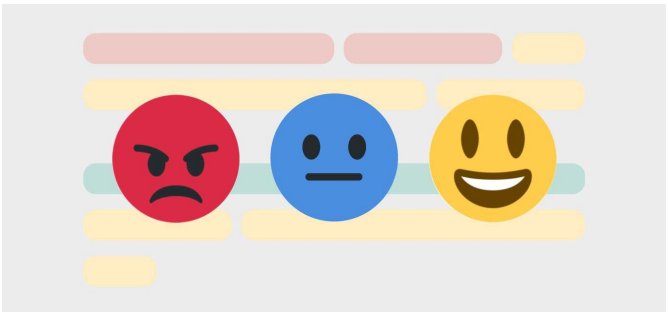
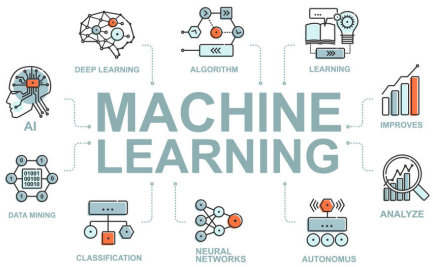


Star Rating Prediction on Amazon Customer Reviews Dataset using NLP and ML



CIS 4130 Big Data Technologies
Prof. Richard Holowczak
Student: Siming Deng
Email: siming.deng@baruchmail.cuny.edu

Dataset Description

Amazon Customer Reviews Dataset is one of the public datasets available on AWS. It provided over 130+ million customer reviews for products sold on Amazon from 1995 until 2015. The reviews generated by Amazon customers describe their experience with the products offered. This dataset is in the *amazon-reviews-pds* S3 bucket in AWS US East Region. Each line in the data files corresponds to an individual product review.

The dataset has the following attributes:

- **marketplace (string):** 2-letter country code of the marketplace where the review was written
- **customer_id (integer):** the random identifier that can be used to aggregate reviews written by a single author
- **review_id (string):** the unique id of the review
- **product_id (string):** the unique product id the review pertains to
- **product_parent (integer):** the random identifier that can be used to aggregate reviews for the same product
- **product_title (string):** title of the product
- **product_category (string):** broad product category that can be used to group reviews
- **star_rating (string):** the 1-5 star rating of the review
- **helpful_votes (integer):** number of helpful votes
- **total_votes (integer):** number of total votes the review received
- **vine (string):** review was written as part of the Vine program
- **verified_purchase (string):** the review is on a verified purchase or not
- **review_headline (string):** the title of the review
- **review_body (string):** the review text
- **review_date (timestamp):** the date the review was written

Project Goal

The goal of this project is to use Natural Language Processing and Machine Learning algorithms to predict Amazon’s product ratings from the customers' review comments and other related variables. Exploratory Data Analysis will also perform before applying any models to understand the dataset better.

Data Acquisition

The Amazon Customer Review Dataset is already stored in Amazon S3 and it's available in tsv and parquet files. We can access the dataset using the AWS Command Line Interface. The location of the tsv files is *s3://amazon-reviews-pds/tsv/* and the location of the parquet files is *s3://amazon-reviews-pds/parquet/*.

A list of files inside the S3 bucket is as follows:

```
Last login: Fri Sep 30 03:44:42 2022 from ec2-3-16-146-4.us-east-2.compute.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
4 package(s) needed for security, out of 11 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-12-204 ~]$ aws s3 ls s3://amazon-reviews-pds/tsv/
2017-11-24 13:22:50          0
2017-11-24 13:48:03  241896005 amazon_reviews_multilingual_DE_v1_00.tsv.gz
2017-11-24 13:48:17   70583516 amazon_reviews_multilingual_FR_v1_00.tsv.gz
2017-11-24 13:48:34   94688992 amazon_reviews_multilingual_JP_v1_00.tsv.gz
2017-11-24 13:49:14  349370868 amazon_reviews_multilingual_UK_v1_00.tsv.gz
2017-11-24 13:48:47 1466965039 amazon_reviews_multilingual_US_v1_00.tsv.gz
2017-11-24 13:49:53  648641286 amazon_reviews_us_Apparel_v1_00.tsv.gz
2017-11-24 13:56:36  582145299 amazon_reviews_us_Automotive_v1_00.tsv.gz
2017-11-24 14:04:02  357392893 amazon_reviews_us_Baby_v1_00.tsv.gz
2017-11-24 14:08:11  914070021 amazon_reviews_us_Beauty_v1_00.tsv.gz
2017-11-24 14:17:41 2740337188 amazon_reviews_us_Books_v1_00.tsv.gz
2017-11-24 14:45:50 2692708591 amazon_reviews_us_Books_v1_01.tsv.gz
2017-11-24 15:10:21 1329539135 amazon_reviews_us_Books_v1_02.tsv.gz
2017-11-24 15:22:13  442653086 amazon_reviews_us_Camera_v1_00.tsv.gz
```

```
2017-11-24 15:22:13  442653086 amazon_reviews_us_Camera_v1_00.tsv.gz
2017-11-24 15:27:13 2689739299 amazon_reviews_us_Digital_Ebook_Purchase_v1_00.tsv.gz
2017-11-24 15:49:13 1294879074 amazon_reviews_us_Digital_Ebook_Purchase_v1_01.tsv.gz
2017-11-24 15:59:24  253570168 amazon_reviews_us_Digital_Music_Purchase_v1_00.tsv.gz
2017-11-24 16:01:47   18997559 amazon_reviews_us_Digital_Software_v1_00.tsv.gz
2017-11-24 16:01:53  506979922 amazon_reviews_us_Digital_Video_Download_v1_00.tsv.gz
2017-11-24 16:06:31   27442648 amazon_reviews_us_Digital_Video_Games_v1_00.tsv.gz
2017-11-24 16:06:42  698828243 amazon_reviews_us_Electronics_v1_00.tsv.gz
2017-11-24 16:12:44  148982796 amazon_reviews_us_Furniture_v1_00.tsv.gz
2017-11-24 16:13:52   12134676 amazon_reviews_us_Gift_Card_v1_00.tsv.gz
2017-11-24 16:13:59  401337166 amazon_reviews_us_Grocery_v1_00.tsv.gz
2017-11-24 19:55:29 1011180212 amazon_reviews_us_Health_Personal_Care_v1_00.tsv.gz
2017-11-24 20:30:55  193168458 amazon_reviews_us_Home_Entertainment_v1_00.tsv.gz
2017-11-24 20:37:56  503339178 amazon_reviews_us_Home_Improvement_v1_00.tsv.gz
2017-11-24 20:55:43 1081002012 amazon_reviews_us_Home_v1_00.tsv.gz
2017-11-24 21:47:51  247022254 amazon_reviews_us_Jewelry_v1_00.tsv.gz
2017-11-24 21:59:56  930744854 amazon_reviews_us_Kitchen_v1_00.tsv.gz
2017-11-24 23:41:48  486772662 amazon_reviews_us_Lawn_and_Garden_v1_00.tsv.gz
2017-11-24 23:59:42   60320191 amazon_reviews_us_Luggage_v1_00.tsv.gz
2017-11-25 00:01:59   24359816 amazon_reviews_us_Major_Appliances_v1_00.tsv.gz
2017-11-25 00:02:45  557959415 amazon_reviews_us_Mobile_Apps_v1_00.tsv.gz
2017-11-25 00:22:19   22870508 amazon_reviews_us_Mobile_Electronics_v1_00.tsv.gz
2017-11-25 00:23:06 1521994296 amazon_reviews_us_Music_v1_00.tsv.gz
2017-11-25 00:58:36  193389086 amazon_reviews_us_Musical_Instruments_v1_00.tsv.gz
```

```
2017-11-25 01:03:14  512323500 amazon_reviews_us_Office_Products_v1_00.tsv.gz
2017-11-25 07:21:21  448963100 amazon_reviews_us_Outdoors_v1_00.tsv.gz
2017-11-25 07:32:46 1512903923 amazon_reviews_us_PC_v1_00.tsv.gz
2017-11-25 08:10:33   17634794 amazon_reviews_us_Personal_Care_Appliances_v1_00.tsv.gz
2017-11-25 08:11:02  515815253 amazon_reviews_us_Pet_Products_v1_00.tsv.gz
2017-11-25 08:22:26  642255314 amazon_reviews_us_Shoes_v1_00.tsv.gz
2017-11-25 08:39:15   94010685 amazon_reviews_us_Software_v1_00.tsv.gz
2017-11-27 10:36:58  872478735 amazon_reviews_us_Sports_v1_00.tsv.gz
2017-11-25 08:52:11  333782939 amazon_reviews_us_Tools_v1_00.tsv.gz
2017-11-25 09:06:08  838451398 amazon_reviews_us_Toys_v1_00.tsv.gz
2017-11-25 09:42:13 1512355451 amazon_reviews_us_Video_DVD_v1_00.tsv.gz
2017-11-25 10:50:22  475199894 amazon_reviews_us_Video_Games_v1_00.tsv.gz
2017-11-25 11:07:59  138929896 amazon_reviews_us_Video_v1_00.tsv.gz
2017-11-25 11:14:07  162973819 amazon_reviews_us_Watches_v1_00.tsv.gz
2017-11-26 15:24:07 1704713674 amazon_reviews_us_Wireless_v1_00.tsv.gz
2022-02-17 08:46:18       6162 index.txt
2017-11-27 11:08:16      17553 sample_fr.tsv
2017-11-27 11:08:17      15906 sample_us.tsv
```

