Finding implicitly related items based on semantic similarities and metadata in a non-hierarchical network of documents

Authors

R. van Ginkel | J. Peters | L. Weerts

Project commissioned by *Perceptum B.V.*

Supervisors

Academic Supervisor	Raquel Fernandez
Company Supervisor	Robrecht Jurriaans
	Sander Latour
	Wijnand Baretta

June 23, 2014
Universiteit van Amsterdam

Abstract

Contents

1 Introduction

This report describes the results of the Second Year's project of the Perceptum team. The project focused on creating a *document link recommender system* to the StarFish website.

StarFish, one of the projects of Perceptum, is a website that aims to share knowledge about the education domain by means of a connected graph. People from all around the world should get access to this knowledge graph in a simple, personalized manner. The nodes in this graph are documents and they are connected with links. These documents can be of all sorts of types - e.g. a good pratice, information, a question. Each document has a set of tags assoicated with it, which describe the different aspects of educational innovation. StarFish is community-driven: both the content of the documents as the links between documents are determined by the users of StarFish.

The drawback of a community driven knowledge graph is that not all the users know the entire document base. Especially when the knowledge base grows it becomes impossible for a user, since one does not know the existence of one or more linkable documents. A possible solution could be to make use of administrators, which can devote more time in getting to know all the documents, but that approach has two main drawbacks. First of all, this would mean that some central authority determines whether or not two documents should be linked. This is not in line with the idea of a community-driven knowledge base. Secondly, if the knowledge base grows even further, it becomes impossible also for an administrator to keep track of all documents. Imagine one person having to link all pages on Wikipedia - an impossible job.

In order to overcome the problem of linking documents in a large knowledge base, this process should be automated. This project therefore focuses on automating making the connections between documents. Though ideally these connections should be made completely automatic, a first step would be to create a recommendation system. When a user adds a new document, he or she can choose from a list of proposed documents the documents he or she deems relevant. This means that the recommendor system does not have to work perfect, but should work reasonably well enough. Defining 'well enough', however, is also a part of this project. Thus, the product vision of the system can be described in the following concise way:

Product vision:

For StarFish users

who search for and edit knowledge in starfish

the starfish document linker

is a starfish core system addition

that finds related documents

unlike moderated or individual linking

our product uses algorithms and data to suggest document links

Within the time span of this project multiple ways of recommending links between documents have been explored. The results of these explorations will be discussed in this report.

2 Product overview

The product created in this project is a python program takes a set of documents and a new document and returns the subset of documents that should be linked with the new document. For this, a descriptor-based approach was used, which consists of three steps. First, each of the documents is transformed into a descriptor: a vector containing numerical values that in some way describes the document (hence the term 'descriptor'). Creating these discriptors is not trivial and during the project several techniques have been explored. Secondly, a ranking is made of all documents based on the similarity of the document descriptors and the descriptor of the new added document. To compare the descriptors the Nearest Neighbour algorithm was implemented, including five different distance metrics that determine how near two vectors are. Thirdly, an algorithm chooses the proper amount of proposed links that must be returned.

```
python documentlinker.py -vectorizer <vectorizername>
-distance <distance metric>
-bayes <true/false>
-threshold <0..1>
```

We will now discuss each of these parameters, since these will give more insight into the approach that was chosen to solve the problem. For the performance of the different parameters we refer to the evaluation section.

2.1 Vectorizer

The first step is to create document descriptors, which is done by algorithms that we call *vectorizers*. Two main paths have been explored: transformation based on text and transformation based on tags.

2.1.1 Text-based transformation

Textvectorizer The text-based vectorizers use the textual content of the documents and are therefore generally applicable to other systems. The textual content is first transformed into a *bag of words*. Then, based on all the documents in the knowledge base, the *TF-IDF* value is calculated for each of the words in the bag of words. *TF-IDF* stands for Term Frequency-Inverse Document Frequency and is a number that represents the importance of a word to a document in a bigger set of documents. Thus, the document descriptor consists of a vector with all TF-IDF values for that document of all words in the corpus.

