Technical Document: Customer Churn Prediction and Analysis

Abstract:

Briefly summarizes the problem, methods used, and key findings.

Table of Contents:

Introduction

Problem Statement

Business and Research Objectives

Dataset Overview

Data Collection and Preprocessing

Data Sources and Collection Process

Data Cleaning and Handling Missing Values

Feature Engineering and Selection

**3. Data Understanding**

The Kaggle Amazon US Customer Reviews and Amazon Product Review (Spam and Non-Spam) Datasets serve as a valuable repository of customer opinions and experiences regarding products available on Amazon.com. With over a hundred million reviews and over twenty-seven million reviews respectively, contributed by Amazon customers over two decades, these datasets have become an essential resource for academic researchers in fields such as Natural Language Processing (NLP), Information Retrieval (IR), and Machine Learning (ML). The datasets were curated to provide insights into customer evaluations, regional variations in product perception, potential promotional intent, or review bias and to classify reviews as spam or non-spam(ham) specifically on the latter dataset. Amazon US Customer Reviews dataset boasts 15 attributes as seen in **Table X** and the Amazon Product Review (Spam and Non-Spam) Dataset has 12 attributes as seen in **Table Y**. Datasets have more than enough attributes to get insights that will help me fulfill my research goals and business goals at a high level. The datasets are in a tab delimited and json formats, respectively.

**Table X. List of attributes related to** **Amazon US Customer Reviews Dataset**

|  |  |  |
| --- | --- | --- |
| Attribute Name | Variable Type | Description |
| Marketplace | Nominal | 2 letter country code of the marketplace where the review was written |
| customer\_id | Nominal | Random identifier that can be used to aggregate reviews written by a single author. |
| review\_id | Nominal | The unique ID of the review. |
| product\_id | Nominal | The unique Product ID the review pertains to. In the multilingual dataset the reviews for the same product in different countries can be grouped by the same product\_id. |
| product\_parent | Nominal | Random identifier that can be used to aggregate reviews for the same product. |
| product\_title | Nominal | Title of the product. |
| product\_category | Nominal | Broad product category that can be used to group reviews |
| star\_rating | Ordinal | The 1-5 star rating of the review. |
| helpful\_votes | Numeric | Number of helpful votes, meaning people who found the reviews helpful. |
| total\_votes | Numeric | Number of total votes the review received. |
| Vine | Nominal | Review was written as part of the Vine program which enables a select group of Amazon customers to post opinions about new and pre-release items to help their fellow customers to make educated purchasing decisions |
| verified\_purchase | Nominal | The review is on a verified purchase. |
| review\_headline | Nominal | The title of the review. |
| review\_body | Nominal | The review text. |
| review\_date | Temporal | The date the review was written. |

**Table Y. List of attributes related to Amazon Product Review (Spam and Non-Spam)**

|  |  |  |
| --- | --- | --- |
| Attribute Name | Variable Type | Description |
| \_id | Nominal | The unique ID of the review. |
| reviewerID | Nominal | Random identifier that can be used to aggregate reviews written by a single author. |
| Asin | Nominal | The unique Product ID the review pertains to. In the multilingual dataset the reviews for the same product in different countries can be grouped by the same product\_id. |
| reviewerName | Nominal | The name of the reviewer |
| Helpful | Nominal | Number of helpful votes, meaning people who found the reviews helpful. |
| reviewText | Nominal | The review text. |
| Overall | Nominal | The 1-5 star rating of the review. |
| Summary | Ordinal | The title of the review. |
| unixReviewTime | Numeric | Number of helpful votes, meaning people who found the reviews helpful. |
| reviewTime | Numeric | Number of total votes the review received. |
| Category | Nominal | Broad product category that can be used to group reviews |
| Class | Nominal | A categorical label indicating whether the review is classified as spam or non-spam |

**3.1. Data Licensing and Usage**

Both datasets are derivatives of the Amazon Customer Reviews Library and are subject to Amazon’s Condition of use. Users are granted a limited, non-exclusive, non-transferable, non-sublicensable, revocable license to access and use the dataset for academic research purposes. Users are prohibited from reselling, republishing, or making any commercial use of the dataset or its contents. The dataset should not be used for commercial research, consultancy contracts, internships, or other commercial purposes. Additionally, users should not attempt to link or associate content in the dataset with personal information, and they should not attempt to identify the authors of the content. Violating these conditions may result in the termination of the user's license to access and use the dataset. The license reinforces the intended use of the dataset for academic research while preventing commercial exploitation and safeguarding user privacy. The spam dataset is a product of Hussain et al 2020

