SEMANTIC CLUSTERING OF QUESTIONS

Maria-Cătălina MOCANU
University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers, Computer Science Department
catalina.mocanu@gmail.com

Supervisors:
Assistant Prof. Andrei OLARU, Professor Adina Magda FLOREA
University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers, Computer Science Department
cs@andreiolaru.ro, adina@cs.pub.ro

Abstract

Nowadays, one main area of focus in natural language processing is information retrieval. Whether we talk about clustering and filtering a set of documents on a specific topic or finding the answer to a question related to a given text, information retrieval is mainly about selecting from large digital datasets the information relevant to the user. In other words, humans ask the questions and computers answers. How about the case when a person has to answer a large set of questions? This is where the idea of semantic clustering of questions steps in. Since large question datasets may contain many similar questions and extracting only the relevant ones is a time consuming task for a person, computer computational power is needed for clustering questions based on their lexical and semantic similarity. We will discuss some theoretical aspects, the tools we used, an architecture overview and results.

Key words: natural language processing, unsupervised learning, document clustering, question answering, semantic distances

1. Introduction

Semantic clustering of questions is another way of bringing the benefits of natural language processing algorithms in our everyday life. It is a field related to information retrieval techniques such as document clustering and question answering.

The most useful application of such semantic clustering algorithms is in real time applications when a large number of questions are asked and there's a limited time for a person to answer them. Let's image two scenarios: an online video conference with a famous person and a presentation held at a congress.

In the first case, people around the world can see the videoconference and interact with the guest using questions sent via social network such as Facebook, Twitter or Google+ or on the site dedicated to the videoconference. The number of questions can be overwhelming. Thus, selecting the questions the guest answers can be a real challenge. Furthermore, sometimes very similar questions can hide other original questions between them. A semantic question clustering application is able to cluster all these questions into clusters based on their semantic similarities. This means that less work is done by the interviewer since most relevant questions are found by the clustering algorithm.

In the other example, let's image that every participant at the conference has a smartphone with an application for uploading questions regarding the current presentation. At the end of the

presentation, in the Q&A section on the screen a clustering of all questions may appear and a highlight on the most asked question. In this way, the presenter can better understand his audience's main interests concerning the presentation, what needs to be explained again, etc. For the audience the benefit is obvious: everyone can ask a question with no time limit.

2. Sections

In the next section, we present Smart Presentation, a project developed on Android that requires question clustering.

Section 4 deals with the problem of clustering; presenting the theoretical aspects of hierarchical clustering and the tools we used to implement it. Similarly, section 5 discusses about some tools for NLP pre-processing of text. Section 6 presents the basic semantic distances between words using WordNet taxonomy.

In section 7, we present our solution – overall architecture and a brief description of classes. Section 8 is dedicated to the evaluation of results and section 9 presents the conclusions.

3. Smart Presentation

Let's consider the following scenario. Dan is holding a presentation in front of his colleagues in the AI course. As he goes through the slides, the students in the audience are able to use their Android devices to navigate independently through the presentation and they can make annotations or write questions. His colleague Alice is unclear on why Dan is using a certain mathematical formula in a slide, so she wants to ask about that, but she sees in the feedback interface that Ben has already created a question about that. So Alice just needs to "plus one" Ben's question. At all times, Dan can just give a quick look on his Android smartphone to see an aggregated view of the annotations and questions, based on semantic similarity and user's reputation. When it is time for questions, he can see the slides with the most questions, which makes it easier for him to make the answers clearer and more detailed. After the presentation, he will be able to see a complete view on the feedback for the presentation, helping him improve it for the next time.

This friendly way of interaction at a conference / lecture is what the Smart Presentation application intends to bring to our daily academic life. In this context, the problem of question clustering is better defined. We are now interested in offering a solution that suits the real time response needs of this scenario: we have to balance accuracy in determining clusters with the computational time needed.

4. Clustering problem

4.1. Hierarchical clustering algorithm – theory

Hierarchical clustering builds a cluster hierarchy or, in other words, a tree of clusters, also known as a dendrogram. Every cluster node contains child clusters; sibling clusters partition the points covered by their common parent. Such an approach allows exploring data on different levels of granularity. A clustering of the data objects is obtained by cutting the dendrogram at the desired level, than each connected component forms a cluster.

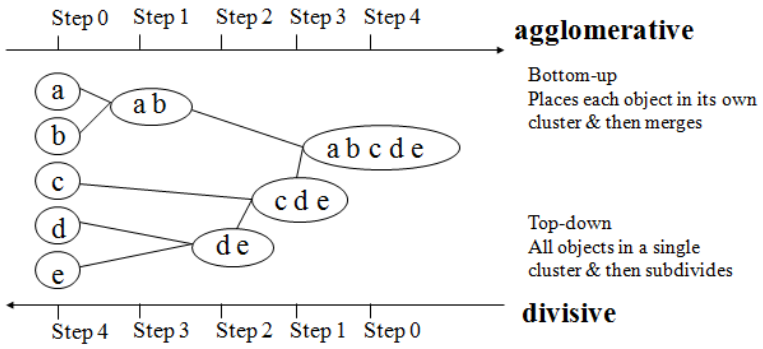


Fig.1 Example of hierarchical clustering

Hierarchical clustering methods are categorized into agglomerative (bottom-up) and divisive (top-down). An agglomerative clustering starts with one-point (singleton) clusters and recursively merges two or more most appropriate clusters. A divisive clustering starts with one cluster of all data points and recursively splits the most appropriate cluster until a stopping criterion is achieved.