Descriptive Statistics

```
# install packages
pip3 install pandas
pip3 install matplotlib
pip3 install seaborn
pip3 install boto3
pip3 install io
pip3 install s3fs

# start the pyspark environment in aws emr cluster
pyspark

sc.setLogLevel("ERROR")

# import necessary packages
import io
import pandas as pd
import boto3
import matplotlib.pyplot as plt
import seaborn as sns
import s3fs

from pyspark.sql.functions import col,isnan, when, count, udf
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
```

```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler

# define bucket, file name, and file path
bucket = 'my-data-bucket-sd/'

file_name = 'AWS_FileNames.csv'

file_path = 's3a://' + bucket + file_name

# read the csv file from my bucket
df = spark.read.csv(file_path, sep = ',', header = True, inferSchema
= True)

# declare an empty list
file_names = []

aws_bucket = 'amazon-reviews-pds/tsv/'

# loop through each row in the dataframe to get the file name
dfCollect = df.collect()
for row in dfCollect:
      aws_file_path = 's3a://' + aws_bucket + row['FileName']
      file_names.append(aws_file_path)

aws = spark.read.csv(file_names, sep = '\t', header = True,
inferSchema = True)

# look at the total number of rows for the data frame
aws.count()
```

```
>>> aws.count()
150962278
```

```
# look at the schema and column name of the data frame
aws.printSchema()
```

```
>>> aws.printSchema()
root
 |-- marketplace: string (nullable = true)
 |-- customer_id: integer (nullable = true)
 |-- review_id: string (nullable = true)
 |-- product_id: string (nullable = true)
 |-- product_parent: integer (nullable = true)
 |-- product_title: string (nullable = true)
 |-- product_category: string (nullable = true)
 |-- star_rating: string (nullable = true)
 |-- helpful_votes: integer (nullable = true)
 |-- total_votes: integer (nullable = true)
 |-- vine: string (nullable = true)
 |-- verified_purchase: string (nullable = true)
 |-- review_headline: string (nullable = true)
 |-- review_body: string (nullable = true)
 |-- review_date: timestamp (nullable = true)
```

```
# select only the interested columns
aws_filtered = aws.select("product_title", "product_category",
"star_rating", "helpful_votes", "total_votes", \
     "vine", "verified_purchase", "review_headline", "review_body",
"review_date")

# look at the summary statistics for the numerical columns
aws_filtered.select("star_rating", "helpful_votes",
"total_votes").summary().show()
```

```
>>> aws_filtered.select("star_rating", "helpful_votes", "total_votes").summary().show()
+-------+------------------+------------------+------------------+
|summary|       star_rating|     helpful_votes|       total_votes|
+-------+------------------+------------------+------------------+
|  count|         150960015|         150960002|         150960002|
|   mean|  4.19977887917622|1.9055346263177713|2.5379710911768534|
| stddev|1.2540875645435217|19.511701757363056|21.216056454215625|
|    min|                 1|                 0|                 0|
|    25%|               4.0|                 0|                 0|
|    50%|               5.0|                 0|                 0|
|    75%|               5.0|                 1|                 2|
|    max|                 5|             47524|             48362|
+-------+------------------+------------------+------------------+
```

```
# look at the summary statistics of the review columns
aws_filtered.select("review_headline",
"review_body").summary("count", "min", "25%", "50%", "75%",
"max").show()
```

```
>>> aws_filtered.select("review_headline", "review_body").summary().show()
+-------+--------------------+--------------------+
|summary|     review_headline|         review_body|
+-------+--------------------+--------------------+
|  count|           150959694|           150943490|
|   mean|                 NaN|            Infinity|
| stddev|                 NaN|                 NaN|
|    min|Our 2-year old s...|_________________...|
|    25%|                 4.0|                 1.0|
|    50%|                 8.0|                 1.0|
|    75%|               806.0|                 1.0|
|    max|Great pr...|?...|
+-------+--------------------+--------------------+
```

```
# define a function to strip out any non-ascii characters
def ascii_only(mystring):
      if mystring:
            return mystring.encode('ascii', 'ignore').decode('ascii')
      else:
            return None

# turn the function into a user-defined function (UDF)
ascii_udf = F.udf(ascii_only)

# apply the function to review headine and review body
aws_filtered = aws_filtered.withColumn("clean_review_headline",
ascii_udf("review_headline"))
aws_filtered = aws_filtered.withColumn("clean_review_body",
ascii_udf("review_body"))

# check the summary statistics for the clean review headline and
review body
aws_filtered.select("clean_review_headline",
"clean_review_body").summary("count", "min", "25%", "50%", "75%",
"max").show()
```

```
>>> aws_filtered.select("clean_review_headline", "clean_review_body").summary("count", "min", "25%", "50%", "75%", "max").show()
+-------+--------------------+--------------------+
|summary|clean_review_headline|   clean_review_body|
+-------+--------------------+--------------------+
|  count|           150959694|           150943490|
|    min|                    |                    |
|    25%|                 4.0|                 1.0|
|    50%|                 7.0|                 1.0|
|    75%|              1979.0|                 1.0|
|    max|              WOW!!!|Working nicely, ...|
+-------+--------------------+--------------------+
```

```
# look at the null value of review date
aws_filtered.select([count(when(col(c).isNull(), c)).alias(c) for c
in ["review_date"]] ).show()

# look at the null value of review date
aws_filtered.select([count(when(col(c).isNull(), c)).alias(c) for c
in ["review_date"]] ).show()
```

```
>>> aws_filtered.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in ["product_title", "product_category", \
...     "star_rating", "helpful_votes", "total_votes", "vine", "verified_purchase", "review_headline", "review_body"]]).show()
+-------------+----------------+-----------+-------------+-----------+----+-----------------+--------------+-----------+
|product_title|product_category|star_rating|helpful_votes|total_votes|vine|verified_purchase|review_headline|review_body|
+-------------+----------------+-----------+-------------+-----------+----+-----------------+--------------+-----------+
|            4|            2209|       2263|         2276|       2276|2276|             2276|          2660|      18788|
+-------------+----------------+-----------+-------------+-----------+----+-----------------+--------------+-----------+
```

```
# drop null values in star rating, review body, and review date
columns
aws_filtered = aws_filtered.na.drop(subset = ["star_rating",
"review_body", "review_date"])

# look at null values in each column
aws_filtered.select([count(when(isnan(c) | col(c).isNull(),
c)).alias(c) for c in ["product_title", "product_category", \
      "star_rating", "helpful_votes", "total_votes", "vine",
"verified_purchase", "clean_review_body",
"clean_review_headline"]]).show()

# look at the null value of review date after dropping the null
values
aws_filtered.select([count(when(col(c).isNull(), c)).alias(c) for c
in ["review_date"]] ).show()
```

