#UNITEDBYHCL

Unsupervised Spider

Synthesis of quality content from the web for any given search query

MultiHumanPerceptron

Prayansh Srivastava

Pratulya Bubna

Rohit Chugh

Sarthak Manchanda

PROBLEM STATEMENT

Create a tool that could scrawl the web on a specific topic and create a synthesis of the content found — the challenge is to mitigate the sources depending on their quality.
Solution should be show content from various sources and rate those sources based on the quality of content. Also, solution should be able to crawl automatically and learn from the content and the source.

WHY US?

- Completely generalised unsupervised Machine Learning algorithm
  - Our algorithm actually ranks on the basis of the content of the website, something that other search engines won't do
  - An algorithm that works on any kind of search query
- Full asynchronous support
  - Providing real time service
  - Supporting a multi-user enviornment while achieveing maximun concurrency

OVERVIEW

- A tool to rank the search results solely on the basis of the content. On the other hand, other search parties do so by keeping in consideration that which article would the user be most likely to click. There are also AdWords or Reserved Words, Paid Appreciation, In-nodes/Out-nodes Basis and Paid Advertisements which rank up a webpage that might not be rich in content.

- Also, most of the major search parties give minimal importance to the content in the webpages during ranking. Therefore, our goal is to remove such discrepancy and rank the webpages solely on the basis of content present in them. Hence, No Clickbait.

- We do this by gathering search results from various search engines and processing the results obtained from them by using our Custom Made Ranking Algorithm which clusters the results and then rank them solely on the basis of the content

STEPS INVOLVED

1. Input the query from the user
2. Convert that text query into a search query url to gather results from various search engines such as Google, Yahoo, Bing and DuckDuckGo
3. Scrape the results webpages using our Custom Parser and store the results in a file
4. Visit each link gathered in the above step and get the content of the webpage
5. Process each webpage and rank them solely on the basis of the content with the help of our Custom Ranking algorithm
6. Render results to the user.

PAST What we had dreamt of

Synchronous Architecture

Django (WSGI web framework)

Multi-threading to achieve concurrency

Prototype on local environment

Naively comparing search results in order to find similarities

PRESENT What we have built

Asynchronous Architecture

Tornado (Alternative to WSGI)

Tornado coroutines

Deployment on Heroku (https://unsupervised-spider.herokuapp.com)

ML based scalable and accurate rating system

Actually finding relationships between search results as a rating metric

FUTURE What we plan to achieve

Hosting on a dedicated server (AWS)

Load Balancing on the servers. (if chose to server on multiple)

Caching of search query results, which would ensure a Search Engine like feel.

WEB CRAWLING SPIDER

The spider is booted with the search query and is allowed to extract the text from links crawled from the 4 major search engines:

- Bing
- DuckDuckGo
- Google
- Yahoo

The relevant websites' links, that are serving the required content, are crawled from the search results' page of these search engines.

To follow the modularity and reusability practices, the spider has been implemented as 2 major components:

1) Parser:

- TextParser: Extracts relevant text from a page.
- SearchResultParser: Abstracts the functionality to extract all the links from a search result page.

  Personalized classes for each Search Engine subclass this class and define their own extractor (SoupStrainer):

  - GoogleParser
  - BingParser
  - DuckDuckGoParser
  - YahooParser

- Why use SoupStrainer class of BeautifulSoup?

  Subclassing SoupStrainer class allows to parse the page selectively, i.e. only for the elements you'd need.

  This helps in achieving faster parsing as a selected parse tree is created.

2) Scraper:

Highlights of Scraper:

- Asynchronous architecture and design.
- Makes non-blocking fetch request to perform the network IO read.
- Each fetch request has a timeout limit of 5 seconds. (modifiable)
- For its execution, scraper maintains two queues:
  - se_queue: Maintains the urls of the search engines, after formatting it with the quoted & escaped search query.
  - links_queue: Stores the search results' links parsed from a search result page.
- Implementation is done using Python's coroutines, generators and yield syntax.

