PROTOTYPE SEARCH ENGINE

PROBLEM:
Develop a prototype crawler tool backed by smart rating algorithm to synthesize quality content from various sources available on the internet for given search query.

THEME:
Artificial Intelligence

UNDERSTANDING:
Ranking Relevance has been the most critical problem since the birth of search. As information was generated and made available the search engines were required to effectively retrieve more relevant data from a prohibitively large corpus. The earliest methodology to solve this synthesized content problem was either a vector space or a probabilistic model.

Traditional methods for information retrieval are not adequate as they fail to generate semantic results and are by far, unable to judge the content itself. Text matching features suffer from the vocabulary gap between queries and document. Enhancements in ranking mainly take into account of the human-generated parameters such as clicks, time on page, hyperlink count, voluntary feedback in form of votes, bounce rate and exit rate, and also some sort of probabilistic model to rank these results. To truly achieve meaningful results we must use techniques of AI(Artificial Intelligence).

The semantic gap between queries and the actual content is the main barrier for improving base relevance. Human behaviour with respect to the provided search results help provide hints to improve relevance, but unfortunately for most tail queries, the human-based parameters are too sparse, or noisy. Furthermore, for a considerable chunk of applications, human-generated parameters are non-existent altogether - especially in cases where the text corpus is processed based on an automated retrieval.

The problem proposed by Ubisoft is a selection and Information retrieval problem, coupled with a ranking metric improved by a model that helps to generate semantically correct results. Ranking and selection are extremely important to the end user and good selection should be the KPI(Key Performance Indicator) for any tool that is developed. The potential customer may be lost if this critical problem of selection and ranking is not solved. The challenge is to provide an AI-based engine that can unambiguously judge the content quality and provide a metric that can help differentiate the content quality of different providers for similar content.

To improve ranking results we need to deeply understand the information and user's intentions behind them. Smart ranking and retrieval coupled with Deep Natural Language Processing(AI techniques) can be used to elucidate how semantic results can be generated which are extremely useful to the end user. Deep learning techniques can help extract semantic and contextual information and thus help in smart ranking and selection.

Our assumptions regarding the characteristics of what makes the article good/better than others:

- Well structured grammar.
- Relevance of all the sentence to the query and related information.
- The influence of the article in extractive and abstractive summarization of the entire corpus.
- Semantic relevance of topics discussed in the article to each other as well as the search query.
- Influence of the article on the subtopics and the relevance of the subtopic in the corpus as well as the query.

METHODOLOGY:

1) Data Collection: The primary source of data is the Universal Crawler that gets all the documents related to the query. Then, the query entered by the user is understood using LDA(Latent Dirichlet Analysis) and then related APIs are used to crawl particular sources of information apart from the information that is obtained by scraping the web. For eg: If the given query is related to travel then the Trip Advisor API is also used to collect information through the API along with the data from the universal crawler(built using BeautifulSoup and urllib2). The information is then stored as a dump in MongoDB, also a dictionary is maintained which maps topics to the additional APIs that will be used to scrape information regarding specific domain related queries.

The ranking is divided into 5 different units with each unit carrying certain weightage that affects the articles' final metric score.

2) Analysing Sentence Structure: POS tagging can be used to understand the sentence structure and sifting for grammatical errors. The documents are also checked for spelling errors. Essentially a sentence must consist of a noun phrase, verb and a subject or an auxiliary verb, subject and a main verb. Each document is assigned the max score score and for the mistakes found in the document, the score is reduced proportional to the gravity of the mistake and it's impact on the content quality.

3) Text Rank: The sentences obtained from the documents will be ranked together using the TextRank algorithm by appending all the documents in a single corpus. The rank will act as the score for each sentence in that document. The score for each document will be obtained by computing the average score of the sentences belonging to the document. Therefore the lower score documents will be ranked higher as lower average score indicates more relevant sentences are present in the document.

4) Summarization Score: Luhns algorithm will be used to generate a summary from all the sentences put together. The documents will be ranked based on it;s influence on the summar. In place of Luhns, we can also use neural networks (like TextSum or NAMAS) to generate a summary and compute its cosine similarity with the document to generate a rank.

