# Fever Code Challenge

Víctor Martínez Palomares

June 23, 2024

## 1 Exploratory data analysis

As part of a basic exploratory data analysis, we analysed the distribution of the target variable. We can see in Figure 1 that the dataset is balanced for the majority of the categories. However, there are some categories with a low number of samples, which could lead to a poor performance of the model in these categories. We see low number of samples for the categories ”Health & Personal Care”, ”Home Audio & Theatre”, ”Musical Instruments”, ”Camera & Photo”, ”Video Games”. We will evaluate the model with a confusion matrix to see how the model performs in these categories.

A suggested approach to balance the dataset would be to apply data augmentation techniques such as back-translation or word substitution with synonyms for the categories with low sample sizes.

Another important aspect is that we see that there is no data for the category ”Buy a Kindle”. We decided to remove this category from the target variables as it would not be possible to train a model for it.

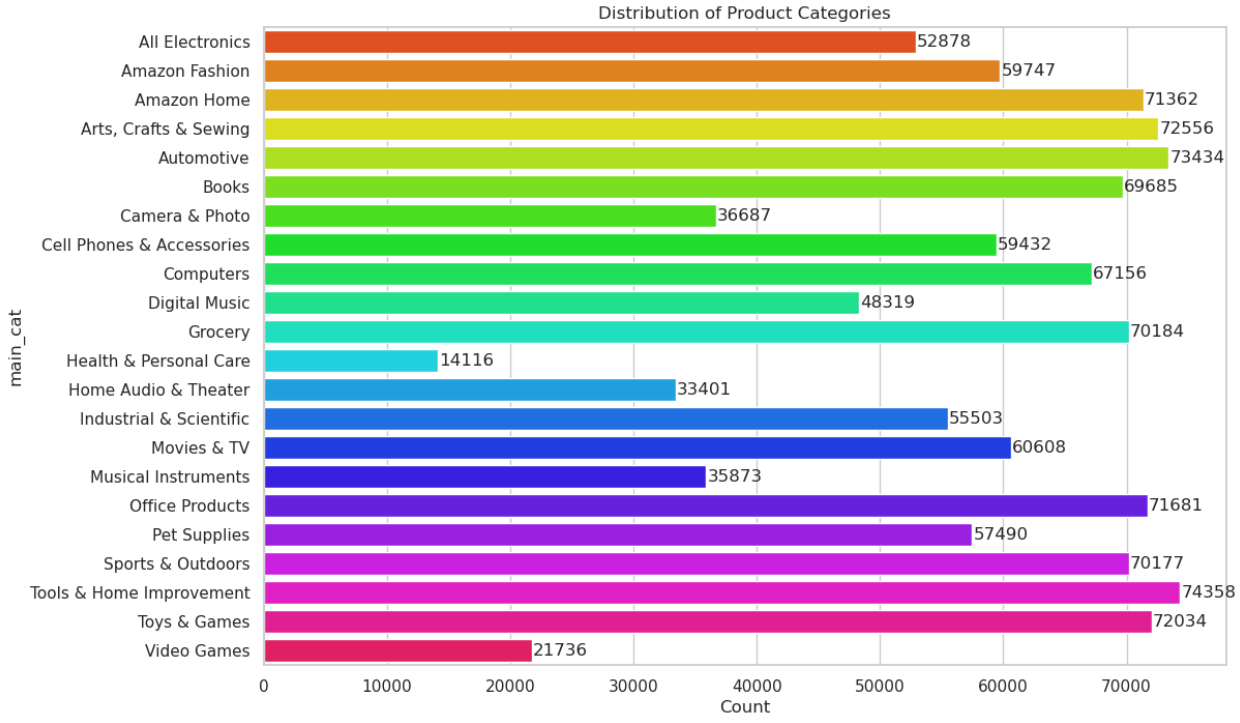


Figure 1: Distribution of the target variable in the dataset

## 1.1 Feature Engineering

For the selection of features, we kept the features containing descriptive text. Therefore, the input for the BERT model is the main text fields combined. We kept "asin", "title, "description", "feature" and "brand" features.