Working of the Scraper:

- Takes as an argument a search query.
- Puts the search engine urls in the se_queue. (cleaned & formatted)
- Boots concurrent workers
    - Workers get a url from queue;
    - Each worker makes an asynchronous request to the url;
    - Workers yield the control until the network IO is complete;
    - Workers resume parsing the scraped page once the asynchronous fetch method returns with the response;
    - Workers put the parsed links into the links_queue.
- links_queue gets filled with required links.
- Again, boot workers with a defined concurrency (adjustable)
    - Workers repeat the asynchronous network IO cycle
    - They parse the respective pages for relevant text
    - Append to result dictionary
- Make a call to the rater with the result.
- Return the rated result.

WHY NOT SYNCHRONOUS ARCHITECTURE?

- The application is Network IO read bound. Also, the results have to be served in real-time.
- The standard network read methods in Python, such as ‘get’ methods implemented by ‘requests’ and ‘urllib’ libraries, are blocking in nature. Therefore, in a single-threaded environment, only 1 request can be served at a time.
- Synchronous architectures tend to be blocking in nature, i.e., a client connected has a dedicated thread in the server, which will not be freed until the expensive, time-consuming and blocking network IO read has finished.

NEXT: Why not Multi-Threading?

WHY NOT MULTI-THREADING?

- In synchronous architectures, a common way-around for its aforementioned caveats, is multi-threading because of its simplicity and synchronous programming style.

- In a multi-user environment, to achieve concurrency and spawning multiple threads for each user:
  -> is extremely inefficient;
  -> requires tremendous CPU resources;

- For example, in a synchronous architecture server, if 1MB of memory is required for each thread, and the web server has 2GB of RAM, this web server would be capable of processing (approximately) 1000 requests (spawning 2 threads per user) until the server stops responding to the new requests.

- Clusters of servers would be required to cater to the increased load.

WHY NOT MULTI-THREADING?

- In multi-thread concurrency, even though when a thread is blocked waiting on a network call to return, other requests can be processed at the same time, yet thread utilisation is lower, even if the number of threads executing is much larger.
- Threads are memory-intensive and their context-switching is an expensive operation.
- There are issues with locking and synchronisation. Single-thread asynchrony doesn't suffer from the same problems.
- Since application is network IO bounded, there would be a cascading delay effect for and with each subsequent request.
- Also, multi-threaded environment is recommended for CPU-bound applications.

WHY ASYNCHRONOUS ARCHITECTURE?

- Due to the application's Network IO read bounded nature, asynchronous execution model fits better.
- In an asynchronous execution model, server still processes requests sequentially and in one thread, but can transfer control to another request handler when there's nothing to do, e.g., while network read is being completed.
- Asynchronous architecture is best-suited for this application which requires real-time serving, while achieving maximum concurrency for the network IO read operations, in a multi-user environment.

HOW DOES TORNADO FIT IN?

- Tornado is a Python web framework and asynchronous networking library. It has a non-blocking network I/O design, ideal for our purpose.
- It offers a full-stack alternative to **WSGI**.
- Tornado uses a single-threaded event loop (called IOLoop). IOLoop checks for IO events on sockets, file descriptors, etc (with help of epoll, kqueue or select).
- One of the greatest benefits of single-thread asynchrony is that it uses much less memory. In multi-thread execution, each thread requires a certain amount of reserved memory. As the number of threads increases, so does the amount of memory required just for the threads to exist. Since memory is finite, it means there are bounds on the number of threads that can be created at any one time.

COROUTINES

- Coroutines are the recommended way to code in Tornado.
- They use the Python yield keyword to suspend and resume execution instead of a chain of callbacks.
- In Tornado, all coroutines use explicit context switches and are called as asynchronous functions.
- Coroutines are almost as simple as synchronous code, but without the expense of a thread.
- They also make concurrency easier to reason about by reducing the number of places where a context switch can happen.

