Better recommender systems based on content recognition in text

Bjorn Vandenbussche

Supervisor(s): Luc Martens

Abstract—**This article tries to extend the limits of content-based text recommender systems by applying content recognition in text. It proposes a new system that is able to classify and cluster text in order to get a better view on a users needs.**

Keywords—**text mining, big data, recommender systems, machine learning, classification, clustering, named entity recognition**

I. Introduction

RECOMMENDER systems help individuals address the challenges of information overload on the Internet. Although a lot of content is available, it remains a though task for users to find the most relevant information. Classical keyword-based recommenders offer just a partial solution. Preferences of users, earlier experiences and already consumed content are not taken into account in typical search engines. That's why recommender systems can be a useful tool in order to select and consume content.

Classical recommender systems work based on consumption behavior (ratings, purchases, clicks), sometimes supplemented by information about content (meta data) like keywords, author, category, resume, etc. The problem with these systems is that the description isn't always perfect. Recommender systems won't work well in those cases, as they expect an exact match of meta data or at least an overlap in used words.

In this abstract we will first discuss what kinds of content recognition can help the recommender system to deliver useful recommendations. Different techniques for information extraction can make sure the recommender gets a better view on the preferences of a user, by really taking the content of items being recommended into account.

II. Concept

Text documents appear under all kinds of file formats (html, doc, pdf, etc). Each format has its own markup and somehow tries to upgrade regular text to something more useful in their own environment. Unfortunately this doesn't work well with content recognition systems, as they require plain text to work with. To address this problem all documents that are processed in this system first pass through Apache Tika [1]. This toolkit allows the extraction of both text and meta data from all kinds of text documents, making it possible for content recognition systems to consume the plain text of different sources. Using this extraction makes it possible to use this system for all kinds of text recommendation problems. The plain text produced by Tika will then be consumed by two systems of content recognition: classification and clustering.

By automatically classifying a text into a predefined structure of categories, we try to deduct context of that item. This classification allows us to gather similar documents and so recommend texts to users that they might like, considering they enjoyed some other content in the same category.

Clustering on the other hand allows to find clusters of similar texts (also referred to as topics). These clusters may be situated inside of an existing category (making classification more precise), but may as well group articles from way different categories since they have no knowledge of classification at all. They don't require any predefinition of structures, but try to calculate document similarity in an n-dimensional space.

By combining the information from both classification and clustering the recommender systems gets a better view on the content of the items and is thereby capable of generating better recommendations.

III. Content recognion

A. *Preprocessing*

The structure of plain text isn't sufficient to be efficiently processed by a computer. In order to comprehend what a text is about, it has to be preprocessed to a feature vector that can be used as input for content recognition systems. The requirements for preprocessing are similar for both classification and clustering. Yet for classification one additional step can be added to lower the dimensionality of the features.

A.1 Tokenization

The first step in preprocessing a text document is tokenization. This process makes sure a document gets split into a stream of words by removing all punctuation marks and replacing all non-textual characters by spaces. The set of different words from all documents gets merged into the dictionary of the document collection. This dictionary then gets used for both classifying or clustering new documents.

A.2 Filtering and stemming algorithms

In order to remove common stop words that have little or no meaning filtering is applied. Examples of removed words include: articles, conjunctions, prepositions, etc. These words tend to add noise to the feature vector, which is something we like to avoid.

Stemming algorithms try to transform a word to their word stem or some kind of root form of a word. It is used to make related words map to the same stem. This can be achieved by converting plural back to singular form or by converting verbs back to their word stem. These words aren't always morphologically correct, but are used for This is achieved by applying Porters algorithm for stemming [2].

A.3 Features selection

Feature selection is a step used to preprocess text for classification systems. It makes sure that irrelevant attributes get removed from the dictionary. In that way a subset of the original dictionary is generated. This mostly reduces processing power and search space required for the vector model. Feature selection is mostly interesting in text mining because of the high dimensionality of the features. A lot of those features turn out to be irrelevant to the classification problem, or can even be seen as noise for the classifier.

B. Classification

For two Dutch datasets (*Het Laatste Nieuws*, 2008 and Wikipedia) multiple classifiers from the WEKA framework [3] were researched. Both datasets are structured as a two-leveled tree. In order to find the optimal classification for each node of the tree, tests were conducted for the top two categories of the dataset. We first try to classify the text in one of the top categories and then classify it into a subcategory. This research solely wants to test what classifiers work best for these datasets, and doesn't describe the algorithms themselves.

