

# Feed Recommendation Model

An approach to generating recommendations with limited user profile and user-content interaction attributes.

## TABLE OF CONTENTS

### OVERVIEW

#### BECOMING ONE WITH THE DATA

[user\\_info.json](#)

[user\\_post\\_interaction.json](#)

[using interaction data to deal with users having no language preferences](#)

[reacted\\_posts.json](#)

[experiment: bringing content to common ground via language translation](#)

#### PREPARING RECOMMENDATION PRE-REQUISITES

[preparing content profile](#)

[preparing user profile](#)

[multi-label content cluster classifier for users](#)

#### GENERATING RECOMMENDATIONS

[recommendations based on user history](#)

[recommendations based on user genre preferences](#)

[trending recommendations](#)

[recommendations single entry point](#)

[obtaining recommendations](#)

## OVERVIEW

This document explains the methodology adopted to generate feed recommendations for users in absence of an extensive attribute set for users and their interactions with various contents. While the attribute set of the aforementioned are limited, the contents have relatively strong attributes, thus serving as a starting point to find patterns in the data available.

The proposed approach uses a combination of unsupervised learning, supervised learning, and rule based approaches to create a feature set and train a multi-label classification model for finding the most suitable subset of contents for a given user.

The final recommendation set includes multiple recommendation types each accompanied with a fallback method to cover-up in case of failure in generating recommendations.

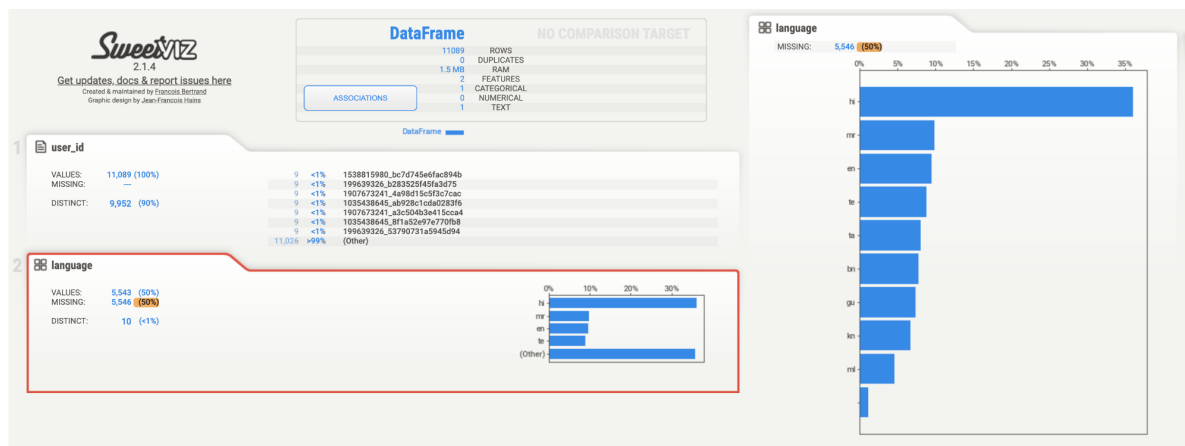
## BECOMING ONE WITH THE DATA

Prior to commencing data preparation and the ultimate task of generating recommendations for a given user, the necessary pre-requisite of EDA is conducted to get a better comprehension of the solving the given problem statement.

### user\_info.json

The user information data comprises of the user\_id and their languages. By the looks of it, it can be interpreted that the languages associated with each user are the languages a user prefers to view contents in. However, while going through the data, it is observed that the **language values are missing for almost half of the users.**