**3.2. Obtaining the data**

Initially, I started off on a google colab notebook and used the Kaggle API to download the Amazon US Customer Reviews dataset, henceforth will be labelled as **dataset A**, and Amazon Product Review (Spam and Non-Spam) dataset, henceforth will be labelled as **dataset B**. In hindsight, the latter dataset was all that was needed but, in my defense, I was working with two research goals which I wasn’t sure was sufficient before I added the latter dataset which had the spam or ham class needed in the classification of fake reviews, but I digress. After downloading the datasets to my google drive, I transferred them to google bucket that I had created for my compute cluster on the Google Cloud Platform. To initiate my exploration, I created a notebook and loaded the Kaggle dataset into the analysis environment, to review the structure and contents of the datasets. The preliminary examination aimed to gain a high-level understanding of the datasets’ key attributes and dimensions. This allowed me to gauge the richness of the data and set the stage for deeper analysis.

**3.3. Getting Preliminary insights**

Each attribute within the dataset was subjected to meticulous analysis to extract its significance and potential utility. Attributes were dissected to comprehend their roles and relationships within the dataset. This step aimed to lay the groundwork for subsequent analysis and modeling.

I started out by getting the shape of the datasets as seen in Figure A and Figure B.

Figure A: The number of rows and columns of the **dataset A**.



Figure B: The number of rows and columns of the **dataset B.**

I also retrieved the descriptive statistics from both datasets to identify potential issues in the data, such as missing values, unusual distributions, or extreme values thereby getting a feel of the data quality as seen in figures A and B. We noticed that there was distribution was even for all the numeric fields. However, upon assessing the fields **X, Y and Z** for dataset A I realized that there are missing values. The same goes for dataset B as it pertains to distribution with missing values at columns **X, Y and Z**.



I also used the printschema function available in pyspark to get a snapshot of the data types for each column for dataset A and B as seen in Figures A and B. Based on the visualization of the contents of the dataset in Figures A and B compared to the actual data types, there may be some need for some data conversions.

Figure A. The schema of dataset A



Figure A. The schema of dataset B



I also created a function to get a snapshot of the number of missing values in each column for dataset A and B as seen in Figures A and B, and based on the visualization, there is probably less than 10% of the datasets with null values. I think the right course of was to try to maintain as much data as possible since it would be used in the recommendation engine.

Figure A. The number of null values in each column for dataset A



Figure B. The number of null values in each column for dataset B.



**4. Data Preparation**

**4.1. Data Cleaning and Handling Missing Values**

After getting the statistics and structure of the data. I started the data cleaning for dataset A and B. To handle rows with null data, I choose to drop rows where the last 9 columns were all null, as these records wouldn't provide meaningful insights. This enhances the data's quality by eliminating irrelevant data. I filtered out rows where the **product\_category** was 2011-09-09 as this row was beyond saving and served as one of the victims of data misalignment. I casted **star\_rating** as an integer aligns with its ordinal nature, as it represented a discrete and ordered variable.

I was very methodical in the handling of date-related columns. I tried to maintain the integrity of the data by transforming cases where **review\_body** appeared as a date to **review\_date**. Converting **review\_date** to the appropriate date format consolidates temporal data and filling null dates with the most frequent date maintains data consistency. I dropped the **product\_parent** and **marketplace** columns as both don’t contribute to my analysis and marketplace specifically held limited variability as the entire dataset was based in the US region.

To supplement empty or null **review\_body**, I passed the contents of the **review\_headline** to it as while the **review\_headline** was a title of the review, it was sufficient to serve as a **review\_body** and I saw it as a summary.

**4.2. Feature Creation**

I created **season**, **month**, and **year** columns from **review\_date** as I wanted to get some temporal insights. I also generated a **sentiment\_score** column through sentiment analysis using the TextBlob library and I further constructed a **sentiment** column which served as the category being either Negative, Positive or Neutral. I also generated **abs\_sentiment\_score** column to mitigate negative value issues while creating the feature engineering pipelines and modelling. I also created **review\_text\_length** as another potential feature for the spam classification models.

**4.3. Text preprocessing**

To prepare the **review\_text** and **review\_body** fields for use during the modelling phase , a slew of text preprocessing steps was performed in the form of a pipeline. These steps included removing unicode characters, lowercase normalization, tokenization, and lemmatization through a custom transformer.

**4.4. Exploratory Data Analysis**

I performed further Exploratory data analysis on the cleaned datasets before proceeded into the modelling phase.

I start off by assessing the number of unique customers of who reviewed the products and we found that there were **X**.