Advantages of hierarchical clustering include: flexibility regarding the level of granularity, meaning that unlike some flat clustering algorithms such as K-means which require the number of clusters to be specified in advance, in the case of hierarchical clustering the dendogram can be cut at any level, obtaining different number of clusters. This is very useful in our case since we cannot know the number of clusters of our corpus would best be divided into. Another advantage is the ease of handling different forms of similarity and distance thus it can be used with many attribute types. In our case, semantic similarity measures will be used.

Like with any other clustering algorithm, hierarchical clustering has also disadvantages: first, the cost efficiency and secondly, the fact that most hierarchical algorithms do not revisit once constructed (intermediate) clusters with the purpose of their improvement as the EM algorithm. Hierarchical clustering is often portrayed as the better quality clustering approach, but is limited because of its quadratic time complexity.

To merge (agglomerative hierarchical clustering) or split (divisive hierarchical clustering) current clusters of points, the distance between clusters has to be obtained from the distance between individual points. Such derived proximity measure is called a linkage metric. In hierarchical clustering the following metrics are used:

- *single-link clustering*
  $$\max\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}.$$
- *complete-link clustering*
  $$\min\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}.$$
- *group-average clustering*
  $$\frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x,y).$$

In *single-link clustering* or *single-linkage clustering*, the similarity of two clusters is the similarity of their *most similar* members. This single-link merge criterion is *local*. We pay attention solely to the area where the two clusters come closest to each other. Other, more distant parts of the cluster and the clusters' overall structure are not taken into account.

In *complete-link clustering* or *complete-linkage clustering*, the similarity of two clusters is the similarity of their *most dissimilar* members. This is equivalent to choosing the cluster pair whose merge has the smallest diameter. This complete-link merge criterion is non-local; the entire structure of the clustering can influence merge decisions. This results in a preference for compact clusters with small diameters over long, straggly clusters, but also causes sensitivity to outliers. A single document far from the center can increase diameters of candidate merge clusters dramatically and completely change the final clustering.

Single-link and complete-link clustering reduce the assessment of cluster quality to a single similarity between a pair of documents: the two most similar documents in single-link clustering and the two most dissimilar documents in complete-link clustering. A measurement based on one pair cannot fully reflect the distribution of documents in a cluster. It is therefore not surprising that both algorithms often produce undesirable clusters. On the one hand, single-link clustering can produce the chaining effect by extending a chain of points without regard to the overall shape of the emerging cluster due to the fact that merge criteria is local. On the other hand, complete-link clustering pays too much attention to outliers, points that do not fit well into the global structure of the cluster.

Group-average agglomerative clustering evaluates cluster quality based on *all* similarities between objects, thus avoiding the pitfalls of the single-link and complete-link criteria, which generalize cluster similarity from the similarity of a single pair of objects. Average-link (or group average) clustering is a compromise between the sensitivity of complete-link clustering to outliers and the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.

Fig. 3 depicts the time complexity of the agglomerative hierarchical clustering algorithm depending upon linkage metric.

method	combination similarity	time compl.	optimal?	comment
single-link	max inter-similarity of any 2 docs	$\Theta(N^2)$	yes	chaining effect
complete-link	min inter-similarity of any 2 docs	$\Theta(N^2 \log N)$	no	sensitive to outliers
group-average	average of all sims	$\Theta(N^2 \log N)$	no	best choice for most applications

Fig. 2 Time complexity of HAC

4.2. Tools – LingPipe

LingPipe is tool kit for processing text using computational linguistics. It provides an API for various tasks such as:

- part of speech tagging, named entity recognition, spelling correction
- word sense disambiguation, sentiment analysis
- classification, logistic regression
- string comparison
- clustering, expectation maximization

We used only the clustering API offered by this library. Since it was designed especially for text clustering, the API was easier to use than the WEKA API. We had to implement an interface (Distance<E>) with our measure of similarity which will be presented in the next sections.

LingPipe provides the two standard aggolomerative clustering algorithms, single-link and complete-link clustering. On the one hand, single-link clustering considers the distance between clusters to be the minimum distance between elements of the clusters, thus creating highly separated clusters. On the other hand, complete-link clustering computes the maximum distance between clusters, thus leading to more tightly centered clusters. In practice, complete link clustering tends to be more useful, though it is more expensive to compute.

The result of the clustering implemented in LingPipe is a set of clusters that forms a partition of the input set. A set of sets is a partition of another set if each element of the set is a member of exactly one set of the partition. In mathematical terms, the sets making up a partition are pairwise disjoint and their union is the original set.