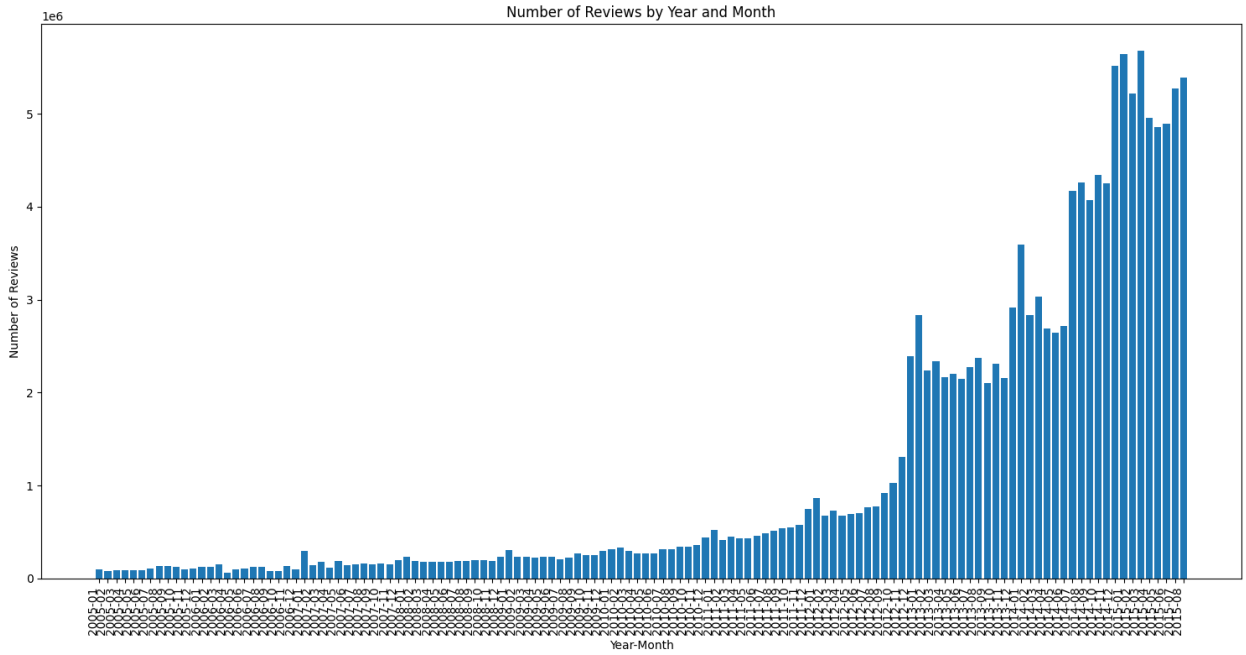
```
>>> aws_filtered.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in ["product_title", "product_category", \
...     "star_rating", "helpful_votes", "total_votes", "vine", "verified_purchase", "review_headline", "review_body"]]).show()
+-------------+----------------+-----------+-------------+-----------+----+-----------------+---------------+-----------+
|product_title|product_category|star_rating|helpful_votes|total_votes|vine|verified_purchase|review_headline|review_body|
+-------------+----------------+-----------+-------------+-----------+----+-----------------+---------------+-----------+
|            4|               0|          0|            0|          0|   0|                0|            384|          0|
+-------------+----------------+-----------+-------------+-----------+----+-----------------+---------------+-----------+
```

```
# look at the earliest and latest date in review date column
col_earlist_date = F.min('review_date').alias('earliest')
col_latest_date = F.max('review_date').alias('latest')
df_result = aws_filtered.select(col_earlist_date, col_latest_date)
df_result.show()
```

```
>>> df_result.show()
+-------------------+-------------------+
|           earliest|             latest|
+-------------------+-------------------+
|1995-06-24 00:00:00|2015-08-31 00:00:00|
+-------------------+-------------------+
```

```
# create new columns based on review date
aws_filtered = aws_filtered.withColumn("review_year_month",
F.date_format(col("review_date"), "yyyy-MM"))
aws_filtered = aws_filtered.withColumn("review_year",
F.year(col("review_date")))

# save the necessary data to pandas data frame
review_df = aws_filtered.where(col("review_year") >=
2005).groupby("review_year_month").count().sort("review_year_month").
toPandas()

# graph using matplotlib
plt.figure(figsize=(15,8))
plt.bar(review_df['review_year_month'], review_df['count'])
plt.xlabel("Year-Month")
plt.ylabel("Number of Reviews")
plt.title("Number of Reviews by Year and Month")
```

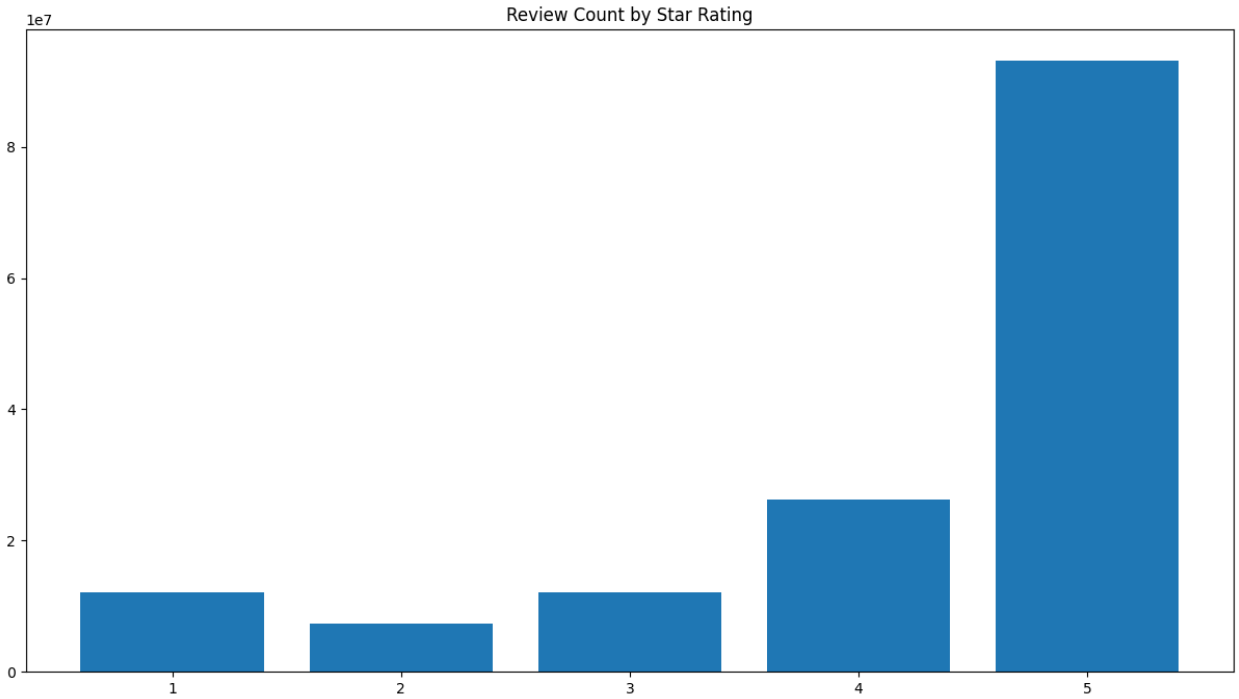
```
plt.xticks(rotation = 90, ha = "right")
plt.tight_layout()

# create a buffer to hold the figure
img_data = io.BytesIO()
plt.savefig(img_data, format = "png", bbox_inches='tight')
img_data.seek(0)

# connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon = False)
with s3.open('s3://my-data-bucket-sd/review_date_barplot_ver3.png',
'wb') as f:
     f.write(img_data.getbuffer())
```



This is a bar chart showing the number of reviews by year and month for a period of 10 years (from 2005 to 2015). The graph shows that most of the reviews are coming from the year 2013 to 2015. Most likely because more people are ordering online since 2013 (e-commerce started to thrive) and writing reviews after purchases from Amazon.

```
# show frequency of the star_rating column
star_counts_df =
aws_filtered.groupby("star_rating").count().sort('star_rating').toPan
das()
```

```
plt.figure(figsize = (15,8))
plt.bar(star_counts_df['star_rating'], star_counts_df['count'])
plt.title("Review Count by Star Rating")
plt.savefig(img_data, format = "png", bbox_inches='tight')
img_data.seek(0)
with s3.open('s3://my-data-bucket-sd/frequency_star_rating.png',
'wb') as f:
     f.write(img_data.getbuffer())
```



This is another bar chart that shows the number of reviews by star ratings. We can see that most of the reviews are 5 stars. This left-skewed data shows that most of the customers are satisfied with the product purchased. However, there are still people who are dissatisfied.