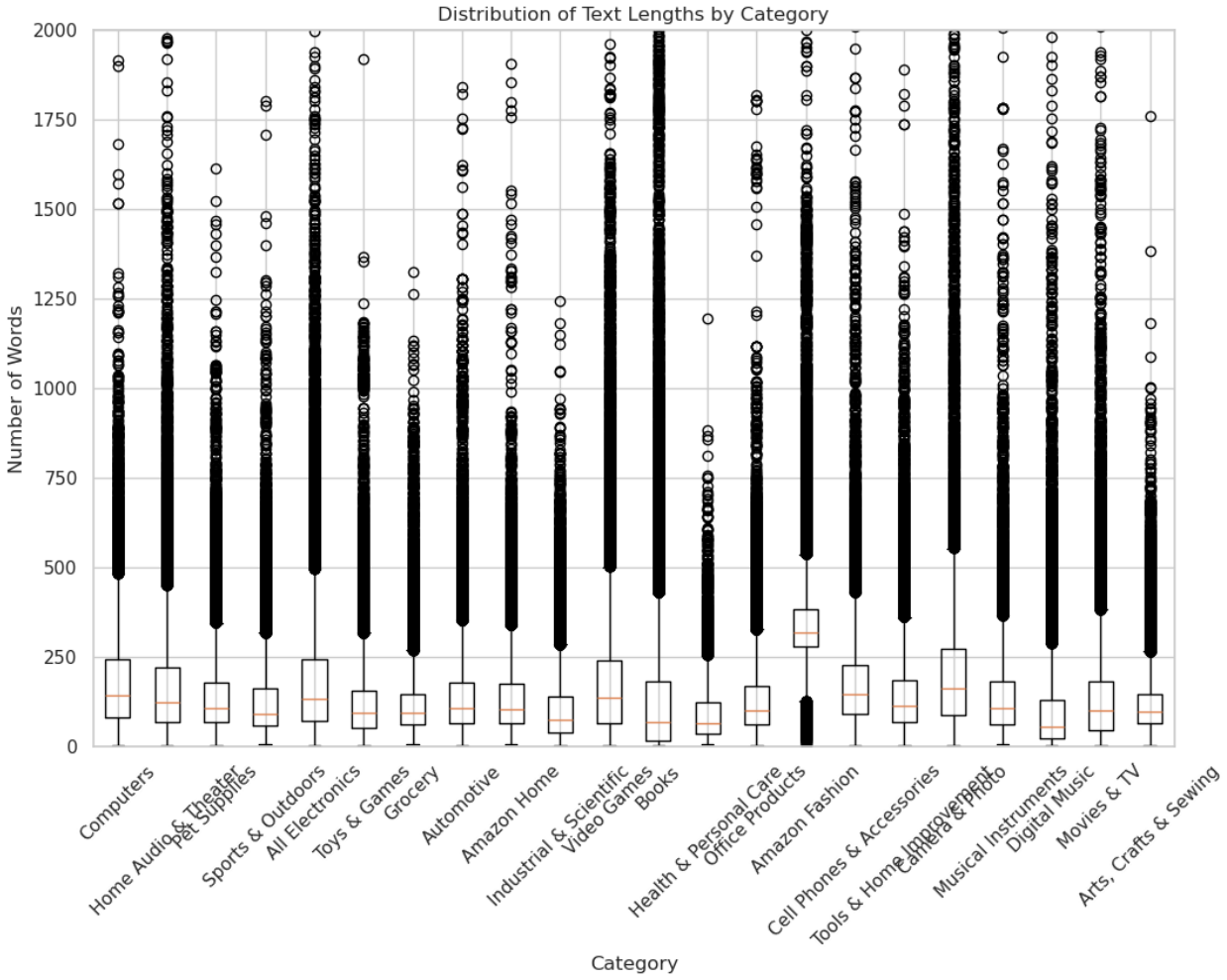


Figure 2: Text length distribution across products by target variable

In Figure 2 we observe that the distribution of the text length for all categories is below or slightly above 250 words. This is important because BERT tokenizer truncates the sentences to a maximum length of 512. To prevent any loss of relevant information for the model, the text is combined with the more relevant features at the beginning of the sentence, so the tokenizer truncates the least relevant part of the text.

An alternative would be to generate a summary from the combined text of maximum 512 words using a summarisation model such as *bert-extractive-summarizer* or *gensim summarizer*. This would allow us to keep the most relevant information for the model.

Further improvements in data pre-processing could help the model by providing more relevant text. The text was quite dirty and sometimes a lot of it was whole HTML syntax without any information. We suggest deleting all the HTML tags as they do not provide any information relevant to the classification task.

Conversely, it is important to note that BERT models tend to perform better when stop words and other elements that can give meaning to the text are retained. Some literature has compared heavy cleaning with lemmatisation and other techniques with simpler cleaning, and the former have performed worse than the latter.

## 2 Design and architecture

Given the heavy weight of text in the features, we chose an NLP model for text classification. We chose BERT, which is a transformer-based model with bidirectional attention that allows it to capture the context of a word in a sentence. It has shown state-of-the-art performance in many NLP tasks. BERT is a masked language model that can be easily configured for classification tasks by adding a classification layer on top of the pre-trained model. Due to computational limitations, we chose a BERT Tiny model, which is the smallest version of BERT available. Further training could be done with a larger model to improve the performance of the model. We fine-tuned a BERT Tiny model from the Hugging Face library on the dataset provided for the classification task.

The project is structured in a FTI (feature, train, inference) architecture, so we have 3 different modules completely decoupled: *data_pipeline*, *training* and *inference*. This is an easy to scale open architecture, meaning that you can implement each part with different technologies.

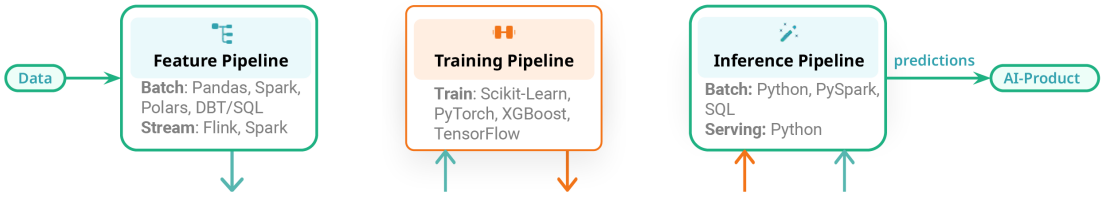


Figure 3: FTI (feature, train, inference) architecture (Source: Hopsworks.ai, 2023)

Other alternatives we considered have been to use Graph Neural Networks to exploit the relationships between "also_buy" and "also_view", but the problem is that in some cases these features could be empty. An ambitious solution to this would be to use a hybrid model that combines a BERT model with a Graph Neural Network to exploit both the text features and the relationships between products. This would require a more complex architecture and more computational resources, but it could provide better results.

## 3 Performance

Table 1: Performance analysis. Metrics of the best BERT model fine-tuning result

| Model | Test accuracy | Test F1 Score | Training accuracy | Validation Accuracy |
|---|---|---|---|---|
| BERT Tiny | 86.92% | 86.11% | 86.92% | 86.3% |

Table 2: Hyper-parameters used for fine-tuning the BERT Tiny model

| Parameter | Value |
|---|---|
| BERT Model | prajjwal1/bert-tiny |
| Batch size | 128 |
| Learning rate | 4e-05 |
| Number of epochs | 5 |
| Optimizer | AdamW |
| Trainable layers | All |

After fine-tuning the model with the parameters in Table 2, we obtained the results in Table 1. The model achieved an accuracy of 86.92% on the test set and an F1 score of 86.11%. The validation accuracy was

86.3% and the training accuracy was 86.92% in the best epoch, so we did not observe any overfitting (refer to Figure 6). The training and validation curve does not show any asymptotic behaviour either, so we could try to train the model for more epochs to see if the generalisation improves further.

