A High-Scale Deep Learning Pipeline for Identifying Similarities in Online Communities

Robert Elwell, Tristan Chong, John Kuner, Wikia, Inc.

Abstract:
We outline a multi-step text processing pipeline employed against tens of millions of English-language documents to identify similarities between Wikia's hundreds of thousands of online communities. We developed cloud-based infrastructure around Stanford CoreNLP to extend the basic use case from ad-hoc parsing to a continuous service-based pipeline for parsing raw text. This allows us to develop and maintain up-to-date parse assets for each individual document. From these parses, we extract and aggregate useful data about named entities within a given community. The most frequent entities in a given wiki are used as initial features for a dimensionality reduction approach known as Latent Dirichlet Allocation (LDA). The output of this process can be used to identify similarities between Wikias, which we use for recommendations and ad targeting.

Introduction

Wikia is a MediaWiki-based platform that empowers communities to generate long-form content on a variety of subject matters. It is one of the top locations on the web for fans to collaborate about what excites them, with over 1 billion monthly page views, over 110 million monthly visitors, and over 400,000 communities hosting more than 30 million pages of content.

With such a large amount of content, we are interested in improving our users' experience with Wikia in two ways. We aim to improve the interconnectedness of different Wikias by pairing users in in a given community with communities that match their interests. Given the large number of communities, manually curating this can be a difficult proposition. Secondly, we aim to improve users' overall advertising experience by matching communities with relevant ad campaigns. Both of these goals rely on finding similarities between wikis, hidden somewhere inside millions of pages of raw text.

We employ a cloud pipeline built on Stanford CoreNLP, Amazon Web Services, Gensim, and Apache Solr to build a model for understanding the underlying of features of a given wiki, based on deep natural language features. We use the output of this model to power recommendations and ad targeting.

Architecture

Parsing Pipeline

The parsing pipeline is built in a way that leverages the power of concurrency and distributed computing in order to maximize efficiency. There are three main tasks comprising the system's architecture: event file generation, autoscaling, and parsing. A machine on Wikia's local network handles event file generation and autoscaling up, while Amazon EC2 instances are launched to handle the actual task of generating parser output. Additionally, a data model allows information to be loaded dynamically from the resulting XML as it is queried.

The event file generation component flexibly handles both content updates and arbitrary

subsets of documents sharing a given field in Solr. To decouple parse event generation from Solr's indexing update queue, Apache Solr query strings indicating the desired scope of documents to be parsed are written to files in a local events directory. These can be either queries against a shared field value, or a recently indexed document ID.

A query writer continually iterates over the events directory, submitting the specified queries to Solr and writing the text contained in the requested documents to a local text directory. Finally, another script polls the local text directory and waits either for enough files to populate it or for enough time to pass, at which point it moves batches of text files to a temporary directory. In the temporary directory, batches are compressed into tarballs, uploaded to an Amazon S3 bucket that serves as a text event queue, and then deleted locally.

A local machine also controls the process of automatically scaling up the number of working parser instances as necessary. Interacting with the Amazon Web Services API, a monitor script watches the size of the remaining event queue and spawns additional parser instances whenever the ratio of remaining events to active instances exceeds a given threshold.

The individual EC2 instances spawned by the autoscaling monitor are based on an image containing two services -- the parser poller and the parser daemon. The parser poller service has three functions: to download parser input from S3, upload parser output to S3, and to automatically self-terminate. A separate parser daemon service is an extension of Stanford CoreNLP's existing functionality, providing improvements over the base suite with regards to concurrency and efficiency.

The poller service continually checks the S3 bucket serving as the text event queue for new tarballs. The poller moves each tarball to a separate S3 bucket for in-progress text events. This avoids a race condition in which multiple EC2 instances attempt to work on the same tarball. It then downloads the tarball and extracts it to a local text directory, deleting the original key in the S3 bucket and fully removing the tarball from the S3 queue.

The service polls the parser output directory for XML files and uploads them to an S3 bucket with the appropriate directory structure. Each Wikia has a directory containing XML files corresponding to the ID of each document comprising it. Upon upload, it sends a text file containing the locations of all newly parsed documents to S3. The target folder acts as a queue for data extraction services to be called. It then deletes the in-process key from the S3 queue, which fully removes the text file from S3.

The poller's final responsibility is to terminate the EC2 instance it runs on. This happens when the poller detects that the text events queue has been emptied, and that the pipeline is already performing above a desired efficiency threshold. This is determined by comparing maximum and minimum instances running with parse rate.