Let's assume the following example:

```
{ "a", "aaa", "aaaaaa", "aaaaaaaaaa" }
```

We will use for the purpose of pointing out what is the difference between the two linkage metrics (single-link and complete-link) a very simple distance to compute the difference between two strings: the edit distance. The edit distance consists in the number of characters that need to be added/replaced or deleted from one string in order to transform it in the other. For our example, for the strings containing only letter "a" or only letter"b", the distance is the difference in length - dist ("aaa", "aaaa") = 2. For a string containing letter "a" and one containing letter "b" the distance is the length of the biggest string - dist("aaa","bbbb") = 4.

Here are the dendrograms produced by the complete-link and single-link clusterers.

Complete- and Single-Link Clusterings	
Complete Link	Single Link

9.0
  4.0
  aaaaaa
  aaaaaaaaaa
  2.0
  aaa
  a
	4.0
	  3.0
	  aaaaaa
	  2.0
	  aaa
	  a
	  aaaaaaaaaa

Note that the clusterings are different trees. The single-link clustering is deeper, whereas the complete link is more balanced.

The numbers that connect the elements of a cluster or the subclusters represent in the case of complete link, the maximum distance between the elements in the clusters that it includes – for our example 9.0 is the distance between "a" and "aaaaaaaaaa". In the case of single link, the number represents the minimum distance between the clusters elements – for our example, the distance 4.0 corresponds to the elements "aaaaaaaaaa" and "aaaaaa".

Here are the steps taken by single-link clustering:

```
a  aaa  aaaaaa  aaaaaaaaaa
{a aaa}:2  aaaaaa  aaaaaaaaaa
{{a aaa}:2  aaaaaa}:3  aaaaaaaaaa
{{{a aaa}:2 aaaaaa}:3  aaaaaaaaaa}:4
```

Note that aaaaaa is linked to {a aaa} at distance 3, which is the distance to aaa, not at distance 5, which is the distance to a. Similarly the final linkage of aaaaaaaaaa to the cluster of the smaller strings is at distance 4, its distance to aaaaaa, the smallest distance among the distances to a, aaa and aaaaaa.

Here are the steps taken taken by complete-link:

```
a  aaa  aaaaaa  aaaaaaaaaa
{a aaa}:2  aaaaaa  aaaaaaaaaa
{a aaa}:2  {aaaaaa  aaaaaaaaaa}:4
{{a aaa}:2  {aaaaaa  aaaaaaaaaa}:4}:9
```

The main difference is that aaaaaa is linked to aaaaaaaaaa instead of to the cluster {a aaa}. That's because the maximum distance of aaaaaa to an element of {a aaa} is 5, whereas the distance between aaaaaa and aaaaaaaaaa is only 4. The final link is at distance 9, which is the maximum distance between elements in the two clusters, specifically between a and aaaaaaaaaa.

There are two ways to extract standard clusterings from dendrograms:
1) Set a distance bound and maintain every cluster formed at less than or equal to that bound. This is equivalent to cutting the standard dendrogram notation at a specified height.

```
Set<Set<String>> clKClustering = clDendrogram.partitionDistance(maxDistance);
```

2) Continue cutting the highest distance cluster until a specified number of clusters is obtained. The following code prints all the possible clustering from one single cluster for all input data to a specific cluster for every string:

```
for (int k = 1; k <= clDendrogram.size(); ++k) {
    Set<Set<String>> clKClustering = clDendrogram.partitionK(k);
    System.out.println(k + "  " + slKClustering);
}
```

5. Tokens, POS tagging and lemmas

5.1. Definitions

In every natural language application, the pre-processing of data plays a very important role in the results of the application. Fist, the text must be divided in sentences, them the sentences in tokens. This is a rather simple task for English language: words are delimited by spaces, but in other languages like German or Chinese, it represents a serious problem.

Part-of-speech tagging is very important since it enables the programmer to extract only the words that are more relevant to the task, or to give certain weights to different part of speech. For example, we are more interested if nouns and verbs are semantically similar than we are about adjectives or adverbs.

Lemmatization is a key tool in computing the distances between two texts. It is used to reduce inflections or variant forms to base form:

- am, are ,is -> be
- car, cars, car's, car'-> car
- the boy's toys are different color -> the boy toy be different color

A lemma is actually the dictionary headword form of a word.

5.2. Tools: Stanford Standard CoreNLP

Stanford CoreNLP provides a set of natural language analysis tools which can take natural language English language text as input and return the base forms of words (lemmas), their parts of speech (POS tagging), whether they are names of companies, people, etc., dates, times, and numeric quantities (named entity recognition), and the dependencies in terms of words and phrases in the structure of sentences, by indicating which noun phrases refer to the same entities (coreference resolution system). CoreNLP is free software available under the GNU General Public License.