Weighted_textvectorizer The weighted textvectorizer is implemented as an extension of the textvectorizer. Besides the descriptor of a document itself, this method also adds the vectors of documents linked to it with some weight. This captures the idea that if a new document resembles some of the documents that are linked to one particular document, it is more likely to be linked to this particular document.

2.1.2 Tag-based transformations

Simple_tag_similarity The tag-based transformations are more StarFish specific, since they make use of the tags that are assigned to the documents. A tag is a keyword that describes a topic/term that is important for that document. For example, 'Online Support and Online Assessment for Teaching and Learning Chemistry' is tagged with 'chemistry', 'e-learning' and 'assessment'. The simple tag similarity vectorizer creates a vector where each value indicates whether or not one particular tag is assigned to the document.

Tag_smoothing The tag smoothing vectorizer uses the co-occurence of tags in estimating document similarity. Even though tags might not co-occur on any document in the data set, they can still provide information about each other. For example, the dataset exists of documents with associated tags like $\{\{t_1, t_2\}, \{t_1, t_3\}\}$. From the co-occurence it does not follow that t_2 and t_3 are related, however by transitivity with t_1 we want to create a small implicit link between t_2 and t_3. The tag smoothing method does this based on work from **?**.

Glossaries_of_tags Another way of capturing tag similarity is by using tag Glossaries. Most of the tags have a Glossary - a special type of document which holds an explanation of a tag. Though glossaries are documents, they cannot be assigned as a link since this should be done by assigning a tag. The glossaries can still be used by applying a text-based transformation on the glossaries to indicate the similarity between tags. Thus, glossaries_of_tags can be seen as a hybrid form of the tag and text-based approaches, where the glossary of a tag is turned into a TF-IDF bag of words. The document descriptor consists of the sum of vectors of each of it's tags.

Weighted_tag_vectorizer This is an extension of glossaries of tags. In the original glossaries of tags, it is assumed all tags contribute the same amount of information to a document's links. In practice some tags provide more information than others. If a certain tag is on nearly all documents in the dataset, it does not provide a lot of insight into linking new documents. In contrast a tag which is only attached to a small subset of documents is much more informative. The weighted tag vectorizer creates descriptors by summing the tag vectors with a weight based on the frequency of that tag in the dataset.

2.2 Distance

The nearest neighbour algorithm loops through all available document descriptors and compares these with the descriptor of the new document. The closer related the descriptors are, the higher their ranking will be. The distance metrics define the closeness of the descriptors - a lower distance means a closer relation. The following were implemented:

Eucledian

Cosine

Bhattacharyya

Correlation

Intersection

2.2.1 StarFish specific adaptations: Bayesian weighting

Both the tag-based and text-based approaches uses some kind of 'semantic similarity' - the similarity of tags or text. However, except for the weighted text vectorizers, no information about possible links is used. For example, the text on a person's profile might be similar to other persons, but within StarFish a person is almost never linked to another person. In the Bayesian weighted vectorizer this is captured by weighting the vectors with the probability that two documents are linked together:

$$P(D_a \rightarrow D_b|t)$$

Thus, the weight of a tag within a vector is equal to the chance that given this particular vector, a document of type a (the type of the newly added document) and a document of type b (equal to the type of proposed link) are linked together.

2.3 Threshold value

The next step in the pipeline is to determine how many of the nearest neighbours should be returned. Depending on the application of the starfish document linker, the desired number might vary. If one wants to immediately link the results, the certainty for relatedness should be high. If the links are presented to a user which can approve or reject them, the relatedness may be lower. Currently, this is configurable by setting the threshold parameter between 0 and 1. Zero will only return the closest document, 1 will return almost all. After exploration of the dataset the default value is 0.3, which roughly returns the same amount of links which is currently average for Starfish.

