

Topper: Client-side Active Learning for Sorting and Filtering Items in an Online List.

Abstract

As the content on the web continues to grow and people spend more time online the ability to filter and sort material according to preference or intent becomes more important. Sources such as blogs, search engines and other social media can benefit in attracting larger audiences if users can personalize their view of their content. Online content personalization is difficult to achieve for several reasons including privacy concerns, scalability and the inability to generalize preferences in a useful way. We describe a browser-based automated learning tool that can be used to learn user preference and intent that circumvent these issues. With this system learned models are built that can be used to sort and filter existing online content across various sources. Our system allows for ad-hoc expression of intent or preference that, by default, is never shared outside of a user's browser. Alternatively models can be persisted, used again or even shared among users.

1. Background

Search engines serve large general audiences by returning the most popular results for a given query. Many queries, however, are ambiguous in user intent. In order to appeal to more people the entire result set will likely need to cover a variety of possible intents. For example the query 'lions' will typically return links to sites related to both the Detroit Lions football team and a large member of the cat family. In order to disambiguate which results will be most relevant for a specific user a search engine needs additional information about the user's preferences or intent.

There are several ways for a search engine to gain additional information about a user's query intent. One method is to ask for or suggest further modifications to the query. Most search engines use augmented query languages to allow users to modify their queries to focus intent. With this method users progressively refine each query until they find the results that are most relevant. For example the query 'lions' can be qualified as 'lions + detroit' or 'lions + cat'. This method puts the onus on the user to figure out the best way to encode the query to achieve the desired results. One disadvantage of this approach is that a user may be unable to find relevant results if he cannot determine the appropriate way to phrase the query. Another problem is that this knowledge cannot be automatically reused on future queries. Instead a user must learn how to phrase queries on a case-by-case basis. Query-specific modifications can disambiguate intent but it is not a general solution that can be used between queries and, more importantly, between users.

Another method used by search engines to learn about user intent is behavioral tracking. This includes the tracking of queries and click behavior of a user over time and generalizing a user's intent on future queries. For example if a search provider knew that

a user was a sports fan based on previous behavior it might know how to bias the results. One problem with this method is the difficulty scaling this to large numbers of users. Both storing and generalizing information requires vast data stores and sufficient computational power. More important, however, are privacy concerns. Many users may not want a search engine to record detailed behavioral information about their habits. Lastly, biasing results toward assumed user intent may be incorrect for that user at that particular time. For example the sports fan may be researching for report on African wildlife. For all of these reasons using behavioral information is not well suited to use for the disambiguation of search query intent. In addition it would be difficult to share, for privacy and scale reasons, behavioral information between users.

The value of understanding user intent can extend beyond the simple disambiguation of search queries. Lists, intended for broad audiences, are pervasive on the web. Online blog sites, aggregation sites like Digg and news portals can be viewed as lists of small chunks of information much like search results. There is a tradeoff inherent with these communities that make personalization appealing in similar ways to web search. Editors of these sites are driven to keep topics focused in order to preserve relevance for their current audience. At the same time the primary means to increase their audience size is to increase the breadth of topics. If user-specific intent can be used to sort and filter articles then such sites could cover a broad scope of topics without sacrificing the depth and relevance for their core audience. It is possible that if user-specific intent-based filtering was available the value of such sites might be better gauged on the quality of content rather than their ability to keep focus. The same is true for many social networks such as MySpace and Facebook. A user's ability to filter data on a social network, such as comments and friend requests, can be greatly enhanced if the intent of the user can be taken into account.

Intent-specific personalization of content in search and other online areas is both attractive and difficult to achieve. Powerful tools in the areas of machine learning and data mining provide automated ways to filter and sort based on user intent. However barriers exist to any automated approach including scalability problems and privacy concerns. In this document we describe a method for the automated learning of user intent in the browser that can be applied to external content sources on the web. We have instrumented this learning system in the browser in order to circumvent issues with scalability and privacy. By creating software that runs in the browser the user is in complete control of whether or not the data is shared with any external sources. As an added benefit, in scalability, all computation is done in the client or user's browser. In addition we describe a method for users to express their intent that work well with list-structured content on the web. By using automatically learned models users can effectively sort and filter content from various sources including search results and blog listings. These intent models can be created ad hoc and, as such, can represent a user's intent at any specific time. Intent models can also be persisted for future use and interchangeability. This persistence also offers users the ability to share intent models easily with other users.

