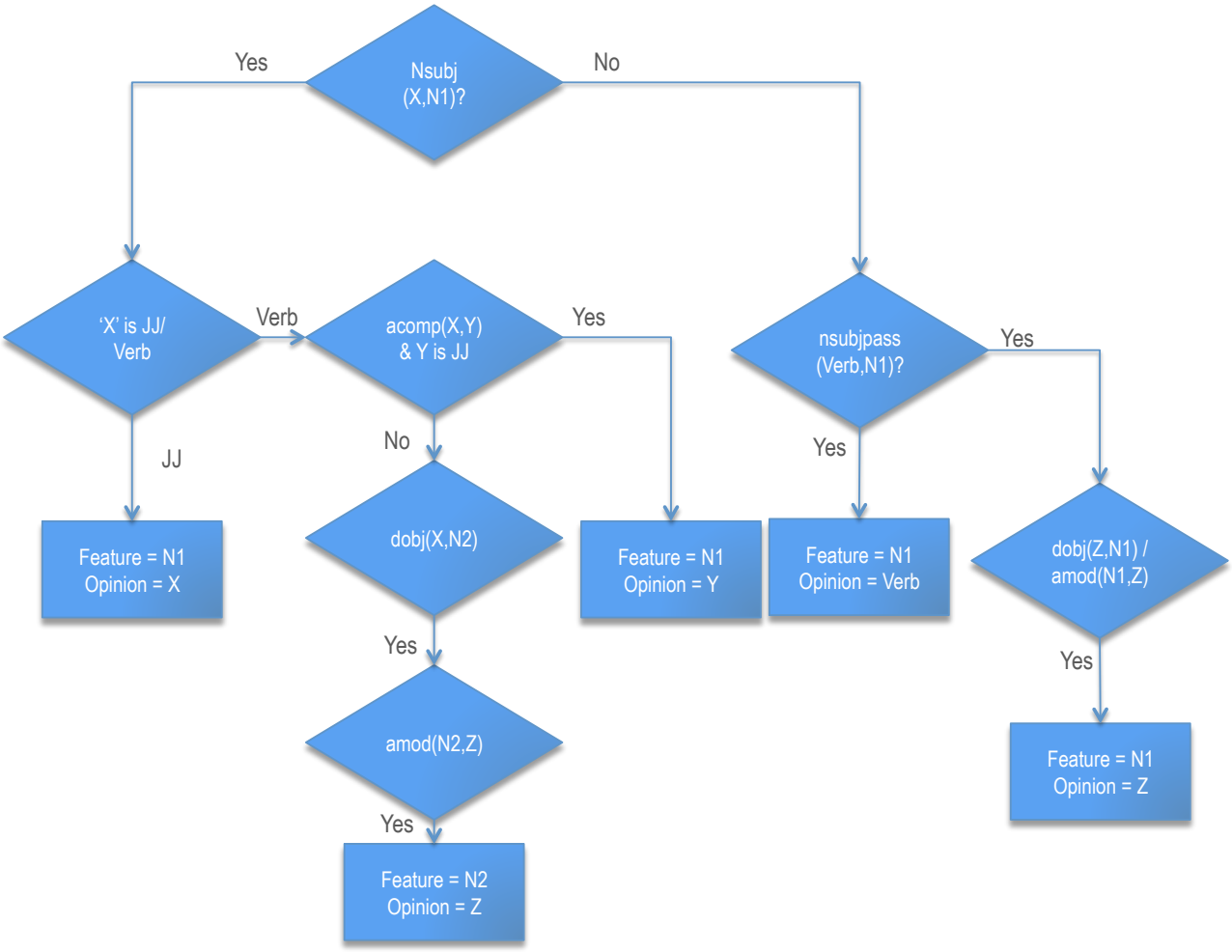
The negation modifier is the relation between a negation word and the word it modifies. “Bill is not a scientist” neg(scientist, not)

“Bill doesn’t drive” neg(drive, n’t)

Step 3: Opinion mining

Using these dependancies, features and corresponding opinions are extracted:

Flow chart for NLP Dependancies:



Once the 'Feature' and 'Opinion' is extracted, they are checked for 'nn' or 'conj' dependancies. This helped us to extract other features or opinions connected with conjunction or advances modifiers.

For the 'Opinion' extracted a check for 'neg' is made to get if the opinion is actually negated.

Rating the review using SentiWordNet

User review rating is a very delicate task as the choice of words to express feelings about a product is very difficult to capture in numbers. The range of words people use to describe their opinion a product can vary between extremes. In our review summary system, user reviews are scored based on the sentence orientation. We have used sentiwordnet to get the positivity or negativity of adjectives that modify the features in each sentence. Also certain nouns themselves might be used in positive or negative connotations and so we have used that score also. The sentiwordnet file is created as an object using the class 'SentiwordnetTable'. We don't load the entire file into memory as we require only the scores, words and corresponding instances for each word. Hence we initially parse the entire sentiwordnet file and load into into a hashmap with the word and instance forming the key and the positive and negative scores being the values. The key is formed using 'word#instance' where word is the actual word and instance is the instance number in the file. Since the same word can have multiple ways it's used in languages, sentiwordnet has scores for all usages of the word. So given a word, the sentiwordnet table class will return all the scores of the word in the table. In our system, we have used the adjectival score

as the boosting factor and the noun's score as the base score since this was a natural method of scoring. The actual method of scoring can be given by the following steps.