WORKING OF TORNADO COROUTINE

- A function containing yield is a generator.
- All generators are asynchronous; when called they return a generator object instead of running to completion. (**as opposed** to normal synchronous functions, which do everything they are going to do before returning).
- the @gen.coroutine decorator communicates with the generator via the yield expressions, and with the coroutine's caller by returning a Future. (eg. Promise in JavaScript)
- The decorator receives a Future from the generator, waits (without blocking) for that Future to complete, and sends the result back into the generator as the result of the yield expression.

ASYNCHRONOUS NETWORKING IN TORNADO

- tornado.ioloop is an I/O event loop for non-blocking sockets.
- tornado.queues module implements an asynchronous producer/consumer pattern for coroutines, analogous to Python's standard library queue.

Sending Request In Tornado

- Tornado provides an AsyncHTTPClient().fetch(url) method that fetches data from url.

How Is It Different From Requests Or Urllib?

- AsyncHTTPClient().fetch method in Tornado is a non-blocking and asynchronous method which returns a generator on call and adds the result to the returned Future object after fetching the response.
- Whereas, requests.get or urllib.urlopen are blocking methods, which block the execution until the response is received.

SCALABILITY AND DEPLOYMENT

Setting Up Local Environment

- pip install -r requirements.txt
- python main.py (requires python >= 3.3)
- open localhost:5000 in your web browser

Production Environment

- https://unsupervised-spider.herokuapp.com

SCALABILITY:

- Deploying the application on a dedicated server (VPS on DigitalOcean).
- Load Balancing on the servers. (if chose to server on multiple)
- Caching of search query results, which would ensure a Search Engine like feel.

RATING ALGORITHM

- Each search result will conceptually belong to a category
  For eg results for the search query MacBook Pro can broadly categorised as:
  - Articles on MacBook
  - E-Commerce websites
  - Forums

- Out of these, we consider that only articles provide us high quality content
  We then rank the best articles based on relevance and quality

STEPS INVOLVED

- Filter each search result by extracting only the important words
  - Data Preparation
- Represent each search result as an array of numbers
  - Vectorization
  - Tf-Idf
- Dynamically find types of categories present in search results
  - LSA
  - SVD
- Cluster search results with respect to their category relevance
  - Clustering
  - Spherical K-Means Clustering
- Rank search results in their respective cluster

RESULTS Full results can be seen in results.txt

- Search query: "Macbook pro"

The result was an easy split between 2 types of categories. Category 1 results were shown in top. Top few results are shown

Category 1: Articles about Macbook Pro [124 results]

- https://www.apple.com/macbook-pro
- http://www.telegraph.co.uk/technology/2016/11/25/macbook-pro-2016-review-apples-almost-perfect-laptop
- https://www.engadget.com/2016/11/14/macbook-pro-review-2016
- http://www.macworld.co.uk/review/mac-laptops/macbook-pro-15-inch-2017-review-3659985

Category 2: E-Commerce websites to buy Macbook Pro [6 results]

- https://www.ebay.co.uk/b/MacBook-Pro/111422/bn_446522
- https://www.amazon.com/macbook-pro/s?ie=UTF8&page=1&rh=i%3Aaps%2Ck%3Amacbook%20pro
- https://www.snapdeal.com/brand/apple/computers-laptops

SCALABILITY

- Page System

  A multipage result system, just like pages in google search can be easily implemented by applying our rating algorithm on each page.

  More number of results per page gives better accuracy but increases the scraping time. So based on the internet speed of our server, we can calculate the optimal number of results per page and make our search fully scalable

RATING ALGORITHM

DATA PREPARATION

- **Removal of stop words:**

  Stop words are common words in the language which convey no semantic meaning

  [This, is, a, sample, array, which, we, are, going, to, vectorize]

  ↓

  [This, sample, array, which, we, going, vectorize]

- **Removal of unique words:**

  No information can be inferred from words which occur in only one document in our entire dataset. Hence they are removed.

- **Removal of frequent words:**

  Words which are present in more than 50% of the documents would not help in clustering documents but will only skew our clusters. Hence they are removed.