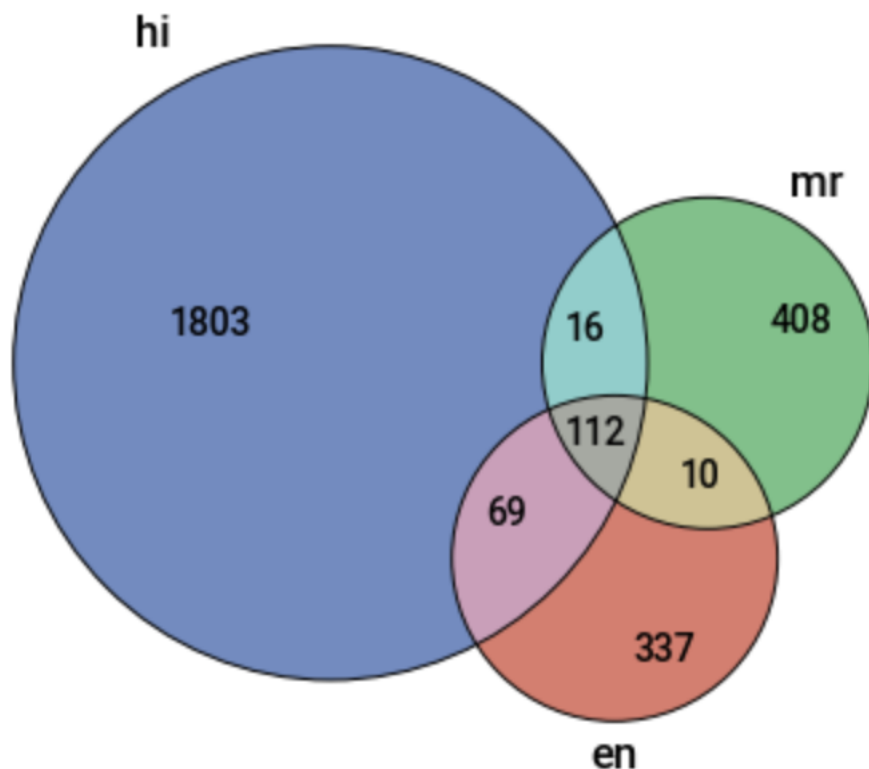
## Language Values Distribution



The language preferences is majorly dominated by 'hi', followed by a somewhat equal proportions of the next 3-5 languages.

In an attempt to find possible rule associations among the language preferences of the users, i.e. finding any potential trends such as " given user X prefers 'hi', he is certain to also prefer 'en' as his language preference ", the focus shifts to users with "hi" as one of the language preferences.

As can be observed from the below Venn Diagram, there is no such strong overlap between the top 3 most frequent language preferences. Thus there is no strong rule association as presumed.



The below results from comparing "hi" with the rest of the languages is supports the above observation.

(NOTE: This comparison is only among the users with "hi" as one of their language preferences)

```

Size of hi = 2000

---Size of mr = 546
-----Intersection Size between hi and mr = 128
---Size of en = 528
-----Intersection Size between hi and en = 181
---Size of te = 491
-----Intersection Size between hi and te = 113
---Size of ta = 446
-----Intersection Size between hi and ta = 115
---Size of bn = 432
-----Intersection Size between hi and bn = 125
---Size of gu = 411
-----Intersection Size between hi and gu = 124
---Size of kn = 372
-----Intersection Size between hi and kn = 121
---Size of ml = 253
-----Intersection Size between hi and ml = 120

```

### OBSERVATIONS

- It is necessary to come up with a method to deal with users with missing language values.
- The most prominent language in the readily available data is "hi".
- There is no strong association between the users and their language preferences (This observation is supported by verifying users from each language preference with the rest of the languages)

### user\_post\_interaction.json

The user content interaction data only includes the basic information of the content\_id viewed by a particular user\_id. Thus, there is no mention of the session duration, feedback, timestamp etc. in the provided data.

A basic inspection of the dataset shows that:

- Of the 15,200 records, only record has null values, thus it can be easily ignored from further consideration.
- Number of { user\_id, post\_id } duplicates are very few ( 9 to be precise ).
- The maximum frequency of { user\_id, post\_id } record = 2.

<b>user_id</b>		
VALUES:	15,200 (100%)	
MISSING:	---	
DISTINCT:	10,000 (66%)	
	615 4%	1941598539_1cbc90e9de2a9b3d
	573 4%	199639326_f234a11971403f3b
	209 1%	199639326_6eaac67e9222ed41
	128 <1%	1186255453_993a5900b1cef45d
	125 <1%	199639326_f25e98cf5090aec9
	37 <1%	199639326_2780e65d968644a6
	28 <1%	199639326_b8b95df715828b1d
	13,485 89%	(Other)
<b>reactions</b>		
VALUES:	15,199 (>99%)	
MISSING:	1 (<1%)	
DISTINCT:	13,322 (88%)	
	14 <1%	olympic_medal_tally_1
	10 <1%	native_hrhie89edf3e1307eaffaa08a91c713d0235
	9 <1%	news_indiaheraldnews_f493d176567632ce1109286931f72c69
	9 <1%	native_vrhi09dcd96927d4f4b238bd6bb3650841d2
	9 <1%	native_hivrdevotional61
	9 <1%	news_cricketnmore_e03c8cfa1a7a87c3093ffe49fdc15bf5
	9 <1%	news_freepressjournal_c31cfc984f5b2d93e74ee8b0dc57c599
	15,130 >99%	(Other)

### using interaction data to deal with users having no language preferences

The analysis of user\_info.json data shows that 5546 users did not have any explicit mention of their respective language preferences.