This tool is easy to use: with a single option you can change which tools should be enabled and which should be disabled by adding a new annotator in the StanfordCoreNLP object's "annotators" property. The code below shows how to create a Stanford CoreNLP object with various annotators for tokenizing, part-of-speech tagging, named entity recognition and parsing.

```
    // creates a StanfordCoreNLP object, with POS tagging, lemmatization,
NER, parsing, and coreference resolution
    Properties props = new Properties();
    props.put("annotators", "tokenize, ssplit, pos, lemma");
    StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
```

At the base of the CoreNLP stand two classes: Annotation and Annotator. Annotations are the data structures which hold the results of annotations, while annotators are some kind of functions that they operate over Annotations instead of Objects. They do things like tokenize, parse, or NER tag sentences and might be dependent upon other annotators in order to be used. For the StandardCoreNLP created with the properties above, this is the code for using specific Annotation classes from the output of the text processing.

```
    for(CoreMap sentence: sentences) {
      // traversing the words in the current sentence
      // a CoreLabel is a CoreMap with additional token-specific methods
      for (CoreLabel token: sentence.get(TokensAnnotation.class)) {
        // this is the text of the token
        String word = token.get(TextAnnotation.class);
        // this is the POS tag of the token
        String pos = token.get(PartOfSpeechAnnotation.class);
        // this is the NER label of the token
        String ne = token.get(NamedEntityTagAnnotation.class);
      }
```

6. Semantic distance

6.1. Basic semantic distance between words

When deriving a similarity measure between two texts, the structure of the text should be taken into account, but we first present a rough model for the semantic similarity of texts as a function of the semantic similarity of the component words. Several measures based on WordNet information were proposed and investigated by computational linguists in the late 1990s.

Next, we present six knowledge-based measures of word semantic similarity that work well on the WordNet hierarchy. All these measures assume as input a pair of concepts, and return a value indicating their semantic relatedness. WordNet provides also a package to compute these metrics.

- **Leacock & Chodorow similarity** (Leacock & Chodorow 1998)

The edge-based approach proposed by Leacock and Chodorow computes some transform of the number of edges (links) between concepts in a graph such as that of the taxonomy in Figure 4. The concepts of *mouse* and *rat* are more similar than that of *mouse* and *cat*.
The formula used:

$$Sim_{lch} = -\log \frac{length}{2 * D}$$

where *length* is the length of the shortest path between two concepts using node-counting, and *D* is the maximum depth of the taxonomy.

- **Lesk similarity** (1986)

The Lesk similarity of two concepts is defined as a function of the overlap between the corresponding definitions, as provided by a dictionary.

- **Wu and Palmer similarity** (Wu & Palmer 1994)

The Wu and Palmer similarity metric measures the depth of two given concepts in the WordNet taxonomy, and the depth of the least common subsumer (LCS), and combines these figures into the following similarity score:

$$Sim_{wup} = \frac{2 * depth(LCS)}{depth(concept_1) + depth(concept_2)}$$

- **Resnik similarity** (Resnik 1995)

It is a node-based approach which begins with the observation that the information content of a concept depends on its location within the graph. More precisely, the lower the probability of encountering a concept, the higher is its information value.
The measure introduced by Resnik returns the information content (IC) of the LCS of two concepts:

$$\boldsymbol{Sim_{res} = IC(LCS)}$$

where IC is defined as:

$$\boldsymbol{IC(c) = -\log P(c)}$$

and *P*(*c*) is the probability of encountering an instance of concept *c* in a large corpus.

In our example, the concept *entity* is associated with all things, so its information value is zero. The concept *rodent* is infrequently encountered in the taxonomy, so its information value is higher (than *entity* and higher than *mammal.* Because *entity* occurs in each pathway in the taxonomy, its probability is 1, and its IC is 0. *Mammal*'s IC would be considerably higher, and *rodent*'s IC higher still.

- **Lin similarity** (Lin 1998)

The metric introduced by Lin builds on Resnik's measure of similarity, and adds a normalization factor consisting of the information content of the two input concepts:

$$Sim_{lin} = \frac{2 * IC(LCS)}{IC(concept_1) + IC(concept_2)}$$

- **Jiang &Conrath similarity** (Jiang & Conrath 1997)

The method involves a combination of edge-based and node-based approaches. The denomitor is the sum of the informational difference between each of the two concepts and their lowest superordinate.

$$Sim_{jnc} = \frac{1}{IC(concept_1) + IC(concept_2) - 2 * IC(LCS)}$$

Note that all the word similarity measures are normalized so that they fall within a 0–1 range. The normalization is done by dividing the similarity score provided by a given measure with the maximum possible score for that measure.

Several tests have shown that the node-based approach (Resnik) is superior to the edge-based approach (Leacock & Chodorow) and that, in turn, the combined measure (JCN) is superior to the node-based approach.

Budanitsky and Hirst compared five measures of semantic distance computed from WordNet with respect to their ability to (accurately) nominate words as malapropisms. Malapropisms are properly spelled words that appear in the wrong contexts. For example, in an essay on dairy farming, misspelling dairy as diary would be a malapropism. The combination (JCN) measure was significantly more accurate at detecting malapropisms than any of the other measures.

Thus, the available evidence combines to nominate the JCN measure as the one with the most potential as a measure of semantic distance in WordNet.

6.2. Tools: WordNet

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The main relation among words in WordNet is synonymy, as between the words shut and close or car and automobile. Synonyms--words that denote the same concept and are interchangeable in many contexts--are grouped into unordered sets (synsets).

