A PDF file explaining the approach in detail and reporting the accuracy and models that were used.

MIDAS@IIITD Summer Internship/RA Task 2021

Task 3: NLP

GOAL:

Use a given dataset to build a model to predict the category using description. Write code in python. Using Jupyter notebook is encouraged.

1. Show how you would clean and process the data
2. Show how you would visualize this data
3. Show how you would measure the accuracy of the model
4. What ideas do you have to improve the accuracy of the model? What other algorithms would you try?

About Data: You have to clean this data, In the product category tree separate all the categories, figure out the primary category, and then use the model to predict this.

If you want to remove some categories for lack of data, you are also free to do that, mention this with explanation and some visualization. Questions are made this way to check if candidates are able to understand this.

Note:- 1) Goal is to predict the product category.

2) Description should be the main feature. Feel free to use other features if it'd improve the model.

3) Include a Readme.pdf file with approach in detail and report the accuracy and what models were used.

Dataset Link: **https://docs.google.com/spreadsheets/d/1pLv0fNE4WHokpJHUIs-FTVnml9STgog05e658qEON0I/edit?usp=sharing**

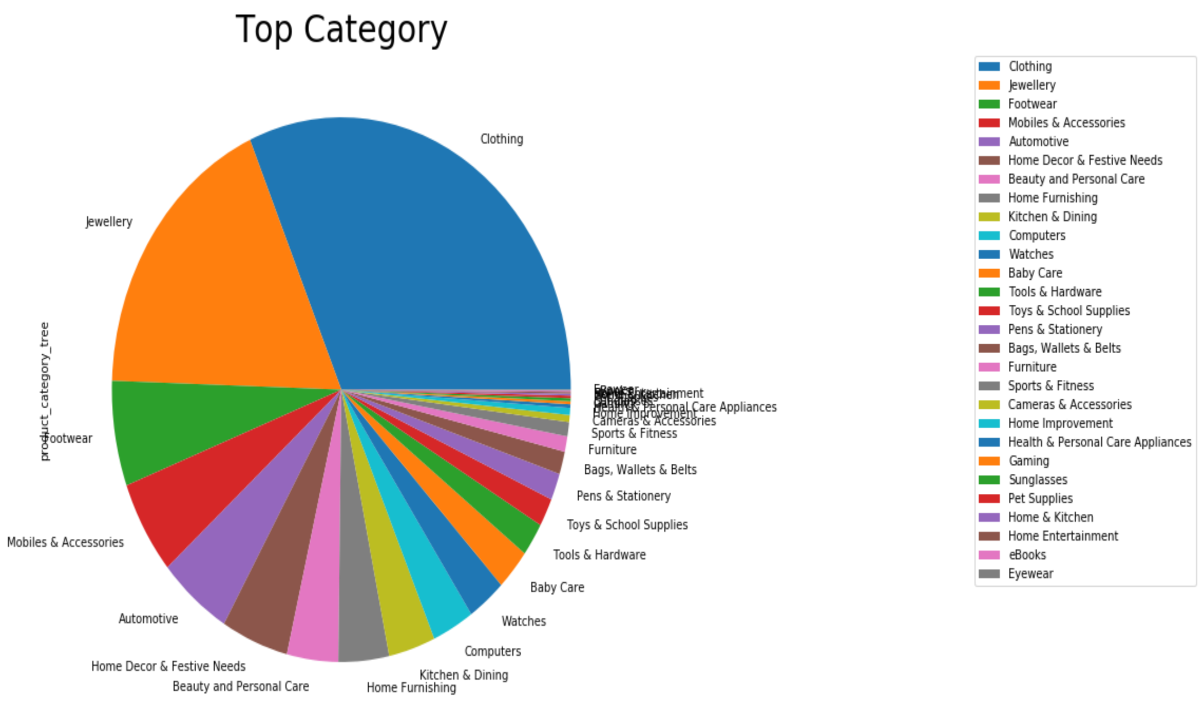
Github Link: https://github.com/ishan17vk/Multimodal-Digital-Media-Analysis-Lab-IIITD-Task

1.Cleaning and processing the dataset

- Began the pre-processing part, by exploring the dataset using describe, info, shape and isnull functions.
- Dropped the null values.
- We observed that there a lot of symbols, characters, numbers and extra spaces which are not useful for us in our model. Therefore, we cleaned them by defining a function and with the help of Regular Expression library(already imported earlier).
- After pre-processing our dataset reduced from 20000 to 19661 rows.
- We decided not to reduce the categories further.