In an attempt to retrieve this information from the user interaction data, an experiment was conducted to find any contents viewed by these 5546 users. The languages of these contents can be retrieved from the reacted\_posts.json file if required.

```
na_user_log = log[log['user_id'].isin(na_users['user_id'].tolist())]
```

The results obtained showed at least one post to be viewed for all these users. Thus, the languages of these contents can be utilised to fill the missing user language values.

	user_id	reactions
0	1186255453_6b96200b883f6581	[{'post_id': 'native_vren8feb44033d85428330529...'}]
1	864995365_a2b572a31e68cd12	[{'post_id': 'publicvibe_1611854902015326419'}]
2	864995365_9f4b8136228b138e	[{'post_id': 'native_vrhic587c122d164501940a5c...'}]
3	1186255453_cf6c4eca3e1f9e84	[{'post_id': 'twitter_1516340979720024071'}]
4	199639326_31fecdd681abf0ef	[{'post_id': 'news_freepressjournal_2e7fb3b377...'}]
...	...	...
5541	864995365_e92cc1084d1a05f1	[{'post_id': 'instagram_CKtWX67p8fz'}]
5542	864995365_6c0516bf8ccc3bf5	[{'post_id': 'news_filmibeat_f50cbf76ef4cf1f36...'}]
5543	1035438645_e08329519b0325cf	[{'post_id': 'youtube_CTGL8qqjL7s'}]
5544	864995365_8abe257b0eb3da4d	[{'post_id': 'news_freepressjournal_64d7eeff21...'}]
5545	199639326_cb6d00e734ecf58b	[{'post_id': 'news_oneindia_6330225528a47bd816...'}]

5546 rows × 2 columns

The updated user\_info data with languages for users with originally no language preferences is prepared and stored at

```
submission/data/complete_user_info.pkl
```

## reacted\_posts.json

This file includes a relatively detailed attribute set for the contents.

The attribute “ml\_interests” is assumed to be the set of genres to which a particular content potentially classifies as.

The “content” attribute comprises of at most 3 key:value pairs for each record, i.e Title, Caption, Description. As per a superficial observation, description is present for all the records, while title and caption occur comparatively less frequently. The 3 sub-attributes occur in many different combinations in the data, as can be seen in the below result snippet.

```
dict_keys(['description'])
dict_keys(['title', 'description'])
dict_keys(['title', 'description', 'caption'])
dict_keys(['title', 'description', 'caption'])
```

```
dict_keys(['title', 'description', 'caption'])
dict_keys(['title', 'description'])
dict_keys(['title', 'description', 'caption'])
dict_keys(['title', 'description', 'caption'])
```

On going through the “content” attribute, it was observed that there are multilingual and mix of languages in many records. Thus, it is necessary to address this issue by exploring the option of machine translation.

### experiment: bringing content to common ground via language translation

In an attempt to unify the text within “content” attribute, multiple approaches to text translation were experimented with to translate the entire text corpus to English for convenient downstream task of vectorising the text to embeddings. The following modules were explored to address this concern:

- **googletrans**

#### googletrans

Compatible with Python 3.6+. Fast and reliable - it uses the same servers that translate.google.com uses  
Auto language detection Bulk translations Customizable service URL HTTP/2 support more features are coming soon. Proxy support Internal session management (for better bulk translations) This library uses


 <https://pypi.org/project/googletrans/>



While this is a hassle free approach to getting good translation results as shown below, the *googletrans* bypasses the original Google Translate API, and is thus unreliable when working with large amounts of data.

#### Googletrans throws a ConnectTimeout error (the handshake operation timed out)

I am currently working on a translator in Python 3.8.5 but I am not getting an appropriate result. My code thus far is from googletrans import Translator text = "Vous êtes français" translator = Translator() dt = translator.detect(text) print(dt) In my terminal, I should be getting the source and the destination which is fr

 <https://stackoverflow.com/questions/65279841/googletrans-throws-a-connecttimeout-error-the-handshake-operation-timed-out>



Original Description= डिस्ले फ्रिज में मजे से पिज्जा खा रहे थे चूहे, वीडियो वायरल

Translated Description= Rats were having fun eating pizza in the display fridge, the video went viral

Original Description= 26/11 हमले के हीरो तुकाराम ओम्बले के नाम पर रखा गया मकड़ी की नई प्रजाति का नाम

Translated Description= New spider species named after 26/11 attack hero Tukaram Omble

Original Description= На официальном сайте портативной консоли Steam Deck нашли пасхалку к утям, которые появлялись до анонса. Она

Translated Description= On the official website of the Steam Deck portable console, they found an easter egg to the leaks that appeared

Original Description= Pour leur premier match du tournoi olympique, les Bleus ont fait chuter la Team USA, portés par un esprit d'équipe

Translated Description= For their first game of the Olympic tournament, the Blues brought down Team USA, carried by an unerring team spirit

- **Google Cloud Translate API**

While Google's API is the go-to option for this task, it is not a completely free alternative for text translation when working with large amounts of data.

