Prediction of YouTube Video Tags using Neural Networks

Suteja Patil
Graduate Student, Khoury College of Computer Sciences
Northeastern University, Boston

Business Use Case

- YouTube has exploded in popularity over the years because of its content and technology and has become a household name. Almost 5 billion YouTube videos are watched every single day. One of the reason for its fame is the efficient search engine which is backed by Google. However most of the content creators who upload their videos on YouTube don't put a meaningful title or an elaborate description of their video but they have an option to select tags relevant to their video in the description section. Tags are words and phrases which can help a video in better ranking for more key words and also increase the chance that they will be displayed in the related and suggested video lists of other videos. So tags can help content creators to increase the views of their videos and also enhance viewer experience. Overall, it will help YouTube grow as a platform.
- So I have developed a YouTube video tag prediction model to predict tags using description under a video. This will help YouTube provide better suggestions for tags and allow content creators to select the relevant ones. For example, if Bob decides to upload a video titled 'A day in my life' then he can choose appropriate tags from a bunch of words such as 'travel', 'food', 'lifestyle', 'gym', 'shopping', 'college', 'movie' which are present in his content. So I am using YouTube's past data to build a prediction model which can be used in future for better recommendation.

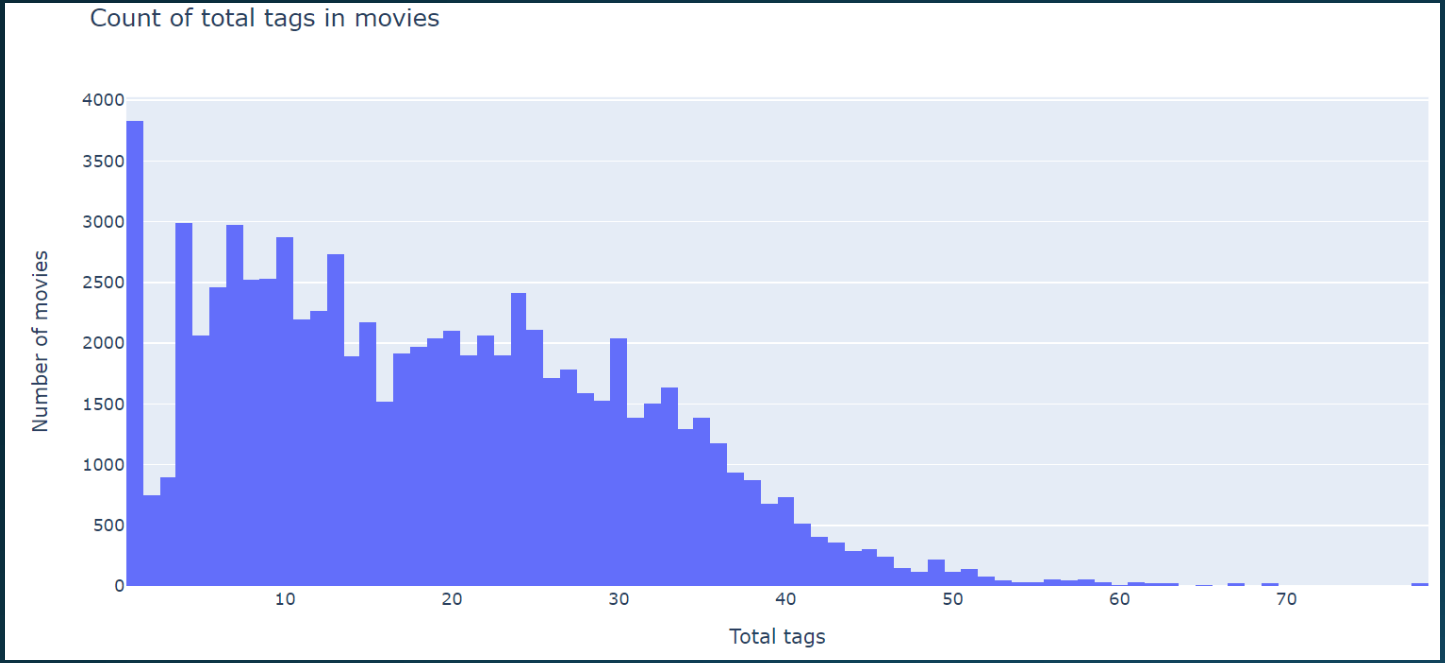
Data Modeling

- Used data sets from Trending YouTube Video Statistics to build the model.
- This dataset includes several months of data on daily trending YouTube videos from 10 different countries. Due to hardware limitations, I could use only two dataset from 10 files- 'USvideos.csv' and 'Gbvideos.csv' that has data of USA and Great Britain respectively.
- The datasets have 16 columns and total combined 79865 rows.
- Column attributes are – video_id, trending_date, title,channel_title, category_id, publish_time, tags, views, likes, dislikes, comment_count, thumbnail_link, comments_disabled, ratings_disabled, video_error_or_removed and description.
- Out of the 16 columns,we need only two-'description' and 'tags'. I have developed a model using description to predict tags.