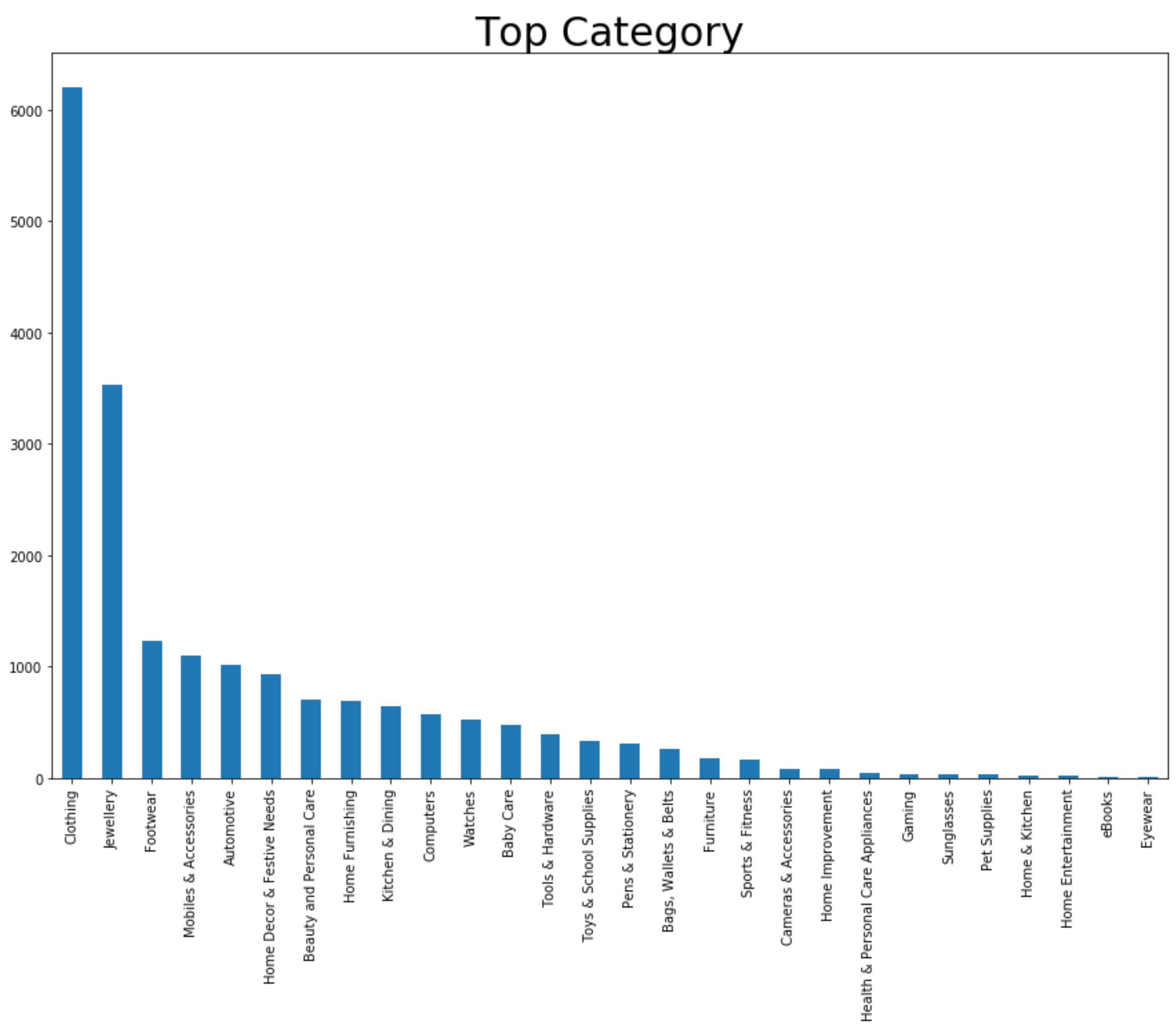
2.Data Visualisation

Pie Chart - A pie chart is best used when trying to work out the composition of something. If we have categorical data then using a pie chart would work really well as each slice can represent a different category.



Bar Plot -Bar plots are an extremely effective visual to use in presentations and reports. They are popular because they allow the reader to recognize patterns or trends far more easily than looking at a table of numerical data.

After visualising our data we got to know that:-
Clothing 6197
Jewellery 3531
Footwear 1227
Mobiles & Accessories 1099
Automotive 1012
are the main categories and they combine for more than 50% of our data.



