Motivation

Description : Autocompletion helps user predict one or more possible words as choices when the user writes the first letter of a word or first word of a sentence.

Implementation Details :

Indexing Mode:

1. In this mode, all the documents in the corpus are processed and an index structure is created using the partial index building method. The index structure is a datastructure with word to a list of documents with extra information like term frequency , document frequency that will be needed during the serving mode. In this method we build partial indices from the subset of documents and write those partial results to the disk as needed. Then we merge all the partial indices into the full index. This method helps to build the index even when the corpus grows. The above approach is not in memory.
2. To implement autocompletion , along with the index structure we wrote our own implementation of WordTree. WordTree is a datastructure that stores the co-occuring words in the corpus along with the frequency of the occurrence. When we process the corpus we populate the wordTree to store the 5-grams phrase that occurs together along with the frequency of the phrase in the entire corpus. For this also we used a partial build method so that we do not restrict ourself as the corpus grows.
3. At the end of index mode, we serialize the index structure to a file along with the WordTree. To make it more efficient we only serialize the phrases that occur more time than a certain threshold. This helped reducing the size of index drastically.
4. We also implemented a TRIE tree to store all the words in the dictionary for prediction of phrases that are not in the WordTree and also for prediction of single word query.

Serving Mode:

1. In this mode, all the information needed for serving like the Index structure , num\_views, pagerank, WordTree and TrieTree are loaded into memory for query serving.
2. After the required structures are loaded, the user can types in a query on our frontend search engine. Using local storage of angular js we check if the user has interacted before with our search engine. If so we read the userID from the local storage. If the user is interacting for the first time we make a server call to generate a id and store in users browser for future use. This unique id for every user helps us to personalize user search.
3. After the user has a valid Id a user can send a query to the search engine. When the user starts typing the first letter of the query , our TRIE tree helps us predicting the first word of the query.
4. After the user has typed the first word and a space our wordTree predicts the sentence.
5. We have used a linear model to rank the suggestion. The following signals are fed into the linear model to predict suggestions for the user.

The signals include ,

* Suggestions from the user session
* Suggestion from the WordTree ( most frequently occurring phrases in the coupus)
* Suggestion by the Trie tre ( if the phrase does not exist in WordTree)

1. Once the user enters the complete query, the query is the handled by the QueryHandler and the response includes the highly ranked documents using document quality feature like pageRank , numViews, nextDoc.