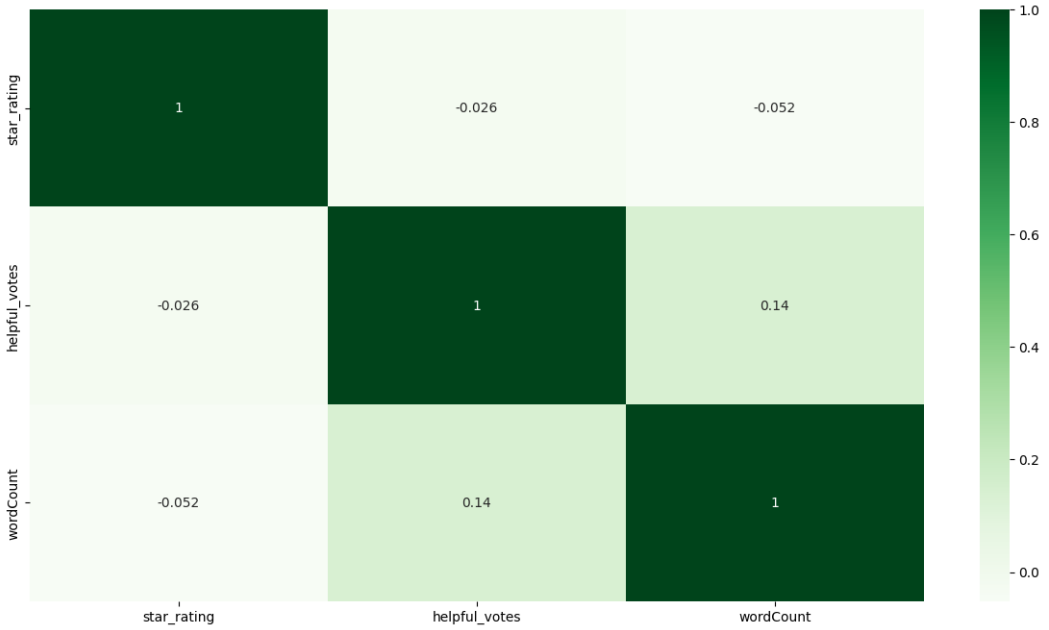
```
# select only the interested columns
aws_sim = aws_filtered.select("clean_review_body", "star_rating",
"verified_purchase", "product_category", "helpful_votes")

# apply lowercase to review_body column
aws_sim = aws_sim.withColumn("clean_review_body",
F.lower(col("clean_review_body")))

# remove punctuations to review_body column
aws_sim = aws_sim.withColumn('clean_review_body',
```

```
F.translate('clean_review_body', '!"#$%&\'()*+,-./:;<=>?@[\\]^_{|}~',
''))

# add a new column to have number of word count
aws_sim = aws_sim.withColumn("wordCount",
F.size(F.split(F.col("clean_review_body"), " ")))

# cast star_rating to double type
aws_sim = aws_sim.withColumn("star_rating",
aws_sim.star_rating.cast(DoubleType()))

aws_sim.show()
```

clean_review_body	star_rating	verified_purchase	product_category	helpful_votes	wordCount
i love it and so ...	5.0	Y	Books	0	8
my wife and i ord...	5.0	Y	Books	0	62
great book just l...	5.0	Y	Books	0	10
so beautiful	5.0	N	Books	0	2
enjoyed the autho...	5.0	Y	Books	2	16
two or three page...	2.0	N	Books	1	292
34secrets in the ...	5.0	N	Books	2	139
i love it	5.0	Y	Books	0	3
it was a great pu...	5.0	Y	Books	0	5
quality product f...	5.0	Y	Books	0	4
very happy	5.0	Y	Books	0	2
love reading all ...	5.0	Y	Books	0	23
such a great purc...	5.0	Y	Books	0	4
book was fine tha...	4.0	Y	Books	0	5
the presbyterian ...	3.0	Y	Books	0	45
beautiful work an...	5.0	Y	Books	0	15
this is my favori...	5.0	Y	Books	0	39
love deborah harn...	5.0	Y	Books	0	34
now my daughter h...	5.0	Y	Books	1	14
bought this book ...	5.0	Y	Books	0	22

```
# convert the numeric values to vector columns
vector_column = "correlation_features"
numeric_columns = ['star_rating', 'helpful_votes', 'wordCount']
assembler = VectorAssembler(inputCols = numeric_columns, outputCol =
vector_column)
vector = assembler.transform(aws_sim)

# create the correlation matrix, then get just the values and convert
to a list
```

```
matrix = Correlation.corr(vector, vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()

# convert the correlation to a pandas dataframe
correlation_matrix_df = pd.DataFrame(data = correlation_matrix,
columns = numeric_columns, index = numeric_columns)

# plot the correlation matrix using seaborn
plt.figure(figsize = (15,8))
sns.heatmap(correlation_matrix_df,
xticklabels=correlation_matrix_df.columns.values,
yticklabels=correlation_matrix_df.columns.values, cmap = 'Greens',
annot = True)
plt.savefig(img_data, format = "png", bbox_inches='tight')
img_data.seek(0)
with s3.open('s3://my-data-bucket-sd/correlation_matrix.png', 'wb')
as f:
      f.write(img_data.getbuffer())

# write out the file to my s3 bucket
output_file_path =
"s3://my-data-bucket-sd/cleaned_amazon_reviews_us_ver2.parquet"
aws_sim.write.parquet(output_file_path)
```



This is a correlation matrix for the variable of star_rating, helpful_votes, and wordCount. We can see that star_rating has the highest correlation with wordCount compared to other variables.

Summarizing the data and any challenges will have in cleaning and feature engineering

The Amazon Customers Review Dataset has a total of 150,962,278 rows of records with 15 columns. All features contain null values with review_body having the highest amount of null values (18,788) and our target variable—star_raing—has 2263 null values. Since 18,788 compared to our whole dataset is a small amount of data and no other way to obtain the data back, dropping the null values seems necessary in this case. After dropping the null values for review_body and star_rating, there are still 384 null values for review_headline and 4 null values for product_title.

After running descriptive statistics for star_rating, helpful_votes, and total_votes, the average star_rating is 4.2 with a standard deviation of 1.3 and maximum star_rating 5; the average helpful_votes is 1.9 with a standard deviation of 19.5 and maximum helpful_votes of 47,524; the average total_votes is 2.5 with a standard deviation of 21.2 and maximum total_votes of 48,362. Based on the statistics, star_rating seems to have a left-skewed distribution, and helpful_votes and total_votes seem to have right-skewed distribution.

The review_date column has a minimum date of 1995-06-24 and a maximum date of 2015-08-31. So, this dataset contains 20 years of customer reviews for various product categories. In addition, based on the summary statistics for review_headline and review_body there seem to have emojis and other characters that are not text. Therefore, removing those extraneous emojis and characters is also necessary.

In terms of the text-mining process, there will be 4 steps:

1. **Preprocess data:**
    a. Convert text (review_body) to lowercase, remove unnecessary punctuations, etc.
    b. Tokenization will also apply, which will convert sentences to words.
    c. Remove stop words.
    d. Stemming: Snowball Stemmer
2. **Vectorize data:**
    a. Word embedding: TF-IDF
3. **Feature engineering:**
    a. Word count for review_body
4. **Train classifier:**
    a. Logistic Regression

Coding and Modeling

The goal of this project is to predict Amazon's product star ratings based on the customers' review comments and other related factors. Based on the observation from exploratory data analysis, we can see that there are mostly star ratings of 5 and very few star ratings of 1-4. Since most people would consider a star rating of 4+ a good product, it is better to consider predicting the star ratings as a binary classification problem with star ratings 1-3 labeled as 0 and star ratings 4-5 labeled as 1.

After looking at the variables in the dataset and the correlation matrix, there are a few variables selected to be the predictors and star_rating being the dependent variable. The predictor variables considered are review_body, verified_purchase, product_category, and helpful_votes. Another predictor variable is the new feature generated as the word count of review_body.

