Construction of Story Telling Graphs

Paul C. Nichols
Department of Computer Science
University of California, San Diego
pnichols@cs.ucsd.edu

Abstract

Stream-like data sources, such as a collection of RSS feeds, can be challenging to summarize as the corpus is ever growing and the nature of the data evolves over time. This project implements an intelligent RSS reader that leverages topic modeling to provide a high-level view of the document stream. LDA Topic modeling is performed at daily intervals and linkages between learned topics from different days are computed to form the edges in what can be considered a "Story Graph". This story graph can be used to answer a number of contextual questions from both the perspective of a topic and a document.

1 Introduction

Stream-like data sources are a common fixture of the current technological landscape. Well known examples of streaming data sources include news content from major media outlets (i.e. through RSS feeds), the collective musings of Twitter's user base (i.e. the Twitter Firehose), and the click-stream generated from surfing internet (i.e. from network logging). Summarizing and contextualizing information in a data stream is a natural and important problem that arises.

The most popular form of summarization on a data stream is mining for trending topics, typically through an unsupervised learning process. Trending topics provide a high level view of the prominent themes present in a data stream during a window of time. A set of trending topic sets, or even multiple sets from different windows of time, can provide insight in isolation. However, contextual questions about a topic's origin, lifetime, momentum, and evolution cannot be addressed. To answer these more advanced questions, topics must be linked beyond the window in time they were discovered. This set of time sensitive topics and the linkages between them is referred to in this report as a "Story Graph".

Given a set of topics, the construction of a story graph is only slightly more complicated than the discovery of the topics themselves. However, not all approaches to topic discovery result in representations where computing linkages between topics is straight forward. Further, the topic modeling should be sufficiently powerful to accommodate complicated real-world mixtures of topics. For both of the above reasons, Latent Dirichlet Allocation (LDA) is used on the bag of word representation of documents to mine for topics.

Queries on a story graph may start and end from both the perspective of a topic and a document. Starting from a topic, the story graph can be traced to another topic or down to a document through the topic weights. Likewise, starting from a document, the story graph can be traced through the topic weights to another topic or again down to a neighboring document. Features of a trace through the graph, such as the length in time covered or the number of topics or documents traversed, serve as the basis for the answers to the contextual questions posed above.

This project investigates story graphs, their construction process, and their relevant queries by implementing a system that digests a stream of documents exposed from a collection of RSS feeds and generates

a summary view of the data through a web portal. Three different datasets are explored from different categories: General news, Business, and Sports. Data was collected by the system for 60 days.

2 Background

Unsupervised methods for discovery and summarization of a corpus have a rich history rooted in information retrieval. Before the more abstract task of a topic discovery became widely known, document clustering was a well studied problem. Document clustering is used to find inter-document similarities and partitions of similar documents in a corpus.

The vector space representation of a document is a common place to begin document clustering. The vector space representation assigns a unique dimension to each term that exists in the corpus with a given documents vector populated from its terms. Various term weighting schemes can be applied to the document vectors such as tf-idf weighting. These schemes enhance the similarity measures by reducing the contribution of unimportant words such as stop words. Similarity between documents can be measured either by the cosign similarity or the euclidean distance between a vector space representation of two documents.

A good first step towards topic discovery is the K-means algorithm applied to corpus of tf-idf weighted documents in the vector space representation. K-means attempts to find a partitioning of the corpus into K groups such that the euclidean distance of a cluster's members is minimized. Iterative refinement from a randomly initialized means can be used to approximate this hard problem. The most highly weighted term indices of the resulting K mean vectors from a clustering job loosely represent the topics present within the corpus. The approach is advantageous for streaming data because the learned mean vectors can be compared across clustering jobs from different time windows easily (using either cosign similarity or euclidean distance between the means). The approach is also desirable because it is simple and fast. However, the technique lacks the richness of topic mixtures by assuming each document can belong to only one mean vector.

Another early approach that improves upon the vector space representation is Latent Semantic Analysis (LSA). LSA computes the singular value decomposition of a document-term matrix to find low-rank approximations of the latent document and term matrices. The resulting matrix factorization has the property that when a pair of documents or a pair of terms are similar, they will be near (using cosign similarity or euclidean distance) in their respective vector spaces. LSA is good step towards finding a representation that can capture synonymy and polysemy, but is not suitable in the streaming paradigm where the process repeats and regular intervals. The resulting matrix factorizations are unique to the document-term matrix they were derived from. There is no straight forward way to compare either the learned term or the document representations from different time windows.

Probabilistic Latent Semantic Analysis (PLSA) addresses the limitations above with LSA by stepping away from the vector space representation of LSA to a probabilistic representation of terms and document. PLSA introduces a hidden random variable to model topics and then learns document-topic and topic-term distributions through the Expectation Maximization (EM) algorithm. PLSA can also overcome issues such as synonymy and polysemy through the topic, and has the advantage of allowing mixtures of topic (or cluster) assignments. Lastly, PLSA is advantageous in the streaming paradigm because the resulting topic distributions are interpretable and comparable across different time windows.

PLSA is the predecessor to LDA, the model used in this project. LDA improves upon PLSA by modeling the document probabilities and word distributions as dependent on Dirichlet prior distributions. This dependence on prior distributions enhances the generality of the distributions learned. An LDA model has a more complicated and the distributions are commonly estimated using Gibbs Sampling.

3 Method

3.1 Graph Creation

4 Architecture

Processing the document stream is accomplished by a nightly process that downloads content from a collection of RSS feeds for the previous day. The entire process is composed a series of stages that happen sequentially. While the production environment schedules the process to handle a single day's worth of data from a cron job, the process can also be run in batch mode to process all data over a provided time range. Importantly, each stage in the processing pipeline is dependent only on the output from the previous stage.

The process begins with the Sync stage by gathering the list of URLs for unprocessed data in the stream using the Google Reader undocumented API (GRAPI). The GRAPI normalizes the many quirks and formatting issues present in a large set of RSS/Atom feeds and exposes a standard representation of a document stream. The output of the Sync stage is a mapping for URL to path on local storage the content should be downloaded to.

The Fetch stage of the processing Pipeline is run after the Sync stage generates output. The Fetch stage is responsible for downloading HTML content, extracting the relevant article content from the raw HTML, and saving the result to local disk. Content downloading is managed by through the Apache Commons libraries. Article extraction was originally handled by extracting text content with an HTML parser. However, the resulting saved documents included too many artifacts of the hosting site such as advertisements, comments, and boiler plate content. The artifacts present in the document made the topic modeling perform poorly. To remedy this problem, a library known as boilerpipe was used to separate article content from artifact with an in-built decision tree classifier.

5 Evaluation

6 Future Work