Flipkart Product Classification

This report consists of intuitions, decisions and the results related to the problem of Flipkart Product Classification. It was undertaken as a part of the application procedure for MIDAS Lab Research Internship process.

Initial Data Analysis

This was the first step taken before thinking about how to solve the task. Firstly, all the relevant feature columns, related to the task are extracted and others are ignored. For our purpose, we take the columns “Description”, “Product Category Tree” and “Product name”. Out of which “Description” remains the main feature. There was some amount of thought on the using the product image as a feature, since classifying the image onto a basic item category would give some extra text information along with the description, but it was ignored as applying a CNN classifier on it seemed to be time consuming and would distract us from the main task.

While going over all the items, there were some of them which had product names as their product category. Since the number of such items occurring in the dataset is quite large to modify by ourselves, we simply ignored these items.

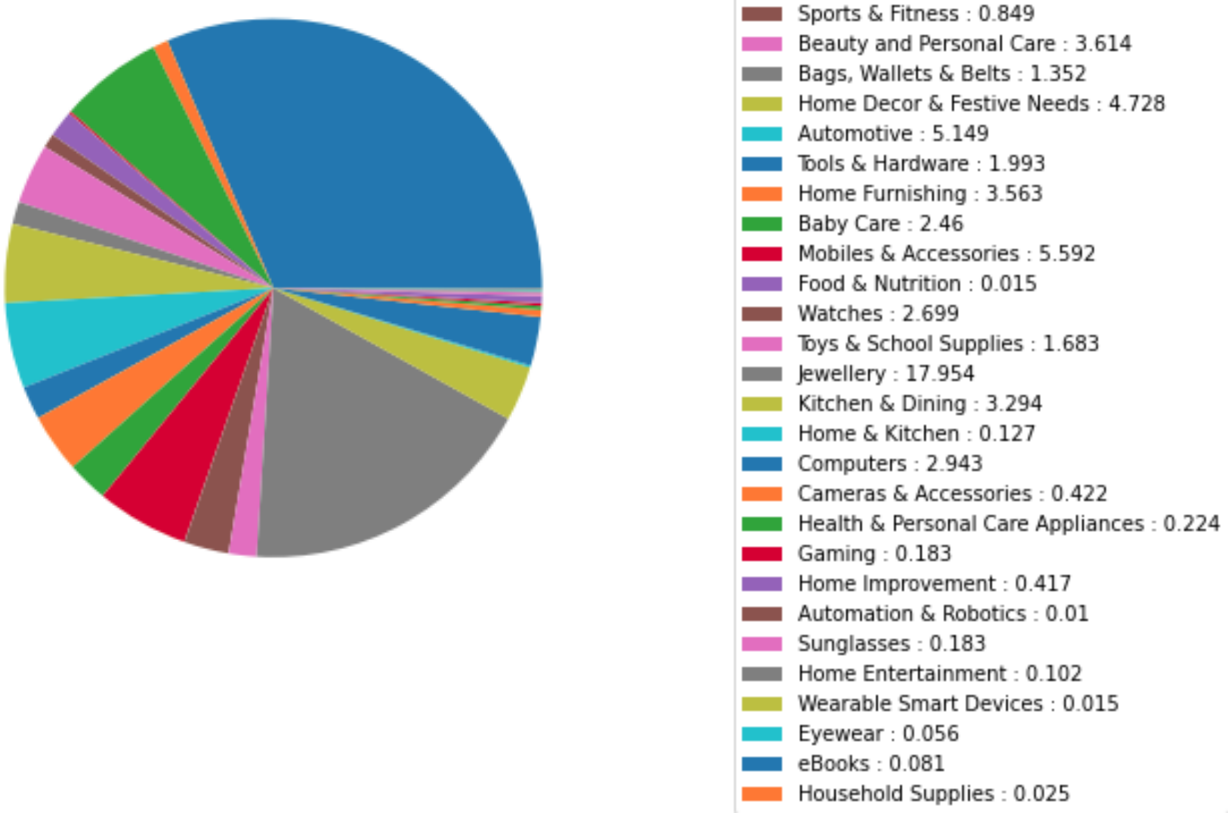
After collecting all the items, we observed the number of item categories and their frequency. In order to illustrate this, Fig1 would be helpful for you.