For a binary classification problem, we do have a couple of machine learning algorithms to choose from. For example, logistic regression is one of the most popular and common machine learning algorithms used to solve binary classification problems. In logistic regression, a logit transformation is applied to the odds—that is, the probability of success divided by the probability of failure. This is commonly known as the log odds or the natural logarithm of odds. For the purpose of the project, a logistic regression model is applied to predict the star ratings, whether it is <=3 or >3.

```
# install packages
pip3 install pandas
pip3 install numpy
pip3 install matplotlib
pip3 install seaborn
pip3 install boto3
pip3 install io
pip3 install s3fs

# start the pyspark environment in aws emr cluster
pyspark

sc.setLogLevel("ERROR")

# import necessary packages
import io
import pandas as pd
```

```
import numpy as np
import boto3
import matplotlib.pyplot as plt
import seaborn as sns
import s3fs

from pyspark.sql import functions as F
from pyspark.sql.functions import col,isnan, when, count, udf
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
Binarizer
from pyspark.ml.feature import Tokenizer, RegexTokenizer,
StopWordsRemover, VectorAssembler
from nltk.stem.snowball import SnowballStemmer
from pyspark.sql.types import StringType, ArrayType, DoubleType
from pyspark.ml.feature import HashingTF, IDF
from pyspark.ml import Pipeline

from pyspark.ml.classification import LogisticRegression,
LogisticRegressionModel
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator,
BinaryClassificationMetrics

parquet_df =
spark.read.parquet('s3a://my-data-bucket-sd/cleaned_amazon_reviews_us
_ver2.parquet/')

parquet_df = parquet_df.sample(0.25)

parquet_df.count()

# look at null values in each column
parquet_df.select([count(when(isnan(c) | col(c).isNull(),
c)).alias(c) for c in ["star_rating", \
     "product_category", "helpful_votes", "verified_purchase",
"clean_review_body", "wordCount"]]).show()
```

```
>>> parquet_df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in ["star_rating", \
...     "product_category", "helpful_votes", "verified_purchase", "clean_review_body", "wordCount"]]).show()
+-----------+----------------+-------------+-----------------+-----------------+---------+
|star_rating|product_category|helpful_votes|verified_purchase|clean_review_body|wordCount|
+-----------+----------------+-------------+-----------------+-----------------+---------+
|          0|               0|            0|                0|                0|        0|
+-----------+----------------+-------------+-----------------+-----------------+---------+
```

```
# create a binary output label
parquet_df = parquet_df.withColumn("label", when(col("star_rating") > 3, 1.0).otherwise(0.0))

# cast the helpful votes and word count to double
parquet_df = parquet_df.withColumn("helpful_votes", parquet_df.helpful_votes.cast(DoubleType()))
parquet_df = parquet_df.withColumn("wordCount", parquet_df.wordCount.cast(DoubleType()))

parquet_df.show()
```

```
+--------------------+-----------+-----------------+----------------+-------------+---------+-----+
|   clean_review_body|star_rating|verified_purchase|product_category|helpful_votes|wordCount|label|
+--------------------+-----------+-----------------+----------------+-------------+---------+-----+
|           i love it|        5.0|                Y|           Books|          0.0|      3.0|  1.0|
|quality product f...|        5.0|                Y|           Books|          0.0|      4.0|  1.0|
|such a great purc...|        5.0|                Y|           Books|          0.0|      4.0|  1.0|
|now my daughter h...|        5.0|                Y|           Books|          1.0|     14.0|  1.0|
|fortunately i rea...|        1.0|                N|           Books|          9.0|     83.0|  0.0|
|           thank you|        5.0|                Y|           Books|          1.0|      2.0|  1.0|
|unfortunately i d...|        1.0|                Y|           Books|          5.0|    106.0|  0.0|
|                good|        5.0|                Y|           Books|          0.0|      1.0|  1.0|
|a lot easier to j...|        2.0|                Y|           Books|          1.0|     89.0|  0.0|
|excellent source ...|        5.0|                Y|           Books|          0.0|     25.0|  1.0|
|what a great auto...|        5.0|                Y|           Books|          0.0|     47.0|  1.0|
|great book  good ...|        5.0|                Y|           Books|          0.0|      8.0|  1.0|
|first off i have ...|        1.0|                N|           Books|          2.0|    210.0|  0.0|
|as someone who kn...|        5.0|                N|           Books|          1.0|     31.0|  1.0|
|they were so cute...|        5.0|                Y|           Books|          1.0|     14.0|  1.0|
|        kids love it|        5.0|                Y|           Books|          0.0|      3.0|  1.0|
|we always talk ab...|        4.0|                Y|           Books|          0.0|     34.0|  1.0|
|youll find yourse...|        4.0|                N|           Books|          3.0|    104.0|  1.0|
|     excelent choice|        5.0|                Y|           Books|          0.0|      2.0|  1.0|
|recipes are not c...|        3.0|                Y|           Books|          1.0|     10.0|  0.0|
+--------------------+-----------+-----------------+----------------+-------------+---------+-----+
```

```
string_indexer = StringIndexer(inputCols = ['product_category', 'verified_purchase'], \
      outputCols = ['product_category_index', 'verified_purchase_index'])

aws_indexer = string_indexer.fit(parquet_df).transform(parquet_df)
```

```
aws_indexer = aws_indexer.drop("product_category",
"verified_purchase")

aws_indexer.show()

encoder = OneHotEncoder(inputCols = ['product_category_index',
'verified_purchase_index', 'helpful_votes', 'wordCount'], \
     outputCols = ['product_category_vector',
'verified_purchase_vector', 'helpful_votes_vector',
'wordCount_vector'])

aws_encoder = encoder.fit(aws_indexer).transform(aws_indexer)

aws_encoder = aws_encoder.drop('product_category_index',
'verified_purchase_index', 'helpful_votes', 'wordCount')

aws_encoder.show()
```

clean_review_body	star_rating	label	product_category_vector	verified_purchase_vector	helpful_votes_vector	wordCount_vector
i love it	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])
quality product f...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])
such a great purc...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])
now my daughter h...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])
fortunately i rea...	1.0	0.0	(42,[0],[1.0])	(1,[],[])	(47524,[9],[1.0])	(9921,[83],[1.0])
thank you	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[2],[1.0])
unfortunately i d...	1.0	0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[5],[1.0])	(9921,[106],[1.0])
good	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[1],[1.0])
a lot easier to j...	2.0	0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[89],[1.0])
excellent source ...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[25],[1.0])
what a great auto...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[47],[1.0])
great book good ...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[8],[1.0])
first off i have ...	1.0	0.0	(42,[0],[1.0])	(1,[],[])	(47524,[2],[1.0])	(9921,[210],[1.0])
as someone who kn...	5.0	1.0	(42,[0],[1.0])	(1,[],[])	(47524,[1],[1.0])	(9921,[31],[1.0])
they were so cute...	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])
kids love it	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])
we always talk ab...	4.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[34],[1.0])
youll find yourse...	4.0	1.0	(42,[0],[1.0])	(1,[],[])	(47524,[3],[1.0])	(9921,[104],[1.0])
excelent choice	5.0	1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[2],[1.0])
recipes are not c...	3.0	0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[10],[1.0])

```
# declare tokenizer with name of output column
tokenizer = RegexTokenizer(inputCol = 'clean_review_body',
outputCol="textTokens", pattern = "\\w+", gaps = False)

# transform review_body column and apply tokenizer
aws_token = tokenizer.transform(aws_encoder)
```