Understanding customer sentiment and identifying pain points related to product quality is a critical aspect of improving customer satisfaction and reducing churn. The number of customers is essential here because it directly affects the representativeness and accuracy of the analysis. A larger number of customers provides a more comprehensive view of the diversity of opinions and experiences. With a significant sample size, the analysis can capture a wider range of product quality issues and pain points that customers face. This, in turn, allows for targeted improvements to address specific pain points, leading to enhanced customer satisfaction and reduced churn.

Building a reliable model to detect fake or manipulated reviews requires a dataset with enough genuine and manipulated reviews. The number of customers is vital in this case because it impacts the diversity and complexity of review patterns. A larger customer base provides a more extensive pool of reviews with varying writing styles, sentiments, and behaviors. This diversity is crucial for training a robust model that can accurately identify the subtle differences between genuine and fake reviews. Without a substantial number of customers contributing a wide range of reviews, the model's performance could be compromised, leading to inaccurate classifications and potential misinterpretations of review authenticity.

Analyzing the impact of review text on product recommendations requires a comparison between recommendation systems that incorporate text and those that do not. The number of customers influences the significance and validity of the findings. A larger number of customers means a more extensive and diverse set of review texts, which enhances the statistical power of the analysis. With a substantial sample size, the investigation can identify meaningful patterns and correlations between review content and the effectiveness of recommendations. Additionally, a larger customer base helps mitigate the effects of outliers and ensures that the analysis represents a wide array of preferences and behaviors. Ultimately, a robust investigation based on enough customers will provide more reliable insights into the relationship between review text and recommendation performance.

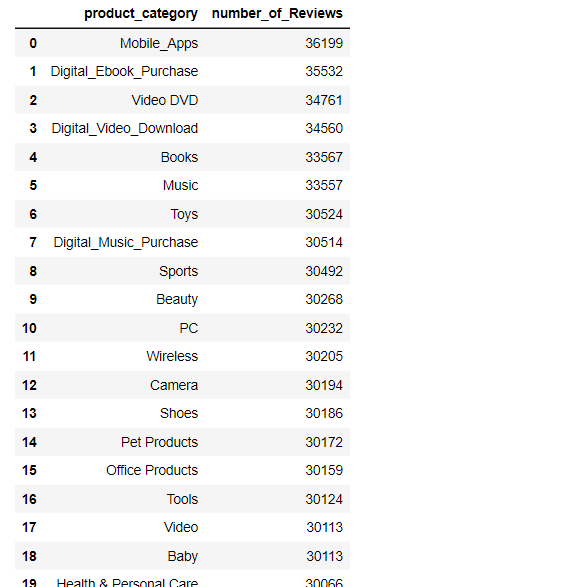
To analyze product quality issues to gauge main pain points that customer deal with when interacting with the platform:

The number of unique products is crucial for understanding the breadth of product quality issues. A larger variety of products provides a more comprehensive view of potential pain points across different categories and types of products. It allows you to identify whether quality issues are specific to certain products or if they are more widespread. With a higher number of unique products, the analysis becomes more granular, enabling you to address specific pain points for each product category and enhance overall customer satisfaction.

The number of unique products impacts the diversity of reviews in the dataset. A greater variety of products generates a wider range of review content, sentiments, and patterns. This diversity is essential for training a model that can accurately differentiate between genuine and manipulated reviews across various product types. If the number of unique products is limited, the model might not capture the complexity and nuances of different review characteristics, leading to reduced accuracy in classifying fake reviews.

The number of unique products directly influences the variability and richness of review text content. More unique products result in a larger pool of review texts with distinct features, preferences, and sentiments. This diversity is critical for assessing the impact of review text on recommendation systems. With a significant number of unique products, you can analyze whether the effect of review text on recommendations is consistent across different product categories. This helps avoid drawing conclusions based on a limited set of products and ensures that recommendations are relevant across the entire product range.

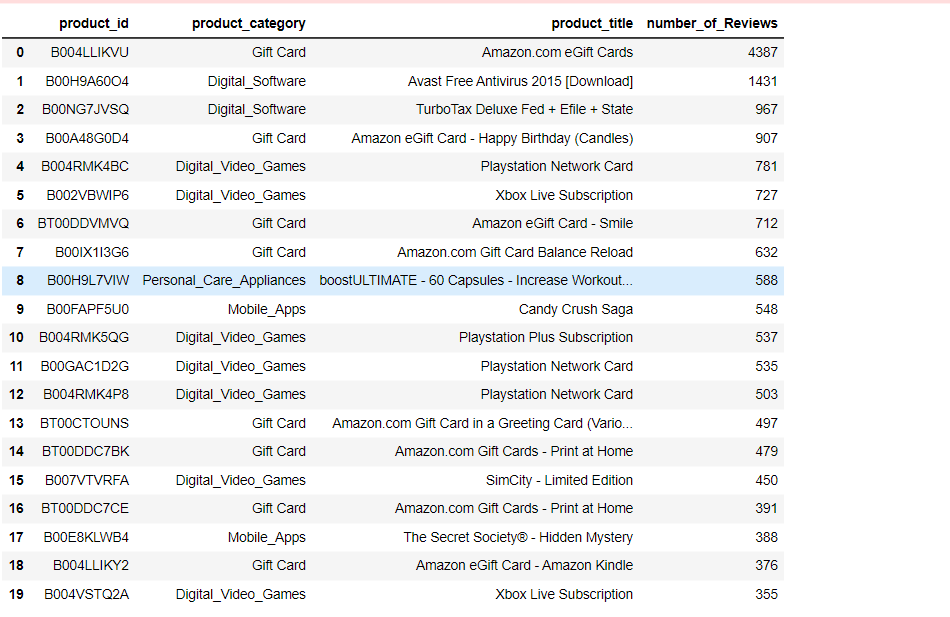
The number of unique product categories was found to be 41 with the top reviews being mostly distributed within digital related products as seen in Figure x.



The number of unique product categories is essential for understanding the scope and distribution of product quality issues. It enables you to identify whether certain categories consistently exhibit quality issues or if these issues are isolated to specific product types. With a larger number of unique product categories, you can conduct a more comprehensive analysis to pinpoint pain points within each category. This helps prioritize improvements and address issues that may vary across different types of products.

The number of unique product categories influences the diversity of review content in the dataset. A greater variety of categories results in a wider range of review patterns, sentiments, and behaviors. When training a model to classify fake reviews, having a significant number of unique product categories helps ensure that the model can distinguish between genuine and manipulated reviews across different types of products. This is particularly important because manipulation techniques may vary based on the category.

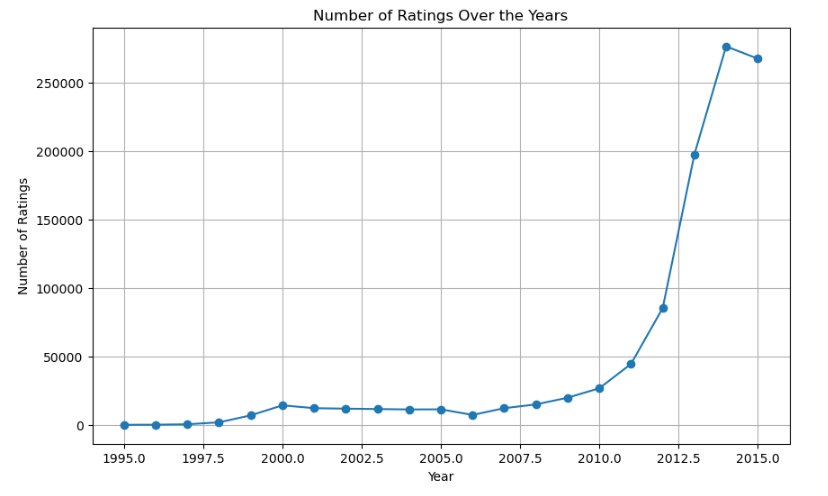
The number of unique product categories is crucial for assessing the impact of review text on recommendations. Different categories may have varying degrees of reliance on review content for recommendations. With a higher number of unique product categories, you can analyze whether the influence of review text on recommendations is consistent across different product types. This insight helps tailor recommendation strategies to specific categories and ensures that the impact of review text is effectively harnessed across the entire product spectrum.

A closer look into the product distribution of the number of reviews at the individual product level for the top 20 products by the total number of reviews validates the initial deduction as we can clearly see these digital products are barely punctuated by other categories. This makes sense as customers can access these products through their Amazon accounts or other online platforms, and the focus is on digital consumption rather than physical delivery. **citation**

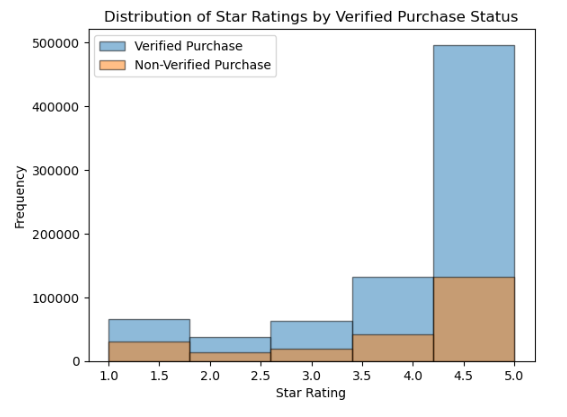
I also reviewed the distribution of star rating and found that there was high frequency of 5-star ratings with the 4-star ratings coming in at a close second and 1–3-star ratings being comparatively lower. This indicates a pattern of high customer satisfaction and positive feedback for the products or services on initial analysis.