2. Description

We have designed and implemented an automated system that allows the application of locally learned models in a browser to external sources on the web. These models can be used to sort and filter web content based on user intent and preferences.

2.1. Expression of Intent

Given a list, such as the results from a search query, a user identifies the items that fit and do not fit his current intent. An item is simply an element of any list such as single search result or a single news article summary. The means by which this is done depends on the medium and implementation. If the medium is search, intent may be expressed as following links to results they like and not clicking on the rest. In other media, such as blogs and aggregation sites, this may be clicking on “like” or “not like” buttons next to the item. Whatever the method the outcome is two subsets of the list items the user has decided either fits or doesn’t fit his current intent. There is no need to identify this intent a priori. The intent is completely expressed in terms of labeled items as either positively and negatively related to the intent.

Many existing systems allow users to label list items in a similar fashion. Voting for the importance of news articles, another form of labeling, on sites like Digg is one example. At these sites votes are used to collectively determine the value of content and then to increase the prominence of articles to the general audience. However these votes are rarely used to personalize the experience for individual users. This is the key difference in what we propose. In our system labels are used to help sort and filter list items for a specific user. Another difference is that act of labeling is independent to the method that generated the list. In this way labeled data can be applied to lists from different sources. For example labeled examples for the query ‘lions’ on a search engine can be used to disambiguate results for the query ‘bears’. Labeling of data is often the first step in building a machine-learned model. Rather than throwing away the preference data or aggregating them as votes we keep them and build a intent model for a specific user.

2.2. Learning

Given a set of labeled items our system automatically creates a model of user intent in the browser. The labeled examples for a single model can be from one or more lists such as multiple queries to a search engine. In this way a model can better generalize to new information (e.g. new queries). As a user browses or performs searches our system can collect examples to help further refine the model. As each new example is added a new model is built. We call these preference or intent models.

An intent model often can be expressed as a set of weights for the features extracted from the labeled examples. The features that are associated with each example include text features such as words and phrases. For example the features for a single search result would be the text from the title, summary and URL of the result. Other indirect features such as tags, images and features of the sites referred to by the URL can be included as well. Using these features a model is built that represents the

intent of the user. This model is then used to score each item in the list, labeled and unlabeled.

The learning itself is automatic and performed in the browser. As such it does not involve any communication with an external server. Because the learning is local there are no privacy concerns. Examples can be collected, a model can be created, and items can be classified without any information leaving the user's local machine. This allows for ad hoc model building (i.e. "one offs") where a user builds a model for his current intent and then discards all information. Using ad hoc modeling a user can perform a search, label a few examples and immediately reorder the result list based on his learned intent. Such ad hoc model building is encouraged because all data and scores are kept private and the mechanism of expressing intent remains simple. Local computation also satisfies the scalability problem because computation is done without server-side storage or computational requirements. The methods for automated learning in our system include common supervised and unsupervised machine learning methods.

2.3. Classification and Recommendation

Using our learned models the probability of any list item being relevant to the user's current intent is calculated in real time. This score is immediately displayed in line with the content of the item. Real-time scoring creates instant gratification for the user. As each example is added or removed the user can see the effect on all items in the list. This property of our system is a key component to the success of continual refinement of the model. A server-side learning system would inhibit the amount of continual refinement because of latency and computational load problems. As the user navigates to other lists, for example by making successive search queries, the model is automatically applied and used to score new items. The same model can be used on many queries on the same or different search engines. Similarly this model can be applied on blogs or aggregation sites to score articles.