By looking at the confusion matrix (see Figure 7) we can see how the model performs for each specific category. We can see that the model performs fairly well in most categories, but we observe a lower performance in the categories "All Electronics" and "Health & Personal Care".

Regarding the category "Health & Personal Care" this could be easily explained by the low number of samples in this category, therefore we could try to balance the dataset by applying data augmentation techniques to this category.

For the "All Electronics" category, the number of samples is similar to the other categories, so data augmentation migth not effectively address the performance issue. The lower performance in this category could be attributed to its generic nature, a category containing a wide range of products, which may cause higher confusion with other categories. Further analysis of the distribution of text in the features of this category might be helpful, for example techniques such as bag of words, vocabulary size, TF-IDF or word embeddings could be used to analyse the distribution of words in the text contained in this category. This analysis could reveal specific features that are causing the confusion and help refining the data_pipeline process.

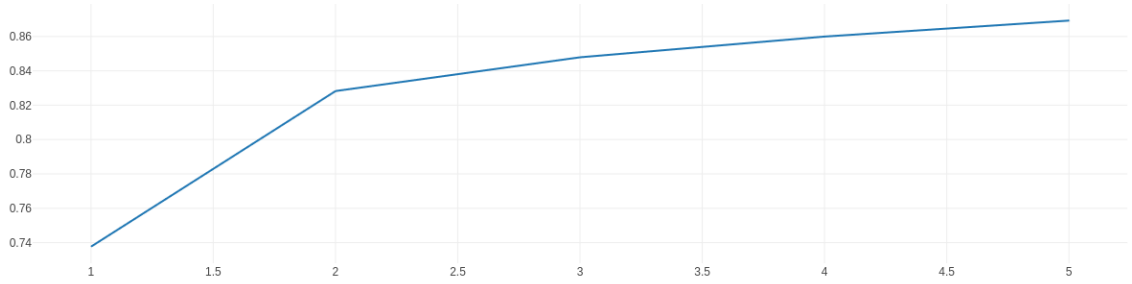


Figure 4: Training accuracy per epoch

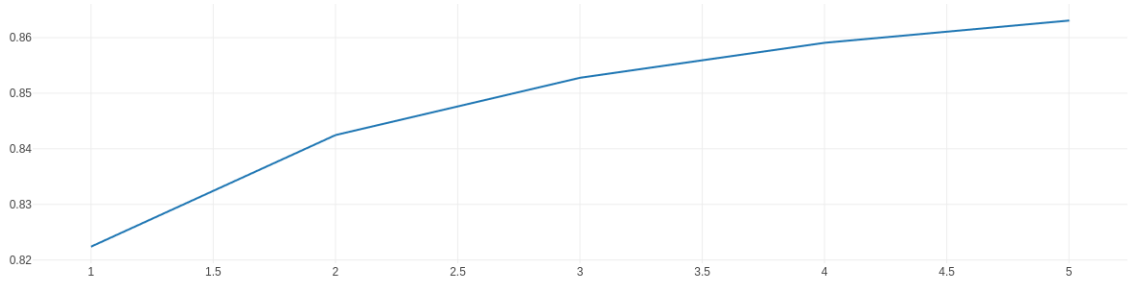


Figure 5: Validation accuracy per epoch

Figure 6: Accuracy per epoch during training and validation phases.

