

# Introduction

This project explores my approach to predicting the helpfulness of Amazon product reviews. The final classifier is built to classify products in the Clothing Shoes and Jewelry category with 88% accuracy and 0.37 f1 score. Features were extracted from review text and fed into both a Naive Bayes and a feedforward network model, complete with hyperparameter tuning. To ensure the pipeline can be executed on a single machine in a reasonable time frame, several techniques were employed, including processing the data in chunks, taking a random sample and choosing model architectures that have been shown to be efficient and effective.

Tools used: [pandas](#), [sklearn](#), [spacy](#), [transformers](#), [PyTorch](#) and [mlflow](#).

## Dataset

The [Amazon reviews dataset](#) is broken out by product categories. Given limited time and resources, I focused on the Clothing Shoes and Jewelry category, which is one of the biggest categories by review count. I chose to focus on one single category instead of sampling across different product categories, in case there are systematic differences in the contents of reviews and what constitutes a helpful review across the different product categories. Additionally, I chose the 5-core version of the dataset, which is limited to products and users with at least 5 reviews, to reduce noisiness of the data and to try to exclude more recent reviews that had less chance to be voted on by readers.

## Process

### Label

The dataset does not provide the number of unhelpful votes on reviews (which seems to have been available in older versions) and number of views, which makes it difficult to normalize the helpfulness of a review by how many people have seen it and age of the review. Seeing that most reviews do not receive any votes, I chose to structure the problem as a binary classification task, with any review that has received at least one helpfulness vote counting as "helpful", and any review without any votes as "unhelpful".

### Preprocessing

Before the data can be fed into a model, some preprocessing was required on the data. Firstly, the JSON review objects needed to be collated into tabular format for easier data munging. Next, the non-English reviews and blank reviews were filtered out (they represent a small percentage of the dataset), since a lot of model features will be based on the choice of words from the reviews.

To handle the large volume of reviews, data was read in, processed, and with results written out in chunks where possible. To speed up the model training process, a round of subsampling was applied to the dataset to keep 35% of the original data volume of ~11M records (25% for training, 5% for validation, 5% for testing).

Since there is a fairly high level of class imbalance in the dataset (~90%+ of reviews never received any votes for being helpful), and the fact that some classifiers work better when the classes are more balanced, downsampling was applied to the majority class of the training set to balance the two classes (an alternative approach would be to assign a higher weight to observations from the rarer class in the loss function).

### Features

Two methods are applied to turn the review text (summary and review fields combined) into vector representations: (1) [TF-IDF](#), and (2) word embeddings from a pretrained language model (i.e. [DistilBERT](#)). The first approach is a more traditional approach to representing text as a bag of words, and obtaining a measure of token frequency downweighted by how common the term is across the training corpus. It is a simple yet often effective technique for modeling textual data, but does benefit from more data cleaning first, including applying lemmatization.

BERT is a bidirectional transformer-based model pretrained on a large and diverse set of text that has shown to be very effective for downstream NLP tasks. DistilBERT is a more lightweight model (66M parameters) trained to learn the behavior of the BERT model (110M parameters). I used DistilBERT because it is less resource-intensive to work with while preserving most of the performance of the full BERT model. The implementation I used came from the [transformers](#) library, which also provides the WordPiece tokenizer for processing text.

I also experimented with engineering features from other fields in the dataset, such as inclusion of images, history of prior reviews by the same user, and popularity of the product, but did not get a chance to test them out due to time constraints.

## Models

For this project, I tried out two algorithms for the classifier: (1) [Naive Bayes](#), and (2) [feedforward neural network](#). Naive Bayes is a classic technique based on Bayes theorem for text classification that is efficient and the [sklearn](#) implementation supports [incremental learning](#), which can be helpful when the size of the dataset is too large to fit into memory. The architecture of technique (2) is shown in the appendix. It was inspired by this [paper](#), which demonstrated that a simple shallow feedforward network can be effective for text classification tasks.

Hyperparameter tuning was applied to the feedforward network to test out the effects of learning rate and hidden dimension size of the layers. More extensive tuning, such as on the dropout rate, could help identify a more optimal configuration.

Results of models (on the held-out test set) are as follows:

Model	Accuracy	AUC	F1
Naive Bayes (tf-idf vectors)	0.876	0.770	0.308
Feedforward Neural Network (DistilBERT embeddings)	0.879	0.814	0.371

As shown, the neural network performs slightly better, but is a lot slower to train, so in practice, the choice between the two models is not too clear.

## Conclusion

To conclude, the classifier models on Amazon reviews in the Clothing Shoes and Jewelry category so far are promising, and there are some ideas for next steps. For example, inclusion of more features aside from the text contents and exploring the use of the attention mechanism so the model can learn to focus more on certain words, such as fine-tuning on a BERT or DistilBERT model (I tried this out but training was taking too long and requires GPUs). If time allows, I would also want to test how well the existing models work on reviews from other categories, and expand the technique so it works on any given Amazon review.

# Appendix

Model diagram:

