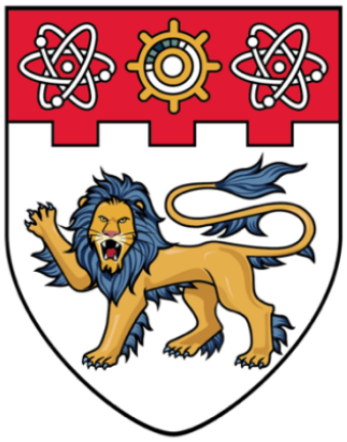


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CX4034: Information Retrieval
Group Project

GROUP 33

Vaidyanathan Abhishek	U1923980D
Iyengar Varun Srikant	U1922219K
Bodipati Kiran	U1922759J
Goel Tejas	U1923301G
Meenachi Sundaram Siddharth	U1923790H
Kartikeya Vedula	U1923891G

Youtube Link: https://youtu.be/Rzt8V3f5I4E
Datasets Link: Submission_4
Codebase Link: Submission_5

Table of Contents

Problem Statement

We propose a system named REDPILS (the REDdit Political Indian's Lexicon of Sentiments) which will serve as a one-stop search engine to retrieve opinions of the public at large on political matters of importance in India. The challenge of obtaining unbiased news on Indian political matters is particularly to contend with due to the multi-party democratic system in India as well as the disparity of politics across various locations across Indian states. We attempt to address this issue by designing a system that can efficiently Reddit comments on various Indian political matters, and simultaneously classify them as being left or right leaning in an effort to provide a comprehensive range of opinions that the user can draw insights from. Due to the anonymity offered by Reddit, we believe that a truly representative mix of opinions of the Indian public at large would be obtained.

Crawling (Question 1)

Reddit (Question 1.1)

As mentioned in the problem statement, we choose Reddit as our source to crawl data from. We make use of the PRAW: The Python Reddit API Wrapper library to interface with and scrape submissions and comments from Reddit.

We chose the 2 biggest subreddits for Indian politics to draw opinions from: r/india and r/IndiaSpeaks. The subreddit r/india is considered largely left-leaning, while r/indiaspeaks is considered largely right-leaning.

In order to find relevant political posts we make use of a Reddit feature known as flairs. A **flair** is a tag associated with a submission on a subreddit to describe its content. For our purpose the relevant flairs were:

1. "**Politics**" and "**Foreign Relations**" for r/india
2. "**Politics**" and "**Defence & Foreign Policy**" for r/IndiaSpeaks.

We devise a crawling method that can be readily extended to be dynamic to continuously mine data on political opinions of the public:

1. We start with the following basic set of keywords `{"BJP", "Congress", "AAP"}`. These keywords represent 3 major Indian political parties.
2. We crawl Reddit comments (the documents for our purpose) using the above 3 keywords and filter by the relevant flairs.
3. The crawled documents are indexed and stored in Solr (details of indexing covered in next section).

4. Using spacy's NER (named entity recognition) tool, we filter out those words in the just-crawled documents which are not considered to be either organizations, proper nouns, or the subject or the object of the relevant sentence. Further, all tokens which are potentially links are filtered out. Finally, any keyword which has previously occurred (this would be only `{"BJP", "Congress", "AAP"}` for the first round of crawling) is filtered out. The remaining filtered tokens are considered important tokens in the context of Indian political discussion.
5. The extracted list however may still be far too large. Using Solr's features, the average TF IDF score across all documents of each extracted keyword is computed. The 5 keywords with the highest scores are then used to further extract documents, as per step 1, and the process can be repeated to build a larger and larger corpus.

During the above procedure, we ensure **deduplication** using each comment's unique comment ID, assigned by Reddit.

Due to practical limitations of the size of the data crawled however, we perform only one round of expansion.

We include here the list of 5 most important keywords after the initial round of crawling:

```
{"kejriwal", "government", "tharoor", "punjab", "education"}
```

The above keywords are truly quite relevant. The term "kejriwal" refers to Arvind Kejriwal, the Chief Minister of India and the leader of the AAP party, while the term "tharoor" refers to Shashi Tharoor, one of the significant figures in the Indian National Congress party. The term "Punjab" refers to the Indian state of Punjab which likely turned up due to the state elections conducted in 2022, about which several comments would have been made.

System Utility (Question 1.2)

As explained in the problem statement, it is desired that the crawled corpus contains a sufficiently wide array of Indian political opinions. Hence, the crawled corpus has the potential to facilitate user queries such as:

1. Queries on key political talking points in India
    a. E.g.: "BJP Ram Mandir"
2. Queries on date ranges, to observe shift in perspective/important talking points
    a. E.g.:
        - "BJP" with a filter of Nov 2022 to Dec 2022
            - We would desire that the above query would return results on the BJP party's Gujarat manifesto as it was released at this time for the state elections.
        - "BJP" with a filter of all time

        - We would desire that the above query would return results which are not necessarily related to the above.
3. Perform geospatial search to gain insights about different political talking points in different locations across India
    a. E.g.: “AAP” with location filter of “Punjab”
        - The above purpose serves the purpose of obtaining comments related to AAP’s campaign in the 2022 Punjab election.
4. Queries with simple boolean combinations
    a. E.g.: “Sikh not Khalistan”
        - The above query serves the practical purpose of obtaining comments where commenters are talking about Sikhs without making a reference to the Khalistan movement, a separatist movement of Sikhs which has had a tendency to be militant.

Corpus Statistics (Question 1.3)

The crawled corpus has the following statistics:

Total documents	6679
Total words	485112
Total unique words	52826

As mentioned before, we perform only a single round of expansions due to the large nature of the individual documents for our setting. This is justified by the computed average length of each comment of **72.63 words**. This is in contrast to other data sources like Twitter where each tweet is capped at 280 characters, which is likely insufficient to even contain ~72 words.

It is due to the above reason that we have a corpus of only about 6679 documents.

Also, the crawled documents have an uneven split of comments from the 2 subreddits. In particular, there are **2495** comments from r/IndiaSpeaks and **4184** comments from r/india. However, given the wide variety of opinions expressed on each of the subreddits, a fairly even split of left-leaning, right-leaning and neutral data can hopefully be obtained. Also, for the purpose of classification, we ensure that we have an even split of data from each subreddit.

System Design (Question 2)

Indexing and Querying

We choose to use Solr for the purpose of indexing and querying.

We were unable to use the third-party pysolr library due to its limited capabilities and support. In particular, it was not maintained up to date with the latest versions of Solr, and had very limited support for the APIs provided by Solr, instead necessitating a great amount of manual configuration.

Hence, we build our own custom Solr interface library that uses Python's requests packages to construct requests for different use cases. All the methods required are encapsulated in a class called "solr_interface". We provide here a short summary of the essential methods of the API we created:

Method	**Description**
create_collection	Creates a collection with a given name, data schema, and any custom field types which are required to be created
delete_collection	Deletes a collection with a given name
add_new_field_type	Adds a new field type to a given collection.
define_schema	Created a new schema in a collection
replace_schema	Replaces an existing schema with a new schema in a collection
push_data	Pushes data into a given collection (thereby automatically indexing it)
delete_data	Deletes all data from a given collection
compute_avg_tf_idf	Computes the average TF-IDF score of a term across the documents in a given collection
compute_query_term_score	Computes a sophisticated score given a phrase query using Solr's extended DisMax parser. The query settings for this method are carefully configured and will be discussed in detail later.

Note that in Solr, a field type is a custom field which can have additional levels of analysis attached to it for the purpose of generating statistics later and facilitating more sophisticated queries. We make use of the add_new_field_type function to create a new custom field type called "**filtered_text**" in Solr, which can then be used to analyze our documents.

The “**filtered_text**” type has the following properties:

1. It extends Solr’s in-built TextField; hence, it is capable of storing text-based documents such as the comments in our use case.
2. It specifies a custom analyzer to be used when **indexing** the documents, the details of which will be delineated in the following paragraph.
3. It specifies a custom analyzer to be used when **querying** the documents, which has the same sequence of analysis as the analyzer used for indexing, with only one additional configuration.

The details of the analyzer are as follows:

1. It makes use of Solr’s standard tokenizer, which uses **whitespace** and **punctuation** as delimiters to determine tokens.
2. After these tokens are generated, a **trim filter** is used which removes leading or trailing whitespaces from tokens.
3. Next, a **lowercase filter** is used to convert all the tokens to lowercase.
4. Then, a **synonym graph filter** is used to treat certain lists of terms as equivalent. We made a curated list of common abbreviations and alternative vernacular used to refer to several relevant terms, in order to ensure that such equivalent terms are indexed as one and the same.
    - E.g. The terms “Congress” and “INC” both refer to the Indian National Congress party.

   For completeness, we include the list of all synonyms in the appendix.
5. Finally, a **stopword filter** is used to remove common stopwords. We make use of ntlk’s list of English stopwords.
6. For the query analyzer alone, Solr’s default similarity metric is overridden and instead Solr’s classic similarity metric is utilized. This allows us to compute TF-IDF scores of terms in the documents directly from Solr.

In order to achieve the above, it was required to build and upload a custom ConfigSet to Solr that contained the required files of custom synonyms and stopwords.

We provide below the schema used for our data collection:

Field	**Type**	**Description**
submission_id	string	ID of the submission from which the comment was retrieved
submission_title	filtered_text	Title of the submission from which the comment was retrieved
subreddit_id	text_en	ID of the subreddit from which the comment was

		retrieved
subreddit_name	text_en	Name of the subreddit from which the comment was retrieved
comment_id	string	ID of the comment retrieved
comment	filtered_text	Body of the comment retrieved
timestamp	pdate (Solr's date type)	Date and time when the comment was posted
url	text_en	Unique identifier beginning of the form r/{subreddit_name}/comments/{submission_id}/{submission_title_truncated}/{comment_id}. This can be prepended with "https://reddit.com/" to obtain a valid URL referring directly to the comment
reddit_score	pint (Solr' integer type)	Score obtained by the comment on Reddit, representing the value (number of upvotes) - (number of downvotes)
redditor_id	text_en	ID of the author of the comment
polarity	pfloat	Floating point score representing the political lean of the comment, where a score closer to -2 represents left-leaning, while that closer to 2 represents right-leaning, and 0 represents a neutral comment

All of the above fields are indexed and stored. In particular, note that the "timestamp" field is indexed. This facilitates querying on date ranges. Also, the fields "comment" and "submission_title" are of type "filtered_text" which allows us to issue queries against them in a special manner that takes into consideration query misspellings, queries using synonyms, case-insensitive queries, shingling for phrase queries, etc. Each of these enhancements are described in detail in the section on searching.

We provide below an example document that was scraped:


```
{
        "submission_id":"zpre2v",
        "submission_title":"\"2 Judges Can't Decide\": BJP MP's Strong
Objection On Same-Sex Marriages",
        "subreddit_id":"3d4x4",
        "subreddit_name":"t5_3d4x4",
        "comment_id":"j0uassz",
        "comment":"Lets take the argument that its not provided in Indian
culture. OK, SO WHAT ? There is no provision saying we have to live
```

```
according to Indian culture only. We have to live according to our rights
and duties in the Constitution and the laws based on it. Dont make Gay
Marriage in Hindu Marriage Act, but Special Marriage Act exists so that
non-dharmic marriages can exist. Whats the problem there ? Forcing people
to live according to your culture is tyranny.",
        "timestamp":"2022-12-19T21:58:40Z",
"url":"/r/IndiaSpeaks/comments/zpre2v/2_judges_cant_decide_bjp_mps_strong_o
bjection_on/j0uassz/",
        "reddit_score":68,
        "redditor_id":"bu6k2huu",
        "id":"04e7546c-ae7d-4b30-8cac-30ee44d16eb7",
        "polarity":0.0,
        "_version_":1762087678752325632}]
  }
```


Backend Server

We utilized the Flask web framework to design the APIs used to send data to the frontend built using React JS (which will be detailed later). This was then connected to the Solr database to query data directly and display it appropriately. The details of the APIs are discussed in further detail in the sections below.

Python's Flask Framework was used to create the backend, and the frontend is provided with the appropriate APIs to facilitate effective data processing and searching. The endpoints and associated parameters have been discussed below:

Query Endpoints

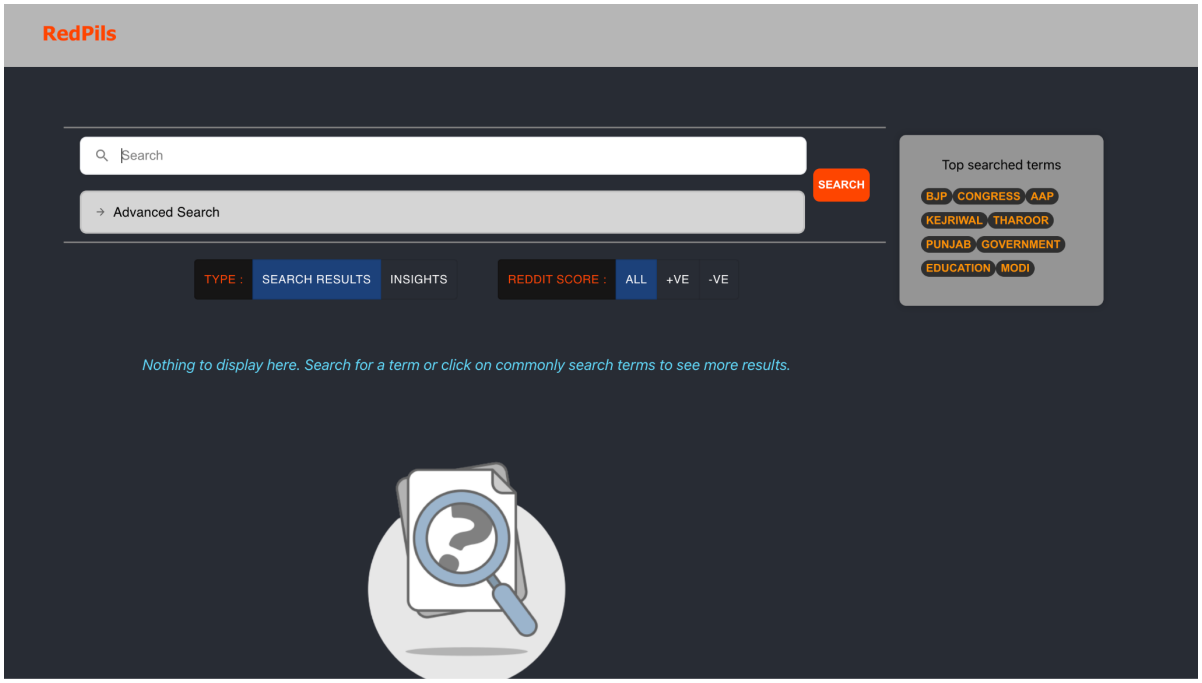
Endpoint	**Parameters**	**Output**	**Description**
/query [GET, POST]	mandatory: query - search query optional: region - for geospatial search	JSON Object contain: querying insights (number of results, search time) Search results, Search insights, Query Graphs	Monolith service that returns the search results with or without filtering, insights on the query and relevant data for visualizations.
/api/query_wordcloud	intitle - bool to	Wordcloud image file	Generates

[GET]	check if in title or not k - top k results to retrieve(def=10) polarity (=left, right, all, neutral) timeframe: -int, last n months to search data from. from, to - Date range to filter from (DDMMYYY) overrides timeframe if both are used.		wordclouds for the query configuration
/api/polarity_wordcloud [GET]	polarity (=left, right, all, neutral)	Wordcloud image file	Generates wordclouds based on the polarity of comments in the database
/api/geoplot [GET]	key(=num_results, reddit_score, score, polarity) colormap: Plotly color_continuous_scale	Plotly GeoJSON object (choropleth map)	Plotly Choropleth map for geospatial visualisation of insights.
/api/timedf [GET]	query	JSON dicts of average polarity, reddit scores and num_results for the time range from 2019 to 2022.	Returns a 3 month average time series of the insights on the query.

User interface

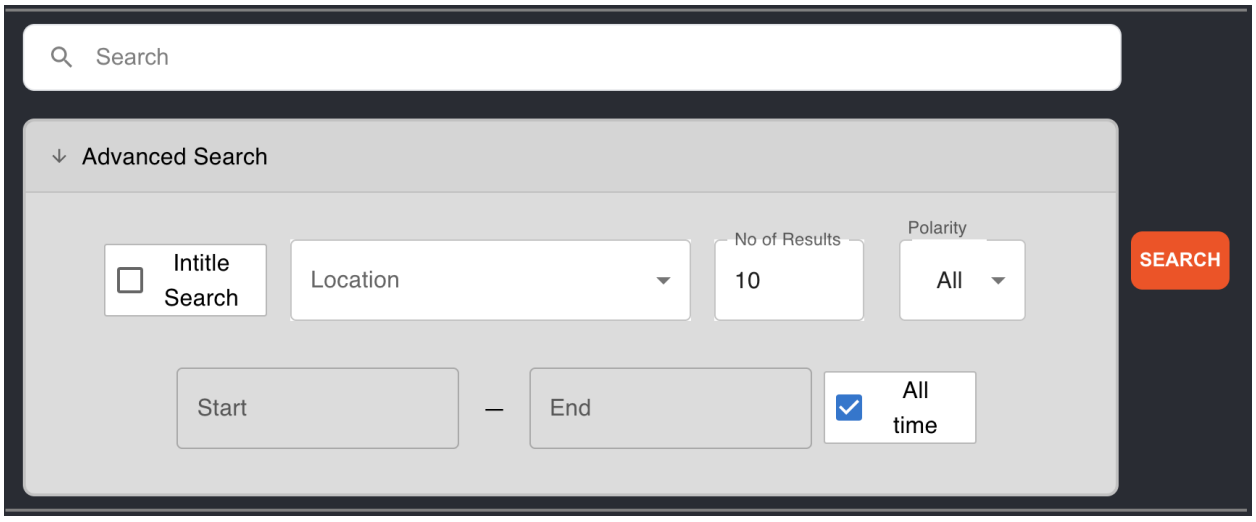
The web interface was designed using React JS. The application was designed to mimic the google search engine with a search bar and filters available to enable efficient search by the user. Furthermore, the web interface also shows the most important keywords used to crawl

Reddit to build our dataset. The details of obtaining these keywords are described in detail in sections above.



Advanced search

The advanced search component consists of different features that can be used to filter the search from Solr. The image below shows the expanded component for search.



The functionalities of the features above are described as follows.

Intitle search: This option output search results from both the reddit comments as well as the submission. It can be considered a useful search feature since the comments following a particular submission need not always contain specific search queries that the user may be searching for. Therefore, this allows the user to view the comments under each submission based on the context of the submission.

Location: Geospatial search is useful for our use case since our topic delves into the political scenario within different regions in India. Therefore, the user can input search queries pertaining to any location inside the Indian subcontinent to view posts discussing the region. Note that the location filter doesn't tap into the actual location associated with the user posting the reddit submission or comment.

No. of results: Allows the user to select the total number of query results displayed in the UI. The documents are ranked using Solr's in-built eDismax scoring with other factors taken into account as described later in the section on Searching.

Polarity: This measure allows the user to select a specific class of data based on the polarity. This filters the queried data as well as the insights to analyze statistics for the particular class of data. Later sections describe the differential analysis conducted for each class of data based on the polarity feature.

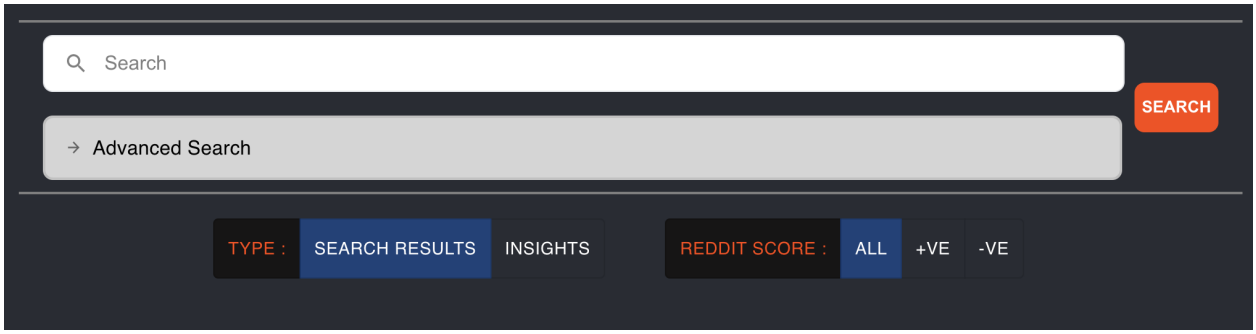
Date range: The date range allows the user to filter the search results to contain results from a specific time period. The results are obtained based on the time range applied to the comments. Besides specifying a given time period, the user can select a specific time range from which the results need to be displayed. This time period is applied on the comments and the results are returned based on the given search query.

From the description above, it is evident that each of these search filters can be used simultaneously. This gives the user greater flexibility to filter and query results.

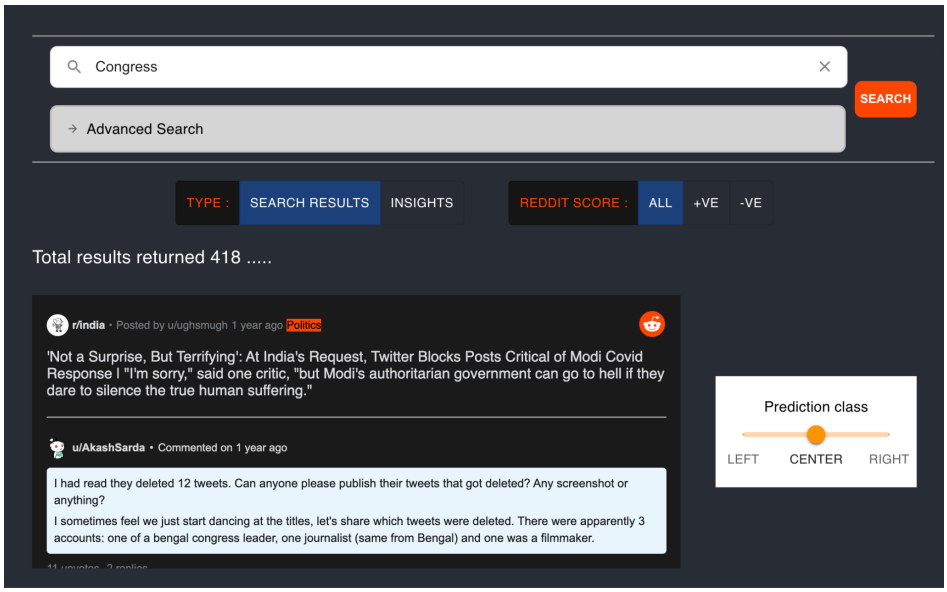
Results filters

Apart from the search results, we also allow users to filter and view relevant posts based on the query results. For example, since reddit allows users to like or dislike comments represented by upvotes or downvotes, each reddit comment is associated with a particular reddit score. This score is calculated by subtracting the total number of downvotes from the upvotes. Therefore, a comment with greater number of upvotes will have a positive reddit score and can be viewed as a more desirable result with respect to the submission.

The image below shows the two such options that can be used to filter the returned query results.

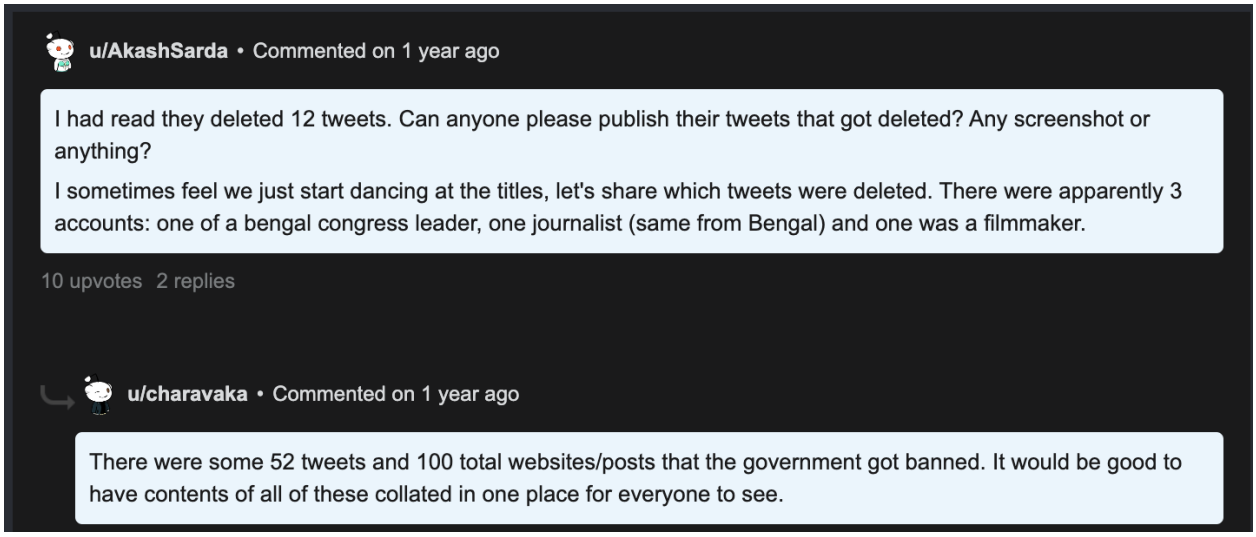


Type: The type indicates whether the user wants to see the actual posts or comments or view analysis based on the search results obtained. Images below show the difference in the two sections.

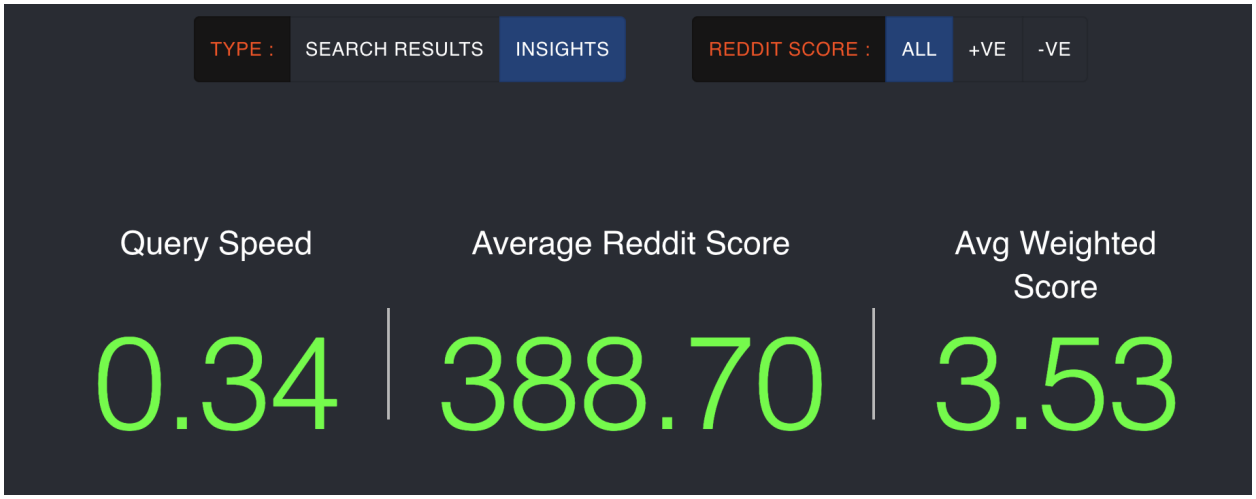


The image above shows that the posts or submissions are returned in the “Search Results” tab. The prediction class obtained from the classification models is depicted to show the sentiment of the given comment. Note that the prediction class represents the sentiment of the comment and not of the submission title.

Each of the reddit submissions or posts show comments up to 3 levels of comments. The user can scroll through each of the posts to view subsequent comments. Furthermore, these posts are clickable which implies that the user can click on each of the posts/submission which will lead them to the original reddit page displaying the post.

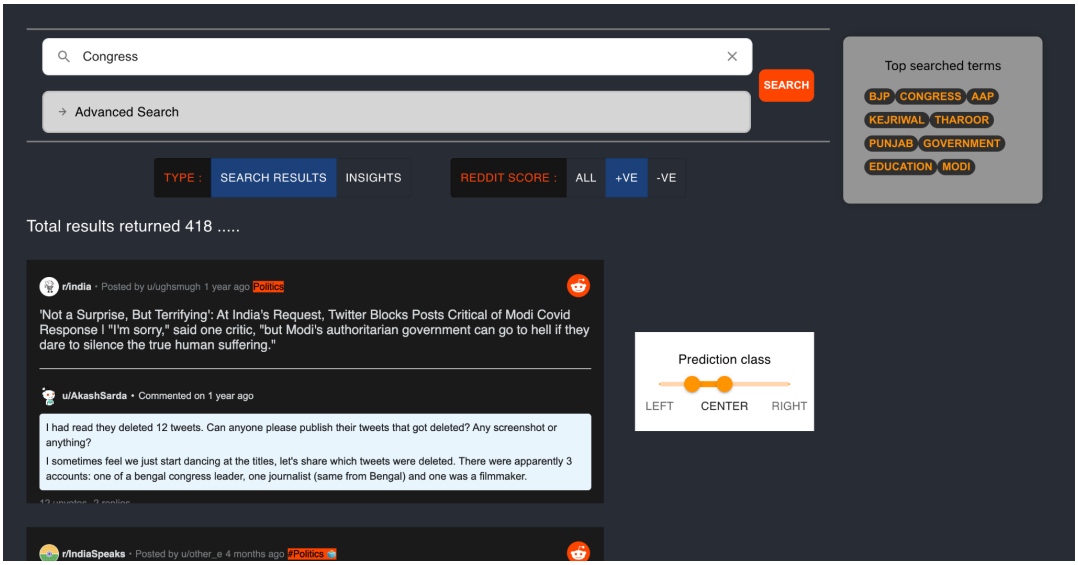


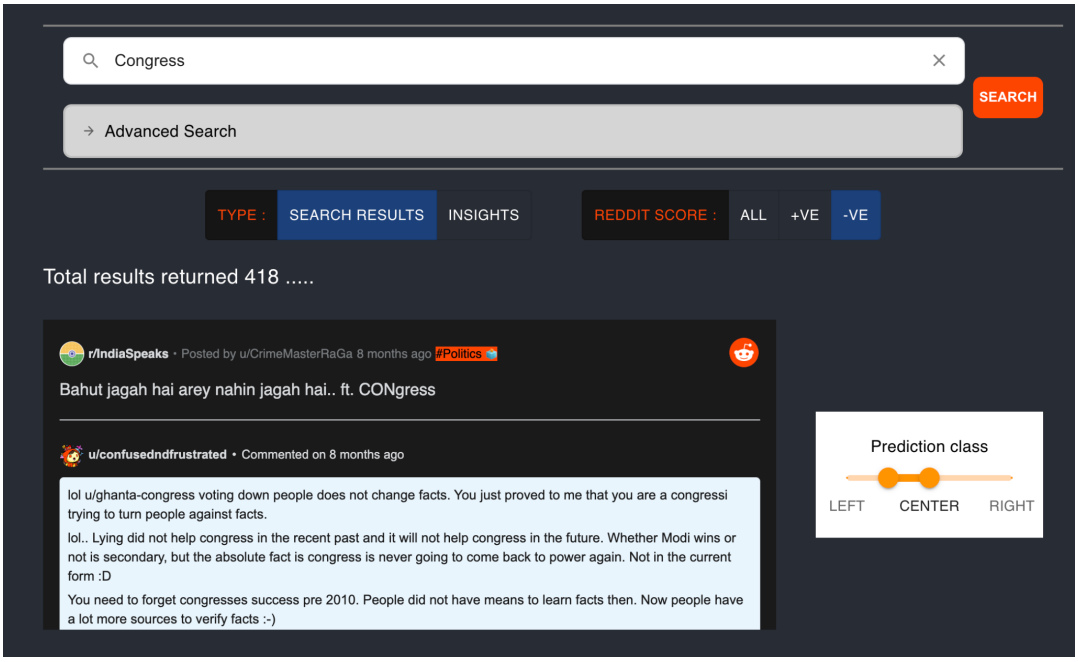
The user can also choose to view insights drawn from each of the query results. For example, the count of leftist or rightist comments and visualizations such as word clouds and geospatial visualizations. These visualizations are explained in more detail in later sections.