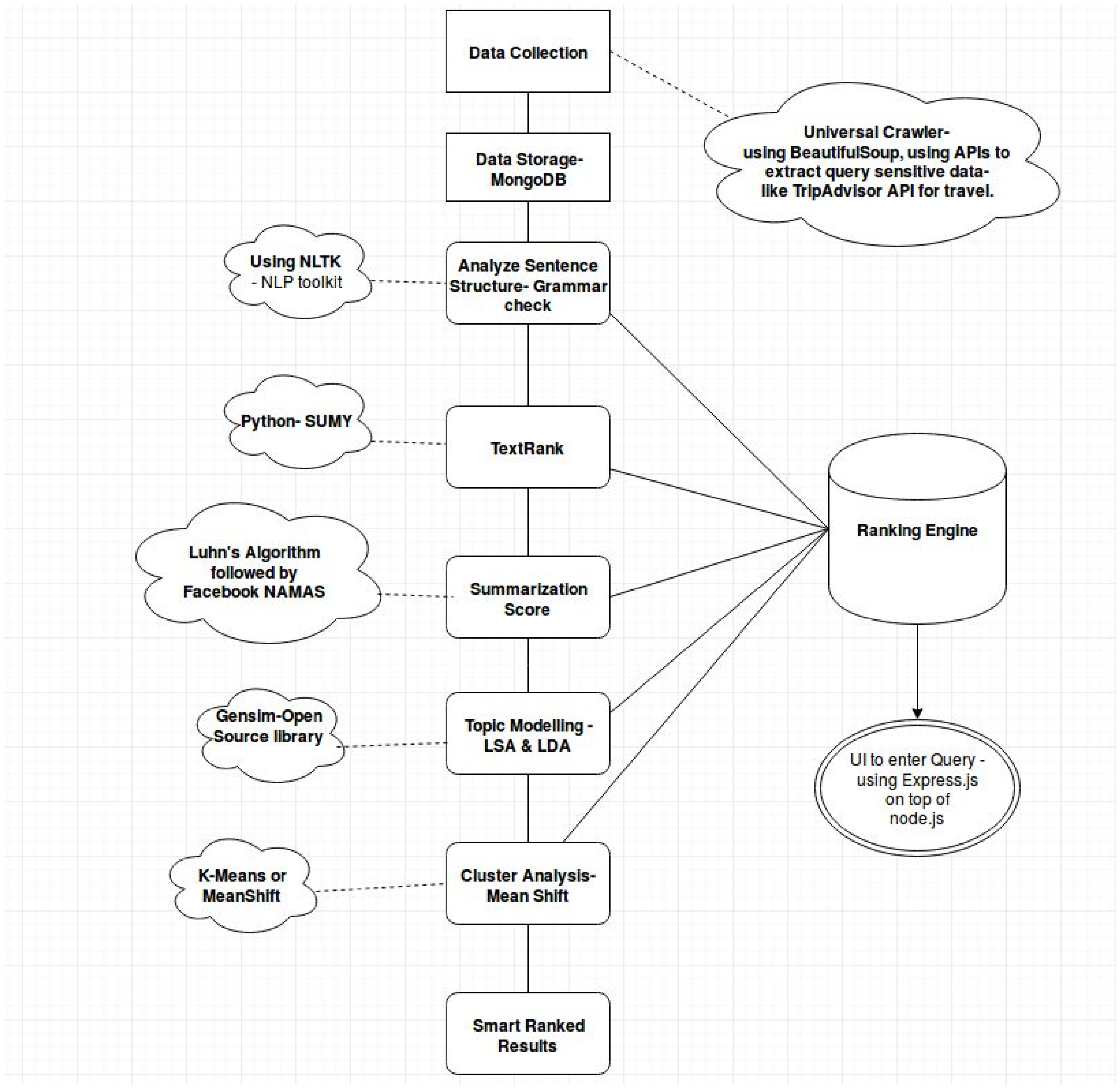
5) Topic Modelling: Topic modelling helps to analyze huge amounts of unlabelled text. Topic modelling helps distinguish between words based on its contextual clues, meanings, and multiple meanings. Hence this is a very important phase which helps to semantically generate relevant content and thereby helps in ranking the content and improve the score of documents. This will be achieved by a combination of LSA(Latent Semantic Analysis) and LDA.

6) Cluster Analysis: Dividing the whole corpus into multiple clusters to find different sub-topics that the entire corpus contains. The size of the cluster and the proximity of the centroid to the search query determines how important each cluster is. Further ranking can to attributed to the articles based on the influence of the article on each cluster and how important the said cluster is in the results. Cluster Analysis also helps marginalize outliers and anomalies that are not relevant to the user. Unsupervised algorithms like K-Means or Mean Shift can be utilized.

The ranking engine takes in input from the above five components and calculates the score for each crawled document - which is representative of the quality of the document.

7) Final Results: After procuring the ranking metric score for each crawled document, we synthesize the content by sorting the results based on the quality score.

FLOWCHART:



Data Collection
Universal Crawler- using BeautifulSoup, using APIs to extract query sensitive data- like TripAdvisor API for travel.
Data Storage- MongoDB
Using NLTK - NLP toolkit
Analyze Sentence Structure- Grammar check
Python- SUMY
TextRank
Ranking Engine
Luhn's Algorithm followed by Facebook NAMAS
Summarization Score
Gensim-Open Source library
Topic Modelling - LSA & LDA
UI to enter Query - using Express.js on top of node.js
K-Means or MeanShift
Cluster Analysis- Mean Shift
Smart Ranked Results

TECH STACK:

Web Crawling: Universal Crawler through Python's BeautifulSoup and urllib2. In addition, domain specific APIs (like TripAdvisor/Amadeus API for Travel, Zomato API for food etc.) are also used, with major focus on gaming, entertainment, travel, retail and finance.

Storage and Indexing: MongoDB for intermittent storage and indexing. Permanent storage and index potentially on a distributed file system - HDFS.

API for front-end users of the Ranking Algorithm: Express.js on top of Node.js.

NLP and Grammar Analysis: Python's nltk supported by custom algorithms.

Summarization Scores: Custom implementation of Luhn's as well as sumy's implementation of TextRank and Luhn's algorithm.

Neural Networks for Abstractive Summaries: NAMAS/TextSum on TensorFlow.

Topic Modelling: Gensim Library for a combination of LDA and LSA.

Cluster Analysis: Python's sci-kit learn package will be used to build unsupervised learning algorithms like K-Means (Agglomerative clustering) / Mean Shift (Hierarchical Clustering) clustering.

Final Output: A web interface for displaying ranked synthesized content. Express.js will be used to build the web server and the middleware, with Python taking care of all processing and computation. Web service API, the web front-end connected through ZMQ queue based communication system API and the mongoDB storage and Indexing connected through mongoose ORM.