Now that we've got our dataset prepared, we're ready to convert it into numbers for our algorithm to understand it

VECTORIZATION

- A vectorizer converts a document into a set of numbers representing it

[This, sample, array, which, we, going, vectorize]

[1.307, 2.603, 1.64, –0.019, 0.016, 0.02, 2.57, 1.157, 3.26, –1.12, 2.58]

- Vectorization is an essential step for any machine learning algorithm.
  - Vectors converts data from text space to number space, and hence can be used for computation
  - Vectors allow us to compute the similarity between two documents. Two documents having similar vectors will be similar themselves.

TF-IDF VECTORIZER

Tf-Idf score of each word in a document is proportional to :
number of occurrences of that word in a document * rarity of that word in the dataset

- Term frequency : $\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$
  - $f_{t,d}$: Frequency of the given term t in given document d
  - $\max\{f_{t',d} : t' \in d\}$: Max frequency among all terms in given document d
- Inverse Document frequency : $\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$
  - N : Total number of documents in dataset
  - D : Total number of documents containing the term t
- For each word in each document compute :

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

We have represented each document in our dataset as an array of numbers
Also we have performed something called as 'feature extraction'
That is, our vectorizer has highlighted the important words in each document

Now we need to modify that array so that we have a way to compare two documents
This will help us to categorise them into 'good' and 'bad' categories

LSA

- Latent Semantic Analysis is a technique for analyzing relationships between a set of documents and the terms they contain, by producing a set of concepts, related to the documents and term.

- We chose LSA here because it is known to model human conceptual knowledge quite well.

  Study: Comparison of Human and Latent Semantic Analysis (LSA) Judgements of Pairwise Document Similarities for a News Corpus (Brandon Pincombe et al.) http://www.dtic.mil/dtic/tr/fulltext/u2/a427585.pdf

- LSA :
  - Mitigates the problem of identifying synonyms
  - Preserves semantic relationships between words of a document
  - Reduces the dimensionality of the dataset
  - Is basically just a fancy name for applying SVD on vectorized text

SVD

- Singular Value Decomposition is a mathematical function which gives us the optimal low rank approximation of a matrix.

  i.e If we take our data and use SVD to represent it in a small number of dimensions, it will be able to identify the best possible dimensions to project our data on.

- SVD discovers a set of concepts (latent dimensions) for representing our documents. That is, it defines every document in our matrix as a linear combination of these concepts, just like a human.

  For eg. an article about 'German Vehicles' can be roughly defined as

  0.3 x 'German' + 0.7 x 'Vehicles'

- This provides us with a very efficient way to compare documents. If two documents are conceptually similar, they will have similar SVD vectors.

SVD

- We take the original input matrix A and decompose it into a product of 3 matrices

$$A_{[d\times w]} = U_{[d\times c]} \sum_{[c\times c]} (V_{[w\times c]})^T$$

d: document w: word c: concept

- A : Document to Word Matrix
- U : Document to Concept Matrix (Left Singular and Orthogonal)
- $\sum$: Strength of each Concept (Positive Diagonal Matrix)
- V : Word to Concept Matrix (Right Singular and Orthogonal)
- c : Concepts (Rank of Matrix A)

SVD

A

	$w1$	$w2$	$w3$	$w4$	$w5$
$d1$	1	1	1	0	0
$d2$	3	3	3	1	0
$d3$	4	4	4	0	0
$d4$	5	5	5	0	0
$d5$	0	2	0	4	4
$d6$	0	0	1	5	5
$d7$	0	1	0	2	2

=

U

	$c1$	$c2$	$c3$
$d1$	0.13	0.02	−0.01
$d2$	0.41	0.07	−0.02
$d3$	0.55	0.09	−0.04
$d4$	0.68	0.11	−0.03
$d5$	0.15	−0.59	0.65
$d6$	0.07	−0.73	−0.67
$d7$	0.07	−0.29	0.32

X

Σ

12.4	0	0
0	9.5	0
0	0	1.3

X

V'

