REPORT

This report is an honest representation of the work I've done towards completing the assignment. It was a challenge to work on this assignment and I enjoyed it quite a lot. Below is the flow of how I proceeded. I have also mentioned my observations and places I could do better. As I proceeded towards completing the task I realized a lot of places which can be performed better but couldn't implement them due to time constraints. Nevertheless I have tried my best and this experience will help me in the future.

Topics:

Duplicate definition:

The problem statement states that the seller could "upload the same listings multiple times", this means that the same exact product down to the exact same size and colour has been uploaded multiple times. So a duplicate listing would have roughly the same title and images, and same colour, size, brand. Also obviously the seller will have to be same, as 2 exact same products can be sold by 2 different sellers and they won't be considered as duplicate listings. So holding this assumption as my base I proceeded forward.

Using the dataset:

The main dataset given was ~7GB, enough to overload my RAM and freeze my computer when I naively tried to open it in a excel viewer. So initially I read the first few thousand lines using pandas and studied the data given. As the problem statement said the task was limited to subcategory 'Tops', this was the first opportunity to reduce the dataset. I only read the column 'categories' and found the total number of unique entries. I found Tops were present in 2 categories. But to drop the rows I had to read the whole dataset. So I had no option but to use my Google cloud account (I do realise this is not an ideal solution, a better solution could be line by line reading but I didn't have enough time to test other methods) and use the extra RAM to read the whole dataset, clean & filter it and return a workable dataset to me. I dropped all rows not containing 'Tops' and also dropped the following columns for the following reasons:

- description: too many NaN entries, also too large
- categories: now redundant as only tops present

- special/discount price: won’t help find duplicates, seller might change discount
- cod,sellerreview etc: not useful for duplicates

This gave me an easy workable csv of 230MB and I could proceed. As the dataset still had around 350000 rows, I had to somehow sort the data into manageable sets. The obvious answer was clustering and I chose ‘titles’ as a good place to start. In hindsight, clustering using the column ‘brand’ would give much better reasons and would be faster, but I had already trained the model for title (which took 10 hours)when I realised this.

Creating text vectors and parallelization:

To cluster using ‘titles’, I had to convert words to vector embeddings. The word2vec module is quite popularly used for this. But as the titles were mostly multi-worded I decided to use doc2vec which is essentially word2vec with functionality to directly use sentences. I trained the model on all the titles overnight. The model takes a sentence and returns a 500 dimensional vector for it. Now I had to get the embeddings for all 350000 titles, quite a lengthy task. And as python is notorious for running everything on a single thread, doing that would take days. So here I had to improvise and implement multithreading by dividing the titles into four sets and creating four threads with their own instance of the model and have 4 inferences parallel. This sped up the process quite a lot and I had all the embeddings in half a day. I implemented multi-threading for the first time in python and it could be used to speed up a lot of processing (even in the next stages of the task, tried it but didn’t put in the source code as it was a rough implementation). But now the next challenge was clustering vectors with 500 dimensions.

Dimensionality reduction of text vectors and clustering:

Dimensionality reduction was very important before clustering. So I decided to use t-Distributed Stochastic Neighbour Embedding (t-SNE). But t-SNE has a problem with scaled up datasets and would take too much time. So as a solution I first reduced 500 vectors to 10 using PCA and then used t-SNE to further reduce it 3 vectors. Both are implemented in sk-learn. Both these processes happen on a single thread. So I did try to implement multithreading here too but only succeeded in doing it for PCA. Doing it for t-SNE would take more testing. Next I used these 3 vectors for clustering using k-means. I tried a lot of variations and found no of cluster =~ (len dataset) / 20 works best. Hence I got clusters of roughly 20 products each.

Using images for reducing clusters:

Now I had clusters of similar products but before moving to compare the hard equals columns, I decided to use images to reduce the size of clusters. I had some experience in this and I know that embeddings are the best way to compare two images (also used for face recognition). So I decide to use the very popular VGG16 model implemented in Keras-Tensorflow. The last layer of this model is a 1000 neuron output layer. I removed it and put a 64 neuron layer in its place, thus giving me 64 dimensional embeddings of an image. Though this method works even without training the model, using the trained weights of this model, would give better accuracy. After getting embeddings for all images in cluster, the Euclidean distance between each is calculated and if it is below a threshold (which means more similar), the 2 images are grouped together and passed for further comparison. This method is somewhat time intensive and after testing I found I'm getting same results even after bypassing this process and not using images at all. Still it has been left in in the source code in hopes of a better way to implement it.

Using some features for hard comparison:

After all the above process I got lists of 2 similar products. At this point I decided I can be 50% sure that they are duplicate. But to actually check if they are duplicate, we need to have some 'hard' comparison which absolutely has to be same. And since we have to check 5 lists for 2 products this part is not that computationally intensive. In the last step there is a simple '==' check for size, colour, brand name and seller, along with a ±10% check on MRP. Each true comparison adds +10% (completely arbitrary) to the similar products and if they reach above 90% they are declared as duplicate and saved.

Observations and results:

Even though I tried my best, I am still not satisfied with the whole pipeline. I didn't get a chance to completely optimise it due to the time crunch. I was basically learning and implementing things at the same time. Some of the problems I could fix given more time are:

- use brand name instead of title for clustering
- use a better way to handle images
- implement multithreading in t-sne

Some tine durations to be noted are:

- time to train doc2vec (j=4) : 10hours
- time to infer 3.5L word-vectors (j=4): 7 hours
- time for pca+kmeans on 3.5L objects (j=4) : 3 hours
- time for pca+tsne on 3.5L objects : 6 hours

I am not 100% confident that I could detect all duplicates and even that the ones detected are actually duplicate. But I believe I can do better with time and experience.

I had fun doing this task and learnt a lot in the process. I hope it lives up to your expectations.