Tag Ontology Construction for QA sites

Introduction

Each tag in the QA sites basically represents a concept and our goal is to organize them so as to understand and perceive their relationships. A Tag Ontology is the hierarchical representation of tags and it's relationships. The relations can be of different types like parent-child, predecessor-successor, sibling etc.

We will be extracting the data from one of the QA sites(in this case we used superuser dataset) which has nearly one million posts and 5500 tags. We will be first filtering out the low frequency tags so as to eliminate unnecessary relation fromations and then based on the co-occurrence relation, we find the similarity between each pair of tags and then build the subsumption relationships. These subsumption relations help us in finding the entropy of each tag which basically represents the broadness of the tag and then build the Ontology.

Each stage is explained in the next slides

Related Works

A. **Subsumption relationships** : Co-occurrence is a common criteria to say that tags are related. Using this the conditional probabilities are calculated and then the hierarchy and eventually DAG which is used to represent the Ontology. Most of this is taken from Folksonomy-Based Visual Ontology Construction and Its Applications which was basically intended for image folksonomy which is applied to QA sites here

B. **Sibling relationships :** Sibling relationships are very important as there are a lot of tag pairs which belong to this relationship.They add more meaning to the Ontology. Word embedding techniques are used to find the semantic similarity between the tag signatures and tag signatures are obtained using the category links of their wikipedia articles as explained in Finding Semantic Relationships in Folksonomies .

1. Tag relationship extraction

a. **Pre-processing** - As the tags to each post are usually tagged carefully by the users and are not random we don't need to do a lot of pre-processing. So we will just be filtering out the tags with very low frequency count and work only with the remaining tags.

b. **Co-occurrent relationships** - If two tags *a* and *b* are co-occurring a lot of times, there is a very high chance of them being closely related. We will be defining the *similarity[*i] between two tags *a* and *b* based on the *google distance*[ii] metric which takes into account the co-occurrences.

   i. ***s(a,b) = (1+ exp(d(a,b)))*** $^{-1}$ where *d(a,b)* represent the google distance as shown below

   ii ***d(a,b) =***

$$\frac{max(logN(a), logN(b)) - log(N(a,b))}{logN_{total} - min(logN(a), log(N(b))}$$

2. Subsumption Relationship extraction

1. Now we have all the tag pair similarities and the next step is to find subsumption relationships between these tags.
2. The subsumption relation between tag *a* and tag *b* is defined as - If *a* subsumes *b*, then whenever *b* is used, *a* can be used without ambiguity.
3. We find these tag subsumption relationships using conditional probability *p(a|b)*. The idea is that, if a subsumes *b* then *a* will have wider coverage or in a sense it stands above *b* in the tag ontology.
4. *p(a|b)* is calculated as follows -

$$p(a|b) = \frac{s(a,b))}{\sum_c s(b,c))}$$

With this subsumption probability we can convert all the tags into a directed graph G where each edges weight is this conditional probability.

3. Tag Hierarchy Construction

The goal if this hierarchy construction is to turn the directed graph to a DAG i.e to remove either of the edges *a->b* or *b->a* from the graph G or to remove the cycles in the G as we need an Ontology.

A DAG(Directed Acyclic Graph) is a graph with no closed chains where a node can have multiple parents. Most of the times the Ontologies are represented in this way.

To calculate the semantic broadness of each tag we define entropy H for each tag using subsumption probability - $H(a) = -\sum_{c} p(a|c)log(p(a|c))$

The idea is that the tag with high entropy is expected to have more semantic broadness and should be above the tags with low entropy than this in the hierarchy.

4. DAG Construction

We calculated subsumption relationships and entropy of each tag, now we need to construct a DAG

$$Input : Tags\ with\ entropies\ O = \{H(a)\} and\ subsumtion\ relationships$$
$$Output : DAG\ of\ Tags$$

This is the algorithm for constructing DAG: (where *comm_threshold* is a threshold which we define)

for *'a' in descending order of entropy* ***do***
    make 'a' as visited
    choose top ranked subsumption relation pairs of 'a', R = {a->b} such that *N(a,b) > comm_threshold*
    if*(H(a) >= H(b))* ***then***
        Add this edge to the output DAG
        Make 'b' as visited
    end **if**
end for

5. Forming Communities and their Ontology

We now have a very large DAG which is for all the tags but for our comfort we have divide the tags in to different communities which helps us in better visualization and understanding of our Ontology.

We will be using InfoMap for community detection. The input will be the edges of DAG along and the output will be the communities.