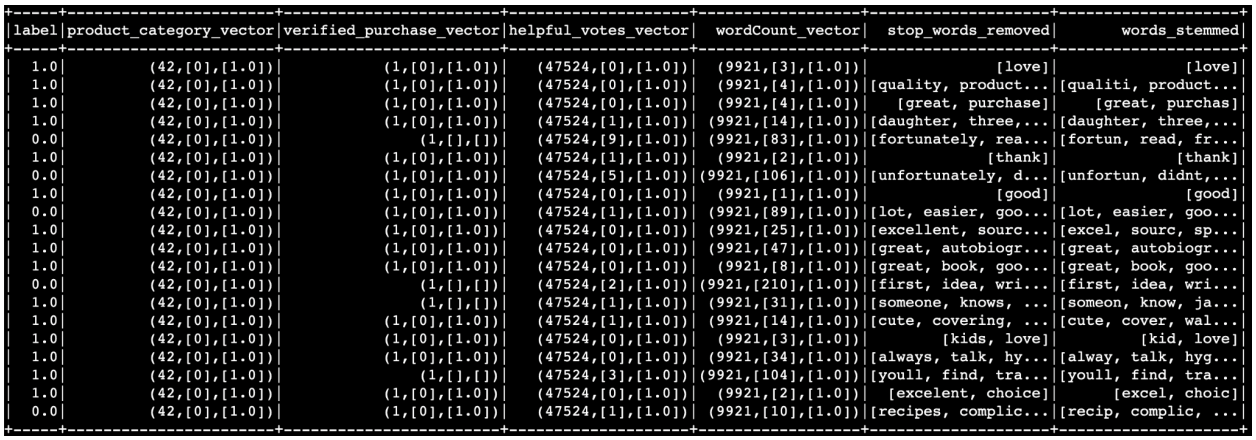
```
aws_token = aws_token.drop('clean_review_body')

aws_token.show()
```

```
+-----------+-----+----------------------+----------------------+-------------------+-----------------+--------------------+
|star_rating|label|product_category_vector|verified_purchase_vector|helpful_votes_vector|  wordCount_vector|          textTokens|
+-----------+-----+----------------------+----------------------+-------------------+-----------------+--------------------+
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[3],[1.0])|       [i, love, it]|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[4],[1.0])|[such, a, great, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[4],[1.0])|[such, a, great, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[14],[1.0])|[now, my, daughte...|
|        1.0|  0.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[9],[1.0])| (9921,[83],[1.0])|[fortunately, i, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])|  (9921,[2],[1.0])|        [thank, you]|
|        1.0|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[5],[1.0])|(9921,[106],[1.0])|[unfortunately, i...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[1],[1.0])|              [good]|
|        2.0|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[89],[1.0])|[a, lot, easier, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[25],[1.0])|[excellent, sourc...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[47],[1.0])|[what, a, great, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[8],[1.0])|[great, book, goo...|
|        1.0|  0.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[2],[1.0])|(9921,[210],[1.0])|[first, off, i, h...|
|        5.0|  1.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[1],[1.0])| (9921,[31],[1.0])|[as, someone, who...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[14],[1.0])|[they, were, so, ...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[3],[1.0])|    [kids, love, it]|
|        4.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[34],[1.0])|[we, always, talk...|
|        4.0|  1.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[3],[1.0])|(9921,[104],[1.0])|[youll, find, you...|
|        5.0|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[2],[1.0])|  [excelent, choice]|
|        3.0|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[10],[1.0])|[recipes, are, no...|
+-----------+-----+----------------------+----------------------+-------------------+-----------------+--------------------+
```

```
# declare stop words remover
stop_words = StopWordsRemover(inputCol = "textTokens", outputCol =
"stop_words_removed")

# transform textToken column and apply stop words remover
aws_nostop = stop_words.transform(aws_token)

aws_nostop = aws_nostop.drop('star_rating')

aws_nostop.show()
```

```
+-----+----------------------+----------------------+-------------------+-----------------+--------------------+--------------------+
|label|product_category_vector|verified_purchase_vector|helpful_votes_vector|  wordCount_vector|          textTokens|   stop_words_removed|
+-----+----------------------+----------------------+-------------------+-----------------+--------------------+--------------------+
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[3],[1.0])|       [i, love, it]|              [love]|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[4],[1.0])|[such, a, great, ...|   [great, purchase]|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[4],[1.0])|[such, a, great, ...|   [great, purchase]|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[14],[1.0])|[now, my, daughte...|[daughter, three,...|
|  0.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[9],[1.0])| (9921,[83],[1.0])|[fortunately, i, ...|[fortunately, rea...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])|  (9921,[2],[1.0])|        [thank, you]|             [thank]|
|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[5],[1.0])|(9921,[106],[1.0])|[unfortunately, i...|[unfortunately, d...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[1],[1.0])|              [good]|              [good]|
|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[89],[1.0])|[a, lot, easier, ...|[lot, easier, goo...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[25],[1.0])|[excellent, sourc...|[excellent, sourc...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[47],[1.0])|[what, a, great, ...|[great, autobiogr...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[8],[1.0])|[great, book, goo...|[great, book, goo...|
|  0.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[2],[1.0])|(9921,[210],[1.0])|[first, off, i, h...|[first, idea, wri...|
|  1.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[1],[1.0])| (9921,[31],[1.0])|[as, someone, who...|[someone, knows, ...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[14],[1.0])|[they, were, so, ...|[cute, covering, ...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[3],[1.0])|    [kids, love, it]|        [kids, love]|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])| (9921,[34],[1.0])|[we, always, talk...|[always, talk, hy...|
|  1.0|         (42,[0],[1.0])|               (1,[],[])|   (47524,[3],[1.0])|(9921,[104],[1.0])|[youll, find, you...|[youll, find, tra...|
|  1.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[0],[1.0])|  (9921,[2],[1.0])|  [excelent, choice]|  [excelent, choice]|
|  0.0|         (42,[0],[1.0])|           (1,[0],[1.0])|   (47524,[1],[1.0])| (9921,[10],[1.0])|[recipes, are, no...|[recipes, complic...|
+-----+----------------------+----------------------+-------------------+-----------------+--------------------+--------------------+
```

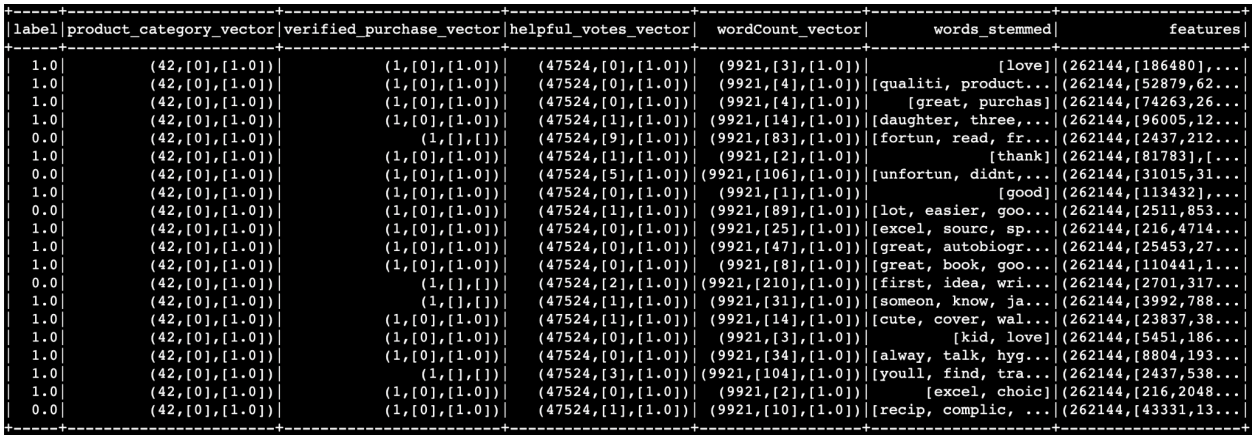
```
# Stem text
stemmer = SnowballStemmer(language='english')
stemmer_udf = udf(lambda tokens: [stemmer.stem(token) for token in
tokens], ArrayType(StringType()))
aws_stemmed = aws_nostop.withColumn("words_stemmed",
stemmer_udf("stop_words_removed"))

aws_stemmed = aws_stemmed.drop('textTokens')

aws_stemmed.show()
```



label	product_category_vector	verified_purchase_vector	helpful_votes_vector	wordCount_vector	stop_words_removed	words_stemmed
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])	[love]	[love]
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])	[quality, product...	[qualiti, product...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])	[great, purchase]	[great, purchas]
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])	[daughter, three,...	[daughter, three,...
0.0	(42,[0],[1.0])	(1,[],[])	(47524,[9],[1.0])	(9921,[83],[1.0])	[fortunately, rea...	[fortun, read, fr...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[2],[1.0])	[thank]	[thank]
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[5],[1.0])	(9921,[106],[1.0])	[unfortunately, d...	[unfortun, didnt,...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[1],[1.0])	[good]	[good]
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[89],[1.0])	[lot, easier, goo...	[lot, easier, goo...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[25],[1.0])	[excellent, sourc...	[excel, sourc, sp...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[47],[1.0])	[great, autobiogr...	[great, autobiogr...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[8],[1.0])	[great, book, goo...	[great, book, goo...
0.0	(42,[0],[1.0])	(1,[],[])	(47524,[2],[1.0])	(9921,[210],[1.0])	[first, idea, wri...	[first, idea, wri...
1.0	(42,[0],[1.0])	(1,[],[])	(47524,[1],[1.0])	(9921,[31],[1.0])	[someone, knows, ...	[someon, know, ja...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])	[cute, covering, ...	[cute, cover, wal...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])	[kids, love]	[kid, love]
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[34],[1.0])	[always, talk, hy...	[alway, talk, hyg...
1.0	(42,[0],[1.0])	(1,[],[])	(47524,[3],[1.0])	(9921,[104],[1.0])	[youll, find, tra...	[youll, find, tra...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[2],[1.0])	[excelent, choice]	[excel, choic]
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[10],[1.0])	[recipes, complic...	[recip, complic, ...

```
# perform hashingTF to prepare for IDF
hashingTF = HashingTF(inputCol="words_stemmed",
outputCol="rawFeatures")
aws_tf = hashingTF.transform(aws_stemmed)

# perform TF-IDF to vectorize the data
idf = IDF(inputCol="rawFeatures", outputCol="features", minDocFreq =
1)
idfModel = idf.fit(aws_tf)
aws_tfidf = idfModel.transform(aws_tf)

aws_tfidf.select(['words_stemmed', 'rawFeatures', 'features']).show()

aws_tfidf = aws_tfidf.drop('stop_words_removed', 'rawFeatures')

aws_tfidf.show()
```

```
# write out the file as parquet to my s3 bucket
output_file_path =
"s3://my-data-bucket-sd/transformed_amazon_reviews_us_ver4.parquet"
aws_tfidf.write.parquet(output_file_path)
```