Metrics such as the query speed, average reddit score and the weighted score using the custom scoring function are displayed in the we interface. This helps to display the basic analysis of the query and the results obtained from the query. Note that the query speed is extremely fast since we leverage Solr's superior indexing methods to obtain our results. As discussed in the previous sections we utilize the various features offered by Solr, to make our information retrieval system more efficient.

Reddit Score: As discussed earlier the user can also choose to hide specific posts based on the reddit score. For example with a search term congress we can see the difference in the posts displayed based on the "Reddit Score" filter.





The interface also shows the total number of results obtained from the given search query. Number of results displayed is still dependent on user selection from the advanced search component.

Search Queries

The basic structure of the search query can be written down as below:

NOT(Word1) AND (word2~3 OR word3*) Word

The components can be broken down as below:

- The search word/phrase
- Boolean operators: AND, OR, NOT
- Parenthesis: Generally used to emphasize order of processing

- * : The wildcard query operator
- ~ : Fuzzy Search operator for edit distance.

The details of each of the components have been mentioned in the latter sections. The user can specifically mention the detailed structure if they wish to. However, default configurations have been provided for each case and advanced filters have been provided as well. The user can hence just enter a phrase/word they want to search for, and choose the advanced search filters provided in the UI accordingly.

Query type	Search Term Query	Other Query Filters	No of results	Description of Top Results	Query Speed (secs)
Wildcard Queries	*	All time	6679	All results from the database are returned	0.51
Wildcard Queries	*dutv*	All time	840	All results contain words starting with “Hind”. Results show posts with words such as “Hindi” and “Hindu”.	0.29
Keyword Query	BJP	All time	1233	We see results with comments containing the term “BJP”	0.27
Keyword Query	BJP	All time with Intitle search	1877	Intuitively, results with BJP in both titles as well as comments are shown. This justifies the increase in the number of results returned.	0.25
Keyword Query + Restriction of search	Congress	Time range between 15/11/2022 and 01/12/2022	5	Post with the search term “Congress”	0.02

space[1]					
Keyword Query + Boolean Query + Restriction of search space[1]	BJP[1]	Time range between 15/11/2022 and 01/12/2022 with location search - Delhi and Gujarat	4	Posts are restricted to those mentioning the given locations. Results are dominated by the “BJP Gujarat 2022 Manifesto”.	0.04
Keyword Query + Boolean Query + Restriction of search space	Kejriwal	Time range between 01/11/2022 and 01/12/2022 and intitle search	5	Posts contained the term “Kejriwal” in either comments or title.	0.03
Phrase query	Islam Hindu	All time	2476	Post containing either Islam or hindu words in the comments	0.33
Phrase query	Islam no Hindu	All time	2476	Post containing either Islam or hindu words in the comments. Note that since “no” is a stopword, it has no effect on the search results.	0.33
Phrase query + Boolean Query	Islam no Hindu	All time with location “Delhi”	89	Posts contained islam or hindu words and has the location term “Delhi”	0.04
Keyword Query	TN	All time	638	Post containing the term “TN”. Note that since “TN” is an abbreviation for “Tamil Nadu”, posts containing the later are also returned	0.10
Proximity Query	tharopw	All time with intitle	147	Posts relating to Shashi Tharoor,	0.2

[1] Search space is restricted by using a datetime index. This is different from a boolean query because no filtering is being done by date; instead, the set of documents being searched is limited.

				which is intended since the query is clearly a keyboard misspelling of “tharoor”	

The above table summarizes different kinds of queries with different search terms. We use “*” as described by Solr to return all records. Analysis from the above table are as follows:

- The query takes about 0.6 seconds to return and display all results.
- Using specific search terms or time periods reduced the query time by almost a third of the original time taken.
- The interface allows effective use of regex matching to retrieve data. Using “Hind*” returns all posts containing words starting with “Hind”.
- This behavior is expected since the database has been indexed on multiple fields allowing for efficient retrieval.
- Details of indexing are discussed in the “Searching” section of the report.
- Text normalization procedures such as the use of synonyms have also been implemented. Queries with the search term “TN” illustrates this technique.
- The search results are processed independent of stopwords in the query. An example using the query “Islam no Hindu” and “Islam Hindu” illustrates this feature.
- Queries with few records barely take any time to display results.

Searching Enhancements (Question 3)

As mentioned in the section on Indexing and Querying, we have configured the relevant fields in Solr in a special manner to facilitate sophisticated queries, and to improve search results. We now discuss the enhancements in querying used in Solr, followed by instances which illustrate the benefits of such enhancements.

Enhancements in Ranking

The extended DisMax query parser of Solr is used to perform querying. This query parser is used to query for simple phrases and allow for fine-tuned configuration of how results are scored. It makes use of the underlying Lucene text search to obtain documents, and then ranks documents based on a custom configured importance for different fields. The following snippet of Python code largely encapsulates how search is performed in our system:

```
qf = "comment^{}".format(term_imp)
pf = "comment^{}".format(full_phrase_imp)
pf2 = "comment^{}".format(bigram_imp)
pf3 = "comment^{}".format(trigram_imp)
```

```
if (intitle):
   qf = qf + " submission_title^{}".format(2 * term_imp)
   pf = pf + " submission_title^{}".format(2 * full_phrase_imp)
   pf2 = pf2 + " submission_title^{}".format(2 * bigram_imp)
   pf3 = pf3 + " submission_title^{}".format(2 * trigram_imp)

query_params = {
            "q": phrase_query,
            "defType": "edismax",
            "qs": qs,
            "ps": ps,
            "bf": "log(reddit_score)^10",
            "qf": qf,
            "fl": "comment,submission_title,score,url,reddit_score,timestamp",
            "pf": pf,
            "pf2": pf2,
            "pf3": pf3,
            "fq": "timestamp:[{} TO {}]".format(start_month, end_month),
            "rows": 7000
        }
```

In the above, `phrase_query` is the input phrase query from the user. We now discuss the various parameters of the constructed query to be issued to Solr:

1. `q`: This represents the actual query that is sent to Solr.
2. `defType`: This signifies the query parser to be used by Solr. Here, we configure Solr to use the extended Dismax (edismax) parser described prior. By default for this query parser, Solr will tokenize using the configured tokenizer and attempt to match documents for each token, i.e., they are treated as phrase queries by default. However, the term-level importance can be fine tuned.
3. `fl`: The “field list” parameter specifies the list of fields whose values are to be returned as a result of the query. The `submission_title`, `comment`, `score`, `url`, `reddit_score`, `timestamp`. Note here that the `score` is a special alias which refers to the computed scores for the documents based on Lucene’s scoring function, combined with the boosting applied as described in the next few parameters. Note that Lucene’s score function is a slight extension of TF-IDF scoring which takes into account the number of terms found in a target document, as well as the boost factors.
4. `qf`: The “query fields” parameter signifies the fields that the query must be issued on, along with any boosting factor to be applied to them. The variable `term_imp` signifies the amount of weighting to be assigned to each query term. We can see in the code block above, that if the intitle parameter is true, then double the `term_imp` is given for

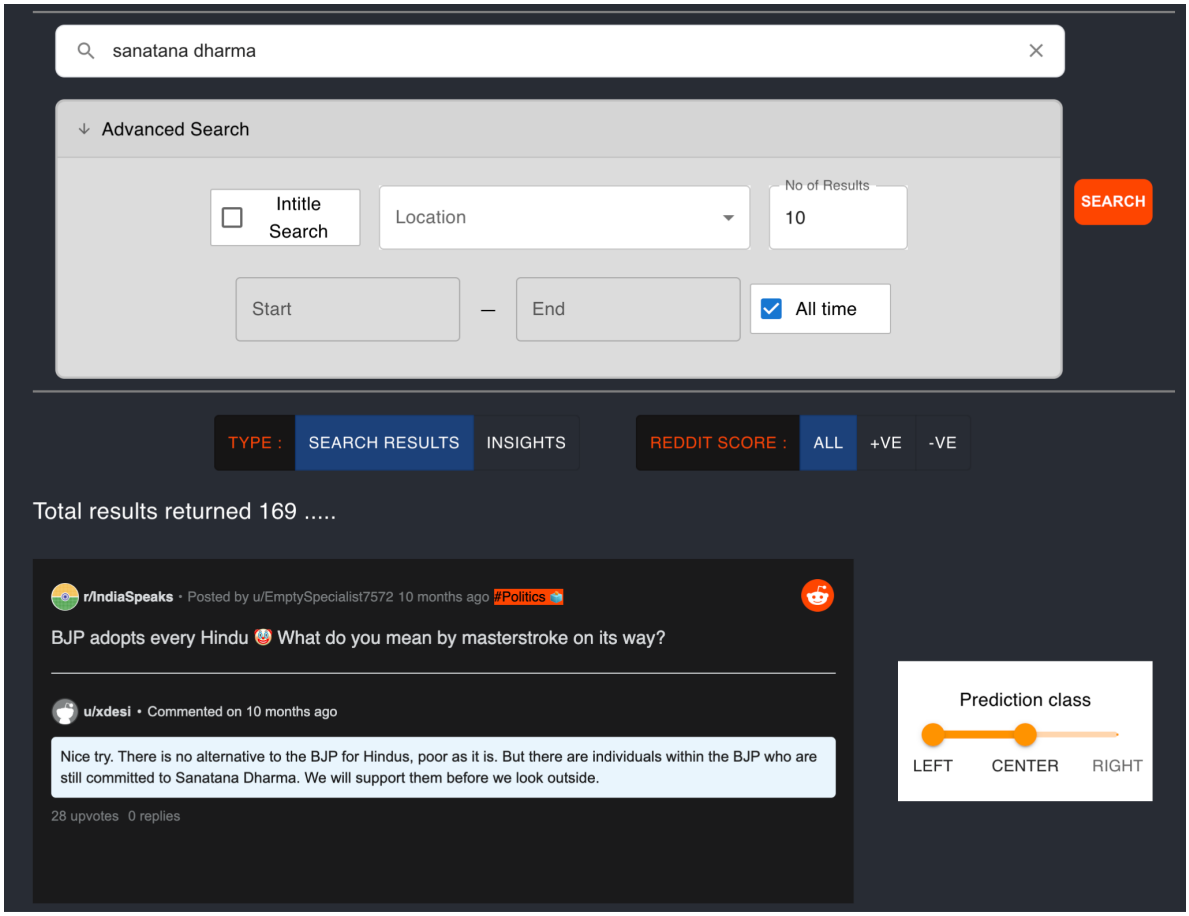
`submission_title`, since they are usually significantly smaller than the comment itself, and are also strongly indicative of the topic of a discussion. So even if the comment does not contain the query term in question, it can match due to the `submission_title`.

5. `pf`: The “phrase fields” parameter signifies a field with a corresponding boost factor, where a document will be boosted if all of the terms in the full phrase appear in close proximity. The variable `full_phrase_imp` signifies the amount of weighting to be assigned to each query term. Once again, double the `full_phrase_imp` is given for `submission_title`.
6. `pf2`: This field is similar to the `pf` but assigns the importance `bigram_imp` for pairs of shingles. Again, double the `bigram_imp` is given for `submission_title`.
7. `pf3`: Once again, this field is similar to the `pf` but assigns the importance `trigram_imp` for triplets of shingles. Again, double the `trigram_imp` is given for `submission_title`.
8. `bf`: The “boost functions” parameter used to specify function query that is used to provide an additive boost for the scores of the documents returned. Logically, this provides a very convenient method to implement a **static quality score**. This will be detailed later when summarizing all innovations in ranking.
9. `fq`: The “filter query” parameter is used to limit the search space of documents when searching. This significantly improves the efficiency of the system. As can be seen, this parameter is used to limit the date range of the documents to be retrieved when searching.
10. `ps`: The “phrase slop” parameter specifies how much individual tokens can “slop” away from each other while still counting as a match.
11. `qs`: The “query phrase slop” parameter specifies how much individual tokens in explicit phrases in the query specified by the user can “slop” away from each other while still counting as a match.
12. `rows`: The “rows” parameter is used to specify the number of documents to be returned as a result of this query.

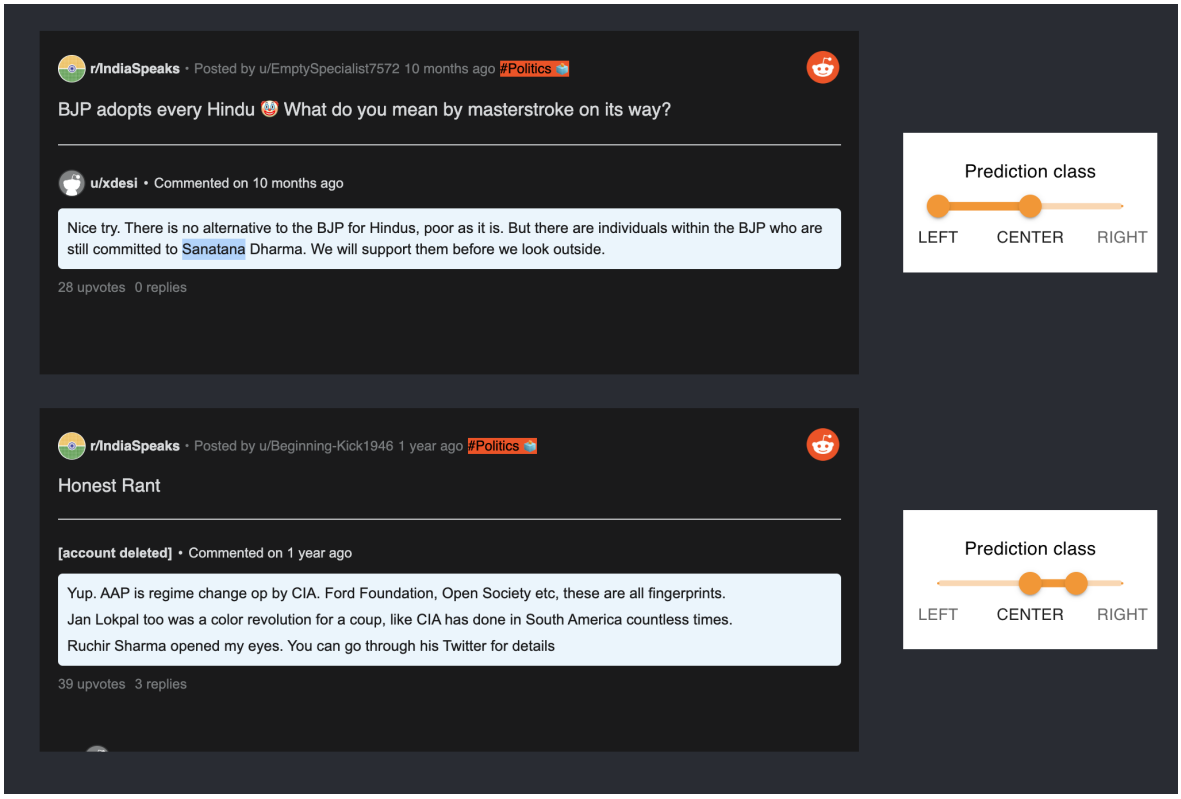
With the context of how queries are issued to Solr, we now summarize the ranking innovations in our system.

Ranking Innovations

1. **Boosting**: In our system, query terms, 2-word shingles, 3-word shingles and full phrases are assigned different levels of importance, which makes the system much more robust. For instance, if the query “BJP in Bengal” is issued, with a standard TF-IDF ranking, it is likely a comment which has several repetition of the term “BJP” would appear near the top, whereas a comment with just one mention of a phrase like “BJP may win the election in Bengal this time” would appear much lower. However, if we set much higher values for `bigram_imp` and `full_phrase_imp` than `term_imp`, and set a `ps` of about 10, then it is much more likely that the latter document would be ranked higher than the former.