Each of WordNet's 117659 synsets (or nodes) is linked to other synsets by means of a small number of "conceptual relations". Each node contains one or more synonymous words (*synsets*). The largest group of nodes (69%) contain nouns; the remaining synsets contain adjectives (16%), verbs (12%), and adverbs (3%). The conceptual relations between synsets are as follows:

Nouns

- *hypernyms*: *Y* is a hypernym of *X* if every *X* is a (kind of) *Y* (*canine* is a hypernym of *dog*)
- *hyponyms*: *Y* is a hyponym of *X* if every *Y* is a (kind of) *X* (*dog* is a hyponym of *canine*)
- *coordinate terms*: *Y* is a coordinate term of *X* if *X* and *Y* share a hypernym (*wolf* is a coordinate term of *dog*, and *dog* is a coordinate term of *wolf*)
- *holonym*: *Y* is a holonym of *X* if *X* is a part of *Y* (*building* is a holonym of *window*)
- *meronym*: *Y* is a meronym of *X* if *Y* is a part of *X* (*window* is a meronym of *building*)

Verbs

- *hypernym*: the verb *Y* is a hypernym of the verb *X* if the activity *X* is a (kind of) *Y* (*to perceive* is an hypernym of *to listen*)
- *troponym*: the verb *Y* is a troponym of the verb *X* if the activity *Y* is doing *X* in some manner (*to lisp* is a troponym of *to talk*)

- *entailment*: the verb *Y* is entailed by *X* if by doing *X* you must be doing *Y* (*to sleep* is entailed by *to snore*)
- *coordinate terms*: those verbs sharing a common hypernym (*to lisp* and *to yell*)

Adjectives

- *related nouns*
- *similar to*
- *participle of verb*

Adverbs

- *root adjectives*

The noun synsets are linked by relationships, notably hypernymy–hyponymy (super- and subordinates; *is-a*) and meronymy–holonymy (*has-a* and *is-part-of*). Figure 4 shows a part of the WordNet taxonomy linking *mouse*, *rat*, and *cat*. *Mouse* and *rat* are both rodents; the feline *cat* is a carnivore. The more general taxonomy, beginning with *placental mammal*, is common to both *rodent* and *carnivore* categories. The root node, *entity*, subsumes all other things in the taxonomy.

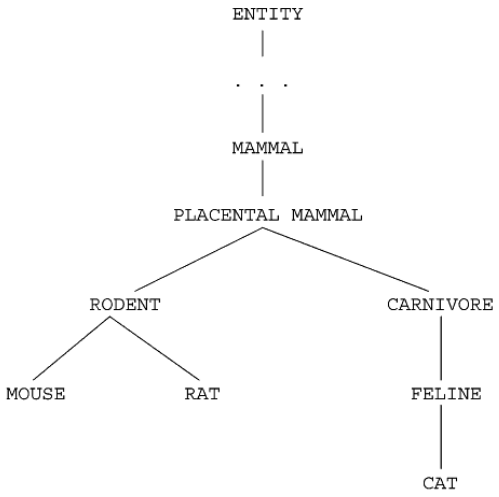


Fig. 3 An example of WordNet taxonomy

7. Solution design

7.1. Overall architecture

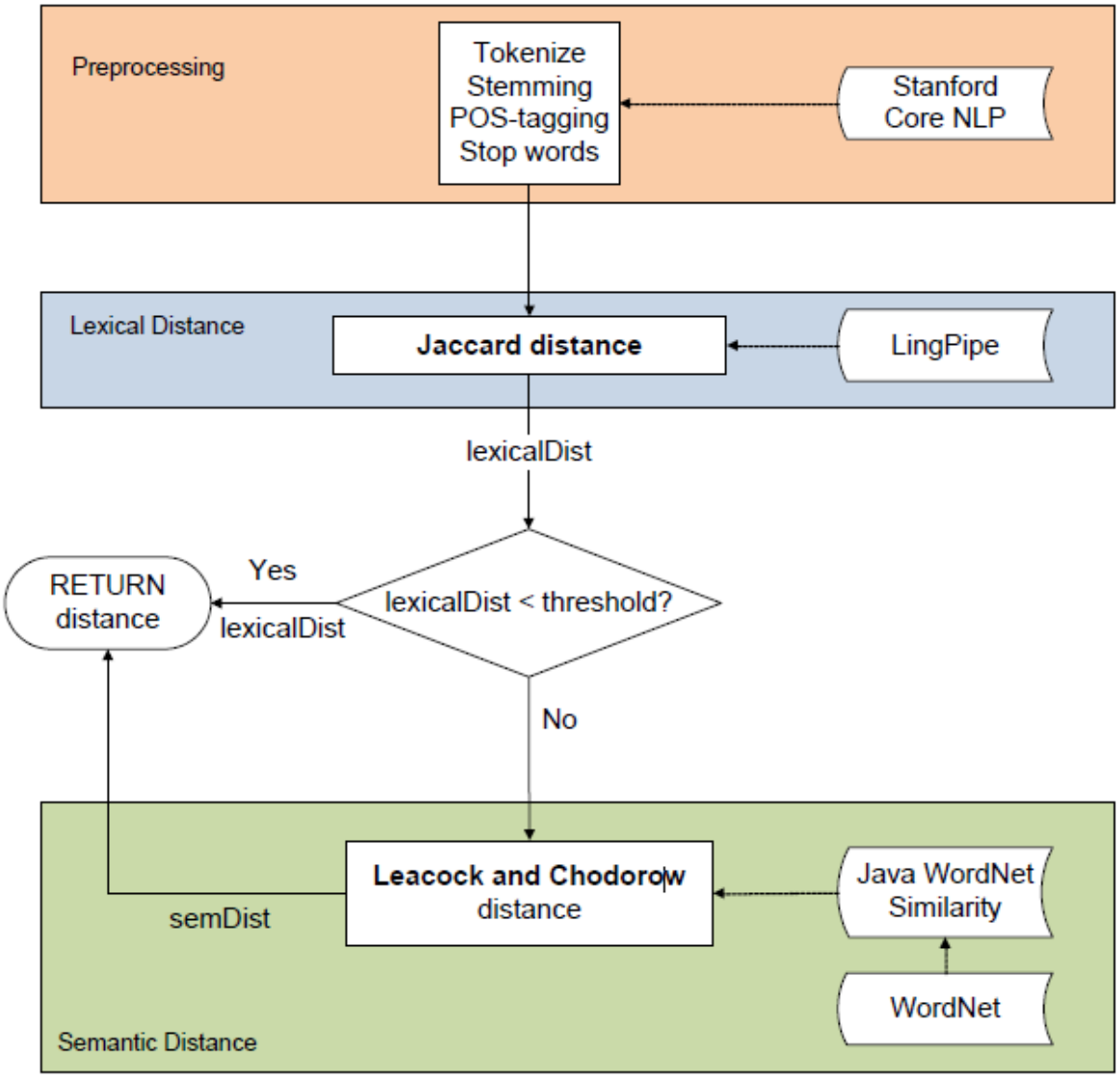
The architecture of our solution is quite simple and straightforward as we define the logical components of the system in close relation to the steps in our algorithm.

Like in any other natural language processing application, we include a pre-processing stage. At this level, we use StandardCoreNLP to extract the relevant features of the questions we intend to cluster. Relevant information is obtained by tokenizing a question, removing punctuation marks and prepositions (those that aren't part of a phrasal verb) from it and extracting specific parts-of-speech and generating the lemma for them. The exact information that we save after this process is presented in the "Input/output format" section.

The next stage is represented by the clustering itself and it involves the complex aspect of computing a relevant distance between two questions that takes into account both lexical and semantic similarity. This brings us to our next two components: the lexical distance, which is mandatory, and the semantic distance computation, which is optional.

The idea is that sometimes, two questions can be so much lexically related that it is very impractical to compute the semantic similarity since semantic distance computation involves time consuming passes along big graphs in the WorldNet taxonomy. The lexical distance computed by Jaccard distance is a number between 0 and 1, where 0 means very similar and 1 not at all similar. So we defined a threshold used to control whether semantic distance should also be computed or not: every pair of questions that have a higher lexical distance than the threshold should have their semantic similarity computed equally. This approach although is every efficient can be quite error prone in the case of very lexically similar questions that are very semantically different. For example: "Who killed Brutus?" and "Who did Brutus kill?".

As seen in the picture, the semantic similarity we use to compare words is based on WordNet's Leacock & Chodorow similarity. More details about the distances used in the next sections.



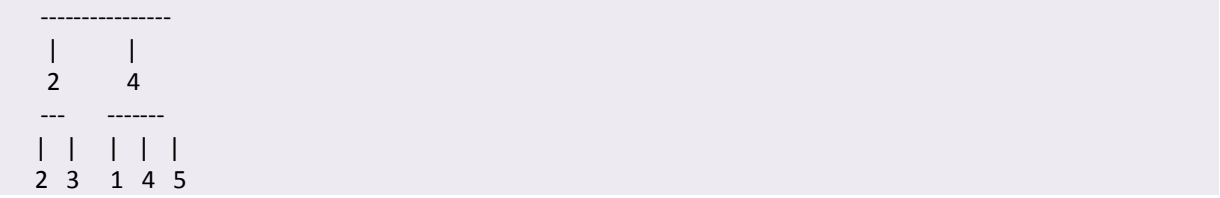
7.2. Input/Output format – Question class

The LingPipe library enables us to use any kind of class as input for the hierarchical clustering algorithm as long as our class contains some form of text and we define a distance to deal with our kind of input.

We chose to create a class named Question that includes the following fields:

- `String originalText` – the original text of the question
- `String parsedText` – the text of the question containing the lemmas of the original words
- `ArrayList<CoreLabel> allTokens` – the tokens resulted from tokenizing the question using the CoreNLP with annotators set to "tokenize, ssplit, pos, lemma,parse"
- `ArrayList<CoreLabel> filteredTokens` – the tokens that are nouns, verbs or adjectives
- `ArrayList<String> nouns` – list of lemmas for nouns in the question
- `ArrayList<String> verbs` – list of lemmas for verbs + phrasel verbs in the question
- `ArrayList<String> adjectives` – list of lemmas for verbs in the question
- `ArrayList<String> lemmas` – list of all lemmas in the sentences (except for lemmas coresponding to punctuation marks)
- `ArrayList<Double> propDistance;`
- `ArrayList<String> stopVerbs` – contains auxiliary verbs such as “be”, “do”, “have”, in order to avoid putting them in the verb list many times;
- `int id` – the general id of the question
- `int setId` – the id of the set the questions belong to. In the SmartPresentation application we are interested in clustering the questions belonging to a specific slide in the presentation.