	$w1$	$w2$	$w3$	$w4$	$w5$
$c1$	0.56	0.59	0.56	0.09	0.09
$c2$	0.12	0.02	0.12	0.60	0.69
$c3$	0.40	−0.80	0.40	0.09	0.09

SVD ANALYSIS

U

	$c1$	$c2$	$c3$
$d1$	0.13	0.02	−0.01
$d2$	0.41	0.07	−0.02
$d3$	0.55	0.09	−0.04
$d4$	0.68	0.11	−0.03
$d5$	0.15	−0.59	0.65
$d6$	0.07	−0.73	−0.67
$d7$	0.07	−0.29	0.32

- SVD finds concepts in our dataset and assigns a 'Concept Value' for every Concept in each document
- U matrix tells us how much a document is based on each concept
- Here d1, d2, d3, d4 are highly based on c1
  Whereas d5, d6, d7 are highly based on c2, c3

Σ

12.4	0	0
0	9.5	0
0	0	1.3

- Every concept has it's own 'Confidence Score', which tells us how confident our algorithm is about that Concept
- Here our algorithm is very confident about Concept 1 with value 12.4 and less confident about Concept 3 with value 1.3

Concepts with a Confidence Score less than a threshold (say 5.0) are ignored.
Hence Concept 3 with value 1.3 will be removed

SVD

- After removing the Concepts with small Confidence Scores, we are left with U and $\sum$ matrices with reduced number of columns

- To get the final SVD vectors, we simply multiply the U and $\sum$ matrices, so as to get the weighted concepts values for each document

	$c1$	$c2$
$d1$	0.13	0.02
$d2$	0.41	0.07
$d3$	0.55	0.09
$d4$	0.68	0.11
$d5$	0.15	−0.59
$d6$	0.07	−0.73
$d7$	0.07	−0.29

U[:2]

X

12.4	0
0	9.5

Σ[:2]

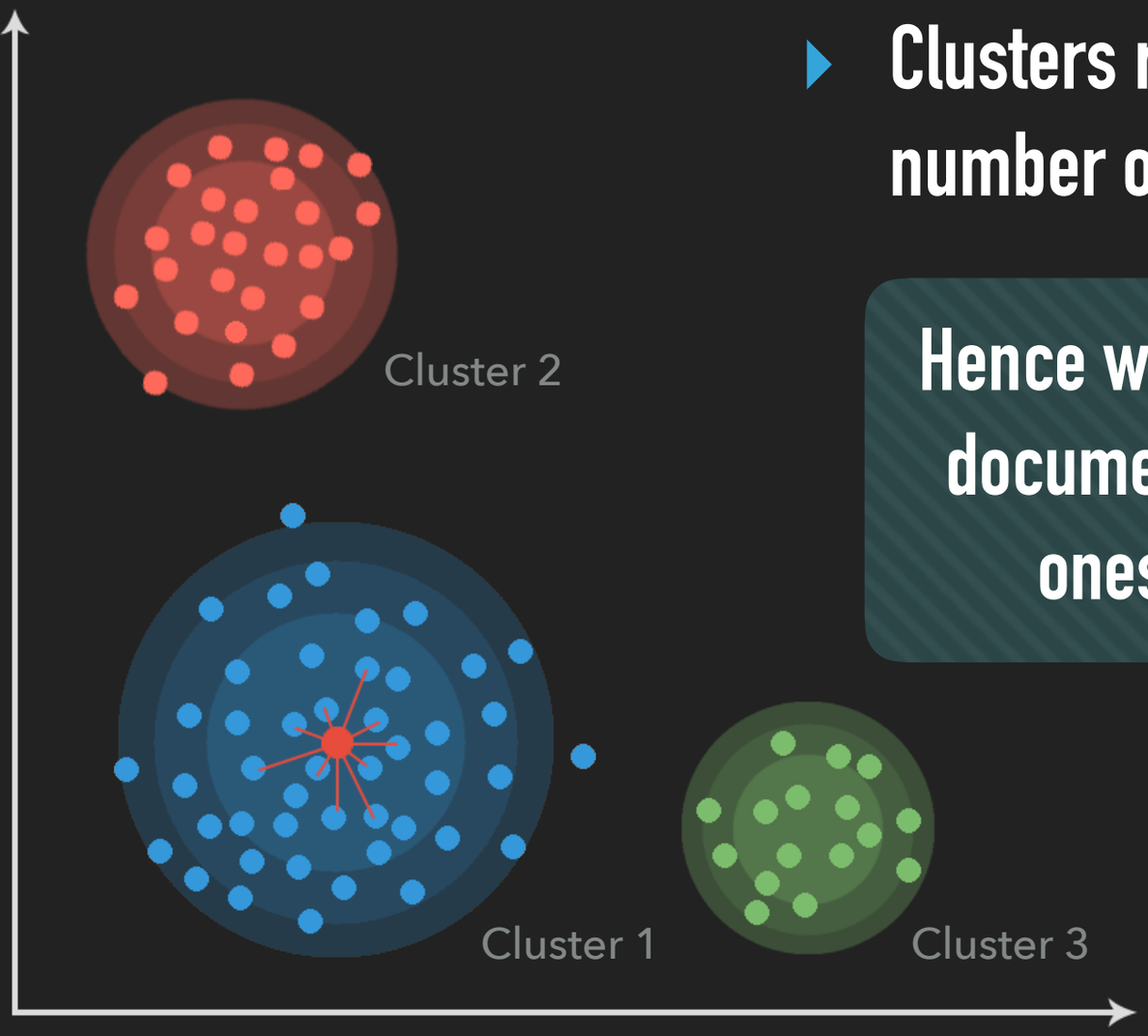
We've defined each document by an array of numbers such that comparing any two documents using any similarity function (We use cosine similarity) will tell us how conceptually similar they are