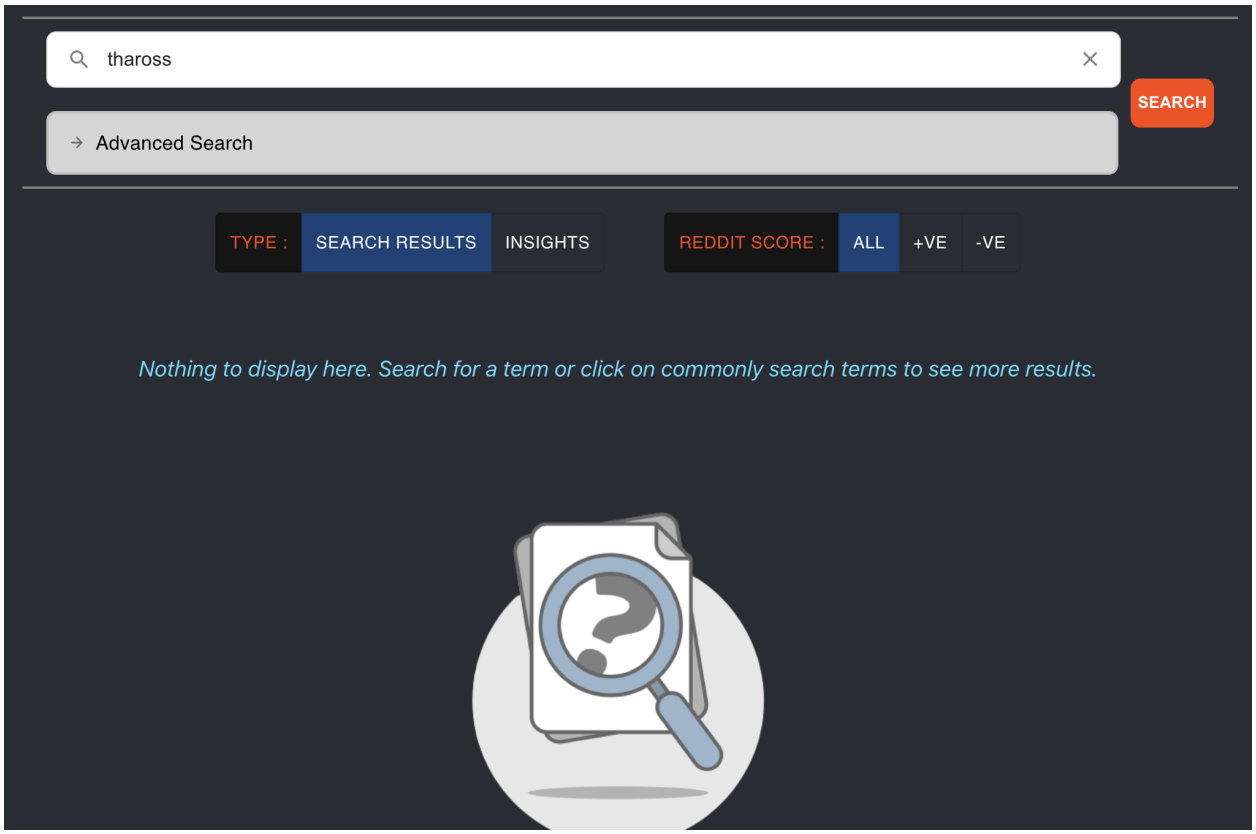


The screenshot above illustrates the results of the phrase query “Sanatana Dharma” on our final system with **extra weighting** for full phrases. As can be seen, the top result is one where the complete phrase matches.

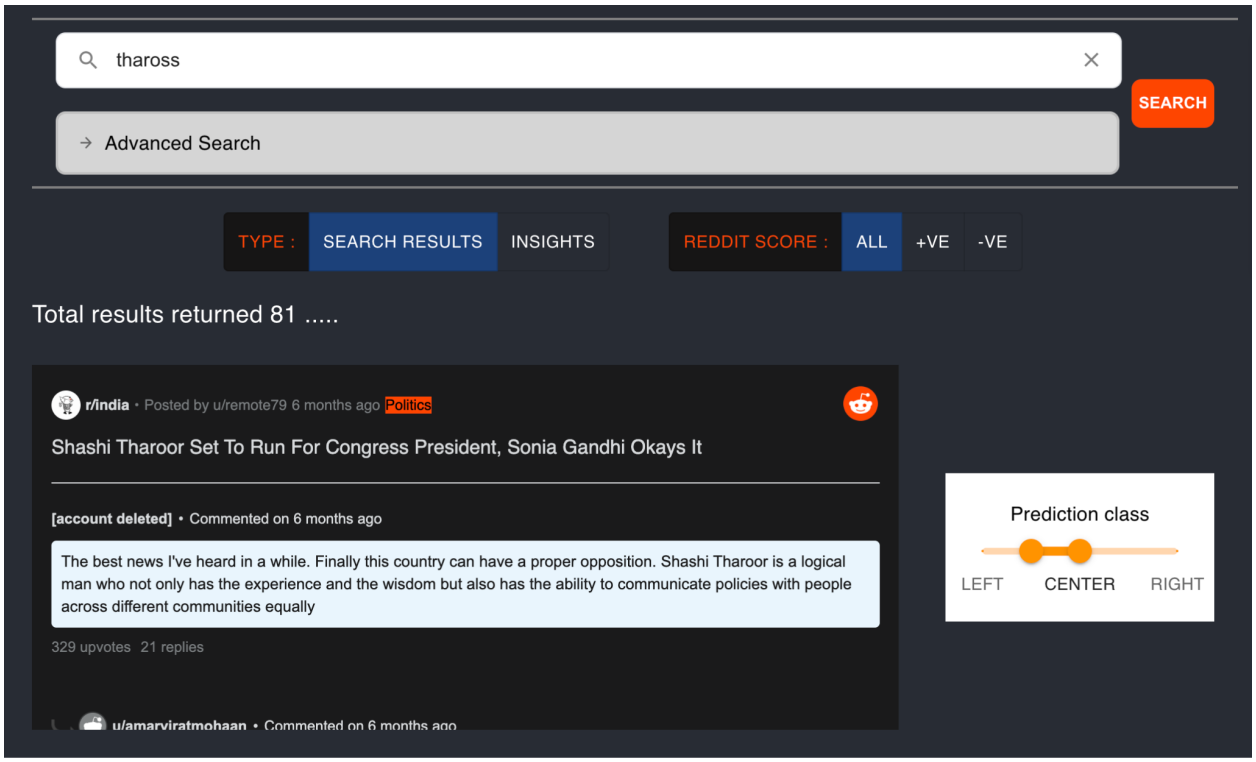


When the same query is issued to the system **without weighting**, the same result appears in the 9th position. This occurs due to the fact that the individual terms are given roughly equal importance as the full phrase, and documents with higher number of hits for individual terms are ranked higher.

2. **Fuzzy search**: In the backend Flask API, the suffix `~3` or `~2` is appended to each term (depending on whether the length of the term is greater than 5 or not respectively) in the input query before being sent to Solr for querying. This suffix indicates that Solr must perform a fuzzy search that matches documents where terms differentiate from any query term by a maximum **Levenshtein distance of 3 or 2**. Hence, in ranking documents, terms a small distance away from the query term are also considered hits, and are ranked high. This is especially helpful in the context of query misspellings. E.g.: `thaross` instead of `tharoor`.

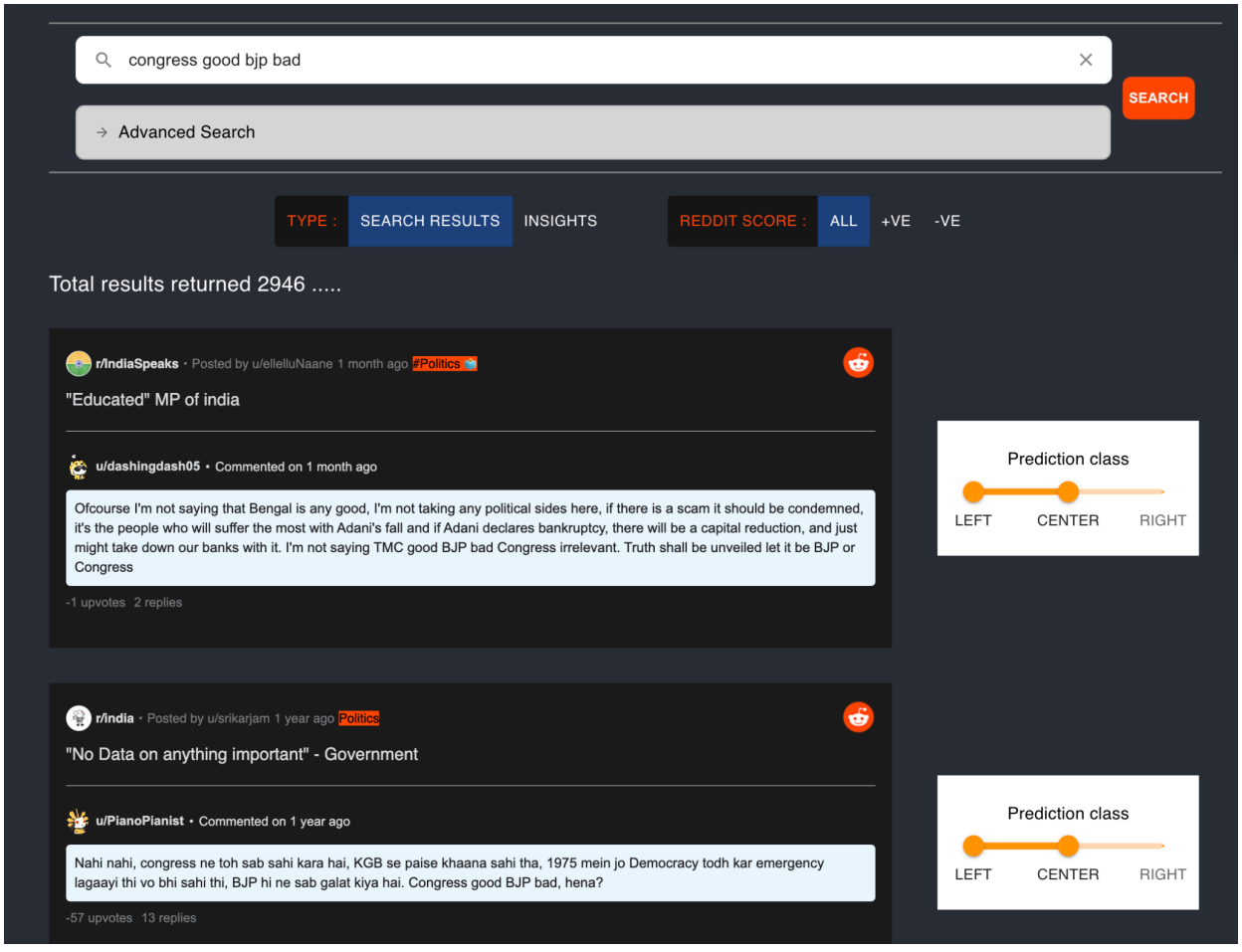


When the query “thaross” (misspelling of “Tharoor”) is issued to our system, when **no fuzzy search** is implemented, no results are obtained.

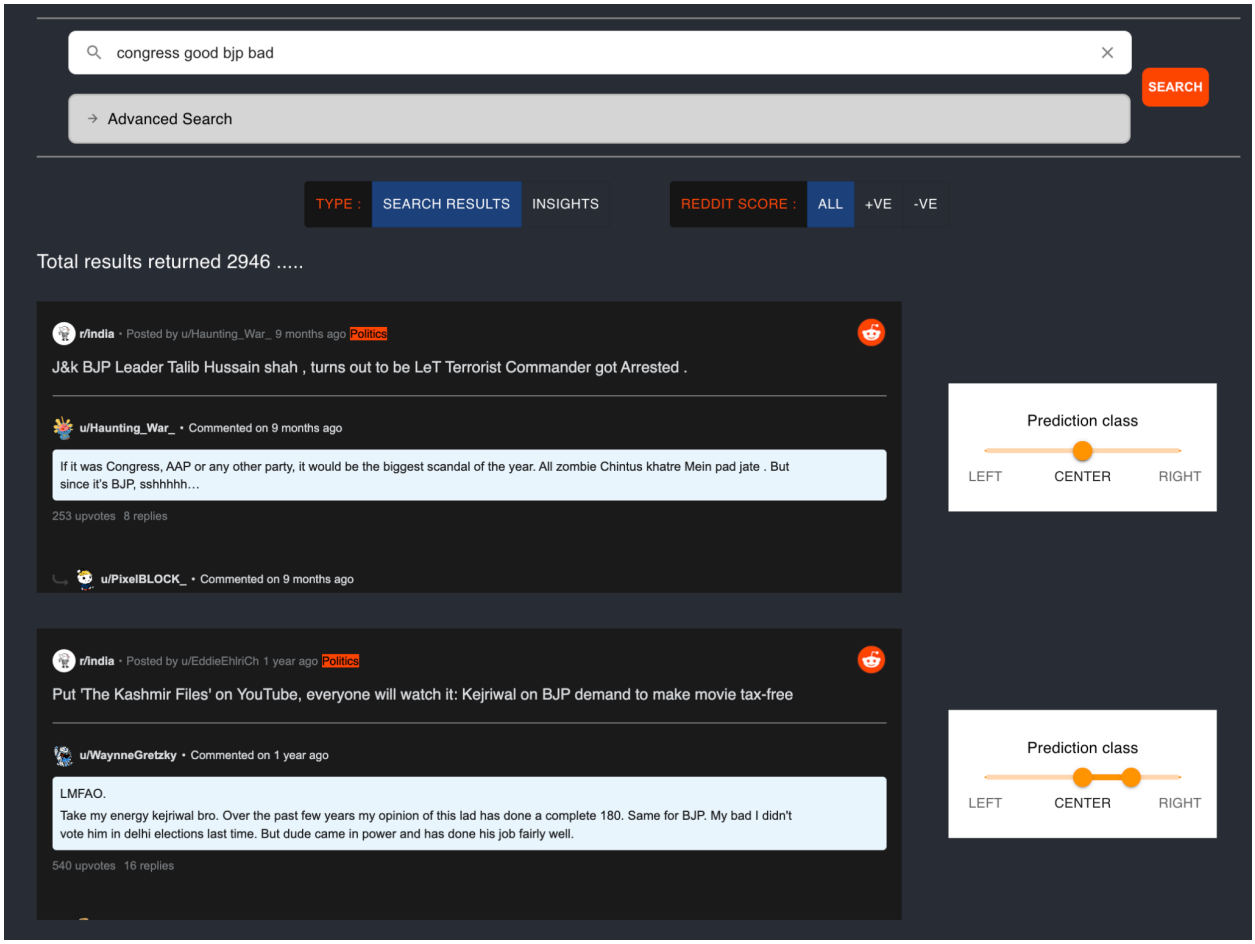


When the query “thaross” is issued to our final system, where **fuzzy search** is implemented, several results containing the term “Tharoor” are displayed.

3. **Static Quality Score**: We choose to have the metric `log(reddit_score)^10` as our static quality score. This is because the Reddit score is a good indication of the authority (quality) of a comment since a higher score signifies that more people agree with the opinion. Also, we choose a log scaling for it to ensure that search results are not simply dominated by irrelevant comments with a high `reddit_score`. Also, we boost the log score by a scale of 10, to compensate for the scales applied to the other terms. We factor this into the scores returned by Lucene by specifying the metric `log(reddit_score)^10` in the `bf` parameter, which provides an additive boost.