The output of the LingPipe clusterer is a dendrogram represented as a set of different clustering possibilities, where each clustering possibility is represented as a set of clusters, every cluster being itself a set of questions. In the SmartPresentation framework, the question clustering application works on the server side and frequently receives requests of clustering from mobile devices. Since efficiency in terms of speed and bandwidth are essential, the output dendrogram has to be modified into a more lightweight data structure. Our idea was to send the resulting clustering as a tree with two levels: the first level corresponds to the ids of the most relevant questions for each cluster, while the second level contains for each of the node on the first level, the ids of the questions that are in the same cluster as them. For example, if after cutting the dendrogram at a specific cost, we obtained the following clusters (each question is represented by its id): { 1, 4, 5} and {2, 3} and let’s consider that question 4 is the most relevant for cluster 1, and 2 for cluster 2, the resulting tree would look like this:



7.3. QuestionHierClusterer class

It is the most important class of the hierarchical clustering process. The QuestionHierClusterer class provides the means to transform a set of Question objects into a dendrogram which can be later processed into a clustering solution represented as a two level tree. When initializing the hierarchical clustering object, two important variables must be specified: the maximum distance accepted between the elements of the same cluster and the distance used for comparing objects in the dataset.

7.4. LexicalDistance class

The LexicalDistance class implements the Distance<E> interface in LingPipe and offers a complex computation of the distance between two Question objects.

As presented in the overview of the architecture, the lexical distance used is the Jaccard distance, implemented also by LingPipe. The Jaccard distance implementation in LingPipe operates at a token level, comparing two strings by first tokenizing them and then dividing the number of tokens shared by the strings by the total number of different tokens in the two sentences. The lexical distance computed by Jaccard distance is a number between 0 and 1, where 0 means very similar and 1 not at all similar. Since Jaccard considers that two words match, only if they are identical, the input texts for this distance should be the texts resulted in the pre-processing stage by replacing every word with its lemma.

So for the following questions:
Does your sister love cats? -> sister love cat
Do you love your sister -> love sister

The Jaccard distance would be: 2/5 = 0.4 -> highly similar

7.5. SemanticDistance class

The SematicDistance class uses the WordNet Leacock & Chodorow similarity, present in the Java WordNet::Similarity package. When computing the similarity between two questions, only the verbs and nouns are taken into account. The similarity between two questions is computed as the mean of the nouns' similarity and the verbs'similarity.

The formula used to compute the similarity for nouns, respectively for verbs is:

$$sim_{sem} = \frac{1}{2}\left(\frac{\sum_{ai \in Q_1} \max\ ssim\ (a_i, Q_2)}{|Q_1|} + \frac{\sum_{bj \in Q_2} \max\ ssim\ (b_j, Q_1)}{|Q_2|}\right)$$

The similarity is computed from both ends: for each noun/verb of Q1 get the maximum similarity to one of the nouns/verbs in Q2, add it to the sum and divide this by the number of terms in Q1; vice-versa for nouns/verbs in Q2 related to Q1.
The distance is then computes as 1 **– (sim_sem_of_nouns + sim_sem_of_verbs) / 2**

7.6. Cache

Since we have to deal with the time constraints of a real-time application that has to respond quickly to clustering requests for questions mostly related to a limited ontology, we came with the idea of implementing a cache system.

We considered that is best to implement two kinds of caches: one containing pairs of words and their similarity, and other containing pairs of Question objects and their similarity. The cache for questions is checked before using the Jaccard distance in the LexicalDistance class. The questions are searched in the cache using their ids and if they are found, their similarity is immediately returned. If not, the similarity is computed as presented in the previous sections and the result is kept in the cache. The word pair cache is used in the SemanticDistance class, right before using the Leacock and Chodorow distance measure for two words. No matter if the words are nouns, verbs or adjectives, they are kept in this cache.

The first type of cache is useful for the clustering process of a set of input questions as it is known that the hierarchical clustering involves serious computation, especially in the case of using complete-link metric. However, the second cache could be also be used for separate input sets of questions: each new clustering would benefit from the comparisons made in the clustering before. This is a subject we intent to investigate upon.

8. Evaluation and results

We have tested our system with a maximum of 143 questions. The dendrogram for such a great number of questions is hard to evaluate at a glance. However, we have done more testing on smaller sets of questions and the results seem to be good enough.
Example parts of dendrogram:

- example 1:

```
0.8702161821518287
  0.8122084287828696
    0.7744157546895316
      0.5
        What hemisphere is the Philippines in ?
        What is the largest city in the world ?
      0.25
        When did Hawaii become a state ?
        When did Idaho become a state ?
    0.7709395042942833
      0.7001254864297131
        Where is the Mall of the America ?
        What strait separates North America from Asia ?
      What river in the US is known as the Big Muddy ?
  What county is Modesto , California in ?
How far is it from Denver to Aspen ?
```

-example 2:

```
0.7646569784324155
  0.753494911695385
    0.5
      0.33333333333333337
        What is the temperature of the sun 's surface ?
        The sun 's core , what is the temperature ?
      What is the earth 's diameter ?
    Why does the moon turn orange ?
  What city had a world fair in 1900 ?
What is Australia 's national flower ?
```

Unfortunately, the corpus we have found is mostly related to the question answering field and most questions are in the form of trivia questions. This means that we could not exploit very much the way the clustering works for questions that are very different lexically but very similar semantically.