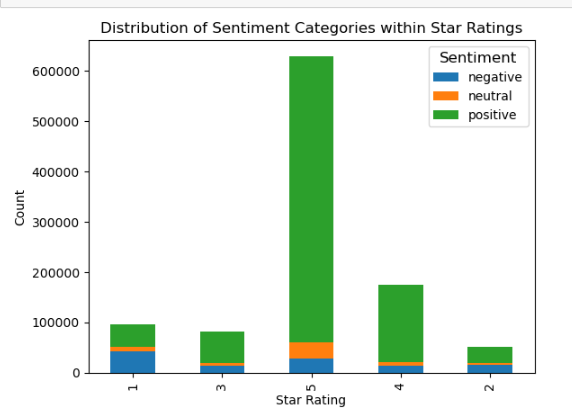
The distribution of the cumulative ratings was tracked over the 20 years and there was a steady growth of the number of ratings being given to products. We see a slight increase between 1995 and the year 2000 with a year plateau between 2000 and 2005. There is also an exponential increase in ratings which can be attributed to the garnering of more customers, due to the expansion of the amazon marketplace and general business model. **Citation.**



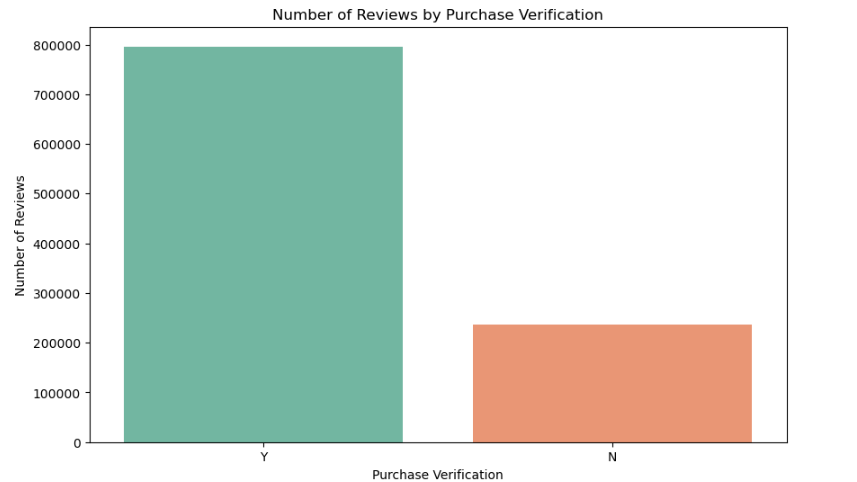
I also wanted to analyze the distribution of ratings as it pertains to whether the product was rated by a customer with verified purchase. The result was that more that 60% of the 5 star and 4-star ratings were done by customers who were denoted as having had a verified purchase.



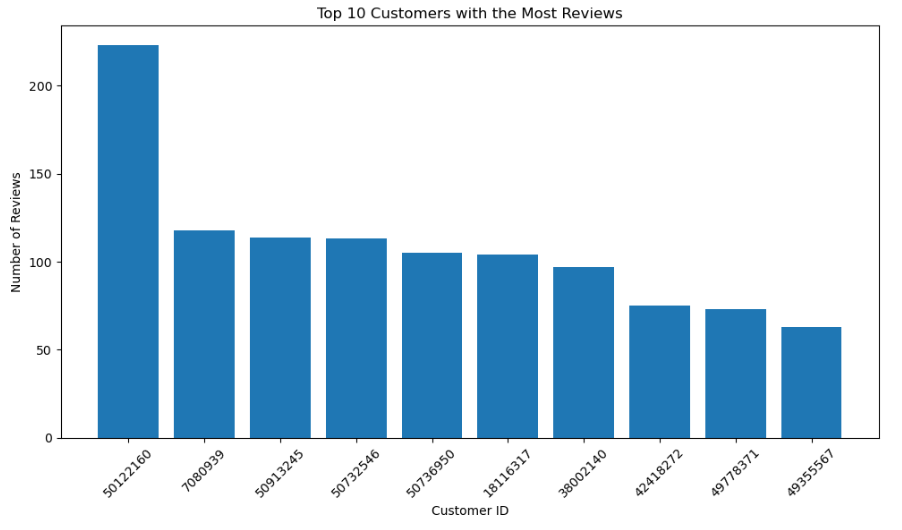
It also appears that most of the ratings are how a lot of positive, especially as it pertains to the 5 star reviews which gives credence to the fact that the distribution of ratings were primarily within 5 stars. This means there is no deviance between a high star rating and the sentiments being expressed in the reviews.