An online tool is used for this: https://www.mapequation.org/infomap/

Now we have communities and all the tags in a community must be closely related. For each community we find a smaller Ontology by filtering out the edges from DAG which are not from this community i.e for a community *i* we take out only those edges *a*->*b* from the entire set of edges only if both *a* and *b* are part of this community.

6. Improving communities

The *comm_threshold* which is mentioned in the section 4 in DAG construction acts as a key to increase or decrease the number of edges in the DAG. These edges are supposed to be given to the infomap for producing the communities.

For some initial threshold which we used the formed edges didn't give proper communities, there were a lot of small communities and the large communities which are supposed to be formed got split into smaller ones. In order to overcome this we decreased the *comm_threshold* value and it increased the edges formed significantly which acted as the connecting links between smaller communities and combined them.

So it is advised to try different threshold values and based on the results decide which value is optimum. But this should only be used for Community detection as using this threshold in DAG formation will give a lot of unnecessary relationships.

7. Different kinds of relationships

We now have defined only one kind of relationship which is subsumption relation which in turn is similar to parent-child relationship i.e subsumption relation *a->b* represents that *a* is the parent of *b*.

But in many cases relationships between two tags need not be of only parent-child, they can be predecessor-successor(like windows-8 and windows-10) or siblings(like ubuntu and fedora). So it is meaningful to define these two relationships for the better output of Ontology.

First we will define Successor-Predecessor which is quite simple and based on Lexical similarity. Tag *a* and Tag *b* are said to have lexical similarity when

$$\frac{len(Initial\ matching\ string)}{max(len(a), len(b))} > some\_threshold$$

6. Successor-Predecessor Relationship

One tag *a* and *b* have met the lexical criteria we need to figure out which is successor and which is predecessor(where tag *a* and tag *b* are tags from same cluster, we basically check this for each combination of tags from a cluster)

1. As most of the times tags of greater length are successors, if lengths of tags *a* and *b* are different then the tag with greater length is made successor and the other as predecessor.
2. If both the lengths of tags are equal then, we can find the difference string which might probably be the version number and figure out which is greater by converting them to floats. (for instance for tags windows-8 and windows-10, the difference strings are 8 and 10 respectively and as they are floats and as 10 is greater than 8 we make windows-10 as successor and windows-8 predecessor)
3. If any of the above conditions are not met then we make the tag with more entropy as successor.

Finally we eliminate subsumption relationships which coincide with sibling relationships for each cluster

8. Improving Succ-Pred

As discussed in the previous section we are using only the lexical criteria for finding succ-pred relationship. The way we defined them by using initial matching string we might get few wrong relationships, for instance *user, users*. These two tags are basically singular and plural but we might get them as succ-pred. So it is advised to convert such tags to either singular or plural.

And the way we defined which to successor and which to be predecessor, might not always work fine. For instance tag pair like *windows-vista, windows-10.* As the length of *windows-vista* is greater then *windows-10*, we get *vista* as successor which is obviously not true. In order to avoid such error we might consider the timeline of the tags, when they were used more often and then decide which is succ and pred. Unfortunately the dataset we considered didn't relevant information about timeline of the tags so we were not able to do this.

As we are using only lexical similarity we might also be missing out few correct succ-pred relationships like *internet-explorer* and *microsoft-edge*

9. Sibling relationships

Till now we have defined parent-child and successor-predecessor relationships, we now move on to the sibling relationship. To define siblings we need a word embedding model for which we will be using GloVe model and wikipedia category links of each tag.

The idea is that the tags which have similar wiki category links are most likely to be siblings. So what we will be doing is that we will find the category links of each tag through wikipedia and then find the word embeddings of each of the category links and find the distance between two tags category links embeddings.

To find the embeddings we will be using GloVe model which is expected to perform better than normal word2vec. We first need to train the model using our own defined corpus. Our corpus consists of wikipedia articles summary of each tag and the wiki excerpt and tag excerpt posts.

10. Forming the Corpus and training the model

Our corpus should consist all the wikipedia tag summaries(which have a wikipage) and the posts which are relevant. Each tag in the dataset is given a wikipostId and an excerptpostId. These two posts are considered as relevant and added to the corpus.

The corpus formed is then used to train model y iterating through each line, then converting into lower case, tokenizing the line and finally removing the punctuations and stopwords. Make sure to scrap out the non-ascii characters as a lot of wikipedia summaries contain them.

After doing the above mentioned process we get a list of keywords for each line which we feed into the model. It is advised to feed this only when the list has a certain length, so as to avoid the noise words. The output of the glove model which is of interest to us is the dictionary containing the word embeddings of all words. We write this into a text file with the first line as dimensions of the dictionary.