From this we get to know, that the class distribution is severely imbalance, with a handful of classes occupying a majority of the pie chart while the rest are severely infrequent. This distribution inequality makes us conscious, and due to this two approaches were being thought:

1) Data augmentation techniques: For the minority classes, some artificial examples could be generated to make up the gap. This approach was a bit tricky to perform in text space since there were very few methods as per my knowledge which could lead to meaningful data augmentation. Furthermore, Data augmentation would lead to increased training time, hence this approach was ignored
2) Cost-Sensitive training: In the presence of imbalanced classes, the models could be trained in such a way that they are aware of such differences so that misclassification of a minority class would be penalized with a greater cost than the other classes. Fortunately, the types of models chosen for this task all had this feature. Hence we went ahead with these.

We didn’t deal with the opportunity of ignoring any category but instead tried to merge similar categories into a single category in certain cases. For eg. Sunglasses and Eyewear, Home & Kitchen with Kitchen & Dining etc.

Fig1: Category Distribution Pie Chart



Preprocessing

All the descriptions are preprocessed to remove numbers, punctuation marks and common stopwords. The remaining words are then tokenized, converted to lowercase and lemmatized to their basic form. The resultant description feature is then vectorized before feeding to the model.

Model Metric

Since we have discovered that the dataset faces severe inequality in terms of its class distribution, relying on intuitive metrics like accuracy would be a folly. In the case of multiclass classification problems, we rely on precision, recall and f1-score metrics. Furthermore, during the model implementation, we are using K fold split on the dataset in order to check on each fold how the model is performing and then we get the average of all the metrics over the folds as our final metric values

Models Used

Since this task roughly belongs to the text classification domain, relying on classical Machine Learning Models would be our best bet. There was some thought given for Deep Learning models, but they seemed to be overwhelming for this task.

The selection of models used for this task include MultinomialNB, Support Vector Classifier, Random Forests and XGBoost.

Intuition: MultinomialNB and SVM are well used in text classification tasks hence these are being used.For Random Forests and XGBoost, these methods involve an ensemble of decision trees which could be used to mitigate the imbalance present in the dataset.

As Random Forests and SVM already have support for balanced class weights, for XGBoost, we need to feed the class weights of each training sample while fitting the model. For the Naive Bayes case, there is an alternate model called Categorical Naive Bayes which is more suited for imbalanced classes, for which class weights are shared as well.

To convert the description features, count vectorizer and TF-IDF vectorizer are used.

Result

Model	Mean Precision	Mean Recall	Mean F1 Score
SVM_TfidfVectorizer	0.905	0.872	0.880
SVM_CountVectorizer	0.855	0.831	0.835
Random_Forest_CountVectorizer	0.894	0.825	0.849
Random_Forest_Tfidf	0.888	0.795	0.825
XGBoost_Count	0.894	0.826	0.848
XGBoost_Tfidf	0.880	0.830	0.847
Categorical_NB_Count	0.774	0.624	0.625
Categorical_NB_Tfidf	0.718	0.601	0.591
MultinomialNB_Count	0.825	0.677	0.717

MultinomialNB_Tfidf	0.556	0.398	0.429

- We observe from the given table that both the models of Naive Bayes perform poorly in the case of both Tf Idf and Count vectorizers as compared to others, which was a bit surprising considering the general utility of Naive Bayes in several text classification tasks like spam classification.
- Apart from SVM, for all other models, converting description features to countvectorizer seems to be more fruitful than TF-IDF vectors
- SVM with TF-IDF vectors gives the best possible result for the task followed by the Random Forest and XGBoost algorithm.

References

1)https://stackoverflow.com/questions/45811201/how-to-set-weights-in-multi-class-classification-in-xgboost-for-imbalanced-data
2) https://scikit-learn.org/stable/modules/classes.html
3)https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane_unbalanced.html
4) https://xgboost.readthedocs.io/en/latest/python/python_api.html