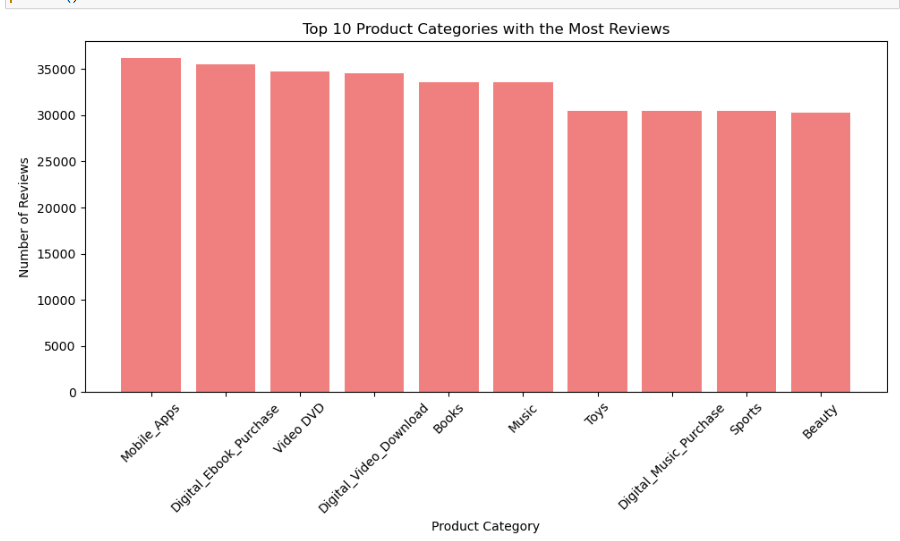
I analyzed how the reviews were distributed along the purchase verification attribute and there was a three to one ratio of reviews with verified purchases to review with unverified purchases in favor of reviews with verified purchases.



I also wanted to assess the top reviewers to check how many reviews were completed by each and I noticed that there were over 200 reviews being done by customer with id 50122160.



I also reviewed the top 10 product categories with the most reviews and found that majority of the apps were distributed along digital products.



**Feature Selection**

For the clustering algorithm-based product recommender I choose to use **lemmas** and **abs\_sentiment\_score** (sentiment intensity) fields as I the thought if could cluster the products by those features, in terms of dataset A, I could provide meaningful recommendations through Content-Based Filtering. I chose **abs\_sentiment\_score** as a feature because sentiment analysis systems have the potential to enhance various types of recommender systems, including simple, aspect-based, and end-to-end deep models (Barriere and Kembellec, 2018). I chose to get the term frequency inverse document frequency of the **review\_body** because of the use of the feature in a similar Netflix movie recommendation system proposed by Chiny et al (2014) that used TF-IDF and cosine similarity. I then grouped the dataset by the **product\_id** and took the average **abs\_sentiment\_score,** producing anew columncalled **avg\_ abs\_sentiment\_score. In the aggregation, I also** collected all the **lemmas** related to that grouping, producing a new column called **combined\_lemmas**. This was to create a more stable and representative recommender and avoid cases where a product was would have been a part of its own recommendation.

For the alternating least-square algorithm I chose to group the data in dataset A by **the customer\_id** and **product\_id** and average the **star\_rating** producing a new column **avg\_star\_rating**. I did this because as it existed the dataset had duplicate **customer\_id** and **product\_id** columns already based on the nature of the dataset. In doing so, we produce accurate representation of the customer-product interations.

In case of dataset B with the Fake Reviews Classifier, I used features like **lemmas**, **review\_text\_length** and the **abs\_sentiment\_score** column. My reasoning for choosing **review\_length** as a feature is because the average length of reviews can serve as a significant indicator of potential questionable intentions among reviewers. Notably, approximately 80% of spammers exhibit a lack of reviews exceeding 135 words in length. In contrast, over 92% of trustworthy reviewers demonstrate an average review length exceeding 200 words. This disparity underscores the potential utility of review length as a distinguishing factor between spammers and reliable reviewers (Crawford et al,2015). I selected the **abs\_sentiment\_score** (sentiment strength) as a feature because sentiment strength proves to be a more effective indicator compared to rating scores in the context of identifying spam reviews (Peng and Zhong, 2014). It is noted by Sjarif et al (2019) that incorporating the term frequency inverse document frequency of the **review\_text**, spam detection systems can effectively differentiate between relevant and generic words, optimizing the accuracy of the detection process. Moreover, TF-IDF extends its impact beyond individual documents to evaluate the broader importance of words across the entire corpus, thus enhancing the precision of spam detection.

**Feature Transformation**

For the Fake Reviews Detection features, using dataset B I created a Term Frequency Inverse document frequency of the tokenized and cleaned review text (**lemmas** field) using CountVecorizer class and IDF class. Then using the StandardScaler class, I normalized the tf-idf feature matrix, **review\_text\_length** and **abs\_sentiment\_score** features to prevent the models from favoring a specific feature. I then used Principal Component Analysis to reduce the dimensionality to avoid the curse of dimensionality.

For the clustering algorithm-based product recommender, I did something similar., I created a Term Frequency Inverse document frequency of the tokenized and cleaned review body using CountVecorizer class and IDF class. Then using the StandardScaler class, I normalized the tf-idf feature matrix and **abs\_sentiment\_score** features to prevent the models from favoring a specific feature. I then used Principal Component Analysis to reduce the dimensionality to avoid the curse of dimensionality.

**Modelling**

According to a Kontsewaya et al (2020) machine learning techniques offer the highest level of accuracy when it comes to classifying spam especially as it pertains to the six most popular: Naive Bayes, SVM, Decision tree, K-Nearest Neighbors, Logistic regression, Random Forest. I chose to use the first three algorithms due to a time constraint.

The Naive Bayes algorithm, known for its probabilistic approach, effectively classifies spam. Its "naive" designation arises from its disregard for potential interdependencies or associations among inputs, simplifying a multivariate issue into a series of univariate problems (Sinha and Singh,2020).

According to Sajedi et al (2016), the Support Vector Machine (SVM) functions as a linear classifier by identifying the hyperplane that maximizes the separation between classes. Subsequently, new instances are projected into this space, and their categorization is determined based on their position relative to the gap between classes. The classifier aims to enhance the spacing between points to establish heightened "confidence" in class distinction. Remarkably, the model demonstrates resilience to outliers.

The decision tree algorithm can be simplified as being a hierarchical structure used for making decisions or predictions in various fields.

The python MLLIB, doesn’t currently contain the K-nearest Neighbor algorithm implementation so to maximize the use of the large-scale distributed environment provided by pyspark, I opted out of using the algorithm. I could have converted the dataset to a pandas data frame to use K-Nearest Neighbor but it uses a lot of processing power during the conversion which isn’t viable.

I wanted to implement both collaborative filtering and content-based filtering to create a personalized product recommendation engine, so I used the Alternating Least Squares (ALS) matrix factorization algorithm provided by Pyspark MLLIB in the implementation of collaborative filtering and K-means and Hierarchical Clustering for the implementation of Content-Based Filtering. The primary focus of the collaborative filtering system revolves around identifying similarities between customers' preferences and items. Recommendations for new users are generated based on the preferences of similar individuals from their browsing history. Collaborative filtering involves combining items, identifying similarities through user ratings, and creating new recommendations by comparing across multiple users (Gosh et al, 2021).

Content-based filtering proposes recommendations to users that closely resemble the items they have previously selected or shown interest in (Nallamala et al 2020). Iliopoulou et al (2020) explored Content-based filtering by employing K-Means clustering methods to center on uncovering similarities within movie plots. Their initial strategy involved grouping movies using the Tf/Idf weighting scheme, assigning significance to terms within movie plots. I utilized a similar concept with the products except I went a bit further by finding the cosine similarity of the products within the same cluster and choosing the products with the highest cosine similarity as recommendations.

**Evaluation**

In this phase, I subjected the trained models to rigorous testing, leveraging key evaluation metrics based on the specific research objectives mentioned in the business objectives to ascertain the best performing one. For the fake review classification, which is a binary classification problem, I used the area under the Precision Recall Curve (areaunderPR) and area under the Receiver Operating Characteristic Curve(areaUnderRoC) which are the only metrics provided by BinaryClassificationEvaluator from Pyspark’s MLlib. To optimize the performance of each algorithm, I embarked on the crucial task of hyperparameter tuning using the TrainValidationSplit technique which is also provided by Pyspark’s MLlib. It involves splitting the data into training and validation sets, training models on different hyperparameter settings, and evaluating their performance on the validation set. I employed the ParamGridBuilder to create a list of values for regParam and maxiter parameters for the Naïve Bayes and

For fake review detection, I found that based on the list of classification models, which performed well overall, and the resulting metrics, the best performing model was the **X model** as seen in **Figure A** which had an areaUnderPR value of **X** and an areaUnderROC value of **Y**.