11. Finding the category links

Both wikipedia summary and category links of a tag are obtained using the library *wikipedia.* Trouble with finding the correct page of the tag in wikipedia is that, most of the tags used her are doesn't have unique name. For instance, for a tag like *python*, there are several wikipedia pages like the films with named python, a species of the snakes - python, etc. So we should be careful to add find correct wikipedia page for such type of ambiguous tags.

For category links, we should filter out the most frequent category links as these usually doesn't hold much value. Category links such as *All articles containing potentially dated statements, All articles needing additional references* are obtained for a lot of tags in wikipedia, so such type of category links are filtered out while calculating the similarity of tag signatures.

12. GloVe model

As we have seen in the previous sections that the output of the glove model is stored in a text file containing all the word embeddings. This text file is then loaded into a *gensim model .* Now for each community apply this algorithm.

for *comb in itertools.combinations(l,2): #where l has all the tags of this particular community*

Cat_1 = category_links(comb[0]) # finding the category links of the tag

Cat_2 = category_links(comb[1])

Key_words_1 = keywords(cat_1) # finding the keywords of category links

Key_words_2 = keywords(cat_2)

if(model.wmdistance*(Key_words_1,Key_words_2)* **< *some_threshold*) # wmdistance stands for word's**

Add (comb[0],comb[1]) to sibling relationships # movers distance defined later

end if

end for

Note: The category_links of tags mentioned here are after filtering out the irrelevant as mentioned in the previous section.

13. Word Mover's Distance

Let us define the embeddings of keywords of category links of a tag as tag signature's. Now for each combination of tags we basically get a two set's of tag signature's which are nothing but two sets of points. These sets might be of different length and are not ordered. So we cannot use normal distance metrics like Euclidean distance instead we use word mover's distance.

Word Mover's Distance(WMD) utilizes word embedding vectors such that the semantic similarity between individual word pairs in two documents is used to calculate distance between them. WMD allows each $\boldsymbol{word_i}$ in a tag signature ***t*** to be transformed to any $\boldsymbol{word_j}$ in another tag signature ***t'*** . The distance between $word_i$ and $word_j$ is measured using ***Euclidean distance.*** In other words, the distance between two tag signatures is the minimum cumulative distance that words from one signature need to travel to match the document cloud of another signature. It can also find the distance even when the lengths of the tag signature's are different.

14.1 Improving sibling relations

1. Keywords of category links in section 12 *Glove Model* are obtained by splitting each category links and removing the stopwords. After finding the keywords of all category links of all tags, very high frequent keywords should be removed as they act as noise points.
2. When calculating the *wmdistance* for two tag signatures as said in the section 12, we can add few more filters such as
    a. *max(len(Key_words_1),len(Key_words_2)) < 2* min(len(Key_words_1),len(Key_words_2))* which basically makes sure that the two tag signatures length are in near vicinity as siblings should have almost similar length tag signatures.
    b. Checking out if both tag keywords are of minimum length, etc.
3. From what we have observed there are few wrong sibling relationships obtained in this method. This is mostly because when calculating the key_words of a category link we are just tokenizing it and removing the stopwords, but the keywords obtained like this may not hold any meaning all the times.

14.2 Using LDA for keywords

To obtain more meaningful keywords, it is advised to used methods like LDA as mentioned in the Finding_Semantic_Relationships_in_Folksonomies. The basic idea is that we use LDA for finding the meaningful keywords for each category link. Labeled LDA is a probabilistic graphical model that is considered one variation of LDA topic modeling technique.

For each Wikipedia category link, a set of N keywords are produced using Labeled LDA. Now these Keywords are used to find the embeddings using Glove and wmdistance is calculated using this. The LDA is trained similar to the Glove model over the corpus which contains wikipedia summary of tags and tag excerpt and wiki excerpt posts.

You can refer to the section III-B-4 of the paper Finding_Semantic_Relationships_in_Folksonomies on how to use the labelled LDA and section IV-A and B on how to train the LDA.

Also make sure the corpus is correct for training with no wring wikipedia summaries.

Collecting the results

Now we have obtained three different kinds of relationships i.e parent-child, Succ-Pred and siblings. These three should be mutually exclusive by definition and the order of confidence of these relationships are succ-pred > siblings > parent-child which means if siblings or parent-child relationships have any relation in common with succ-pred they should be deleted and so on. Each of these relationships are to be represented for each community.

These relationships can be visualized using a tool called Gephi.