#### Cloud Translation documentation | Google Cloud

Allows programmatic integration with Google Translate.

 <https://cloud.google.com/translate/docs>



## Cloud Translation - Basic

The following pricing information applies to the `detect` method call and the `translate` method call.

Feature	Metered usage	Price
Total usage for <a href="#">language detection</a> and <a href="#">text translation</a> by using the NMT model	First 500,000 characters* per month	Free (applied as \$10 credit every month)†
	500,000 to 1 billion characters* per month	\$20# per million characters
	Over 1 billion characters* per month	<a href="#">Contact a sales representative</a> to discuss discount pricing.


- **translate**

This is a free to use Python module, that can be utilised when working with large amounts of data. However, this module does not cover the languages in this use case. Thus, this is also not a suitable option for further consideration.

**translate**

Translate is a simple but powerful translation tool written in python with support for multiple translation providers.

<https://pypi.org/project/translate/>



As a result, the idea of text translation was dropped.

## PREPARING RECOMMENDATION PRE-REQUISITES

Before jumping into the main task of generating the actual recommendations, the observations and analysis of the available data was utilised to reformat, preprocess, and add to the raw attribute set, so as to make it suitable and convenient to be consumed for recommendation.

### preparing content profile

`/submission/content/`

The scripts under this directory orchestrate the creation of content profile attributes that shall be utilised in the downstream tasks of unsupervised learning, i.e Clustering of contents.

The motivation behind clustering is to group contents on a criteria not solely based on their category and that also utilised their textual information. The entire pipeline involves multiple subcomponents, which during the process further add to the attribute set

(such as content source, text language etc.)

The content profile creation includes the following sub-procedures:

- Fetching Content Attributes
- Merging Content Attributes
- Processing Content Attributes
- Preparing Feature Set for Unsupervised Learning

- Clustering Contents

## ✓ FETCHING CONTENT ATTRIBUTES

```
/submission/content/fetch_attributes.py
```

### splitting content attribute

Here, the "content" attribute, comprising of any potential combination of {title, caption, description} is split into 3 separate attributes.

The idea is to avoid using description for generating text embeddings due to its large body and mix of languages. Thus a new attribute 'merged\_text' is created in later steps, that keeps either title + caption, or the description, if title and caption are missing in the text.

### creating source attribute

every post\_id mentions the source of the post, which can be useful information when finding relations among contents. Thus a new attribute 'source' is created by splitting the post\_id attribute.

```
self.data[SOURCE] = [post_id.split("_")[0] for post_id in self.data[POST_ID]]
```

## ✓ MERGING ATTRIBUTES

```
/submission/content/merge_attributes.py
```

This file merges the acquired attributes. The procedure involves the following sub-components:

- 1) Choose to proceed with description or title+caption for each record
- 2) Check for semantic overlap between title and caption to avoid redundancy.
- 3) Create an additional attribute indicating the text language

### checking for semantic overlap

If both title and caption are available for a content, the description is dropped from further consideration. However, the semantic overlap between the title and caption is still checked. This is done to avoid considering redundant information, because it is possible that title and caption consist of the exact same text. If the semantic overlap exceeds the threshold, only the title is considered for further tasks.

### language detection

An additional attribute 'text\_language' is created using the text in 'merged\_texts' to detect the language of the text in the latter. Since it is possible for the "ml\_language" field of a content to have multiple languages, the text\_language field will help in ascertaining which one of them is present in the text being considered. This attribute is used in the following ML tasks as well.