2.4. Sorting and Filtering

The automatic scoring of items provides the user with instant gratification that his labeling efforts are paying off. Another intrinsic value of the scores is their ability to affect the ordering of the items in the list. This can be done by filtering out content that ranks below some score from the model. For example a user can opt to only see news articles that have a high probability of being relevant to his current model of preference. Similarly a sorting interface can be added that would allow users to change the list ordering by blending the list rank with the model score. For example a sorting tool can be added to a search engine that allows users to change the ordering of the list from using the original search rank to an order entirely based on model score.

2.5. Persistence and Sharing

Our system allows for the ad hoc building of intent models for online lists. In some cases a user may want to persist his preference model for future use or sharing with other users. This storage can be done locally but for convenience we can store the preference data on an external server for sharing between users. There are no latency problems in using these external servers since they are used solely as a data store and do not need to bear the computational load of machine learning. In this way users can choose to keep their preferences private or share them for other to use. This is an opt-in choice and users must explicitly agree to store their labeled examples externally. No offline storing is done by default. By persisting these intent models users may create a set of models and switch between them when convenient. For example on one occasion a user may be looking for recipes and at another time looking for restaurants. In either situation the query for 'grilled steak' may be considered differently under separate intent models. With our system the user would be able to filter or sort results differently based on his intent at different times. In addition an experienced restaurant critic may want to share his model in order for other users to gain from his experience in searching for local restaurants.

3. Instrumentation

The machine learning tools in our systems are derived from well-known published algorithms. In order to run these methods in a browser we have implemented them entirely in JavaScript. JavaScript is a well-known computer language that can be executed within various media including most web browsers. Porting these algorithms to JavaScript is the primary reason why we can create models locally on browsers. To build models and score items quickly on a browser the execution speed of the learning method is critical. To this end we have ported what we have deemed as, currently, the fastest machine learning algorithms for the type of data used by our system. Specifically our selected methods work well with short excerpts of text such as those returned by most search engines.

4. Topper Search

Topper Search (<http://www.toppersearch.com>) is an example of our automated learning system built on top of search results. Specifically this Topper learning example classifies search results from Yahoo Web Services (<http://developer.yahoo.com/about>). A user begins by making a query and clicking the button 'Topper Search'. Topper Search relays this query to Yahoo Search and retrieves the top 100 results. Before the results are displayed one of two possible processes occur. If you do not already have a Topper 'hat' the top 10 results will be displayed in the original rank order from Yahoo Search. A Topper 'hat' is a learned model that you have either built yourself or borrowed from someone else that is tuned for classifying search results. If you already have a Topper 'hat' the results will be reordered according to a blend of the original rank and the Topper score. Following this reordering the top 10 results according to this blend of rank and score will be shown. You can change this blend and thus change what results are in the top 10 using the radio buttons in search bar. Using these buttons a search result with a high rank but low Topper score will be pushed lower as you move the buttons to the

right. Selecting the button on the far left will reorder based on search rank alone while selecting the button on the far right will sort purely based on the Topper score.

Topper models can be built using the 'positive' and 'negative' labeling buttons to the left of each item. When a labeling button is selected the item is added to the model training examples. Instantaneously Topper will use this new example to create a new model and re-classify the entire set of 100 search results. As user moves between searchers the same model is used to classify new sets of results.

5. Summary

In recent years there has been considerable effort to create repositories of knowledge from people across the web. Some projects have tried to use simple factual assertions in the form of binary questions to better understand the structure of the human mind. Other projects, such as OpenMind, encourage people to contribute factual, more abstract, statements about common sense knowledge. The problem with these approaches is the difficulty in rewarding a user's contribution of knowledge. This difficulty accounts for the lack of cooperation and accountability among users leading to smaller datasets. Prediction markets, such as NewsFutures, encourage users to share knowledge through competition. The challenge with prediction markets is overcoming the complexity and difficulty in the expression of many problems as a market. The intent of Topper similarly attempts to a build repository of knowledge that will enable us to create software that better represents human behavior. However Topper encourages users to share knowledge by providing immediate rewards for each bit of information they contribute. In addition Topper uses an interface that can be layered on top of everyday user activities online such as search, blogs and other types of social media. For these reasons Topper encourages users to contribute general knowledge about human behavior as it applies to their lives online.