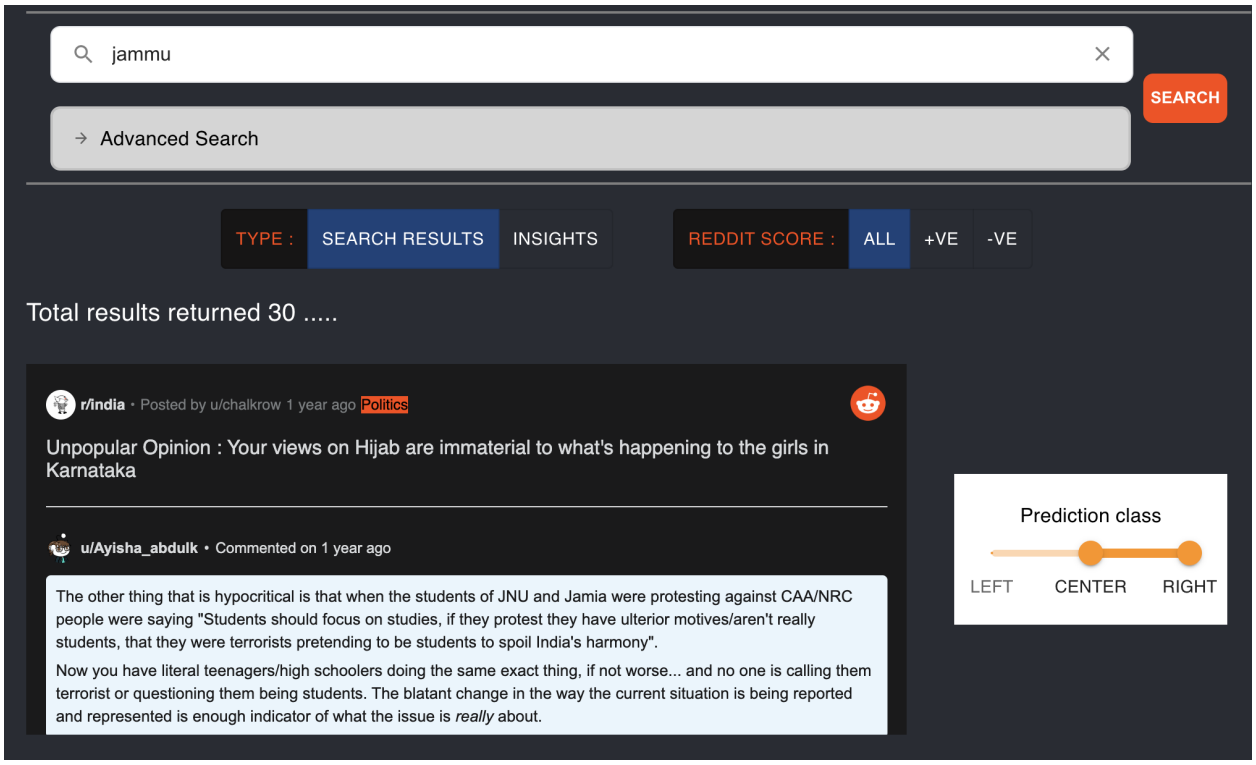


When the phrase query “congress good bjp bad” is issued to the system **without static quality scores**, the top 2 results are comments with negative Reddit scores as shown above, clearly indicating that the opinions are unpopular.

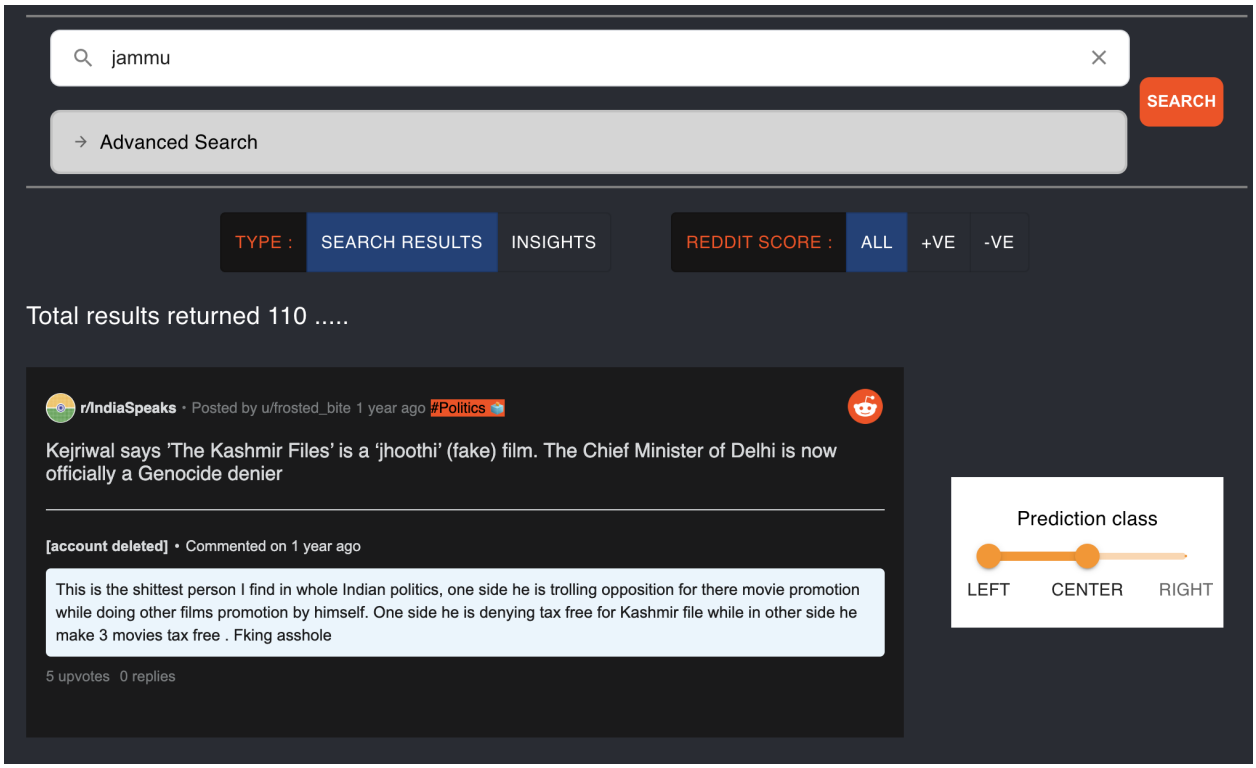


When the **static quality scores are taken into account**, the top 2 results are changed to more popular opinions with a high number of upvotes.

4. **Synonym Filtering**: The synonyms list provided in the appendix is specified for the `filtered_text` for query analysis ensures that equivalent terms and abbreviations are interpreted when ranking documents. For instance, a query issued as “AP election” would previously likely match only with documents related to the AAP election campaign. However, a query like “AP election” is likely in reference to elections in the state of Andhra Pradesh, which is encoded in the synonyms list. Now, issuing the same query after factoring in synonyms will rank documents containing “Andhra Pradesh” and “elections” much higher than “AAP elections”.



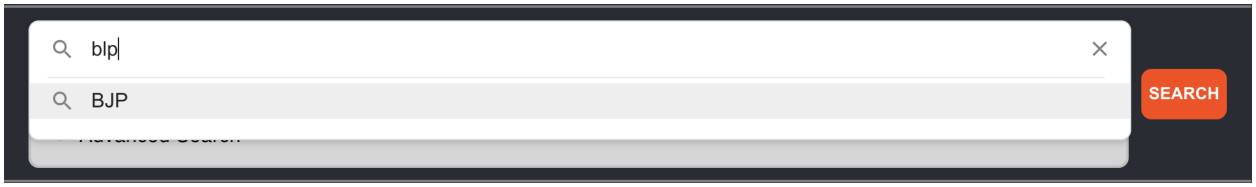
When the term query “jammu” is issued to the system when **synonyms are not taken into account**, only 30 results are obtained. Further, we see that the top result matches not due to the occurrence of the term “jammu”, but rather due to the occurrence of “Jamia” which has a Levenshtein distance of 2 from “jammu”, a **false positive**.



When the same term query is issued to our final system which **takes into account synonyms**, 110 results are obtained, and the top results are in fact related to the Indian state of “Jammu and Kashmir”, which is the intended result of the query.

Autocomplete

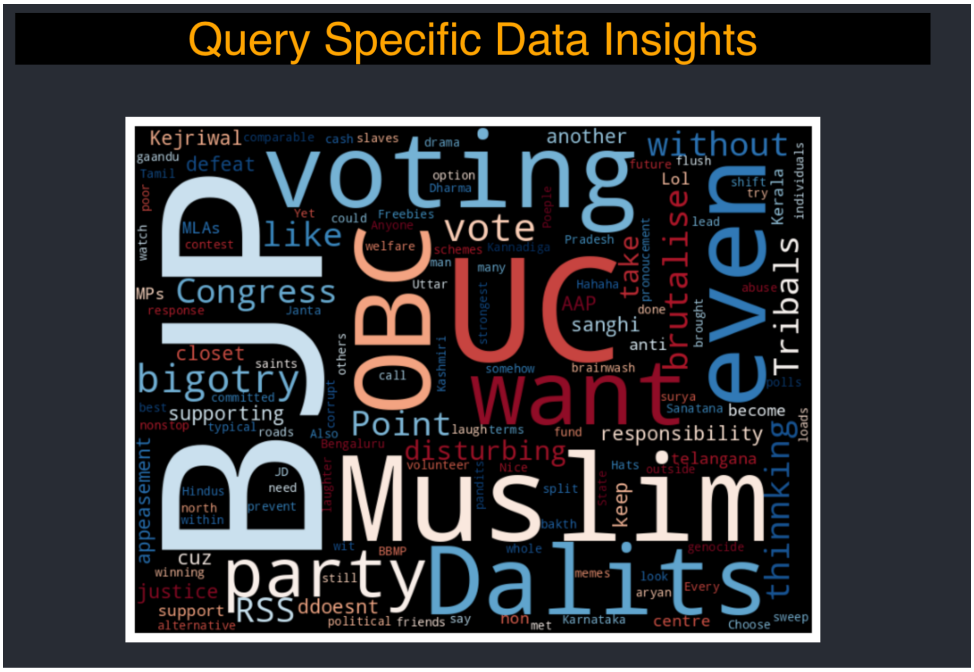
For ease of searching the user interface has a search bar with autocomplete feature. The autocomplete function is based on fuzzy search to recommend possible search terms while typing. Since our application deals with Indian political comments, we use a static set of keywords to recommend possible search terms for the user. This is done to ensure that relevant correction terms are used as recommendations rather than a general set of words from the English language. Evident from the image below, the recommended search terms are case insensitive.



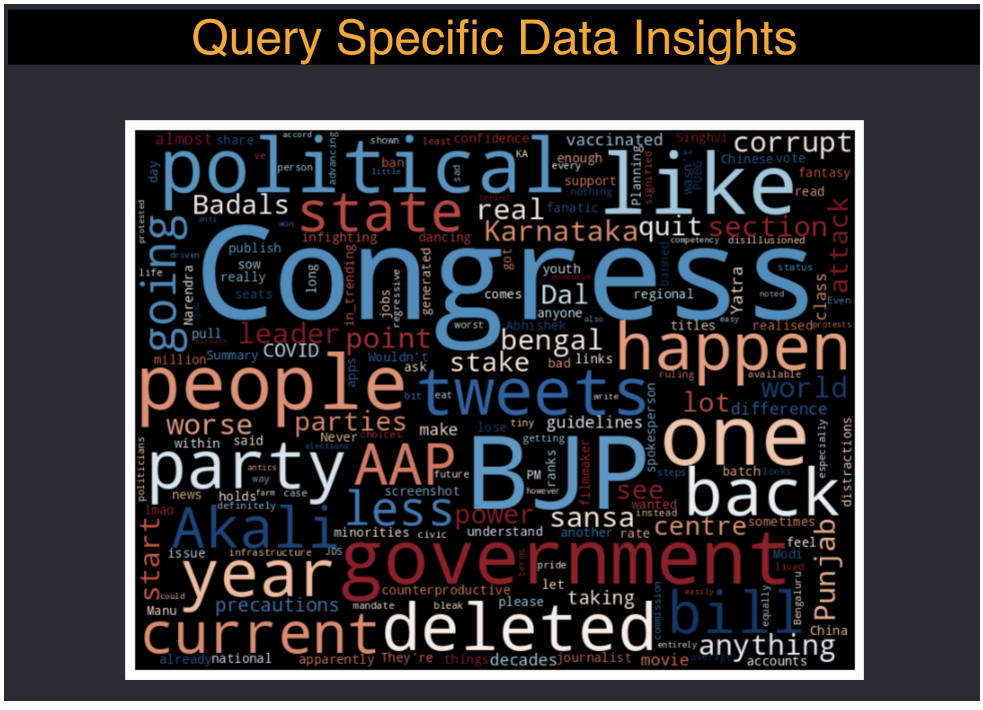
Visualization-Enhanced search

Valuable insights can be drawn from the most occurring words or terms in each category. For the purpose of text based visualization, we make use of wordclouds to highlight the most commonly occurring terms. This form of visualization is effective since it draws the user’s attention to such terms with the use of increased size. We therefore, display wordclouds for each query term inputted by the user. These wordclouds are filtered using advanced query features to give insights for each use query. This type of searching enables the interfaces to dynamically update the results based on each user query. The image below shows a sample wordcloud generated from the search query.

The wodcloud below shows the word cloud generated from all query results with query term BJP. AS expected, the term “BJP” is one of the highest occurring terms, followed by few others which may be closely related to the context of the search query.

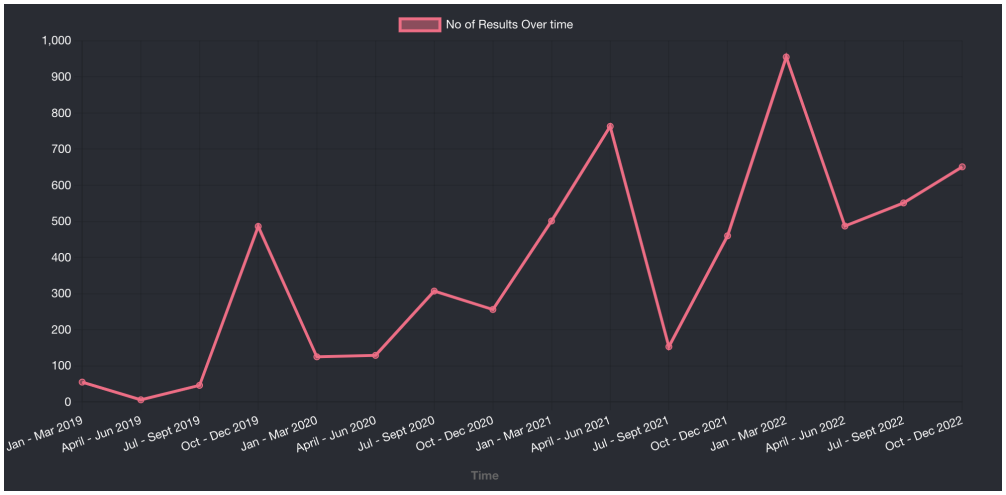


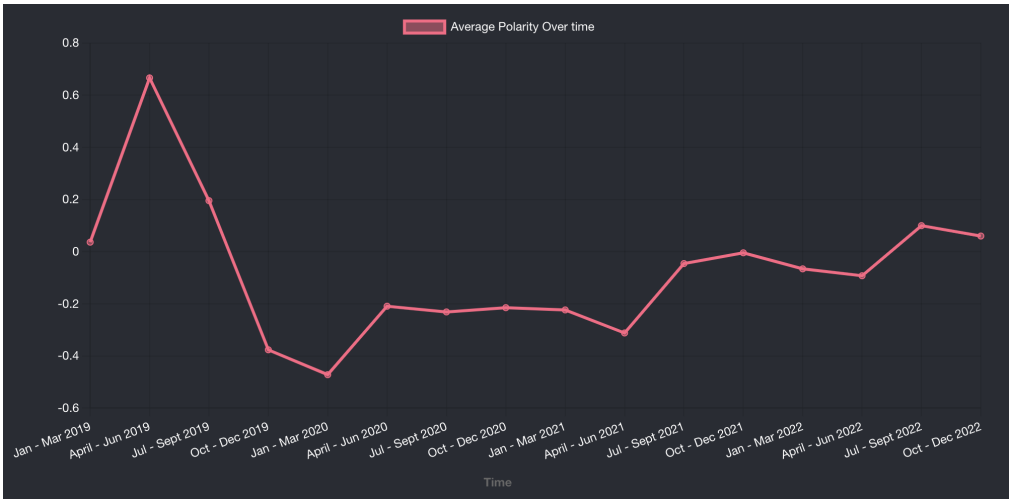
On the other hand, the word cloud generated using the search term “Congress” shows that congress occurs the most number of times.