## ✓ PROCESSING ATTRIBUTES

```
/submission/content/process_attributes.py
```

This script includes basic text preprocessing methods to stabilise the text before generating their embeddings. The preprocessing methods include:

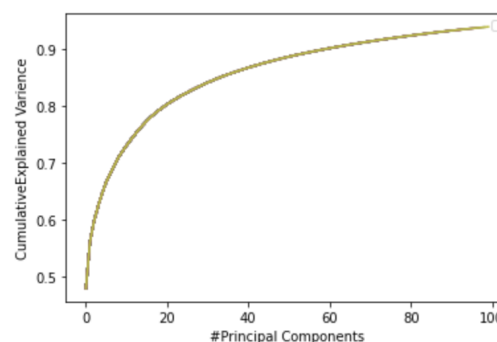
- removing redundant hyperlinks, ASCII/Unicode strings.
- lowercasing the entire corpus.
- removing non-alpha numerics.
- removing redundant whitespaces.
- text lemmatisation

### ✓ PREPARING FEATURE SET

```
/submission/content/prepare_feature_set.py
```

This script prepares the content attributes to be fed into a clustering model. The steps involved as a part of this process are:

- One-Hot encoding categorical attributes
- Generating Text Embeddings for “merged\_text”
  - The word embeddings are calculated using the sPaCy “`en_core_web_lg`” pre-trained model.
  - Only the English text is used for generating embeddings, the other language texts are set to Zero Vectors.
- Dimensionality Reduction
  - Including the (300,) word embeddings for each content record, the entire attribute set for each content accounts for 400+ features.
  - Using PCA the feature dimensions are reduced to handful of most representative attributes.
    - In order to verify the optimal dimension value before performing the actual PCA, the variance of its principal components is observed.
    - As a result, PCA is computed on for N = 65



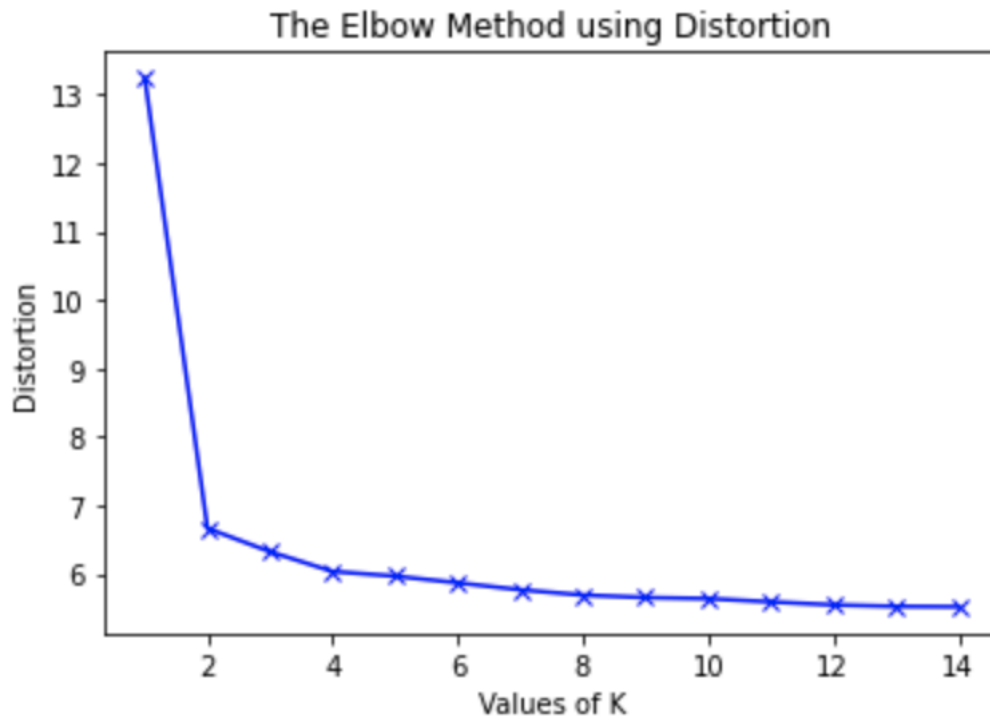
### ✓ PREPARING CLUSTER LABELS

KMeans Clustering Algorithm is used for segmenting contents into various clusters.

KMeans is a simple yet effective way to cluster attributes based due to its runtime computational complexity of  $O(m)$ ;  $m = \text{\#nodes}$

The optimal K value is computed and observed using the elbow method.





Thus the KMeans Algorithm is run over the 65-sized content vectors. As a result, a “cluster” attribute is prepared for each content. The clustering results are stored at:

```
/submission/content/intermediates/all_cluster_labels.pkl
```

In order to execute the content profile generation procedures, use the `main.py` file at:

```
/submission/content/main.py
```

Any intermediate files/results are stored at

```
/submission/content/intermediates/
```

### preparing user profile

```
/submission/user/
```

The scripts in this directory create user profile attributes to be used for downstream supervised neural model training. The attribute set comprises of 3 sub-components:

- 1) **User Info:** contain information about user's language preferences

- 2) **User Interaction:** contain information about each posts\_id viewed by every user
- 3) **Content Info:** attributes particular to contents, including the obtained clustering results

#### ✓ FETCHING USER INFORMATION ATTRIBUTES

```
/submission/user/fetch_info_attributes.py
```

Includes basic processing of user\_info attributes, i.e. one-hot Encoding Language Attribute

#### ✓ FETCHING INTERACTION ATTRIBUTES

```
/submission/user/fetch_interaction_attributes.py
```

Includes basic exploding and one-hot encoding of content info attributes, such as "ml\_languages", "ml\_interests", etc. Same steps over the "reactions" attribute in the user interaction data.

The attributes not required for model training are dropped from further consideration.

In order to execute the content profile generation procedures, use the main.py file at:

```
/submission/user/main.py
```

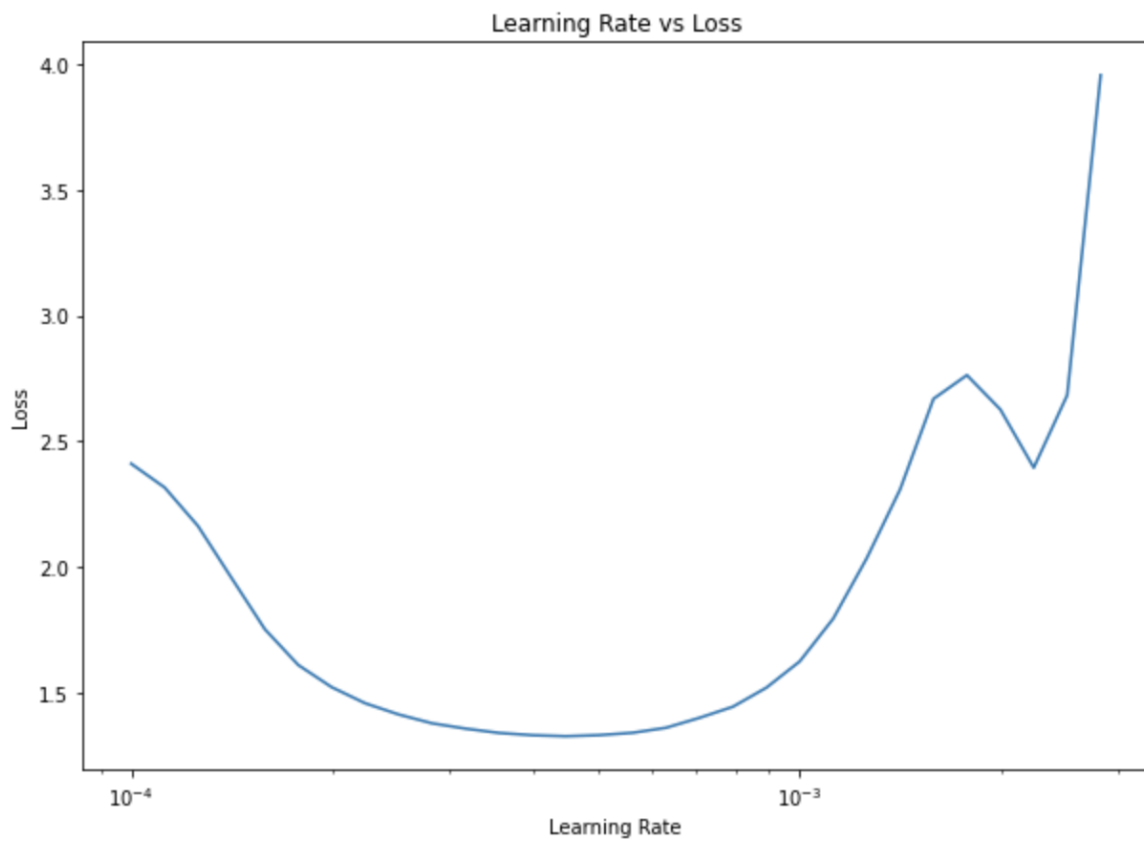
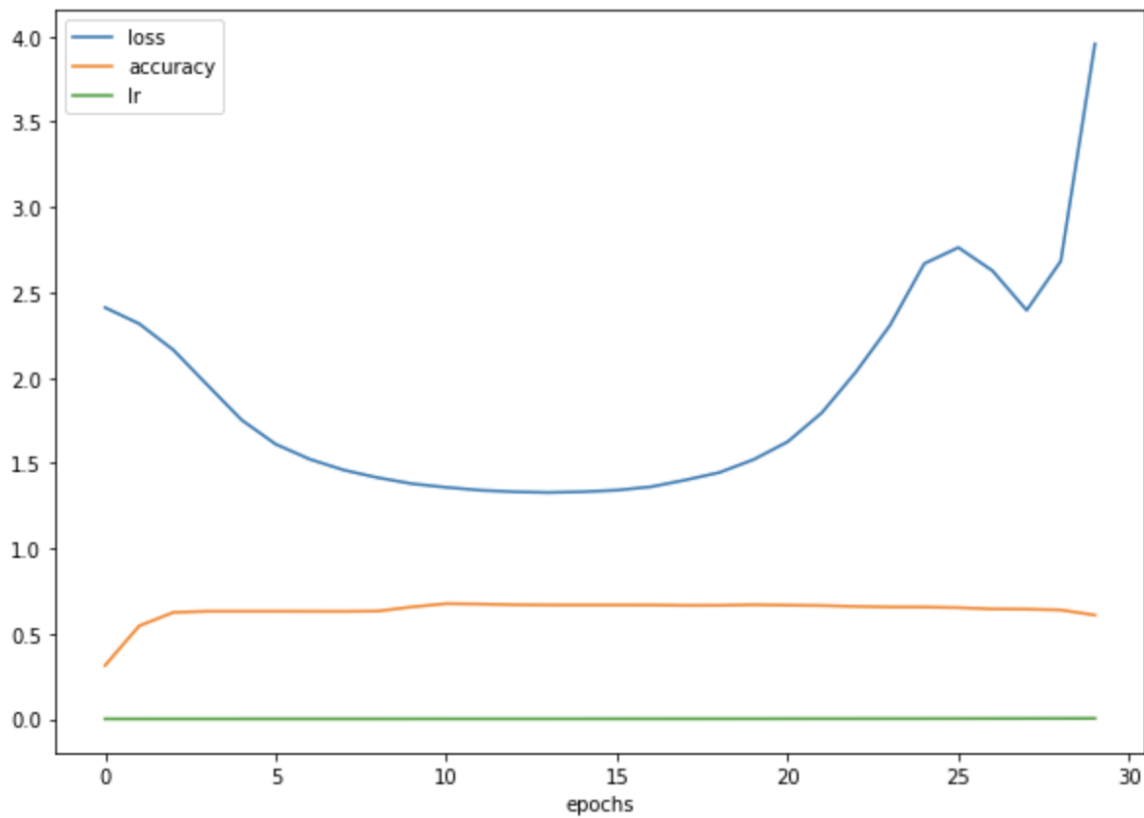
Any intermediate files/results are stored at

```
/submission/user/intermediates/
```