Now we need to use this similarity property to categorise our dataset into 'good' and 'bad' categories

CLUSTERING

- Clustering is done to group documents based on similar Concepts together
- It will help filter out results which have unwanted concepts in them

  For eg, concepts such as forums, social networks, e-commerce

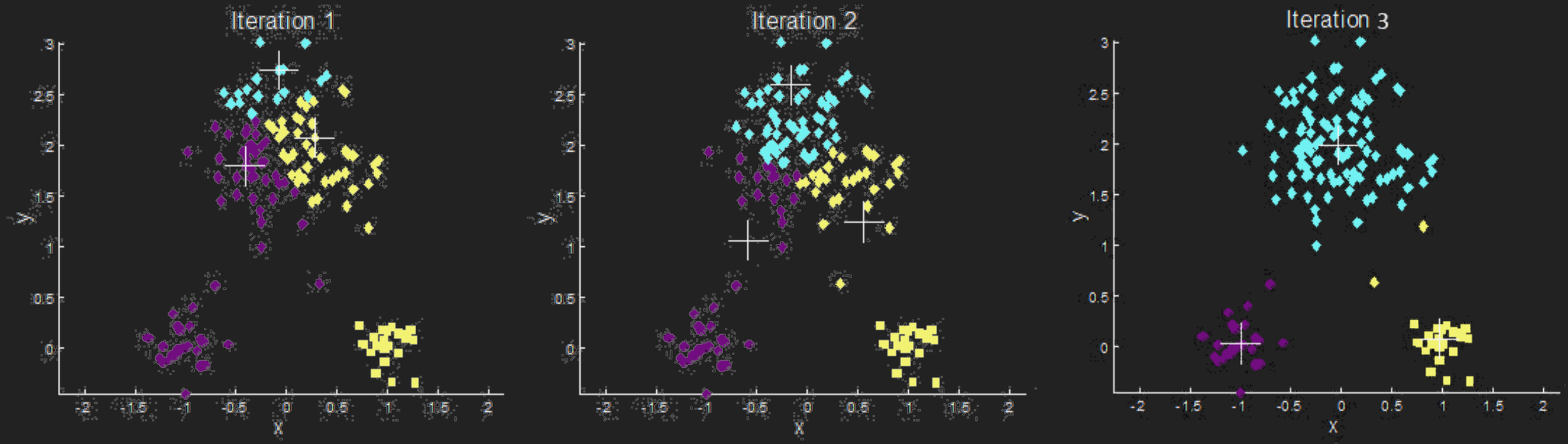


- Clusters representing 'bad' concepts will have less number of results in them, as they are very specific