Algorithm: Scoring

1. Get the feature from the sentence and find the appropriate adjectives affecting the feature

2. Get the score for the feature and the adjectives together from sentiwordnet table.

3. Calculate the total sum of positive and negative scores for all instances of adjectives within sentiwordnet.

4. Total adjective score is given by difference between total positive score and negative score.

5. Final score is given by 'noun score x total adjective score'.

6. If the noun score is zero, then final score is just the total adjective score.

7. If there is a negative modifier for the entire sentence modifying the adjective, then the total score is inverted i.e. multiplied by a -1.

8. The final score is then multiplied by 100 to give a score in the range of [-100 – 100] and if the scores exceed this limit, they are clamped.

We have used a scoring system in the range of [-100 – 100] because it makes the actual scoring more natural giving the user a way to effectively compare two products or features.

Database Schema - Design and Description

We have used MongoDB as our NoSql database. MongoDB is an open source, document-oriented database designed with both scalability and developer agility in mind. Instead of storing your data in tables and rows as you would with a relational database, in MongoDB you store JSON-like documents with dynamic schemas. The goal of MongoDB is to bridge the gap between key-value stores (which are fast and scalable) and relational databases (which have rich functionality).

The following are the advantage of using MongoDB as our Database.

1. Highly Scalable

2. Highly Available

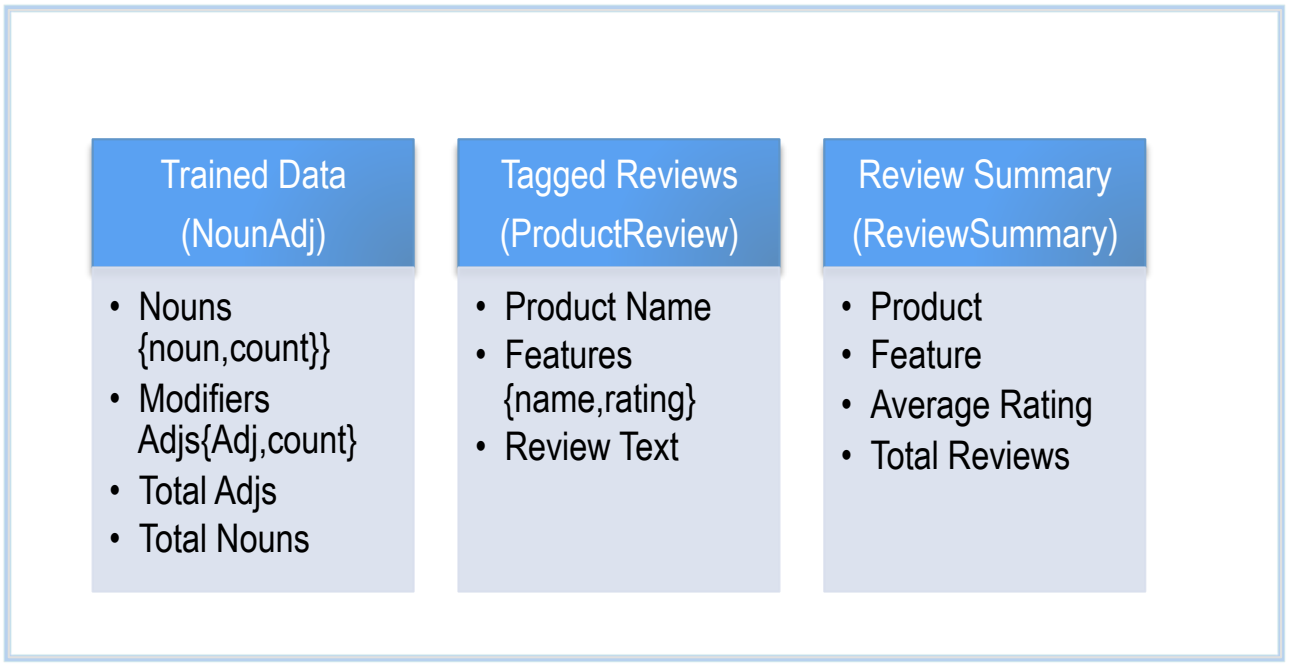
3. Document Oriented

4. Database is exposed as Rest services

5. Easy to manage

6. Large Community Support

For the effective use of NoSql database one has to have a very good database schema. We have designed our database schema keeping in mind scalability and access pattern by different sub-processes in our project. Our database schema can be represented as shown in figure 1.



For each product category we are maintaining 3 Collections in the database.