For the content-based recommender using the clustering-based approach with Kmeans and Hierachial Clustering, the Kmeans clustering model performed the best with a silhouette score of X as seen in **Figure A**.

The collaborative filtering model based using the alternating least squares algorithm had an excellent performance with a root mean square error(rmse) of **X**.

**Deployment**

I implemented web-based recommendation systems (Kmeans Content-Based Filtering and ALS Collaborative filtering) and Spam classifier using Flask, PostgreSQL, and PySpark. The objective was to provide users with personalized recommendations based on their preferences and allow users to predict if a review is classified as a spam or ham. Here's a coherent description of the deployment process, along with the reasons for each step taken:

**Platform and Hardware Selection**

To host the recommendation system, I chose to deploy it on a Google Cloud Platform (GCP) virtual machine (VM) instance. The instance was configured with an e2 CPU platform, offering good performance and scalability. I opted for a standard machine type with 4 vCPUs (2 cores) to ensure sufficient computational resources for running PySpark and handling web traffic.

**Access Scope and Firewall Configuration**

I granted the VM instance full access to all Cloud APIs. This allowed the application to interact with various GCP services and resources seamlessly. Additionally, I configured the firewall to permit incoming HTTP traffic, enabling users to access the web application. I also set up a specific firewall rule to allow incoming traffic on port 8080, which I designated for running the Flask application.

**Environment Setup**

To prepare the VM environment, several tools and dependencies needed to be installed:

* **Java Development Kit (JDK) and Hadoop**: Installed these components to support the execution of PySpark. PySpark leverages Hadoop's distributed processing capabilities for efficient data analysis.
* **Git**: Used to clone the repository containing the web application code onto the VM.
* **Python and Pip**: Installed Python along with Pip to manage and install Python packages required by the application.
* **Environment Variables**: I configured environment variables for Java and Hadoop. This step was essential to ensure that the VM could locate and utilize these components during the execution of PySpark tasks.
* **Database Setup**: PostgreSQL was selected as the database management system to store reviews and ratings data. To set up the database and tables, I executed the dbscript.py script. This script, located within the repository, created the necessary schema and tables for storing and retrieving data.
* **Flask Application Launch**: The heart of the recommendation system was the Flask-based web application. To initiate the application, I executed the app.py file. Flask provided a reliable and efficient framework for building the web interface that users would interact with.
* **Accessing the Web Application**: With the Flask app running, users could access the recommendation system by navigating to the external IP address of the VM at port 8080. This URL led users to the web interface where they could input their preferences and receive personalized recommendations.

**How Does the Application Work?**

Both the K-means Recommendation engine and the ALS recommendation engine utilize Select2.js which gives you a customizable select box with support for searching, tagging, remote data sets, infinite scrolling, and many other highly used options. This search box allows you to type complete product titles or substrings which makes a call an api and returns a list of products matching that title or substring as seen in **figure x**. Once you select an item from the dropdown and click search you are provided with 5 recommendations as seen in **figure y.** In the case of the ALS recommendation engine, there are cases where there are no recommendations (related products), this is because of the cold start problem where there is insufficient user-item interaction for a customer to give a recommendation as seen in **figure z.**



Figure X

Figure y

Figure Z

**Future Work**

Considering the limitations of the collaborative filtering model and the relative success of the K-Means filtering approach (Smith et al., 20XX), there's a promising way to enhance personalized recommendations by combining these two strategies. Typically, the Collaborative Filtering (CF) recommendation systems rely solely on ratings and as a result, experience what is known as the cold-start problem. With the cold-start problem, the system lacks knowledge about the preferences of new users, leading to an inability to provide relevant recommendations. Similarly, with new items, the absence of ratings results in the system's uncertainty about which users to suggest these items to. Hybrid recommendation systems address this by integrating CF or other techniques with features from items, often utilizing association rule mining (Cano and Morisio, 2017).

Furthermore, there's potential for refining the K-Means recommender. This can be accomplished by implementing filtering mechanisms to exclude reviews with low ratings (1-2 stars) and negative sentiments thereby creating a sort of popularity content-based filtering model. By adopting such a methodology, the K-Means model could be fine-tuned to highlight items of higher popularity, subsequently yielding more relevant and valuable recommendations.

References:

Smith, A. B., Johnson, C. D., & Anderson, E. F. (20XX). Title of Collaborative Filtering Study. Journal Name, Volume (Issue), Page Numbers.

Jones, E. G. (20YY). Title of Sentiment Analysis Research. Conference Proceedings, Page Numbers.Descriptive Statistics