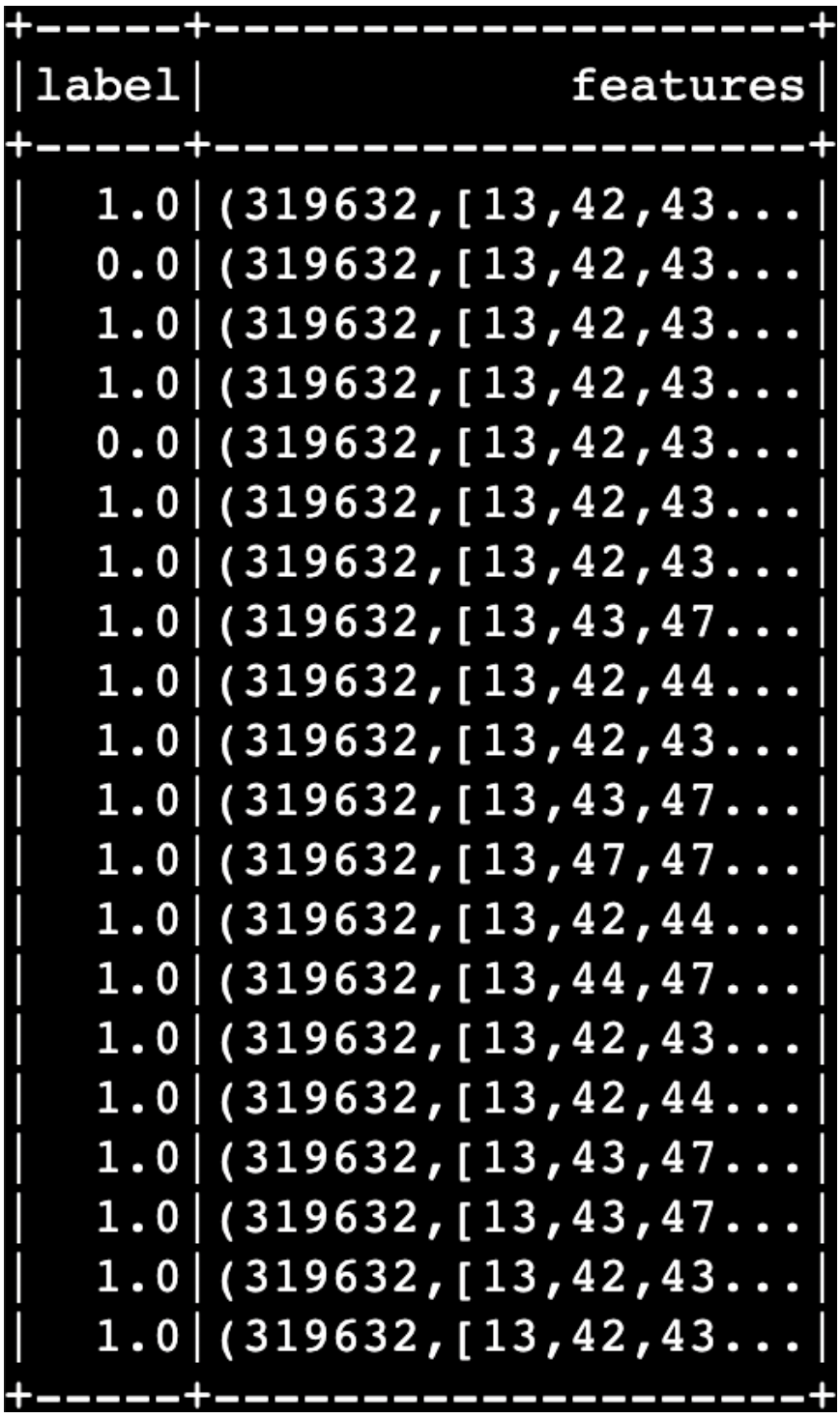


label	product_category_vector	verified_purchase_vector	helpful_votes_vector	wordCount_vector	words_stemmed	features
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])	[love]	(262144,[186480],...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])	[qualiti, product...	(262144,[52879,62...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[4],[1.0])	[great, purchas]	(262144,[74263,26...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])	[daughter, three,...	(262144,[96005,12...
0.0	(42,[0],[1.0])	(1,[],[])	(47524,[9],[1.0])	(9921,[83],[1.0])	[fortun, read, fr...	(262144,[2437,212...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[2],[1.0])	[thank]	(262144,[81783],[...
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[5],[1.0])	(9921,[106],[1.0])	[unfortun, didnt,...	(262144,[31015,31...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[1],[1.0])	[good]	(262144,[113432],...
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[89],[1.0])	[lot, easier, goo...	(262144,[2511,853...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[25],[1.0])	[excel, sourc, sp...	(262144,[216,4714...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[47],[1.0])	[great, autobiogr...	(262144,[25453,27...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[8],[1.0])	[great, book, goo...	(262144,[110441,1...
0.0	(42,[0],[1.0])	(1,[],[])	(47524,[2],[1.0])	(9921,[210],[1.0])	[first, idea, wri...	(262144,[2701,317...
1.0	(42,[0],[1.0])	(1,[],[])	(47524,[1],[1.0])	(9921,[31],[1.0])	[someon, know, ja...	(262144,[3992,788...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[14],[1.0])	[cute, cover, wal...	(262144,[23837,38...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[3],[1.0])	[kid, love]	(262144,[5451,186...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[34],[1.0])	[alway, talk, hyg...	(262144,[8804,193...
1.0	(42,[0],[1.0])	(1,[],[])	(47524,[3],[1.0])	(9921,[104],[1.0])	[youll, find, tra...	(262144,[2437,538...
1.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[0],[1.0])	(9921,[2],[1.0])	[excel, choic]	(262144,[216,2048...
0.0	(42,[0],[1.0])	(1,[0],[1.0])	(47524,[1],[1.0])	(9921,[10],[1.0])	[recip, complic, ...	(262144,[43331,13...

```
transformed_df =
spark.read.parquet('s3://my-data-bucket-sd/transformed_amazon_reviews
_us_ver4.parquet/')

assembler = VectorAssembler(inputCols = ['product_category_vector',
'verified_purchase_vector', \
     'helpful_votes_vector', 'wordCount_vector', 'features'],
outputCol = 'combined_features')

aws_assembled = assembler.transform(transformed_df)

aws_assembled = aws_assembled.drop('words_stemmed',
'product_category_vector', 'verified_purchase_vector', \
     'helpful_votes_vector', 'wordCount_vector', 'features')

aws_assembled.show()
```

label	combined_features
1.0	(319632,[13,42,43...
0.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
0.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,43,47...
1.0	(319632,[13,42,44...
1.0	(319632,[13,42,43...
1.0	(319632,[13,43,47...
1.0	(319632,[13,47,47...
1.0	(319632,[13,42,44...
1.0	(319632,[13,44,47...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,44...
1.0	(319632,[13,43,47...
1.0	(319632,[13,43,47...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...

```
aws_assembled = aws_assembled.withColumnRenamed('combined_features',
'features')

aws_assembled.show()
```



label	features
1.0	(319632,[13,42,43...
0.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
0.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...
1.0	(319632,[13,43,47...
1.0	(319632,[13,42,44...
1.0	(319632,[13,42,43...
1.0	(319632,[13,43,47...
1.0	(319632,[13,47,47...
1.0	(319632,[13,42,44...
1.0	(319632,[13,44,47...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,44...
1.0	(319632,[13,43,47...
1.0	(319632,[13,43,47...
1.0	(319632,[13,42,43...
1.0	(319632,[13,42,43...

```
# split the data into training and test sets
training_data, test_data = aws_assembled.randomSplit([0.7, 0.3], seed
= 42)

# create a logistic regression estimator
lr = LogisticRegression(maxIter = 100)

# create a grid to hold hyperparameters
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])

# build the parameter grid
grid = grid.build()

print('Number of models to be tested:', len(grid))
```

```
>>> print('Number of models to be tested:', len(grid))
Number of models to be tested: 18
```

```
# create a binary classification evaluator to evaluate how well model
works
evaluator = BinaryClassificationEvaluator(metricName =
'areaUnderROC')

# create the cross validator using the hyperparameter grid
cv = CrossValidator(estimator = lr,
      estimatorParamMaps = grid,
      evaluator = evaluator,
      numFolds = 3,
      seed = 42)

# train the model
cv = cv.fit(training_data)
cv.avgMetrics

# test the predictions
```

```
predictions = cv.transform(test_data)

# calculate auc
auc = evaluator.evaluate(predictions)
print('AUC:', auc)
```

```
>>> print('AUC:', auc)
AUC: 0.8930547253759209
```

```
# create confusion matrix
predictions.groupby('label').pivot('prediction').count().fillna(0).sh
ow()
```

```
>>> predictions.groupby('label').pivot('prediction').count().show()
+-----+------+-------+
|label|   0.0|    1.0|
+-----+------+-------+
|  1.0| 89320|8866646|
|  0.0|557634|1808670|
+-----+------+-------+
```

```
cm =
predictions.groupby('label').pivot('prediction').count().fillna(0).co
llect()

def calculate_precision_recall(cm):
      tn = cm[1][1]
      fp = cm[1][2]
      fn = cm[0][1]
      tp = cm[0][2]
      precision = tp / (tp + fp)
      recall = tp / (tp + fn)
      accuracy = (tp + tn) / (tp + tn + fp + fn)
      f1_score = 2 * ((precision * recall) / (precision + recall))
      return accuracy, precision, recall, f1_score
```

```
print("The model accuracy is:", calculate_precision_recall(cm)[0])
print("The model precision is:", calculate_precision_recall(cm)[1])
print("The model recall is:", calculate_precision_recall(cm)[2])
print("The model f1_score is:", calculate_precision_recall(cm)[3])
```

