Semantic text features from small world graphs

Jurij Leskovec[1] and John Shawe-Taylor[2]

[1] Carnegie Mellon University, USA. Jozef Stefan Institute, Slovenia.
jure@cs.cmu.edu
[2] University of Southampton,UK
jst@cs.sotton.ac.uk

Abstract. We present a set of methods for creating a semantic representation from a collection of textual documents. Given a document collection we use a simple algorithm to connect the documents into a tree or a graph. Using the imposed topology we define a feature and document similarity measures. We use the kernel alignment to compare the quality of various similarity measures. Results show that the document similarity defined over the topology gives better alignment than standard cosine similarity measure on a bag of words document representation.

1 Introduction

In various domains we often deal with objects having thousands sparse features. Text documents are of such kind, where we have thousands of words, but each document contains only about a hundred of them. Text mining techniques mostly rely on single term analysis of text, such as the vector space model (bag of words model). To better capture the structure of documents, the underlying data model should be able to represent semantic relations between features in the document.

In kernel based learning methods [1] the choice of the kernel, which can be thought of as a problem specific similarity measure, is crucial for the performance of the system. There has been work [3][5] where people tried to learn or create kernels (similarity measures) which would exploit semantic similarity between the terms. Attempts to incorporate some notion of term similarity also include the latent semantic indexing [6], semantic networks [8] and probabilistic methods [4].

In this paper we present a set of methods for organizing a collection of documents into a tree or a graph. The induced topology implicitly defines feature similarity measure which outperforms standard cosine similarity measure.

2 Motivation

Given a set of documents D our goal is to build a graph where the each node N_i contains a set of features (words). There is an edge from node N_i to node N_j if a node N_i extends the vocabulary of node N_j is some particular way.

To illustrate this let's take a small toy example and let's limit ourselves to trees. In the root of the tree we expect to find very general words *(stopwords)* occurring in most

of the documents. We then expect to find increasingly more specific terms in the lower level nodes of the tree. We can have a node with general *computer science* terms, which has among the others also the children *computer architecture* and *machine learning*. Each of the children extend the vocabulary of *computer science* node in their own way.

To move away from the trees we can also think about general graphs. One may expect that *computer architecture* node will also share a lot of terms with *electrical engineering* node. In a similar way a *machine learning* node may extend/enrich vocabularies of both *computer science* and *statistics* nodes.

So we are eventually looking for the clusters of words which extend some terminology. Along with the clusters we also maintain the structural information. There is a directed edge between nodes (word clusters) N_i and N_j if node N_i extends the vocabulary of N_j. By "extends" we mean that a node N_i uses more specific terminology (on the same topic) than node N_j.

One can also see our ideas from a viewpoint of thematic categories or taxonomies. Documents are often organized into taxonomies or topic hierarchies. For instance on the web pages are organized into taxonomies like Yahoo or Open Directory. A path from the root of the hierarchy towards the leaves specifies an increasingly fine-grained description of web-page's topic. Since words define the meaning of the document we expect to find very general topic unspecific documents residing in the higher levels of the taxonomy using very broad and unspecific terminology. As we move down the hierarchy tree we expect documents to be more specific and using topic specific terms.

Our main goal is not to create full document topic hierarchy but rather a hierarchy of increasingly fine-grained sets of features. The feature sets and the topology can later be used to define a better feature similarity measure which could lead to improvement of text mining algorithms.

3 Proposed algorithms

In the following section we describe the algorithms for creating a semantic representation of a collection of text documents. Given a set of documents we connect them into a tree or a graph. Each node in a graph contains a set of words and a topology of the graph naturally defines the distances between the features.

3.1 Basic Tree

We are given a set of text units $1 \dots m$. We can think of them as documents or paragraphs. We use bag-of-words model to present each text unit (document) D_i. We use D_i to denote the i^{th} document and also the set of words that the document contains.

We take the documents one by one and compose them into a tree. Each node in a tree corresponds to a document and contains a subset of document words.

Suppose we already built the a part of the tree and we want to insert a new document D_i into the tree. We create a new node N_i and connect it to a node N_j so that D_i has the largest intersection of words with the words found in the nodes on the path from N_j to the root of the tree. We populate the node N_i with words from document D_i which do not exist in nodes on a path from N_j to the root of the tree.

We now formally describe the algorithm. Given a document D_i which we are about to insert in the tree, we first find a node N_j that maximizes the score

$$score(D_i, N_j) = \frac{|D_i \cap (\cup_{p \in P(j)} N_p)|}{|D_i \cup (\cup_{p \in P(j)} N_p)|} \quad (1)$$

where $\cup_{p \in P(j)} N_p$ denotes a set of words from nodes on a path from N_j to the tree root.