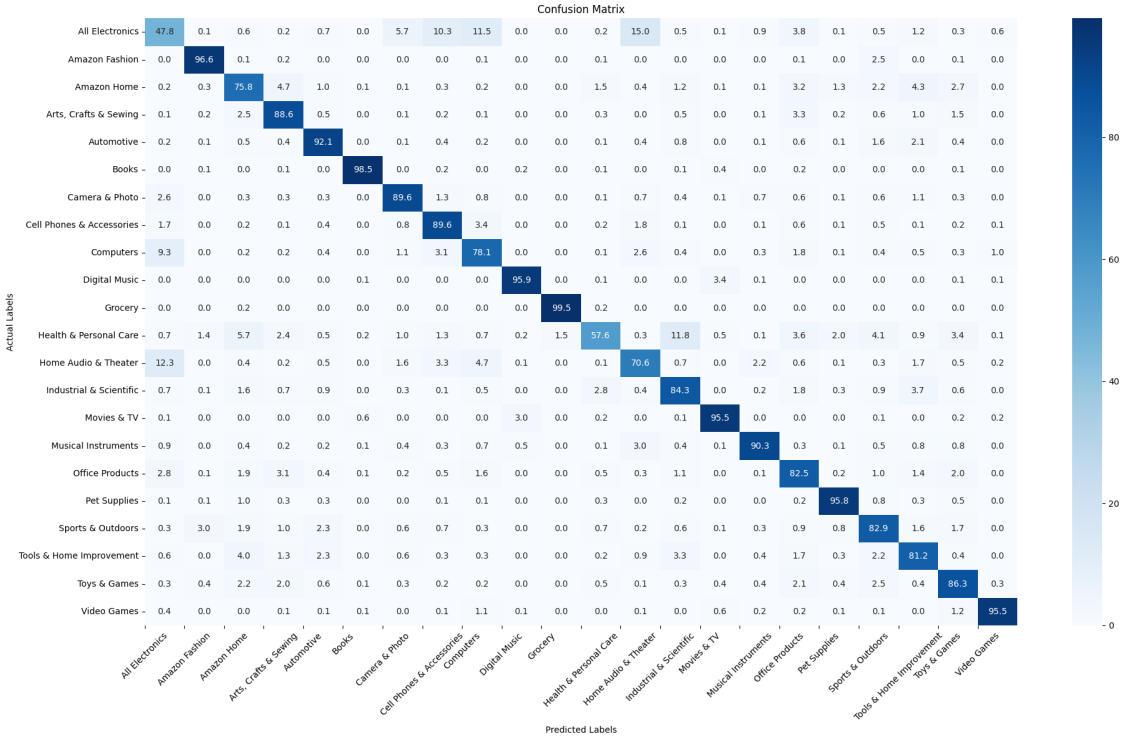


Figure 7: Confusion matrix on test dataset

# 4 Questions

## 4.1 What would you change in your solution if you needed to predict all the categories?

The classification task would change from a multi-class classification task to a **multi-label classification** task, where the model would have to predict multiple categories for each product. The following changes would be required:

- The **label encoding** would need to be changed to a multi-label encoding where the target variable is represented as a multi-hot encoded vector of the size of the total number of categories signaling in a binary way the presence of a category in the product. The Scikit Learn library provides a MultiLabelBinarizer class that can be used to encode the target variable in this way from the list of possible categories.
- In order to compute the predictions, the **activation function** of the output layer of the BERT model would need to be changed from softmax to sigmoid so that the probability of each category is output independently. A threshold would also need to be set to determine the presence of a category in the product. A common practice is to set the threshold to 0.5 for a positive prediction.
- The **loss function** would need to be changed from cross-entropy loss to binary cross-entropy loss to calculate the loss for each category independently and sum them up.
- The test accuracy metric would not be representative of the overall performance of the model, as the model could correctly predict some categories and not others for a given sample. Therefore, the **evaluation metrics** would need to be changed to F1 score, precision and recall as they consider the performance of the model for each category independently.