By default, the Stanford CoreNLP software is called with a finite set of input files. Before parsing can begin all models need to be loaded. This loading time is on the order of 5 minutes. The parser daemon extends the default behavior such that new input files can be added to the queue as soon as they appear in the EC2 instance's local text directory. At capacity, this is continuous. This is advantageous because it means the parser does not have to incur a protracted startup cost every time a new batch of text files is extracted. The parser is also invoked with as many threads as there are CPU cores on the server, allowing for greater efficiency.

The XML output written by the parser contains various types of syntactic and semantic data about the original text. Constituency and dependency parses provide information about sentence structure, like which sequences of words go together as cohesive phrases, which phrases serve as subjects and objects, and how words in a sentence are related to one another. Named entity tags indicate whether or not phrases are classified as people, locations, or organizations. Coreference resolution information clarifies the expressions referred to by any pronouns present in the text.

Sentiment data gives insight into the attitude of the author towards the topic he or she is addressing. Wikia added this data to the default XML output provided by CoreNLP. This is a valuable asset as the sentiment analysis implementation in CoreNLP is state-of-the-art as of October 2013. The approach introduces graph-based inferences over sentiment scope that is not available in traditional bag-of-words approaches.

Extracting Data from Parses

Once each document has a corresponding parse stored in S3, we begin extracting data and storing its denormalized form in S3. In this case, we aim to find the most frequently mentioned entities for a given wiki. We first identify noun phrases and any coreference chain they belong to as candidate entities. Those candidate strings are then referenced against two data sources. We use all titles in Wikipedia as a data source due to its high reliability, as promoted by engineers at Facebook. We also cross-reference candidate entities against the titles for the wiki we are currently operating against, as each community is also capable of building a specific ontology using the structure of their Wikia. We are interested in retrieving the entities and their frequencies aggregated across all documents for a given Wikia.

We also extract and aggregate semantic heads from each sentence of each document in each wiki. Semantic heads are the root node of a dependency parse, and can be found from syntactic parses via a set of recursive rules. These words tend to be tensed verbs, except for in such cases as copular verbs or sentences involving modal verbs. They generally offer insight into the kinds of events being discussed in a wiki, and can even serve as indicators of prose, style, and tone.

Unsupervised Learning

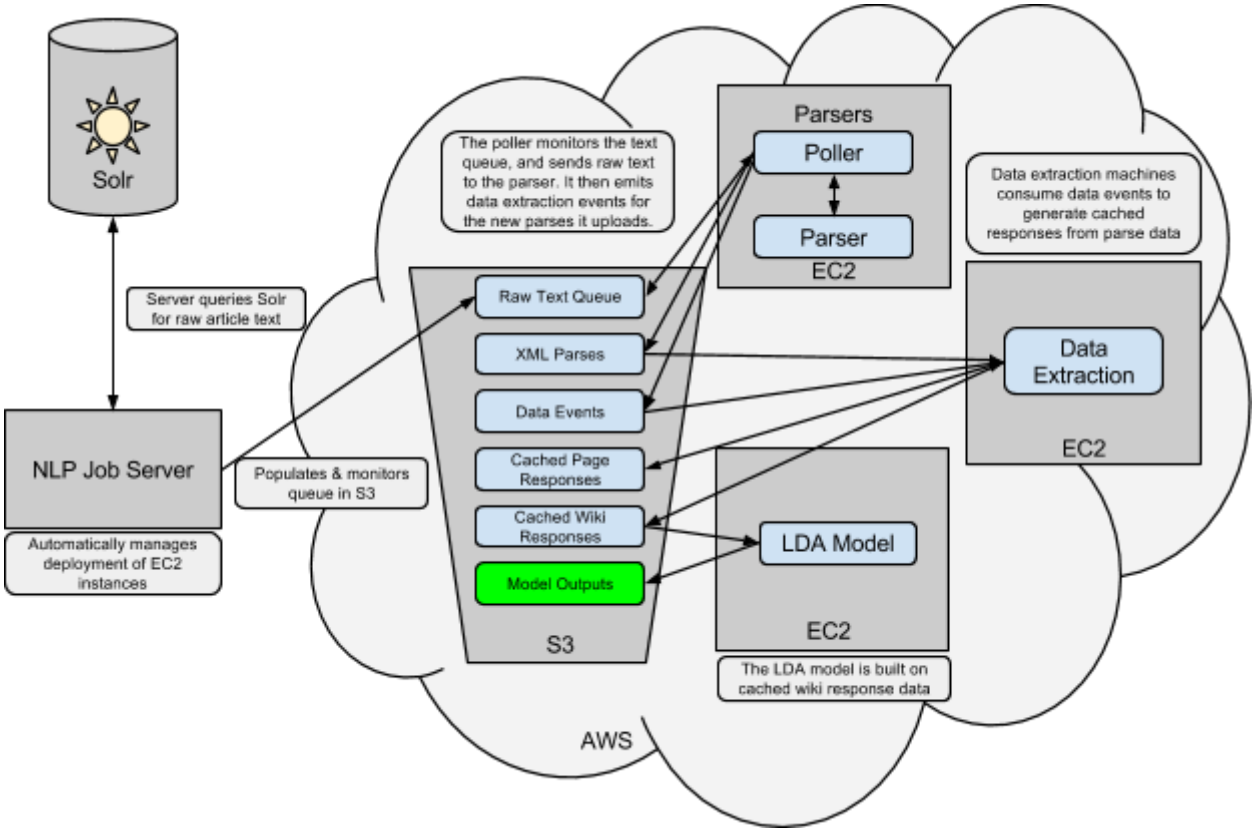
Extracting this data for a given Wikia gives us insight into the most important topics of conversation, as well as that wiki's domain. However, finding which of those data points are responsible for determining similarity between two wikis continues to be problematic. Given the multitude of topics each community may be discussing, we find ourselves with unique features numbering in the hundreds of thousands. Since one of our intended implementations revolves around integration with Doubleclick for Publishers, we are also constrained on the length and number of features we can use when implementing this data. This problem, referred to as "the curse of dimensionality", is one encountered quite often when working with natural language. We address this problem using a dimensionality reduction technique known as latent Dirichlet allocation.