Sometimes the application makes what may appear as odd choices for clustering, but this is due in most cases to the fact that semantic similarity does not return a distance as we would expect it. A good example in such a case is the semantic distance between profession – mother, and profession – musician: all WordNet similarity measure consider that that profession is more related to mother, than to musician.

Below is a graph regarding the way execution time is affected by the number of question while using word pair cache:

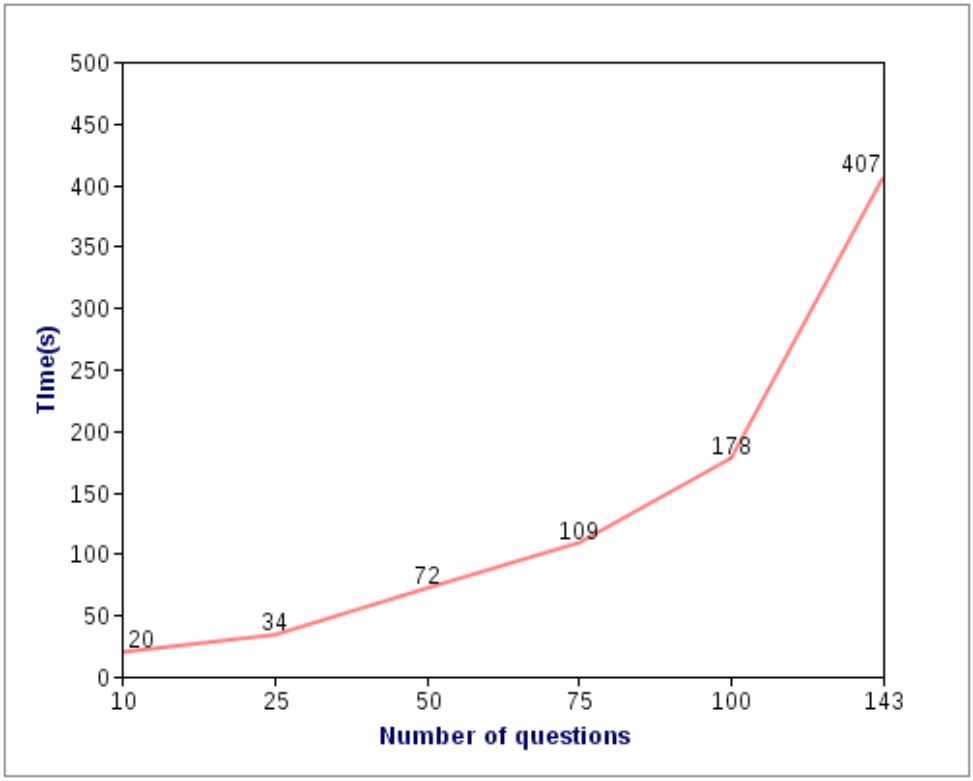


Fig.4 Graphical representation of algorithm performance considering number of questions

9. Conclusions

We have managed to create a working application able to cluster a set of questions by using several NLP tools available freely on the Internet and designing a solution to integrate all of them.

On the small corpus we had, the algorithm seems to work well. However, extensive testing should be done in order to catch any bugs that can change the outcome of the clustering. We also have to do more tests with semantically similar questions.

Although the tools we used were indeed very helpful, they lack flexibility. Since we talk about real-time applications, speed of computation is everything and the most expensive operation is the computation of semantic similarity between words in WordNet. This is because it involves many passes through big graphs in the taxonomy. A function that would stop looking for the exact distance after a certain time and declare the words as not similar would have been helpful. A future development would be to try to develop such a function using the sources available online.

Our next goal is to integrate our application with Smart Presentation and make the necessary changes needed.

10. Bibliography

[1] Benjamin C. M. Fung, Ke Wang, Martin Ester - *Hierarchical Document Clustering*, Simon Fraser University, Canada

[2] Pavel Berkhin - *Survey of Clustering Data Mining Techniques*, Accrue Software, Inc.

[3] William S. Maki, Lauren N. McKinley, and Amber G. Thompson - *Semantic distance norms computed from an electronic dictionary (WordNet)*, Texas Tech University, Lubbock, Texas

[4] Rada Mihalcea, Courtney Corley, Carlo Strapparava - *Corpus-based and Knowledge-based Measures of Text Semantic Similarity*

[5] Deepa Paranjpe - *Clustering Semantically Similar and Related Questions*

[6] http://alias-i.com/lingpipe/demos/tutorial/cluster/read-me.html

[7] http://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-clustering-1.html

[8] http://wordnet.princeton.edu/

[9] http://trec.nist.gov/