## 4.2 How would you deploy this API on the cloud?

1. **Containerisation:** Use Docker to containerise the model and the API modules. This allows to create a portable and reproducible environment for the deployment.
2. **Choosing a Cloud Provider:** Select a cloud provider that best fits the project requirements. Options include Amazon Web Services, Google Cloud Platform, or Microsoft Azure.
3. **Container Orchestration:** Deploy the containers using orchestration tool such as Kubernetes, which allows for automated deployment and management of containers, leveraging auto-scaling and load balancing features. We create a Kubernetes cluster on the cloud provider and deploy the containers to the cluster. We rely on either Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE) or Azure Kubernetes Service (AKS) to manage the Kubernetes cluster.
4. **Load Balancing:** Implement a load balancing solution using an Ingress Controller like NGINX or HAProxy, integrated with Kubernetes. This setup will distribute incoming API requests across multiple instances of the application deployed in different containers.
5. **Decoupling Model Serving and API:** Decouple the model serving component from the API to allow each to scale independently based on demand. For model serving we could use tools like Torchserve.
6. **Monitoring and Logging:** Set up monitoring and logging tools to check the performance of the model and the API in real time. We could use the ELK stack (Elasticsearch, Logstash and Kibana).
7. **Continuous Integration and Deployment (CI/CD):** Implement CI/CD pipelines using tools such as GitHub Actions, GitLab CI/CD, or Jenkins to automate the process of integrating code changes, running tests and deployment processes to make sure there is a continuous and reliable deployment of the API.

Other considerations, such as authentication and authorisation mechanisms for the API, should also be taken into account when deploying the API in the cloud.

## 4.3 If this model was deployed to categorise products without any supervision which metrics would you check to detect data drifting? When would you need to retrain?

To answer this question, we must first distinguish between the different types of data drift that can have a negative impact on model performance. The main types of data drift that we will cover are:

- **Concept drift:** The relationship between the input features and the target variable changes. This can lead to a decrease in model performance. In our product classification task, concept drift could occur if the relationship between some of the features, such as ”brand” or ”title” and its corresponding category, changes over time. For example, a brand that sells a new type of product from a category that was not previously sold in the training dataset.
- **Feature Drift (Covariate Drift):** The distribution of input features changes over time. It does not necessarily imply a drift between the input features and the target variable. In our product classification task, feature drift could occur if some features change in style or format over time. For example, a change in the way a product from certain categories defines its features.

In the following points we explain some metrics that could be used to detect each type of data drift. If the evaluation of any of theses metrics exceeds certain security thresholds, we could trigger automated processes to run hypothesis tests to assess whether the observed changes are statistically significant, identify the nature of the data drift and retrain the model.

- **Concept drift**: To detect concept drift, we could monitor metrics such as model accuracy, F1 score, precision and recall to detect potential changes in model performance. If the performance of the model drops significantly over time, this may indicate that there is concept drift. The confusion matrix could also be used to detect concept drift in specific categories.

Alternatively, if we do not have access to the true labels for the new data to monitor the evolution of model performance, we could use unsupervised techniques to detect concept drift. For example, we could use the reconstruction loss of an autoencoder trained on the features of the training data as a baseline to compare with the reconstruction loss of the new data. If the reconstruction loss is significantly higher in the new data, this may indicate that the relationship between the input features and the target variable has changed.

- **Feature Drift (Covariate Drift):** To detect feature drift, we could compute the embeddings of the features from the new data and compare them with the embeddings from the training data to see if they were generated from the same distribution. We could use the Maximum Mean Discrepancy (MMD) on the BERT embeddings of the features. If the p-value (e.g. 0.05) is below a certain threshold, we reject the null hypothesis and conclude that there is indeed a feature shift and retrain the model.

  In addition to retraining with new data, analysing the features that contribute to feature drift and modifying or eliminating them and keeping more robust features, could help to leverage retraining in old data by reducing the impact of the data drift.