Latent Dirichlet allocation (LDA) is a generative model that allows us to specify a number of unobserved groups, or topics, whose composition explains the similarities between observed

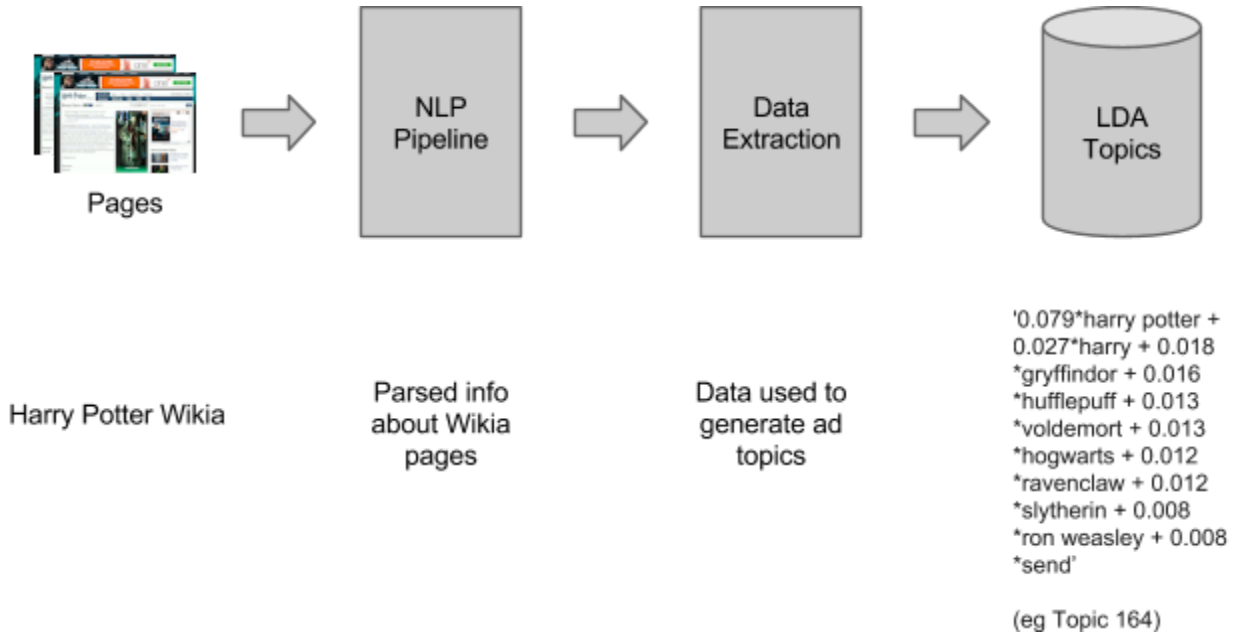
data. This specific model is known to provide state-of-the-art improvements in user interaction for content-based advertising using deep linguistic data.