We create node N_i and link it to node N_j with maximum *score*. In this and all further algorithms we weigh each link (N_i, N_j) by its score: $score(D_i, N_j)$.

We also have to decide on a set of words residing in a node N_i. We set the words to be: $N_i = D_i - \cup_{p \in P(j)} N_p$. This means a node N_i contains words from D_i that are new for the path from node N_j to the root of the tree.

We expect to find very common and general words on higher levels of the tree and increasingly more specialized sets of words as we move deeper into the tree.

3.2 Basic Tree with stopwords node

The inspection of trees created by the Basic Tree algorithm revealed that each path from the root of tree contains its own set of stopwords. It is very natural to introduce a stopwords node and start building the tree with a root node N_0 containing stopwords — common and general words that have little or no meaning by themselves.

We experimented with several different English stopword lists (8, 425, 523 stopwords). There was very little difference in performance between them. At the end we decided to use 8 English stopwords (and, an, by, from, of, the, with). We also combined them with words that occur in more than 80% of the documents in the dataset. The stopwords node usually contained around 20 words.

3.3 Optimal Tree

The tree created by the Basic Tree algorithm depends on the ordering of the documents. Since the Basic Tree is choosing the documents in order, a random permutation of the documents will generate a different tree. We now propose a modified Basic Tree algorithm called Optimal Tree which generates trees independently of document ordering.

We start with a pool of documents. In case of Basic Tree algorithm we pick a documents from the pool in some arbitrary random order. In case of Optimal Tree algorithm we always take the best document from the pool — a document which maximizes the score given by equation 1. We use the same rules to create links and to determine the node words as in case of Basic Tree with stopwords node algorithm.

3.4 Basic Graph

So far he have been thinking of documents organized into a hierarchy of topics. But if we take a closer look we notice that topic hierarchies are not really hierarchies, they are graphs. Open Directory, for instance, is a graph — a hierarchy tree with cross-links between the nodes of the tree. We now propose an algorithm for connecting a set of documents into a graph.

Let's assume that we already built a portion of the graph. To insert a document D_i, we create a node N_i and link it to all nodes N_j where $score(D_i, N_j) > threshold$. A set of words in node N_i is composed of words from document D_i which are *new* for the whole graph: $N_i = D_i - \cup_k N_k$, where k runs over all nodes in the graph.

In the later sections we describe the use of a graph shortest path distance as a feature distance measure. Given a document collection we expect that most of the documents contain some stopwords. This means the distance between two nodes would always be either 1 (a direct link between the nodes) or 2 (a path through a stopwords node). To prevent this we remove the stopwords node after the graph is built.

4 Feature similarity measure

We have built the topology implicit on the nodes based on the distance in the graph. This suggests that the features should not be treated as independent (i.e. bag of nodes) but rather the geometry of the feature space could be adapted to reflect the dependency.

This could be done by defining the similarity S_{ij} between two features i and j based on its distance in the graph. Then using this similarity matrix S to define the inner product between two feature vectors x and z:

$$\kappa(x, z) = x' S z \tag{2}$$

Feature distance is defined by the length of the shortest path connecting the nodes which contain features i and j. We can treat the distances of links to be all equal to 1 or we can take $1 - score$ as the length of the link. The similarity between the features i and j is then defined by $(1 + dist(i, j))^{-1}$.

In case of Tree algorithms a single feature can be present in many nodes. If a features i and j are present in more than one node we take the average shortest path distance between all nodes which contain i and all nodes which contain j.

5 Experimental evaluation

We use the Reuters corpus volume 1 [7] which contains more than 800,000 documents. Each document in the corpus also belongs to one or more of 103 categories.

We take 100 random documents from the Reuters corpus. Using these documents we build a semantic structure using the algorithms described in section 3. We create the feature distance matrix S (Eq. 2) using the distance measures from section 4 and evaluate the performance. We repeat this procedure 10 times, each time using a different set of documents. We compare various topology generation algorithms in combination with different distance measures.

We measure the quality of the representation by the kernel alignment [2] which captures the degree of agreement between a kernel and a given learning task. Intuitively it compares the sum of within class distances with the sum of between class distances.

The alignment between the kernel κ and the matrix A that has $A_{ij} = 1$ if documents i and j belong to the same category and 0 otherwise is given by

$$\frac{\sum_{ij} \kappa(x_i, x_j) A_{ij}}{\sqrt{\sum_{ij} \kappa(x_i, x_j)^2 \sum_{ij} A_{ij}^2}} \tag{3}$$

Table 1. Mean and associated standard deviation alignment values for all algorithms using various topology feature distance measures.