B.1 Individual classifiers

The first dataset tested is Het Laatste Nieuws. For the first level, top classifiers include DMNBtext (89.86%), NaiveBayesMultinomial (88.75%), SMO (86.98%), ComplementNaiveBayes (86.45%) and Logistic (82.83%). These ratings are the percentages of texts they can classify correctly being trained with one month of data (Januari 2008, 10121 articles), using 10-fold cross validation.

For the second dataset (Wikipedia) training data gets generated with 1000 random articles chosen for each subcategory. The classification problem for this dataset seems to be a lot harder as the class structure is not as strictly defined as with the first dataset. Wikipedia tends to use an acyclic graph representation of articles, since they want to link multiple topics over different categories to inform a user of their encyclopedia. This results in much poorer classification. Top classifiers include ComplementsNaiveBayes (51.44%), NaiveBayesMultinomial (51.33%) and DMNBtext (50.0%).

B.2 Combining classifiers

By combining good individual classifiers it is possible to achieve better classification. Combination algorithms used are voting, stacking and grading. Results for the first dataset show a 0.6-0.12% improvement to classification precision for the first dataset. The beste combination algorithm found was for this dataset was stacking the top 4 classifiers and using linear regression to combine them (89.98%).

The second dataset also benefits from using a combination of classifiers. Best results are achieved by using a voting scheme (52.04% correctly classified).

C. Clustering

Clustering is used to group documents with a similar content. Each cluster contains some documents. It is important that documents inside of a cluster show big similarity (intra-cluster similarity), but differ with documents from other clusters (intra-cluster similarity). This makes up for a good measure of validity of a clustering.

C.1 Named entity recognition

With text documents, clusters tend to be related to certain named entities (NEs). To achieve better clustering natural language processing (NLP) gets added to the preprocessing using the Stanford Named Entity recognizer [4] [5]. The idea that NLP can help finding relevant clusters gets fueled by *(a)* the fact that NEs remain the same over different documents (it is difficult to paraphrase a name), *(b)* the frequency of NEs in a text shows its importance to the events being described in the text, and *(c)* two different stories containing the same NE most probably belong to the same cluster.

Because not all texts contain NEs it is very difficult to conduct clustering solely based on them. Because of that fact, in this system, NEs get an artificial higher term frequency than other (ordinary) words. Using this system interesting clusters get build with a good mixture of NEs and important keywords. On top of that, clusters (and thus recommendations) get more comprehensive for users of the recommender system.

C.2 K-means algorithm

In this system, Apache Mahout is used to process documents to a clustering by applying one of the best-known algorithms: k-means clustering. It tries to partition a given dataset $X = x_1, ..., x_n, x_n \in R^d$ in k different subsets (clusters) $C_1, ..., C_k$. It expects a certain value for the number of clusters k that should be constructed. This requirement is a big drawback for this algorithm. If one chooses a k that is too big, clusters will be too small and won't bring any relevance to users. If on the other hand k gets chosen too small, clusters will tend to be bigger and their topic will be much more general than we want it to be.

In order to overcome the issue of finding the optimal k, canopy clustering is used. The main idea with canopy clustering is that the calculations that are necessary in order to calculate distances can diminish if the dataset first gets divided in overlapping subsets. Distances can then be calculated between each pair of items that belong to the same subset. The strength of this algorithm lies in the speed by which clusters are formed. This strength is also its largest weakness, as clusters that get formed by canopy clustering aren't always accurate. Luckily the output of canopy clustering (and so the optimal k) can be used as input for the k-means algorithm. This algorithm is able to find good and relevant clusters for our dataset.

IV. Recommender system

The recommender system used to process the input from both the classification and clustering is content based (built with Lenskit [6]). It uses the information as tags (classifciation tags, the cluster the document appears in and some relevant keywords) and tries to find relevant items based on those tags. This recommender was compared to a TFIDF (Term frequency-inverse document frequency) content based recommender. This baseline uses the contents of an item to find similar documents.

V. Conclusion

The recommender evaluation shows a vast improvement over the TFIDF recommendation system. By finding relevant items really based on their contents, a recommender can help users finding and consuming more relevant content. On top of that items can get recommended way faster because of the few tags used in this new system.

References

[1] The Apache Software Foundation, "Apache Tika," .

[2] M.F. Porter, "An algorithm for suffix stripping," 1980.

[3] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten, "The WEKA data mining software," *ACM SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[4] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, vol. 53, pp. 55–60, 2014.

[5] Soto Montalvo, Raquel Martínez, Víctor Fresno, and Agustín Delgado, "Exploiting named entities for bilingual news clustering," *Journal of the Association for Information Science and Technology*, vol. 66, no. 2, pp. 363–376, feb 2015.

[6] Michael D. Ekstrand, Michael Ludwig, Jack Kolb, and John T. Riedl, "LensKit," *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*, p. 349, 2011.