3 Method

3.1 Data

- How big is the dataset
- Types of documents
- Properties of documents (title, author ect)
- Tags: meaning, aliases and glossaries
- Linking between documents: probabilities & standard linking of authors

3.2 Text vectorization

3.2.1 Textvectorizer

The first set of vectorizers focuses on the texts of the documents. The *textvectorizer* is a very generic approach that can be used on any corpus of textual documents. In the StarFish context, we define 'content' as the title and text-fields of a document. The only exception on this are Persons, of which we wil use the xx and xx.

The textvectorizer makes use of a bag of words representation. If two documents cover the same subject(s), they are likely to contain similar keywords. To capture this similarity, the documents can be transformed into a list of all words that are present within that text. Instead of counting the frequency of each word within a document, the more suffisticated TF-IDF value was used. This was implemented using the python kit xxx. Though the TF-IDF values of words that are used very often should be low, common words such as 'and', 'or' and 'of' are still present in the vectors. This could be caused by the different types of documents. For example, a Question often structured in a less complex way than a Project description. Adding a standard English stopword list to the vectorizer improved it's performance

Table x shows the performance of the textvectorizer on the data set. It shows that of all (valid) documents, about 20% of the links that were returned were correct. If the results are split into types of documents, it can be seen that the textvectorizer performs best in questions. This can be explained by the nature of questions: they often contain important keywords that indicate the subjects the question is about. If a document is a Person, on the other hand, only xx

percent of the proposed links is correct. Table x shows a possible explanation of this phenomenon. The textvectorizer often proposes other Persons if a new document is a Person. However, within StarFish a Person is seldomly linked to another Person. This implies that a more StarFish specific approach that does make use of links is more reasonable.

3.2.2 Weighted textvectorizer

The weighted text vectorizer is an extention of the textvectorizer that takes into account the links of the proposed documents. The vectors of the links of a document are added with some weight to the vectors of the documents themselves. Intuitively, this would add semantic information about a document based on it's links. For example, a Person is likely to write other documents about his or her subjects of expertice. Knowing not only the biography of a Person, but also the content he or she has added to StarFish, gives a more complete image of what documents could be related to that Person.

The vectors of links of a document are added in a recursive way, where documents that are linked directly have a higher weight than documents that are linked transitively. The algorithm is displayed in figure xx.

The weighted text vectorizer performs better than the normal text vectorizer.

3.2.3 Text-based approach limitations

Overall, both textvectorizers are slow in performance even though the corpus is small. Additionally, the the bag-of-words approach imposes a few limitations on the document linker. Firstly, it performs bad when different languages are used. Figure x shows the vectors of three texts when an English text is used, combined with an English proposed document and a Dutch proposed document. The English and Dutch vectors have only little words in common - luckily the keywords are in this case English, but if they are not this is a problem that cannot be overcome by simply looking at texts. Secondly, the current StarFish network consists of mainly textual content. However, in the future this is likely to be extended with images, videos and other non-textual content. These sources should then somehow be converted to text.

3.3 Tag vectorization

3.3.1 Simple tag vectorizer

- Motivation: use more starfish specific things
- Implementation: binary vectors
- Evaluation: works surprisingly well, only works bad on questions because no propper tagging, works extremely fast

3.3.2 Tag smoothing

- Motivation: use connection between tags
- Implementation: based on the paper
- Evaluation: very slow, similar results as the simple tag vectorizer, 'magic' so cannot really see why something happens

3.3.3 Glossaries of tags

- Motivation: find underlying network between tags by using their glossaries
- Implementation: hybrid form of text and tag vectorizer
- Evaluation: does not work well (find out why!)

3.4 Distance metrics

- Eucledian
- Cosine
- Bhattacharyya
- Correlation
- Intersection

3.5 Bayesian weighting

Explain motivation, implementation and short results

3.6 Thresholds

Explan motivation, implementation and short results

4 Experiments

Overview of performance over the entire pipeline

5 Conclusion

- Which vectorizers do not work
- Which vectorizers do work
- How well does the bayesian layer perform
- How well does the thresholds perform

6 Future Work & Recommendations

Latent dirichlet allocation because tags in dataset are not so good.
Also create incoming links