Algorithm	Alignment
Baseline methods	
Random	0.5382
Cosine similarity	0.5848 (0.0217)
Basic Tree without stopwords node	
Graph path + Feature distance	0.5911 (0.0279)
Graph path + Node distance	0.5696 (0.0280)
Weighted path + Feature distance	**0.5981** (0.0270)
Weighted path + Node distance	0.5756 (0.0284)
Basic Tree with stopwords node	
Graph path + Feature distance	0.6186 (0.0220)
Graph path + Node distance	0.5832 (0.0199)
Weighted path + Feature distance	**0.6270** (0.0251)
Weighted path + Node distance	0.6169 (0.0250)
Optimal Tree with stopwords node	
Graph path + Feature distance	0.6113 (0.0211)
Graph path + Node distance	0.5768 (0.0222)
Weighted path + Feature distance	**0.6291** (0.0252)
Weighted path + Node distance	0.6161 (0.0241)
Basic Graph	
Graph path + Feature distance	**0.6280** (0.0235)
Graph path + Node distance	0.6202 (0.0242)
Weighted path + Feature distance	0.6258 (0.0245)
Weighted path + Node distance	0.6192 (0.0254)

We compare our algorithms against two baseline methods: *Random:* the similarity between the documents is chosen uniformly at random. And *Cosine similarity* where we take standard vector space model (bag of words). A document is represented with a vector v where $v_i = 1$ if feature i is present in the document and $v_i = 0$ otherwise. We use the cosine similarity — inner product of the feature vectors.

To measure document similarity we always use the average shortest path length in a graph. In table 1 we denote *Graph path* if each link had length of 1 and *Weighted path* if we used $1 - score(D_i, N_j)$ for a distance of a link when calculating the shortest path.

Feature distance means that we used the topology to measure feature similarity and then equation 2 to calculate document similarity. Since each node basically represents a document we can directly measure the document similarity as a shortest path distance of the nodes representing the documents. We use *Node distance* to denote this.

From table 1 we observe that Basic Tree without stopwords node has about the same performance as cosine similarity baseline measure. Introducing a stopwords node increases the alignment by 3% (a relative improvement of 5%). One can see this as a small improvement, but the improvement of alignment between purely random document similarity and cosine similarity is 5%. This suggests 3% increase to be more significant than one might expect at first sight.

Further we notice that Basic Tree with stopwords achieves similar performance as Optimal Tree and Basic Graph. This suggests that the quality of Basic Tree semantic representation does not suffer much from the order of how documents are inserted into the topology.

Comparing various distance measure strategies we observe that Feature distance in combination with Weighted shortest path consistently gives very good results.

In case of Tree algorithms alignment is consistently higher when using weighted $(1 - score)$ shortest path distance between the nodes. The performance increase over un-weighted (link length is 1) shortest path is around 1.5%. For Basic Graph the performance is about the same no matter how we define the length of the path.

Also the Feature distance always outperforms Node distance. The performance of Node distance is around 1% lower than feature distance. On the other hand the Feature distance is much more expensive to compute while the cost of Node distance is just one calculation of shortest path algorithm.

We also performed experiments with paragraphs instead of documents. The idea was to create a topology from the paragraphs and then measure the alignment using whole documents. The alignment performance around the one of cosine similarity.

6 Conclusion

We have proposed and compared three different methods to model the semantic similarity between then documents. Given a set of document we used a simple algorithms to connect them into a tree or a graph. Using the imposed structure we defined a feature and document similarity measures. We used the kernel alignment to compare the quality of various similarity measures. Our results show that the document similarity defined over the topology gives better alignment than standard cosine similarity measure on a bag of words document representation.

References

[1] N. Cristianini and J. Shawe-Taylor. *An introduction to support Vector Machines: and other kernel- based learning methods*. Cambridge University Press, 2000.

[2] N. Cristianini, J. Shawe-Taylor, A. Elisseeff and J. Kandola. *On Kernel-Target Alignment*. Advances in Neural Information Processing Systems 14 (NIPS), 2002.

[3] N. Cristianini, J. Shawe-Taylor and H. Lodhi. *Latent Semantic Kernels*. Journal of Intelligent Information Systems 18 (2002), 127–152.

[4] T. Hofmann. *Probabilistic latent semantic indexing*. In Research and Development in Information Retrieval, 1999.

[5] J. Kandola, J. Shawe-Taylor and N. Cristianini. *Learning Semantic Similarity*. Neural Information Processing Systems 15 (NIPS), 2002.

[6] T. A. Letsche and M. W. Berry. *Large-scale information retrieval with latent semantic indexing*. Information Sciences 100, 1997.

[7] T. Rose, M. Stevenson, and M. Whitehead. *The Reuters Corpus Volume 1—from yesterday's news to tomorrow's language resources*. Third International Conference on Language Resources and Evaluation, 2002.

[8] G. Siolas and F. d'Alché-Buc. *Support vector machines based on a semantic kernel for text categorization*. In IEEE-IJCNN, 2000.