### multi-label content cluster classifier for users

The idea behind creating this neural network, is to use the obtained content clusters as the target variable to identify the most suitable subset of content clusters for a given user feature set. The results from this neural network will be used in the recommendation process to filter out the contents to be considered for further consideration.

In order to create a multi-label neural model, the learning rate was experimented using a scheduler during the training process, the results obtained are as follows:

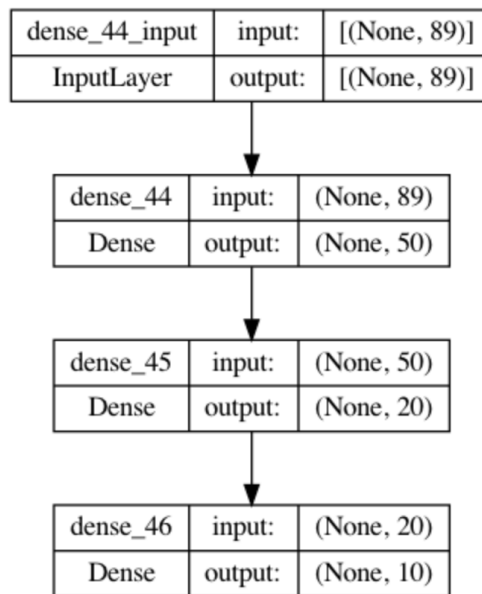


As can be observed, the increase in learning rate increases the loss drastically for greater values.

Thus the model was experimented with learning rate in the vicinity of  $10e-4$ . The observations showed no significant learning, with the best training accuracy of 0.45.

In an attempt to increase the model performance, the model was trained on variable learning rate with early stopping to avoid model degradation. The results showed significant improvement (though not good in ML terms) of training accuracy of 69.9% and test accuracy of 65.4%.

Experiments were conducted by adjusting the depth and width of the neural network and using other non-linear activation functions. However, tuning the hyper-parameters for deeper neural networks did not show any significant improvement in model training.