We choose the top 50 most frequent entities, and the top 50 most frequent semantic heads, as features for the learning model. The features themselves are modulated by their frequencies within each instance. Based on best performance in prior work, we build a model using Gensim to reduce dimensionality from any hundreds of thousands of features to 999 topics. We use Gensim's distributed LDA training model capabilities to improve the time to build the model. This machine cluster is created in a virtual private cloud within AWS. This infrastructure allows us to scale our learning model for much larger problems, such as document- rather than wiki-based LDA.

The result of the output is a Zipfian distribution of topics to wikis, which can also be described in terms of "fat head, long tail". The output is a CSV file that keys the document ID to the weights for each topic. These weights can be understood as probability distributions over groupings of words.



A Diagram of the Full NLP Pipeline



An Illustration of the Transformation from Wikia to Topics

Use in Advertising

In order to use this data for advertising, we store the top five topics, ordered by weight, as key values to be sent to Doubleclick for Publishers on each pageview, along with a variety of other demographic- and content-oriented data points. From this, we can identify what topics correlate positively with clickthrough rate, and activate specific campaigns for all wikis exhibiting those topics. The most active communities may have some obvious similarities (for example, all science fiction movie Wikias). With some manual effort, these similar communities can be targeted for ad campaigns.

The derived topics add value by revealing non-obvious connections around shared concepts. For example, our most popular Harry Potter Wikia shares a topic with several derivative Harry Potter Wikias (eg fan fiction). Augmenting the manually derived topics with the generated topics has resulted in a substantial lift in user interaction for ad campaigns in which we have employed this approach.

The current test has been limited to shared topics at entire community level (eg glee.wikia.com). Future work will investigate shared topics at a page level, which should result in more nuanced clusters. In addition, other content clustering is also under investigation.

Generating Recommendations

As a data source for recommendations, we make the assumption that documents with similar probability distributions over similar groupings of words will have similar content. In order to test this assumption, we inject topic data into Solr documents corresponding to wikis. Each topic is treated as a dynamic float field. For topics that a given wiki does not exhibit, we inject a

zero value for that topic. Otherwise, a weight between 0 and 1 is stored at that value.

Starting with a given Wikia, we then aggregate a list of non-zero topics for that wiki, and treat them as vectors in N-dimensional space, where N is the length of that list. We then calculate Euclidean distance against each Wikia, and sort it by closest -- or smallest distance. This is done by treating that initial set of topic fields as vectors in each other wikia in the data set. Vectors are compared by using Solr's `dist` function while sorting. We assert that the shorter the distance, the more likely a wiki is to be similar to the wiki in question.

Values for recommendations can be denormalized as a document field in Solr, stored as metadata using Wikia's global variable scheme, or simply exported to CSV for further manipulation. Since these values should not change often, and the calculation averages in the hundreds of milliseconds, we err towards heavy caching with regular updates for maintaining this data asset.

Future Work

We plan to explore the use of these data assets and statistical approaches for a number of additional internal and product features. On the statistical side, applying LDA to video metadata may provide an effective recommendation algorithm for the Wikia Video Library. Furthermore, we can use entity data on a page to provide high-fidelity recommendations from a page to a video using the same model.

The sentiment data included in the Stanford CoreNLP pipeline cross-referenced with entity extraction will allow us to identify user attitudes about certain topics within different communities. We can identify the subject matter expertise of authors within a given community by cross-referencing entity data with author centrality and article quality score data. By scaling this approach across multiple wikis, we can identify experts who may be able to improve the quality of other wikis, or identify the most authoritative subject matter experts across Wikia.

Summary & Conclusion

As the breadth and depth of popular user-generated content grows at Wikia, we aim to apply massively concurrent statistical techniques to serve our users and continue to grow our ecosystem. We believe that we will increase user engagement by continuing to tailor the ad and recommendation experience to the user's interests and the communities that represent those interests. At a scale of hundreds of thousands of communities, automation promises to play a major role in providing this service. In doing so, we will facilitate exploration and discovery, ultimately bringing communities together, improving network density, and continuing to grow our repertoire of user-generated content.