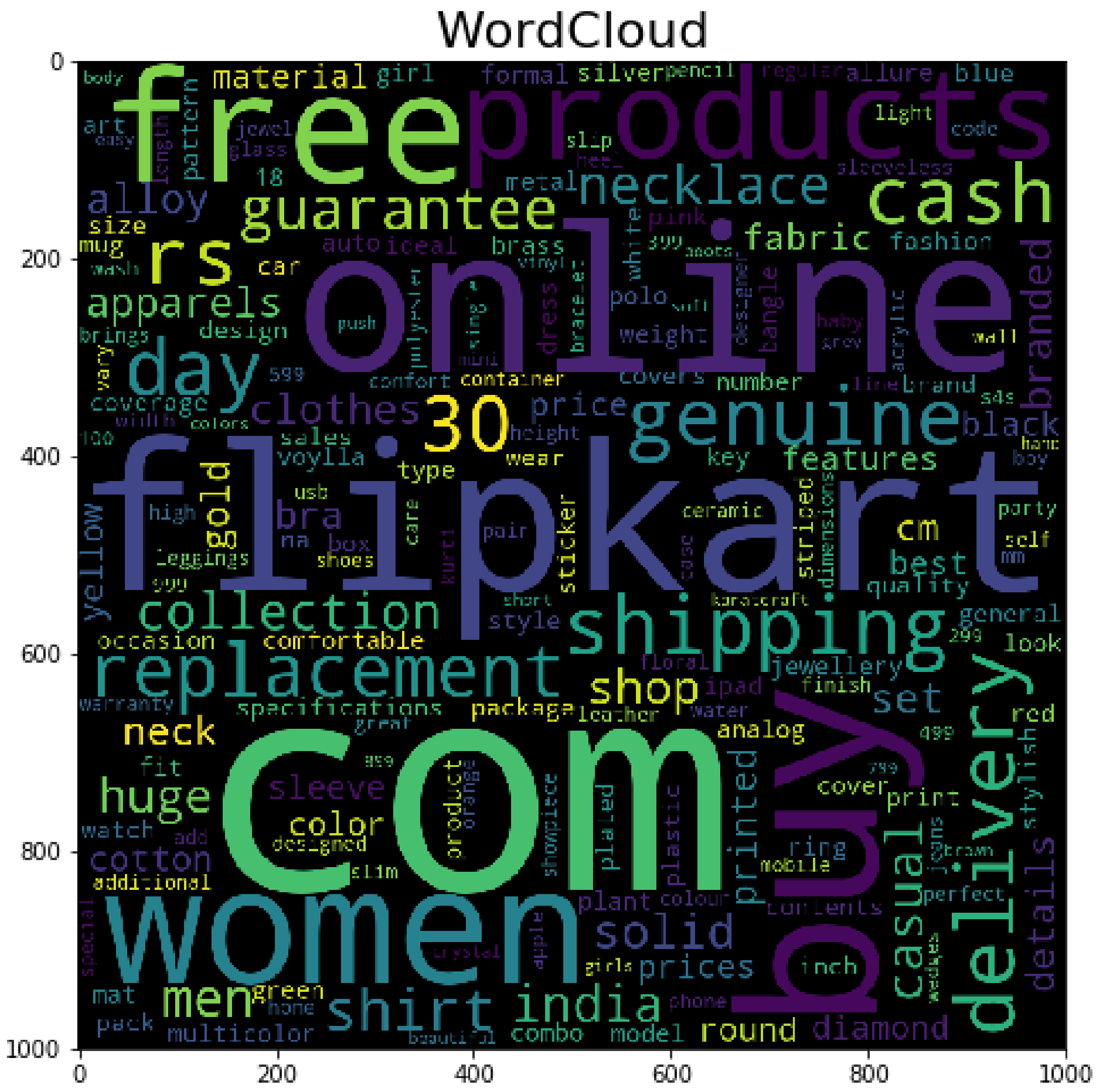
3.Model and Accuracy of The Model

For Natural Language Processing (NLP) to work, it always requires to transform natural language (text and audio) into numerical form. Text vectorization techniques namely Bag of Words, Count vectorizer and tf-idf vectorization, which are very popular choices for traditional machine learning algorithms can help in converting text to numeric feature vectors.

Our main issue with our data is that it is all in text format (strings). The classification algorithms that we've learned about so far will need some sort of numerical feature vector in order to perform the classification task. There are actually many methods to convert a corpus to a vector format. The simplest is the the **bag-of-words** approach, where each unique word in a text will be represented by one number.They are not generally used when length of text is large.

In CountVectorizer we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped is in processing our data more efficiently.

💡 Word cloud is a technique for visualising frequent words in a text where the size of the words represents their frequency. One easy way to make a word cloud is to search 'word cloud' on Google to find one of those free websites that generate a word cloud. You can possibly customise how it looks like. Quick and easy!



We split our dataset in the two categories as follows:
Training Data - 75%
Testing Data - 25%

We have achieved 95% accuracy which is a good result.

We have achieved 92.65% accuracy with the testing dataset.

Out of the (4916,) target categories which we considered, the model predicted 4555 c
ategories correctly

4.What ideas do you have to improve the accuracy of the model? What other algorithms would you try?

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

After the counting, the term weighting and normalization can be done with TF-IDF, using scikit-learn's TfidfTransformer.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

See below for a simple example.

Example:

Consider a document containing 100 words wherein the word cat appears 3 times.

The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

We have achieved 88% accuracy which is not as good as the count vectorizer method.

We achieved 85 % accuracy which is very less when compared to the count vectorizer method

Rather than focussing on and using too many models, I have confined myself to two models as well as presenting the notebook in a detailed manner.
There is a markdown text before each line of code suggesting what it does and the notebook has been divided into 4 sections as asked in the task.

Considering that we didn't left out the categories and trained on 19661 rows our model achieved 95% accuracy on the training data and 92.65% on the testing data.

To conclude, I have tried to explain everything and each code line by line. The Jupyter notebook along with this PDF File can be also very useful for a beginner and help them to understand the Natural Language Processing easily.

Thank You