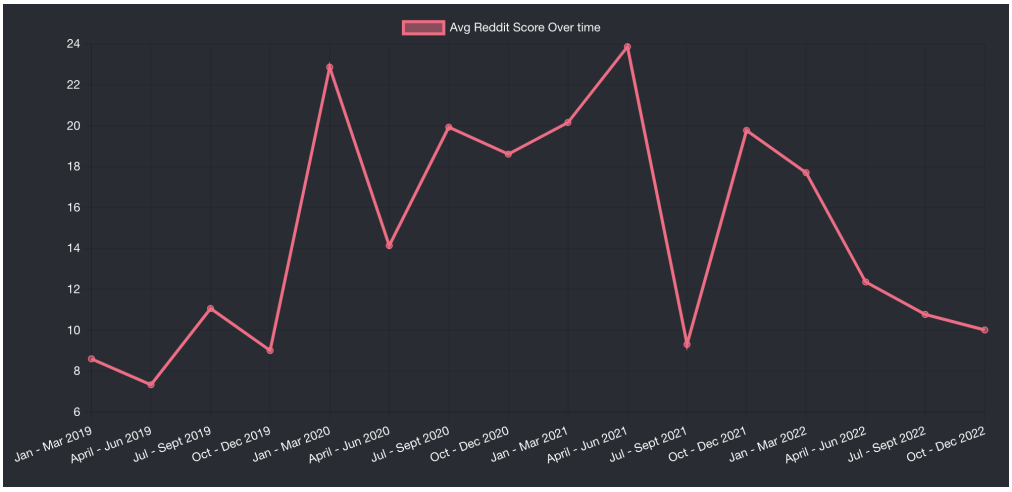


Apart from the above visuals, the interface displays **sequential data** showing change in statistics for different metrics over time. For the purpose of analysis three main time plots were

generated. The plots show the change in polarity scores, number of posts and average reddit score across time.







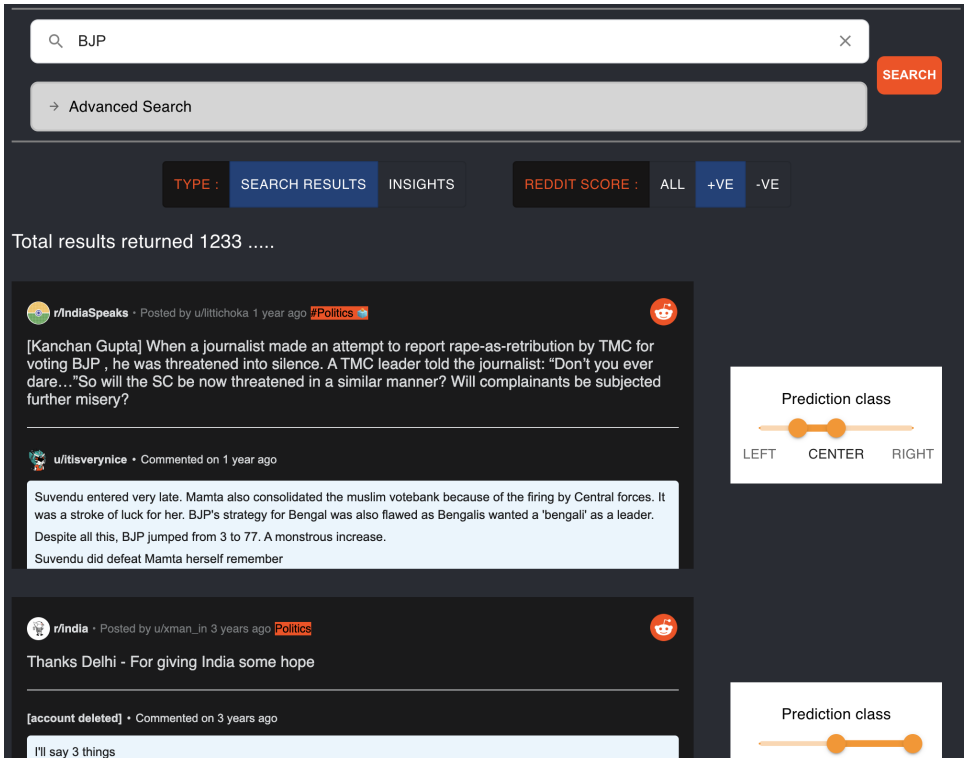
These results provide us with some useful insights. If we draw back from historical events and compare the polarity scores during those particular events, we can gain the political trend towards that topic, which can also be reinforced by natively searching for that particular event.

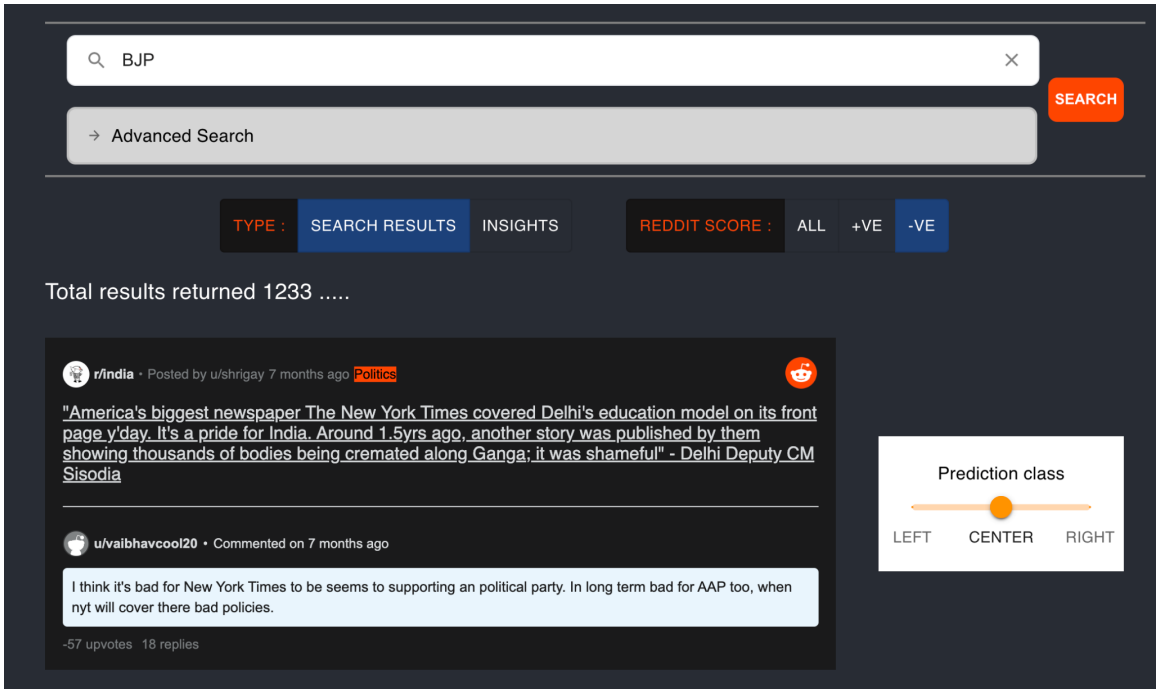
For example, a highly discussed topic was the 2019 Pulwama Attack which was a border skirmish between India and Pakistan, which happened around Feb 2019. During the period, nationalist and conservative sentiments were high, and the general political leaning was right wing during that period.

Likewise, the Indian government introduced the “Citizenship Ammendment Act” at the end of December 2019. This was a legislation that was met with severe backlash from the general public, and this can be reflected in the polarity chart, where the political sentiment for the period was leaning towards the left.

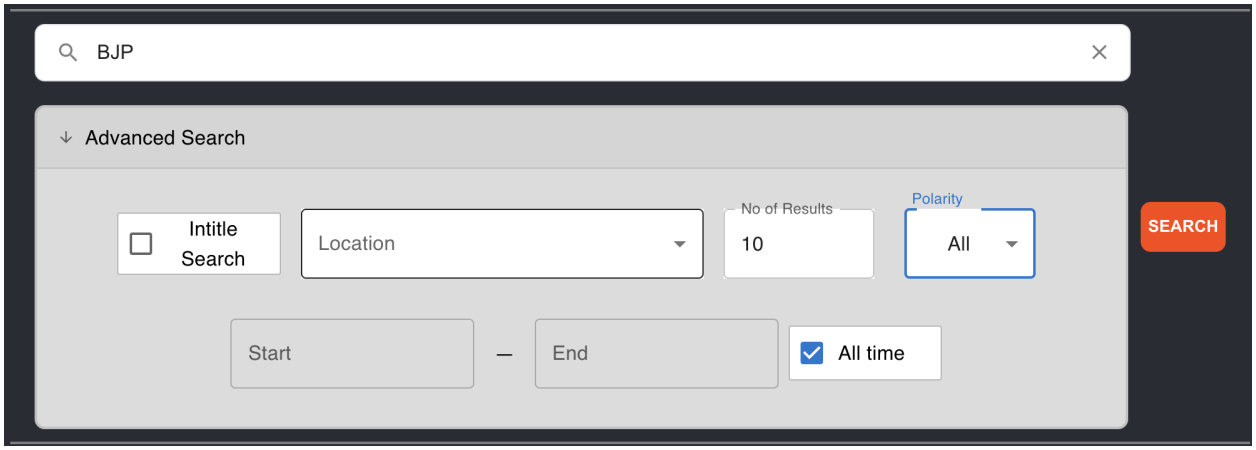
Interactive search

As described in the previous sections, the interface allows the user to filter posts that have positive or negative reddit score. Such a feature helps the user to view only relevant posts based on their search query. The images below show the difference in the search results based on positive or negative reddit scores. We can see that selecting only a negative reddit score with the search query “BJP” returns only one result, while the positive score returns most of the results. The filter helps the user to determine what users perceive to be unacceptable with respect to certain subreddits.



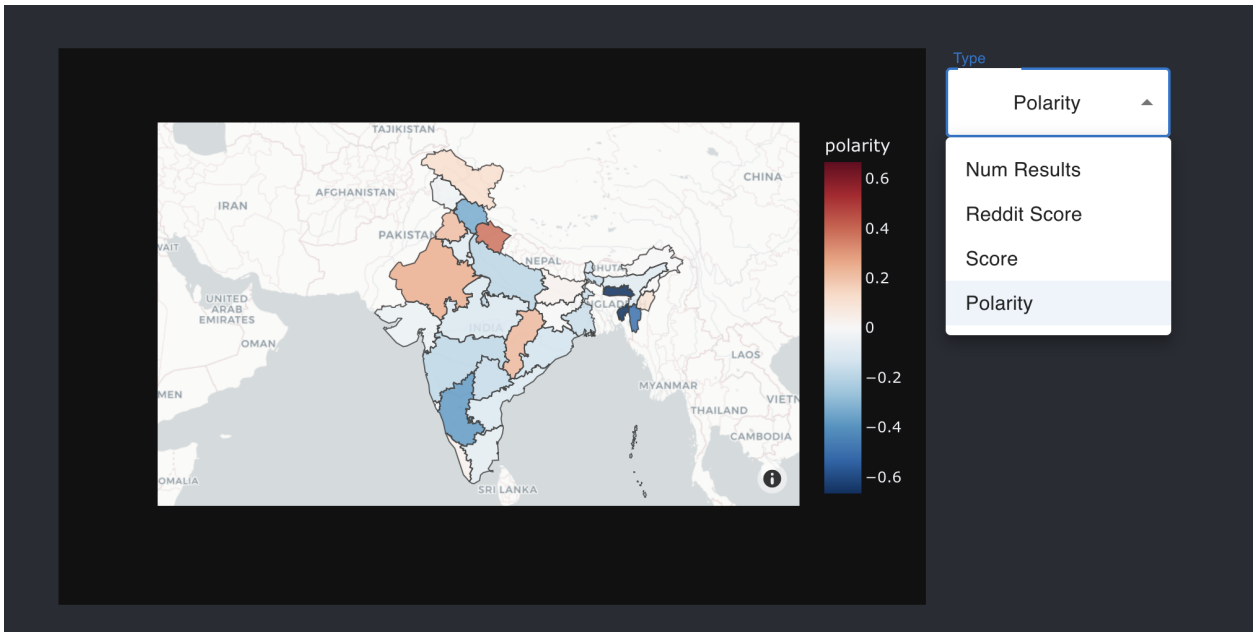


Similarly with the search query “BJP” in the time frame of 15th November 2022 to 1st December 2022, none of the posts have a negative reddit score.



Geospatial search

Since we enable the user to search using specific locations, the interface displays the count of posts returned specifying each of these regions. This helps to analyze the intensity of comments referring to different parts of India especially around the time of elections.



The image above shows the geospatial analysis based on different parameter filters. The visual above shows the intensity of the average polarity score across different locations. In the state of Karnataka (South India), there was a controversial Hijab ban, which lead to several left wing statements. Such analysis can help the user to craft custom queries to analyze the posts leading to such a score.

Similarly other filters such as Reddit score, Number of results mentioning different locations, etc., can be used to filter and analyze the results for different locations.

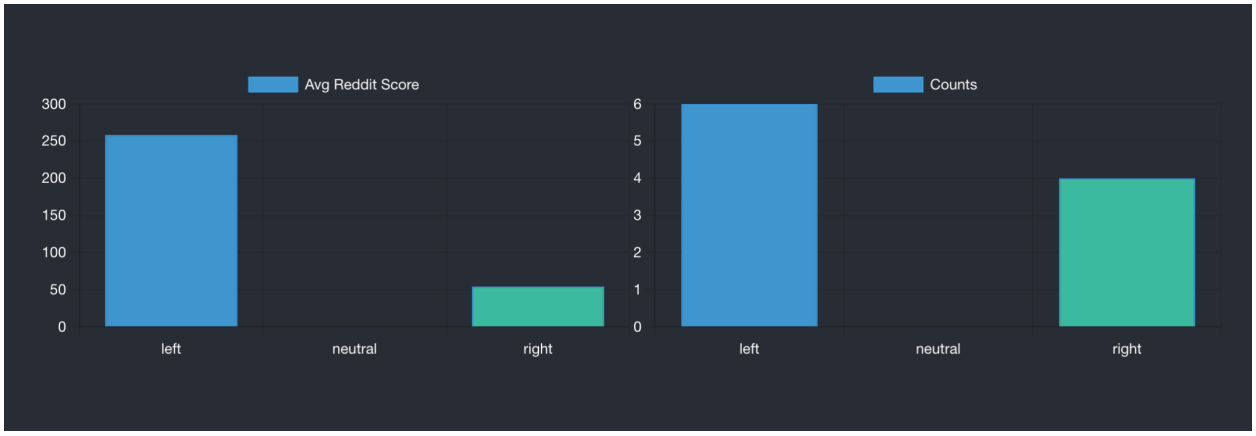
Multifaceted Search (Question 4)

The interface shows three broad types of visuals,

- bar graphs displaying counts for different class labels
- Text visualization, wordclouds to display counts of different terms or words
- Geospatial visualization based on location searches by the user.

The above visualization techniques are encoded with different class labels to provide in depth analysis for each category. This helps the user to analyze the different metrics based on the class labels obtained from the classification model.

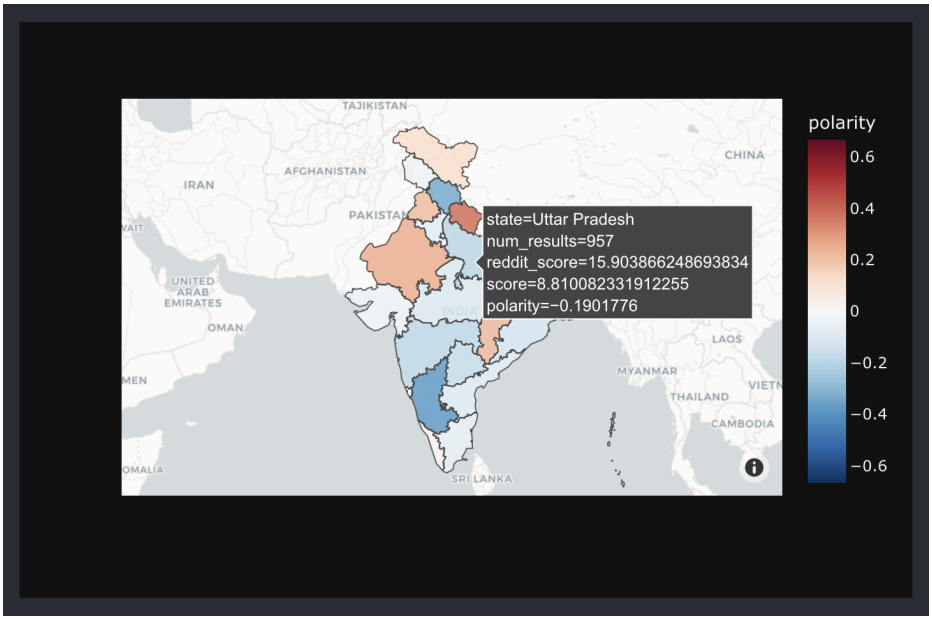
We display bar graphs showing the average reddit score and number of posts associated with the class of labels. We can see that the search term “Congress” doesn’t have neutral posts or comments.