Configuration of the trained multi-label classifier

This multi-label classifier can be trained using the file

```
/submission/user/model/train.py
```

The trained model is stored at

```
/submission/user/model/01_userContentClusterClassifier
```

## **GENERATING RECOMMENDATIONS**

```
/submission/recsys/
```

The recommendation set comprises of 3 recommendation types:

- Recommendations based on user history
- Recommendations based on user Genre Preferences
- Trending Contents

## recommendations based on user history

```
/submission/recsys/history_based_recommendations.py
```

- This recommendation type builds on the predictions obtained from the multi-label classifier to get a subset of contents most suitable for a given user.
- If the user has at least one record in user interaction data, the user is considered to have history.
- The most recently viewed content is compared with the contents in the clusters returned by the classifier to get the most similar contents.
- The content similarity is computed by comparing the text embeddings of the contents.
- The results are prepared in decreasing order of similarity scores with `rec_type` label as "HISTORY"
- If the user is found to have no history, the contents in the classifier suggested clusters is returned as the result, but with the `rec_type` label as "NO\_HISTORY"

💡: For any given recommendation case, it is ensured that only those contents are considered for recommendation which are in the language preferences of the given user

## recommendations based on user genre preferences

```
/submission/recsys/genre_based_recommendations.py
```

- The "ml\_interests" of the contents viewed by a user are considered to be the user's genre preferences. This recommendation type filters the content on the basis of their "ml\_interests", i.e. whether a content's genre is one of the user's genre preferences? If yes, consider for recommendation, else skip.
- Since no scores are involved in this method, it is possible that different users with overlapping genre preferences are given the same recommendations. In order to avoid this issue, the result is shuffled to introduce randomness and variability in the recommendations of two users with same preferences.
- If user has some genre preferences, the recommendations are given from those specific genres in shuffled order with `rec_type="GENRE"`
- If user has no genre preferences, the recommendations are solely based on the language preferences of the user with `rec_type="NO_GENRE"`

## trending recommendations

```
/submission/recsys/trending_recommendations.py
```

The basic notion behind generating trending contents is to evaluate the contents as a function of popularity and age. The score of the content must decrease with age so as to ensure freshness of content.

Since trending scores heavily depend on feedback, and this is absent in the current data, the total views is the sole metric to computing popularity, i.e.

```
popularity(content_X) = total_views(content_X, user_interaction_data)
```

Since the timestamp is also absent in the user interaction data, it is assumed that each user interaction record is recorded at a new new timestamp, such that timestamps of user interaction entries are sequentially increasing order.

```
age(content_X) = max(index(content_X, user_interaction_data))
```

As observed during initial analysis of data, that the maximum frequency of content viewed = 2, it seems obvious that popularity will have less contribution to the trending score. In order to overcome this issue, the formula used to generate trending scores is:

```
trending_score(X) = popularity(X) / age(X)^0.2
```

The above reduces the influence of age to increase the range of trending scores.

The file storing the trending scores is stored at:

```
/submission/recsys/data/trending.pkl
```

## recommendations single entry point

```
/submission/recsys/main.py
```

The entry point consolidates recommendations from all the recommendation types into a single recommendation set.

Each recommendation use case is accompanied by a fallback function, which can generate some recommendations if in case, the model fails to run its mainstream method.

## obtaining recommendations

open your Terminal and go to the directory:

```
/submission/recsys/
```

run the following command

```
flask run
```

## **Execution Screenshot**

POST http://127.0.0.1:5000/recommendation Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2  "user_id": "1035438645_0085cccaab2be238"
3
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1646 ms Size: 5.95 KB Save Response

Pretty Raw Preview Visualize JSON

```
90  },
91  {
92    "post_id": "news_oneindia_e3d883fa35e6f87731df58900e8e0be1",
93    "rec_type": "HISTORY"
94  },
95  {
96    "post_id": "news_feeds-bollywoodlife_71f5cfe327c8675c3225db36269063e6",
97    "rec_type": "HISTORY"
98  },
99  {
100   "post_id": "videobytes_chingari-61b32f4d817291086ce5c210_626ad7c2a07b400987733be5",
101   "rec_type": "HISTORY"
102  },
103  {
104    "post_id": "youtube_t080praEXtE",
105    "rec_type": "HISTORY"
106  },
107  {
108    "post_id": "videobytes_chingari-5f93f284cdadba0949ae7a69_627803411e534184ea565319",
109    "rec_type": "GENRE"
110  },
111  {
112    "post_id": "native_hivzhealth192",
113    "rec_type": "GENRE"
114  },
115  {
116    "post_id": "native_hihzdance164",
117    "rec_type": "GENRE"
118  }
119 }
```