UNIVERSITEIT VAN AMSTERDAM

Proposing related documents within the knowledge graph of StarFish

Second Year's Project of the Bachelor Artificial Intelligence

ROBBERT VAN GINKEL 60606060
JORN PETERS 60606060
LOTTE WEERTS 10423303

Supervisor: RAQUEL FERNANDEZ
Client: PERCEPTUM

Informatics Institute, Faculty of Science, Universiteit van Amsterdam

Abstract

This is totally an abstract. Like totally

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:

Figure 1: Caption

Figure 2: Caption 2

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> -threshold <'auto' or a fixed number>
```

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 extention 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 documens. 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.

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, where a weight is assigned to the tag vectors.

2.1.3 StarFish specific adaptations

Bayesian Weigthed Vectorizer 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 Weigthed 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.

2.2 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

Algorithm 1 SLIC Segmentation

Require: $K =$ number of super pixels

1: Initialize K cluster centres $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid
2: Perturb cluster centres in an $n \times n$ neighbourhood, to the lowest gradient position
3: **repeat**
4: **for all** cluster centre C_k **do**
5: Assign the best matching pixels from a $2S \times 2S$ square neighbourhood around the cluster centre according to the distance measure in eq. 1
6: **end for**
7: Compute new cluster centres and residual error E($L1$ distance between previous centres and recomputed centres)
8: **until** $E \leq threshold$
9: Enforce connectivity

2.3 Threshold value

In the end, an algorithm chooses the proper amount of proposed links that must be returned. If the threshold value is set to a fixed number, e.g. 10, than only the proposed links of rank 1-10 are returned. If the threshold is set to *'auto'*, the number of returned links is based on the gradient of the distances. For example, if the algorithm is only certain that the first two links are correct, it returns no more than two links. For user-friendliness a maximum of 15 links is returned.

ALGORITHMS!!!!

MATHSSS!!!!

$$
\begin{aligned}
d_{lab} &= \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \\
d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\
D_s &= d_{lab} + \frac{m}{S} d_{xy}
\end{aligned} \tag{1}
$$

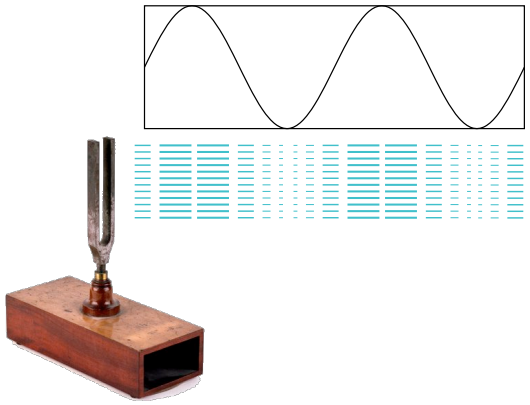

Figure 3: Sound waves propagate through a medium such as air molecules

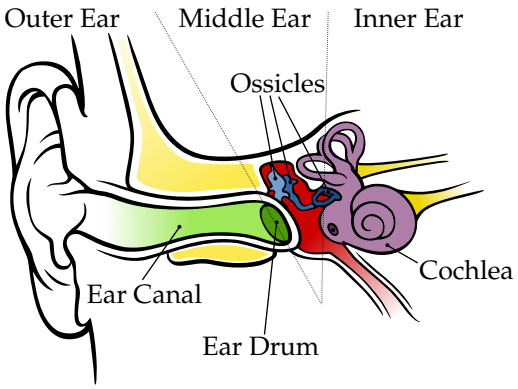

Figure 4: Internals of the ear

3 Implementation

Een ander vet iets dat je kan doen is svg in Inkscape openen en opslaan als pdf met latex output! Dan kan je dus latex tekst hebben binnen plaatjes zoals in figuur 4.

4 Conclusion

Bla.

References

Robrecht, Harrie, and Lotte. Totally bitching paper. *Journal*, 1(1):1–5, January 2013.