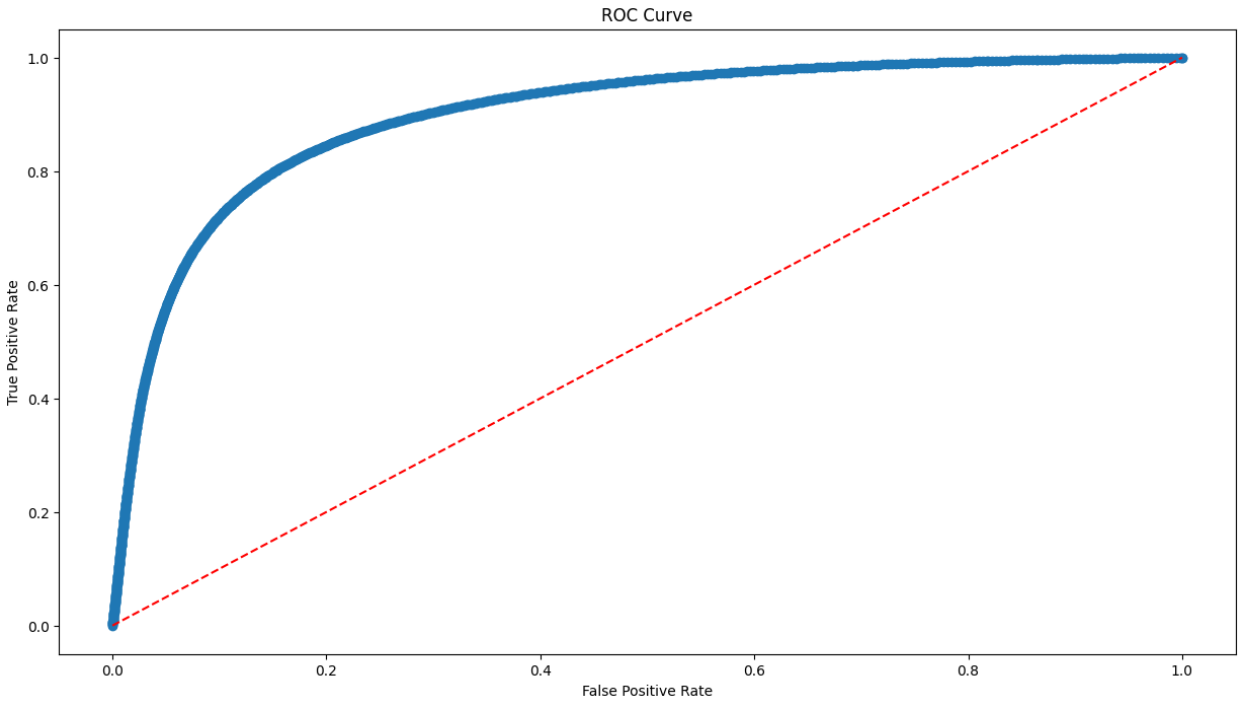
```
>>> print("The model accuracy is:", calculate_precision_recall(cm)[0])
The model accuracy is: 0.8323666543899766
>>> print("The model precision is:", calculate_precision_recall(cm)[1])
The model precision is: 0.8305745703452713
>>> print("The model recall is:", calculate_precision_recall(cm)[2])
The model recall is: 0.9900267598157474
>>> print("The model f1_score is:", calculate_precision_recall(cm)[3])
The model f1_score is: 0.9033180818247122
```

```
parammap = cv.bestModel.extractParamMap()

for p, v in parammap.items():
     print(p, v)
```

```
LogisticRegression_169a83480b4a__aggregationDepth 2
LogisticRegression_169a83480b4a__elasticNetParam 0.0
LogisticRegression_169a83480b4a__family auto
LogisticRegression_169a83480b4a__featuresCol features
LogisticRegression_169a83480b4a__fitIntercept True
LogisticRegression_169a83480b4a__labelCol label
LogisticRegression_169a83480b4a__maxBlockSizeInMB 0.0
LogisticRegression_169a83480b4a__maxIter 100
LogisticRegression_169a83480b4a__predictionCol prediction
LogisticRegression_169a83480b4a__probabilityCol probability
LogisticRegression_169a83480b4a__rawPredictionCol rawPrediction
LogisticRegression_169a83480b4a__regParam 0.2
LogisticRegression_169a83480b4a__standardization True
LogisticRegression_169a83480b4a__threshold 0.5
LogisticRegression_169a83480b4a__tol 1e-06
```

```
# grab the best model
mymodel = cv.bestModel

plt.figure(figsize = (15,8))
plt.plot([0,1], [0,1], 'r--')
```

```
x = mymodel.summary.roc.select('FPR').collect()
y = mymodel.summary.roc.select('TPR').collect()
plt.scatter(x, y)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")

# create a buffer to hold the figure
img_data = io.BytesIO()
plt.savefig(img_data, format = "png", bbox_inches='tight')
img_data.seek(0)

# connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon = False)
with s3.open('s3://my-data-bucket-sd/review_roc_curve.png', 'wb') as
f:
      f.write(img_data.getbuffer())
```



This graph illustrate the ROC curve of the best model after running the GridSearch for Logistic Regression. We can see that most of the areas are under the curve—about 89%.

Summary of the project and the main conclusions drawn from the data

Logistic Regression is used to classify whether the star rating is <=3 or >3 using predictors such as the review body text, helpful_votes, product_categories, verified_purchase, and word counts. Hyperparameter tuning is implemented by setting regParam = [0, 0.2, 0.4, 0.6, 0.8, 1] and elasticNetParam = [0, 0.5, 1]. The total number of models being tested is 18.

Our best model has an elasticNetParam of 0 and regParam of 0.2. That means that our model is reduced to a ridge regression model. The best model after hyperparameter tuning has an AUC of 0.89 with an accuracy of 0.83, a precision of 0.83, a recall of 0.99, and an f1 score of 0.9. This means that this model has a good performance on recall—the model can classify 99% of all the >3 ratings correctly. If we want to focus on predicting >3 ratings correctly then this would be a good model for that. If we want to focus on getting higher correct prediction for all predicted positive samples then we want to have higher precision. Depending on what purpose we want to accomplish we can modify and/or improve our model.

Some future improvements include

1. Try different machine learning models and hyperparameter tuning (e.g. Random Forest Classifier and/or Multinomial Naive Bayes)
2. Increase the sample size, but that would also increase the computational cost
3. Try different data preprocessing methods for text data (e.g. Lemmatization)

References

Amazon Customer Reviews Dataset. s3.amazonaws.com/amazon-reviews-pds/readme.html. Accessed 8 Sept. 2022.

"Basic Statistics." *Basic Statistics - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/ml-statistics.html.

"Classification and Regression." *Classification and Regression - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/ml-classification-regression.html.

"Extracting, Transforming and Selecting Features." *Extracting, Transforming and Selecting Features - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/ml-features.

Huilgol, Purva. "Precision vs Recall: Precision and Recall Machine Learning." *Analytics Vidhya*, https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/.

"ML - Linear Methods." *Linear Methods - ML - Spark 1.5.2 Documentation*, https://spark.apache.org/docs/1.5.2/ml-linear-methods.html.

"ML Pipelines." *ML Pipelines - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/ml-pipeline.html.

"ML Tuning: Model Selection and Hyperparameter Tuning." *ML Tuning - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/ml-tuning.html.

"Parquet Files." *Parquet Files - Spark 3.3.1 Documentation*, https://spark.apache.org/docs/latest/sql-data-sources-parquet.html.

"PySpark Tutorial for Beginners: Python Examples." *Spark by {Examples}*, 25 Aug. 2022, https://sparkbyexamples.com/pyspark-tutorial/.

"What Is Logistic Regression?" *IBM*, https://www.ibm.com/topics/logistic-regression.