On the other hand with search query “BJP” has more number of neutral comments in comparison to the other two classes. This may be indicative of underlying trends or terms which can be analyzed from generating word clouds.

In the image below, we show multifaceted search for wordclouds and geospatial visualizations. In the case of wordclouds, we generate a word cloud analyzing words from all posts returned from the query. Furthermore, we generate separate wordclouds for each class to display the most common terms in the category. This visualization allows the user to draw insights for the most common talking points for the different political ideologies. The interface allows the user to visualize the wordclouds based on the specific polarity as shown in the image below.

On the other hand, the geospatial visualization has markers encoded with different statistics for each location. The image shows the different statistics on hovering over the map.


TAJIKISTAN
AFGHANISTAN
IRAN
PAKISTAN
CHINA
UNITED ARAB EMIRATES
OMAN
LAOS
MYANMAR
THAILAND
CAMBODIA
SRI LANKA
state=Uttar Pradesh
num_results=957
reddit_score=15.903866248693834
score=8.810082331912255
polarity=−0.1901776
polarity
0.6
0.4
0.2
0
−0.2
−0.4
−0.6

Classification (Question 4)

Labeling

We labeled our comments on a 5-point Likert scale of -2 to 2 which indicates:

-2	-1	0	1	2
Left	Center-Left	Neutral	Centre-Right	Right

Although our objective is to classify a post as left/right-winged, we also chose to capture the intensity of political polarity during labeling in order to better understand sentence semantics and get more reliable features.

Since our classification task is a niche domain, we were unable to find relevant datasets online that could be used for training. Thus, we hand labeled 3000 posts (out of 6687 total), each with 2 annotators. The distribution of classes after labeling is shown below:

```
 0    1577
-1     554
 1     437
-2     219
 2     213
Name: label1, dtype: int64
```

```
 0    1344
-1     704
 1     496
-2     233
 2     223
Name: label2, dtype: int64
```

To assess reliability of annotations, we calculated the Cohen-Kappa score between 2 annotators. We have defined *agreement* between labels as a distance less than 1. This is justifiable as we attempted to capture the intensity of sentiment in comments, and equality of intensity would be very unreliable. The calculations and results are as follows:

```python
def inter_annotator_agreement(judge1, judeg2):
    def prob_chance(label):
        j1_chance = ([judge_1[i]==label for i in range(3000)].count(True)) / 3000.0
        j2_chance = ([judge_2[i]==label for i in range(3000)].count(True)) / 3000.0
        p_e = j1_chance * j2_chance
        return p_e

    p_a = ( [abs( judge_1[i]-judge_2[i] ) <= 1 for i in range(3000)].count(True) ) / 3000.0
    p_e = sum(prob_chance(label) for label in [-2, -1, 0, 1, 2])
    kappa = (p_a - p_e) / (1 - p_e)
    print(f"agreement: {p_a:.3f}; by-chance: {p_e:.3f}; kappa score: {kappa:.3f}")

judge_1 = text_df['label1'].to_numpy()
judge_2 = text_df['label2'].to_numpy()

inter_annotator_agreement(judge_1, judge_2)
```

✓ 0.0s

```
agreement: 0.971; by-chance: 0.314; kappa score: 0.958
```

A score of 95.82% implies a strong agreement between the annotators.

Train & Test Datasets

The manual labels for each row were averaged, and the score divided by 2 to get a target variable in the range [-1, 1]. We then performed a random train-test split to get 2000 posts for the training dataset, and 1000 posts for the evaluation dataset (which is 15% of the total dataset). The distribution of classes in each of the datasets is shown below:

```
Train data target distribution:
-1.00    104
-0.75     83
-0.50    271
-0.25    182
 0.00    841
 0.25    140
 0.50    203
 0.75     79
 1.00     96
Name: target, dtype: int64
```

```
Evaluation data target distribution:
-1.00     46
-0.75     47
-0.50    148
-0.25     94
 0.00    414
 0.25     61
 0.50    100
 0.75     41
 1.00     49
Name: target, dtype: int64
```

These datasets were used for the final training and evaluation of the main models selected for classification.

Cleaning and Preprocessing

After crawling comments from the two Indian political subreddits on Reddit, we observed that the data has a lot of noise in the form of acronyms, subreddit-specific slangs, links and tags, etc. Hence, data cleaning was an important and necessary step in order to standardize and reduce noise in the data.

The following steps were taken to clean the data:

1. **Removal of Quotations**

   In Reddit, a common text formatting used is quotations. A quotation often includes text or comments that a user singles out as important to address (such as other users' comments, internet quotes, etc.), but not content authored by the user themself. We, hence, decided to remove quotations as this added ambiguity to the text's sentiment.

   Quotations are formatted as sentence starting with ">" and ending with "\n". They can be removed using simple regex substitution.

2. **Removal of URLs**

   URLs do not contribute towards the political sentiment of a comment, and hence, we choose to remove them from the URLs. However, note that we retain the "display text" of the URL links.

3. **Removal of special characters**

   Based on our domain-specific understanding, non-alphanumeric and special characters do not greatly influence the political sentiment of the text. It is possible that emojis and others special characters enrich the sentiment of a comment, however these are difficult

to process and embed by traditional means. Hence, we chose to focus only on the meaningful words and semantics of the comment and retain only alphanumeric characters and certain special characters relating to punctuation.

4. **Lowercasing**

   Lowercasing all words helps to reduce the complexity of the dataset and standardizes the text for further processing.

Code Snippet for Cleaning Code based on above steps:

```
def clean_quotations(text):
    temp = re.sub(r'>.*?\n', '', text)
    return temp

def clean_url(text):
    temp =
re.sub(r'(http(s)?:\/\/.)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-z
A-Z0-9@:%_\+\.~#?&\/=]*)', '', text)
    return temp

def clean_special_chars(text):
    temp = re.sub(r'[^a-zA-Z0-9\s\.,?!\'\"]', ' ', text)
    return temp

def lowercase(text):
    temp = text.lower()
    return temp
```

The true power of most ANN-based models comes from contextual information in the data. Hence, many lossy preprocessing steps such as stopword removal, lemmatization, or stemming have not been performed on the data. We perform additional preprocessing steps for model-specific cases.

Ablation Study

We experimented with a variety of ML approaches ranging from simple hard-coded rule based classifiers to more sophisticated models using transformers (and transfer learning). The motivations and results are explained in the following sections.
The Fine-tuned Indian-sBERT model can be considered to be closest to a pre-existing state-of-the-art model, as we could not find any other models that performed as decently on the mixed English/Hindi data. However, we did not consider this model to be satisfactory on the task. Hence, we iteratively experimented with a variety of models, enhancing each model based on results and providing justifications and motivation for the same.

Rule Based Classification

Our first approach was to use an unsupervised rule based classifier along with word counts for classifying a comment. The idea is to classify sentiment of a text based on the term frequency of politically right/left keywords. The process was as follows:

1. Define a hard-coded knowledge base of right-winged and left-winged keywords. (Refer to Appendix for list of keywords).
2. Preprocess text by removing all numbers and special characters.
3. Generate term frequency dictionary of unigrams and bigrams of words in each comment.
4. Count the occurrences of right-winged and left-winged terms using the above-defined knowledge base.
5. Infer class label as the political spectrum which dominates in occurrences.

The evaluation report is as follows:

	precision	recall	f1-score	support
Left	0.32	0.18	0.23	975
Neutral	0.54	0.77	0.64	1256
Right	0.26	0.22	0.24	769
accuracy			0.44	3000

Clearly, the accuracy of the model is too low and is hence, very unreliable for the classification task.

<u>Enhancement: Incorporating some context using toxicity and submission title</u>

In reddit, each comment is posted under a parent post which has a “submission title”. Often these comments are a direct response to the parent post. For each comment, we also retrieved the toxicity score using SenticNet API. In this model, to infer a class label, we make following adjustments:

1. If a comment is detected as Left / Right, but is a highly toxic comment (>=0.5 toxicity), we invert the label.

2. If a comment is detected as Neutral, but is a highly toxic comment (>=0.5 toxicity), we assign it the label opposite that of its parent post.

The above steps are justified because a highly toxic comment is most likely being toxic towards the political spectrum which it most frequently mentions or the post which it responds to.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.32	0.29	0.30	975
Neutral	0.67	0.69	0.67	1256
Right	0.27	0.31	0.29	769
Accuracy			0.49	3000

Although the accuracy and f1-scores of this model is significantly higher, it is still not reliable enough to be used as a classifier. However, the idea of using a text's toxicity to classify political sentiment is one we explore further.

Enhancement: Incorporating sentiment score

Simply counting the occurrences of left-wing and right-wing terms is not enough, as it does not account for people of the opposing political spectrum criticizing the talking points. We attempt to incorporate sentiment analysis to account for this - if the sentiment of the sentence is negative, it can be considered to be the opposite label.

The actual implementation assigns a score based on a few factors, described below:

$$score = \frac{k \times sign(rw - lw) \times \sqrt{lw + rw}}{n_{words}}$$

where *lw* and *rw* represent number of left- and right-wing terms from the knowledge base respectively, n_{words} represents the number of words in the sentence, and *k* is a normalization factor based on experimentation.

This score is discretized into labels using a threshold. The following classification evaluation results were obtained:

	precision	recall	f1-score	support
Left	0.52	0.36	0.42	335
Neutral	0.58	0.77	0.66	414
Right	0.37	0.32	0.35	251
Accuracy			0.52	1000

These are the best unsupervised results we could obtain using a rule-based approach. However, the sentiment analysis models we tried were trained on English text, and could not effectively handle Hindi text. This could be one cause for the low accuracy.

Knowledge-Based Feature Extraction

We attempt to encode more information from the knowledge base through the creation of term-frequency embeddings. These embeddings are enhanced with sentiment scores from the NLTK and TextBlob libraries. These embeddings are then classified using an XGBoost classifier. The implementation is as follows:

1. For each sentence, create an array of the number of occurrences of each knowledge base term.
2. Append compound sentiment score from NLTK VADER, and subjectivity and polarity scores from TextBlob
3. Train and evaluate an XGBoost Classifier with following parameters:

   { learning_rate=0.1, n_estimators=100, max_depth=4 }.

The following evaluation report was obtained:

	precision	recall	f1-score	support
Left	0.57	0.41	0.48	335
Neutral	0.57	0.88	0.69	414
Right	0.63	0.29	0.39	251
Accuracy			0.57	1000

Clearly, this method significantly boosts performance over the previous rule-based method, as we do not need to manually try and optimize the score function.

Random Forest Classifier

A random forest classifier is a supervised model that ensembles multiple decision tree classifiers, where each tree is based on a random subset of features. Due to the randomness introduced by feature sampling and bagging, random forest classifiers are generally suitable for high-dimensional noisy data. The implementation is as follows:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

   { max_features=20000, analyzer=”char_wb”, ngram_range=(3,7) }.
2. Retrieve toxicity score from SenticNet API to use as a feature for training the model.
3. Training and evaluation of a Random Forest Classifier with following parameters:

{ n_estimators=100, max_depth=None }.

These parameters were selected by using Grid Search.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.52	0.47	0.45	335
Neutral	0.55	0.80	0.65	414
Right	0.61	0.25	0.36	251
Accuracy			0.55	1000

Support Vector Machine

A support vector machine is a supervised method that aims to learn a maximum-margin hyperplane to classify data in high-dimensional vector spaces. Non-linear kernels also makes it more versatile and efficient than other methods. The implementation is as follows:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

   { analyzer="char_wb", ngram_range=(3,7) }.
2. Retrieve toxicity score from SenticNet API to use as a feature for training the model.
3. Training and evaluation of a Support Vector Classifier (SVC) with following parameters:

   { kernel='linear', C=1, probability=True}.

   These parameters were selected by using Grid Search.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.45	0.48	0.46	335
Neutral	0.55	0.70	0.62	414
Right	0.62	0.33	0.44	251
Accuracy			0.53	1000

Fine-Tuning Indian-sBERT

Indian-sBERT is a pre-trained transformer model available under the HuggingFace transformers library. It is based on RoBERTa, an XLM (cross-lingual language model) model. It is pretrained on the task of feature extraction, and the transformers library pipeline allows it to be plugged directly into a classification pipeline using a linear NN prediction layer.

The implementation is as follows:

1. Instantiate the model for classification using AutoModelForSequenceClassification with num_labels = 1 (allows for regression as well as classification)
2. Freeze all but the last 3 layers of the model for fine-tuning
3. Instantiate an AutoTokenizer with the following parameters:

   { padding='max_length', max_length=256, truncation=True }

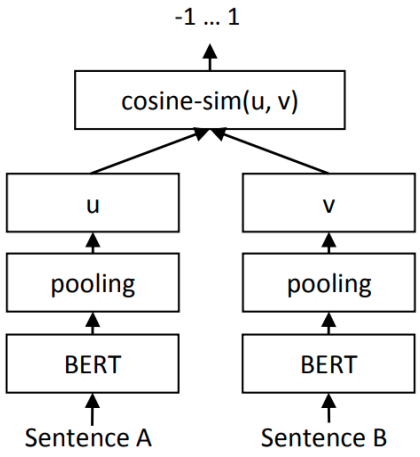
4. Use a train loop to fine-tune the model on train data with MSE regression loss on the target variable.

The regression scores can be discretized into labels with a threshold. The model obtained an RMSE of 0.42 and an accuracy of 0.45.

Unfortunately, it appears that the size of the hand-labeled training data is too small to sufficiently fine-tune the model.

BERT Siamese Embeddings

We attempt to address the issue faced in the fine-tuning model by creating a Siamese model instead. Siamese models accept pairwise inputs, and the combination of pairs increases the dataset size. Siamese models aim to fine-tune the model on creating the maximum distance between any two embeddings of opposing classes. The simplified architecture is seen below:



The “true” label is calculated as $sign(l_1 * l_2)\sqrt{abs(l_1 * l_2)}$. The sign is positive or negative based on whether the labels are the same, and the magnitude is the geometric mean of the individual magnitudes.

The model is trained using MSELoss for regression. The trained BERT model component of the siamese network is then used to generate embeddings for the dataset, and a separate classifier is trained using those embeddings as input.

A neural network classifier was trained on these embeddings to classify them. Unfortunately, the training did not put a sufficient distance between the embeddings, and the standard deviation of each dimension was very small. The best accuracy obtained was 49%.

Indian-sBERT Frozen Embeddings

Based on experimentation across a few cross-language models on Indian languages, we found Indian-sBERT to have a good distance between embeddings generated. Thus, we used a frozen pretrained Indian-sBERT model to extract features from sentences, and put them through various classifiers for prediction. XGBoost was found to be the best performing model for this task, trained with the parameters { learning_rate=0.1, n_estimators=100, max_depth=5 }.

The evaluation results are shown below:

	precision	recall	f1-score	support
Left	0.48	0.44	0.46	335
Neutral	0.57	0.71	0.63	414
Right	0.42	0.29	0.35	251
Accuracy			0.52	1000

Of the classifiers based on pretrained BERT models, these were the best results obtained.

Transformers

A simple transformer model was tested considering the attention based mechanism. This may have possibly helped to capture important phrases or parts of the sentence to classify them into the appropriate classes. However, due to insufficient data, the model generated poor results with a maximum accuracy of around 35%.