Data Preprocessing

- Data cleaning increases overall productivity and allows for the highest quality information in one’s decision-making. It helps in minimizing errors when multiple sources of data are at play.
- Since the data is from multiple countries, data cleaning helped in focusing on words that add meaning and help in better modeling.
- Before moving into data cleaning, I have kept the attributes that are useful. In my case I need columns- ’description’ and ‘tags’ .
- I have implemented following data cleaning methods –

1. Removed entries with null values
2. Removed all non-alphanumeric and whitespace characters
3. Converted text into lowercase
4. Expanding contractions like I’ve to I have
5. Removed single characters
6. Removed stop words



EDA

- Exploratory Data Analysis involves generating summary statistics for numerical data in the dataset and creating various graphical representations to understand the data better.
- I added a new column 'total tags' to count the total tags under a video.
- Minimum number of tags is 1 and maximum is 78.
- These are the numbers without preprocessing of data.

Feature Engineering Techniques

- I have split the data into 80% training and 20% testing data and applied following Feature Engineering techniques –

1. MultiLabelBinarizer
   - Scikit-learn's MultiLabelBinarizer converts input labels into multilabel labels, each example can belong to multiple classes. Here the label (can have multiple classes) is transformed into a binary vector such that all values are zero except the indexes associated for each class in that label, which is marked with a 1.
   - In my case, tags are the output data so I have first split them into tokens using str.split() function and then transformed the training and testing tags into binary vector using MultiLabelBinarizer.
2. TfidfVectorizer
   - Term Frequency-Inverse Document Frequency (tf-idf) value of a term in a document is the product of its tf and idf.
   - Term Frequency (tf) is the number of times a term appears in a particular document while Inverse Document Frequency (idf) is a measure of how common or rare a term is across the entire corpus of documents.
   - TfidfVectorizer is the base building block of many NLP pipelines. It is a simple technique to vectorize text documents — i.e. transform sentences into arrays of numbers — and use them in subsequent tasks.
   - In my model, I have used TfidfVectorizer to vectorize the input labels, that is ‘ description’.

Model Selection

- I have used Keras Sequential Model to build this prediction system. It is mainly used when you have multiple inputs and multiple outputs and as my dataset had multiple description words and multiple tags to be predicted, I found this algorithm to be the most suitable one.
- I have trained the model with an input of 10000 features.
- The input layer is of 1000 Dense blocks while next two layers are of 500 and 100 blocks/units. I have applied ReLu activation function for all these layers as the model just needs a positive value to be passed to the output layer.
- The output layer is of the size of training data of tags where I used sigmoid function as I needed result in binary which we will inverse transform later to get the tags.
- I have used Binary cross entropy as the loss function compares each of the predicted probabilities to actual class output which can be either 0 or 1.
- I have used Adam as the optimizer to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Also, results of the Adam optimizer are generally better than every other optimization algorithms and have faster computation time.
- For evaluating the performance of the model, I have used Accuracy as the metric.

Results

- ***Note- The results are solely based on a model built by using 2 datasets having around 70k entries because of limited hardware capacity. The performance of the model can be improved using large amount of data, higher RAM and processing capacity.***
- I fit the training input and output on the built model and executed it for 50 epochs with a batch_size of 100.
- The training accuracy achieved is around 11% .
- The predicted tags are printed for reference to show the efficiency and correctness of this model.
- Even if we increase the epochs, the accuracy wont increase beyond the current result as the data and hardware capacity is limited on my system.
- But the model will definitely perform better with more data and higher capacity hardware.

Alternate Approaches and Future Scope

- Sequential Model API is a way of creating deep learning models and as a known fact, deep learning models tend to work well with more data.
- Also, the recent advances in Natural Language Processing to produce valuable insights from text-based data will help in using tags to understand users interest.
- We can also use comments under videos for Sentimental Analysis.
- This model can also be expanded to improve Search Recommendation system using history of tags previously used by user.
- We can also try to implement the model using RNN methods-Long Short Term Memory(LSTM) and GRU(Gated Recurrent Unit).

Thank you!