NounAdj – Stores the trained data for the Review-Classifier.
ProductReview – Stores the complete information about the review, including the complete review text.
ReviewSummary – Stores the summary for the product that includes the average rating for that product and feature and other statisitcs like total review counts, etc.
Few features of the this design are

The database is independent of the product category as is our other project components. So it can be easily extended to other products.

The only ReviewSummry collection is accessed by the client in most senarios, until user asks for the complete review text. And the review tagger access the ProdcutReview collection, making this to more or less independent and improving the response time for both.

Sample document from each of the collection is shown below

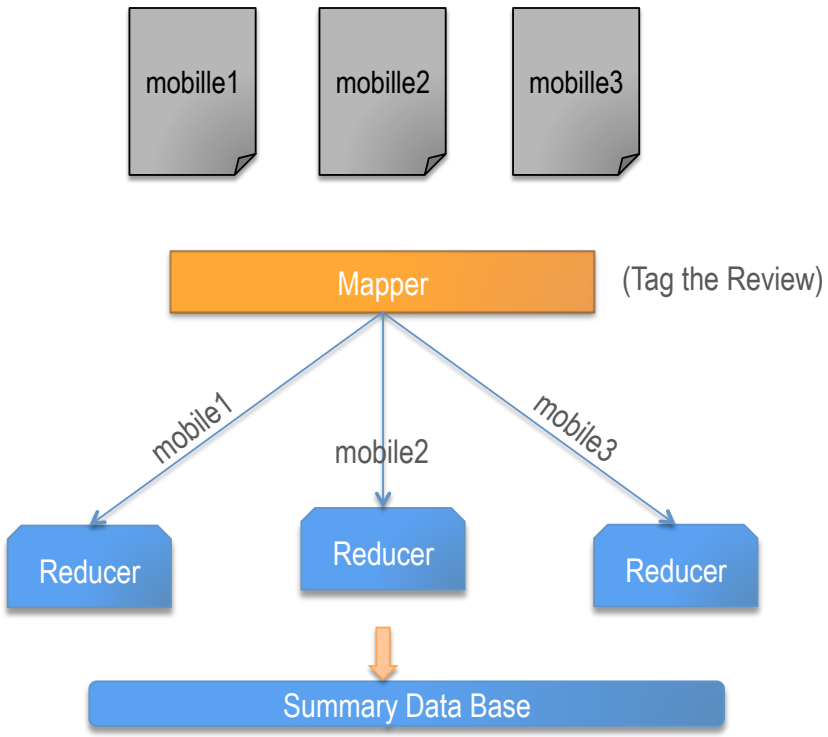
```
> db.ProductReview.findOne()
{
        "_id" : ObjectId("4ecb8ab7e4b0bdabbb039531"),
        "name" : "CanonS100",
        "review" : "I want to start off saying that this camera is small for a reason.",
        "features" : [
                {
                        "name" : "camera",
                        "rating" : -95
                }
        ]
}


> db.ReviewSummary.findOne()
```

```
{
	"_id" : ObjectId("4ecb8ab7e4b0bdabbb039530"),
	"product" : "CanonS100",
	"feature" : "camera",
	"totalReviews" : 120,
	"avgRating" : 3.549999999999998
}

> db.NounAdj.findOne()
{
	"_id" : ObjectId("4ecb4a24e4b08e3cc1dddfdd"),
	"nouns" : [
		{
			"noun" : "form",
			"count" : 2
		}
	],
	"adjs" : [
		{
			"adj" : "ever tested",
			"count" : 1
		}
	],
	"totalNouns" : 2,
	"totalAdjs" : 1
}
```


Parallelizing with HADOOP



Mapper

Mapper is given a input of text files containing reviews of each product. Each file contains the reviews of a particular product and is named accordingly.

Example: 'CanonSD500' - Contains reviews of CanonSD500.

Function of mapper is to read each sentence of the review, tag the sentence, wrap it in a serialized object(named 'Review') and send the object to reducer.

Mapper:

input (key, value) = (offset, review text)

output (key, value) = (product_name, Review object)

'Review' is a serialized object which contains:

1. Review sentence
2. Feature list
3. Rating
4. Product_name

Reducer

Reducer deserializes the 'Review' object and stores the object into Mongo database.

The summarized ratings for the corresponding product features is also updated.

Tools used and External dependancies

1. NLP

- Stanford Parser
- Wordnet (Synonyms)
- Sentiwordnet

2. Cloud Framework : Apache Hadoop 20.2 (http://hadoop.apache.org/)

3. NoSql Database : MongoDB 2.0.1 (http://www.mongodb.org/)

Hadoop Cluster Details

We have used the cluster of 3 nodes. Each had the following configuration.

- OS : Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)
- Processor : Intel(R) Core(TM)2 Duo CPU-E7500 @ 2.93GHz
- Memory : 2 GB

Future Enhancements

- Dependance relationships can be further analyzed for improved feature and opinion extraction.
- Synonym match can be improved with Wordnet::Similarity for better grouping of similar features.
- Database access can be further optimized for blazing performance of the system.
- Preprocessing of user query can be performed for better results.