Hence we sort the clusters based on the number of documents in them, putting 'bad' clusters, i.e the ones with unwanted concepts, at the back

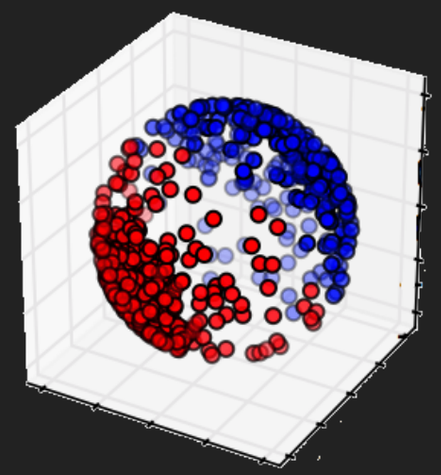
SPHERICAL K-MEANS

- First let us explain normal K-Means clustering algorithm



- First N random cluster centers are defined (the three +)
- Then each point is assigned to the cluster with the closest cluster center. This closeness is calculated by a 'distance function'
- Each cluster center is moved to the average of the points in a cluster
- This step is repeated until displacement of cluster centers between iteration is not very small

SPHERICAL K-MEANS



- For text classification, Spherical K-Means works better than normal K-Means at the expense of execution speed

Refining clusters in high dimensional text data (Inderjit S. Dhillon * Yuqiang Guan, J. Kogan et al)
https://grid.cs.gsu.edu/~wkim/index_files/papers/refinehd.pdf

But as we only need to cluster a few hundred documents at max, reduced speed is not an issue

- In it, the SVD result vectors are first normalized. i.e their value is transformed between 0 and 1.

- This is done to be able to use Cosine Similarity rather than the Eucledian Distance as the 'distance function' for clustering

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Phew! One final step left

We've successfully segregated good and bad categories

We've also sorted the categories such that good categories come before the bad ones

But for each category, we also need to rate their respective documents

THE FINAL STEP

- The final step is quite straightforward:

  For each cluster:

    Sort the documents by increasing order of distance from that cluster's center

  We do this because the documents closer to the cluster's center will 'belong' more to that cluster than the documents far away

Clusters are sorted, documents in each cluster are sorted
Append their respective links in order in a list and return!

IMPROVEMENTS

To be implemented before the onsite round

- ## Search engine weighting

  A small weight should be added for results based on google ranking, this weight should be exponentially decreasing.
  For eg. first 100 google search results for query 'Manchester United' :

  1. Has manutd.com as the top result, which does not contain any textual information about the club
  2. Does not contain manutd.com/en/History.aspx, which does contain a textual history of the club

  To counter this problem, a weighted biasing is required

- ## Dynamically setting number of clusters

  Set number of clusters based on number of search results and scraped data

TECHNOLOGY STACK

- Python3
  - Numpy
  - Sklearn
  - BeautiulSoup
- Tornado
- MaterializeCSS

DISCLAIMER:

- For prototype submission, we chose to opt for ease in deployment, and hence chose Heroku.
- However, Heroku has a serious drawback in concern with the working of our application, that is, it closes the client connection within 30 seconds, irrespective of any thing.
- Even though we've scaled and adapted our application to serve the data well within 20 seconds, there might be a minimalistic chance that an error occurs.
- If the issue persists, please retry or try in local environment.

For the final submission in UK, we plan to deploy our application on a dedicated server, as mentioned under the "SCALABILITY" section.

THANKS!