TF-IDF + XGBoost

XGBoost is a machine learning method that uses distributed regularized gradient-boosted decision trees for classification. The biggest advantages of XGBoost are its execution speed and model performance, which almost always makes it superior to other traditional models - particularly with limited data. The implementation is as follows:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

   { analyzer=”word”, ngram_range=(1, 2) }.
2. Retrieve toxicity score from SenticNet API to use as a feature for training the model.

3. Training and evaluation of a XGBoost Classifier with following parameters:

{ learning_rate=0.05, n_estimators=100, max_depth=4 }.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.48	0.41	0.44	335
Neutral	0.54	0.76	0.63	414
Right	0.60	0.30	0.40	251
Accuracy			0.53	1000

Enhancement: Sub-word TF-IDF Embeddings

TF-IDF can be performed using sub-word character n-grams, ending at word boundaries. This generates a larger embedding size, but also encodes more potential information. For our use-case in particular, the sub-word embeddings likely capture Hindi text much better, as the spellings of hindi words can be ambiguous when written in English. The implementation is:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

{ analyzer=”char_wb”, ngram_range=(2, 10) }.

2. Training and evaluation of a XGBoost Classifier with following parameters:

{ learning_rate=0.05, n_estimators=100, max_depth=4 }.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.58	0.40	0.47	335
Neutral	0.56	0.85	0.67	414
Right	0.56	0.31	0.40	251
Accuracy			0.56	1000

Enhancement: Restricting Vocabulary

Since we know that certain terms included in our knowledge base are important towards prediction, we can bias the TF-IDF model to consider related terms more. We do this by training a TF-IDF vectorizer, selecting terms from its vocabulary that include knowledge-based terms, and re-training a new TF-IDF vectorizer on this limited subset.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.59	0.41	0.49	335
Neutral	0.59	0.86	0.70	414
Right	0.53	0.33	0.41	251
Accuracy			0.58	1000

Enhancement: Dimensionality Reduction and Sentiment Scores

As before with the Feature Extraction classifier, we can use dimensionality reduction in combination with sentiment analysis scores to encode more information in a smaller embedding size. However, due to the massive size of TF-IDF embeddings and their storage in sparse matrices, PCA cannot be used directly. Instead, TruncatedSVD is used to achieve the same goal. The embedding has additional values for sentiment scores appended to it. This is passed to the XGBoost model for classification. The implementation is as follows:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

   { analyzer=”word”, ngram_range=(1, 2) }.

2. Fit a TruncatedSVD model on train embeddings and transform them, using n_components=8. Transform evaluation embeddings using the same model.
3. Append NLTK VADER’s sentiment compound score, TextBlob’s polarity and subjectivity scores to each embedding.
4. Train and evaluate an XGBoost Classifier with following parameters:

   { learning_rate=0.001, n_estimators=100, max_depth=4 }.

The following evaluation report is obtained:

	precision	recall	f1-score	support
Left	0.54	0.49	0.52	335
Neutral	0.61	0.80	0.69	414
Right	0.49	0.30	0.37	251
Accuracy			0.57	1000

Data Augmented TF-IDF

A recurring issue in previous models appears to be the bias towards the neutral label due to its prominence in the support. One method we implemented as an attempt to tackle this was the

augmentation of data to bias the TF-IDF model towards words from the knowledge base. The methodology for data augmentation is as follows:

1. Select a row from the training data having cleaned text and a label
2. Use regex to find all matches of the knowledge base terms in the text
3. Replace each occurrence of a right-wing term with a randomly selected left-wing term, and vice versa
4. Negate the label

This effectively doubles the size of the data. Although the sentences no longer make sense, they are still usable by context-independent models such as sub-word TF-IDF. The models should ideally identify the fact that only these specific terms changed and caused the label to become the opposite. This augmented dataset is passed into the sub-word TF-IDF classifier described above.

The following evaluation report is obtained:

	precision	recall	f1-score	support
Left	0.62	0.38	0.47	335
Neutral	0.56	0.89	0.69	414
Right	0.52	0.27	0.36	251
Accuracy			0.57	1000

2-Step Prediction Model

The models described above show very low accuracy owing to the low recall values of Left & Right prediction classes. This may be due to the imbalance in our training dataset causing a bias towards Neutral class. One possible solution is to randomly undersample the majority class in order to balance the dataset; however, this would cause loss of a big chunk of training data.

This motivated us to approach the problem as a 2-step classification task rather than a single 3-class task. We split the task into two subtasks: i) classify text as neutral or non-neutral and, ii) classify non-neutral tasks as left or right. This would also help to create balanced datasets for each subtask, and improve performance. The implementation is as follows:

1. Vectorize the data using Tf-idf vectorizer with following parameters:

   { analyzer=”char_wb”, ngram_range=(3,7) }.

2. Retrieve toxicity score from SenticNet API to use as a feature for training the model. We also incorporate Knowledge Based Feature Extraction and Vader polarity score as additional features for this model.
3. Subtask 1 - training and evaluation of model for Neutral vs Non-neutral posts
    a. Convert Left & Right labels in train and test data to Non-neutral
    b. Training and evaluation of a XGBoost Classifier with following parameters:

{ n_estimators=100, max_depth=4, learning_rate=0.05 }.

c. Predict labels for test dataset

4. Subtask 2 - training and evaluation of model for Left vs Right posts
    a. Prepare train and test datasets by filtering out neutral posts from the train data and predicted labels respectively.
    b. Training and evaluation of a XGBoost Classifier with following parameters:

        { n_estimators=100, max_depth=4, learning_rate=0.05 }.

    c. Predict labels for test dataset
5. Combine labels from results of above 2 subtasks to generate final predicted labels.

The evaluation report for this model is shown below:

	precision	recall	f1-score	support
Left	0.51	0.53	0.52	335
Neutral	0.65	0.72	0.68	414
Right	0.47	0.35	0.41	251
Accuracy			0.57	1000

This classifier has a good accuracy, and a comparatively higher right-wing recall than other methods. This makes it quite useful in the final model ensemble described in the section below

Ensembled Model

Based on the above discussions, we have the following models (best selected from each architecture with enhancements):

Rule Based	SVM	Random Forest	Feature Extraction	Indian sBERT	Siamese
52%	53%	55%	58%	52%	45%

Constrained Tf-idf	Augmented Tf-idf	Tf-idf Subword	Tf-idf with PCA	2-step
58%	57%	56%	57%	57%

Our final approach is to ensemble selected models from the above list of classifiers. Ensembling models is a method of combining predictions of multiple other models to infer a final prediction. It helps to overcome the technical challenges of a single classifier including high variance, high bias, and low accuracy.

Our base classifiers each have accuracy better than random guessing, and the training methods are slightly different to account for covariance. Hence, we can expect an improvement in results. To aggregate predictions, we utilize the Majority Voting and Weighted Average methods.

Using Majority Voting

This method makes a final prediction based on the majority voting of classes predicted by individual models. Ties are broken in order of model preference determined by listing order. We defined the following function for inferring predictions using majority voting:

```
def majority_voting(pred_list):
    # provide in decreasing order of confidence for tie-breaks
    n = len(pred_list[0])
    y_ensembled = []
    for i in range(n):
        temp = [x[i] for x in pred_list]
        y_ensembled.append(Counter(temp).most_common(1)[0][0])
    return y_ensembled
```

For this method, the classifiers selected were (in decreasing order for tie-breaks):

- Constrained TF-IDF
- Knowledge-based Feature Extraction
- Constrained TF-IDF with Dimensionality Reduction and Sentiment Scores
- 2-Step
- Sub-word TF-IDF
- Feature Extraction with PCA and Sentiment Scores

- Data Augmented (Constrained) TF-IDF

The following evaluation report is obtained:

	precision	recall	f1-score	support
Left	0.59	0.47	0.52	335
Neutral	0.59	0.86	0.70	414
Right	0.61	0.29	0.40	251
Accuracy			0.59	1000

Although the accuracy improves slightly, the recall for right-wing labels is lower than we would like.

Using Weighted Average

This method makes a final prediction based on the weighted average of classes predicted by individual models. This adds greater flexibility to the results, allowing us to boost certain metrics (such as right recall) by increasing the weights of models that perform well on them. The weights are hence selected using manual testing. We defined the following function for inferring predictions using weighted average:

```
def weighted_avg(pred_list):
    # pred_list should contain tuples of (y_pred, weight)
    n = len(pred_list[0][0])
    y_ensembled = []
    weight_sum = sum([m[1] for m in pred_list])
    for i in range(n):
        score = 0
        for m in pred_list:
            score += m[0][i]*m[1]/weight_sum
        y_ensembled.append(score-1)
    return y_ensembled
```

For this method, the classifiers selected (and weights assigned) were:

Classifier	**Weight**
Constrained TF-IDF	6
Knowledge-based Feature Extraction	6
Constrained TF-IDF with Dimensionality Reduction and Sentiment Scores	5

Sub-word TF-IDF	5
Feature Extraction with PCA and Sentiment Scores	5
2-step	6
Data Augmented (Constrained) TF-IDF	3

The following evaluation report is obtained:

	precision	recall	f1-score	support
Left	0.59	0.52	0.55	335
Neutral	0.61	0.79	0.69	414
Right	0.53	0.35	0.42	251
Accuracy			0.59	1000

This approach has a similar 3-class evaluation accuracy as the majority voting model, but has the added advantage of a better right recall and f1-score. It appears to be slightly less biased towards predicting “neutral”, which is useful for providing some valuable insights on unseen data.

Final Classification Model

The weighted average ensemble model described above was the final model selected for the application. It was used to propagate labels to all 6.7k comments, which are then displayed on the UI.

The final 3-class evaluation accuracy of 59.1% is substantially higher than the random guessing probability of 33%, and also an improvement on the results of the closest state-of-the-art pretrained model we could find (Indian-sBERT, which on standard fine-tuning gave 45%).

Scalability

The main scalability constraint is with respect to time for classification, as the model sizes remain unchanged after training and the feature sizes are constant. The following table shows time required to classify 6674 labels for each individual classifier in the final ensemble:

Classifier	**Time for 6674 labels (s)**
Constrained TF-IDF	0.382
Knowledge-based Feature Extraction	11.261
Constrained TF-IDF with Dimensionality Reduction and Sentiment Scores	0.510
Sub-word TF-IDF	4.496
Feature Extraction with PCA and Sentiment Scores	11.156
2-step	16.601
Data Augmented (Constrained) TF-IDF	0.386
Total for Ensemble	44.792

Based on the above results, we obtain a performance of **6.71ms per label.**

Scope for Improvement

1. Sarcasm Detection - the models struggle to detect sarcasm in comments, leading to certain data being completely misclassified. The following is an example of such a comment that we found to be labeled wrong due to sarcasm:

> How dare you assume that filthy idol worshippers like Hindus have any right to celebrate their misogynist, casteist, racist, homophobic, islamophobic, humanphobic, animalphobic, alienphobic festival?
>
> Only we, followers of **true faith** have right to celebrate whatever we want, anyway we want, wherever we want.
>
> >_____
> /s

Here, the “\s” is Reddit slang that clearly indicates sarcasm - however, very few comments explicitly state that they are being sarcastic, and this behavior was not encoded in our models.

2. Transliterate Hindi text - many users on Reddit post Hindi comments (and sometimes, also mixed languages) written in English. Often these are difficult to detect, and even more difficult to transliterate. In fact, we had attempted to incorporate this preprocessing step but were constrained by lack of public APIs that could effectively handle mixed Hindi and English text. One such example of a comment is:

> Bhai Rastragaan bhi toh gaane ko bol raha hai, paxtaan ka gaane ko kehte tb sochte bhi or 15 Aug ko patang udaate hai toh sb ke sb toh aane se rahe, toh zyada se zyada bnde toh 14 ko hi aayenge. (Idk me in chizo me itna nhi hu toh normally dekha jaye toh aisa hi hoga) + dekha jaye toh saalo se schools me bhi 14 ko function rkhte isiliye pak independence day wala point wrong lgta hai in ma opinion baaki zyada gyaan nhi hai mere pass in chizo ki toh gaaliya na dena pls, chhote bhai smjhkr smjha dena mujhe

3. Sufficient labeled Training Data - We consider this to be a large reason for the ineffectiveness of deep learning solutions on this problem. Most indian political datasets found online are unlabeled. A larger training dataset would undoubtedly lead to stronger results from transformer models, especially XLM BERT models pre-trained on Indian languages.

Appendix

Synonyms list:

```
AFSPA,Armed Forces (Special Powers) Act
AAP, Aam Aadmi Party
BJP,Bharatiya Janata Party,
TMC, AITC, Trinamool, Trinamool
BSP,Bahujan Samaj Party
CE,Common Era
ICSSR,Indian Council of Social Science Research
IPT,Indian Political Thought
IT,Information Technology
INC,Indian National Congress
MIPT,Modern Indian Political Thought
MRPS,Madiga Reservation Porata Samithi
TRS, Telangana Rashtra Samithi
OBCs,Other Backward Classes
RPI,Republican Party of India
SCs,Scheduled Castes
SP,Samajwadi Party
ST,Scheduled Tribe
US,United States
Andaman and Nicobar Islands, A&N islands, Andaman, Nicobar, Andaman and
Nicobar
Andhra Pradesh, Andhra, AP
Arunachal Pradesh, Arunachal
Assam
Bihar
Chandigarh
Chhattisgarh
Dadra and Nagar Haveli and Daman and Diu, Dadra, Nagar Haveli, Diu, Daman,
Daman & Diu, Daman and Diu
Delhi, New Delhi, NCT
Goa
Gujarat
Haryana
Himachal Pradesh, HP, Himachal
Jammu and Kashmir, JK, J&K, POK, Jammu, Kashmir, Jammu & Kashmir, Azad JK,
Gilgit, Gilgit Baltistan
Jharkhand
Karnataka
Kerala
```

```
Ladakh
Lakshadweep
Madhya Pradesh, MP
Maharashtra
Manipur
Meghalaya
Mizoram
Nagaland
Odisha, Orissa
Puducherry, Pondicherry
Punjab
Rajasthan
Sikkim
Tamil Nadu, TN
Telangana
Tripura
Union territory, UT
Uttar Pradesh, UP
Uttarakhand, Uttaranchal
West Bengal, WB, Bengal
India, Hindustan, Bharat
China
Pakistan, Pak
Sri Lanka, SL
Britain, UK, United Kingdom, British
Bangladesh
Afghanistan, Afghan
```

Stopwords - In addition to NLTK Stopwords

```
don't
http
you're
r
https
u
you
ki
ke
can't
www
won't
hai
```

Knowledge Base for Right-wing terms:

```
right = """right wing, RW, authority, hierarchy, order, duty, tradition,
reaction, nationalism, conservative, right-libertarian, \
neoconservative, imperialist, monarchist, fascist, reactionaries,
traditionalist, traditional, death penalty, \
religion, Bhajpa, BJP, Shiv Sena, RSS, MNS, Sanatan, dharm, Hindutva,
Islamophobia, Narendra, Modi, Amit, Shah, \
mandir, ram, valmiki, ramayan, Bharatiya, Janata, Democratic Alliance, NDA,
AIADMK, Janta Dal, bhakt, CAA, NRC, hindu majority, \
hindu unity, hindu pride, nationalist, sangh, sanghi, yogi, brahmin,
brahman, smriti irani, hindu rashtra, jai shri ram, \
pm cares, pmcares, adani, hindu""".lower()
```

Knowledge Base for Left-wing terms:

```
left = """left wing, LW, leftists, freedom, equality, fraternity, rights,
progress, reform, internationalism, anarchist, communist, socialist, \
democratic socialist, social democrat, left-libertarian, progressive,
social, liberal, western, Congress, UPA, RG, mamata, \
Aam, aadmi, AAP, CPI, Welfare, Protectionism, Commies, Rahul, gandhi,
indira, yatra, arvind, kejriwal, inclusivity, \
libby, libbies, sjw, libtard, hinduphobia, LGBTQ, masjid, pappu, christian,
muslim, secular, minority, minorities, Shashi, Tharoor, \
gay, lesbian, transgender, trans, reservation, abrahamic""".lower()
```