

# Proiect BigData – Amazon Product Reviews

Toma Sabin-Sebastian 412 – Group 412

Link dataset: <https://www.kaggle.com/datasets/arhamrumi/amazon-product-reviews>

## 1. Dataset

Setul de date utilizat în acest proiect conține **recenzii de produse de pe Amazon**, incluzând informații atât despre produs, cât și despre opinia utilizatorului. Fiecare rând din setul de date reprezintă o recenzie și este descris prin următoarele coloane:

- Id – identificatorul unic al recenziei
- ProductId – codul unic al produsului recenzat
- UserId – identificatorul utilizatorului care a lăsat recenzie
- ProfileName – numele public al utilizatorului
- HelpfulnessNumerator / HelpfulnessDenominator – scorul de utilitate al recenziei (voturi pozitive / totale)
- Score – **scorul recenziei**, de la 1 la 5 (obiectivul de clasificare în proiect)
- Time – timestamp-ul recenziei
- Summary – un titlu scurt oferit de utilizator (ex: „Great taffy”)
- Text – textul complet al recenziei (opinia detaliată)

În cadrul acestui proiect, vom folosi doar **coloanele Summary, Text și Score**, celelalte nefiind necesare pentru sarcina de clasificare.

## 2. Obiectivele proiectului

Obiectivul principal al acestui proiect este de a construi un **sistem automat de clasificare a recenziilor în funcție de scor**, folosind metode de **Machine Learning (ML)** și **Deep Learning (DL)** aplicate pe conținutul textual (Summary și Text). Proiectul va urmări:

1. **Preprocesarea și curățarea datelor textuale** cu ajutorul Spark.

2. **Aplicarea a două metode ML** (ex: Naïve Bayes Logistic Regression – via Spark MLlib) pentru a prezice Score.
3. **Aplicarea unei metode DL** (ex: LSTM) cu TensorFlow pentru a compara rezultatele cu metodele clasice ML.
4. **Crearea unui pipeline de procesare.**
5. **Evaluarea performanței** fiecărei metode prin metrici relevante: accuracy, precision, recall și F1-score.

Scopul final este de a determina în ce măsură este posibilă prezicerea corectă a unui scor de review doar pe baza conținutului scris de utilizator.

### 3. Preprocesare și curățare dataset

S-a realizat preprocesarea unui set de date cu recenzii Amazon, începând cu citirea fișierului Reviews.csv folosind biblioteca pandas. S-a convertit coloana Time într-un format de dată lizibil (ReadableTime), s-au eliminat valorile lipsă și scorurile invalide, păstrând doar recenziile cu scoruri între 1 și 5. A fost implementată o funcție de curățare a textului pentru coloanele Summary și Text, care include conversia la litere mici, eliminarea link-urilor, semnelor de punctuație și a cuvintelor comune (stopwords), precum și lematizarea cu ajutorul pachetului nltk. Scopul acestor pași a fost obținerea unui text curat și uniform pentru a putea fi folosit în mod eficient în clasificarea scorului recenziei. La final, setul de date curățat a fost salvat într-un fișier nou (newReviews.csv), ce va fi folosit pentru antrenarea modelelor de Machine Learning și Deep Learning.

```
import pandas as pd
```

```
import re
```

```
import nltk
```

```
from nltk.corpus import stopwords
```

```
from nltk.stem import WordNetLemmatizer
```

```
import os
```

```
nltk.download('stopwords')
```

```

nltk.download('punkt')

nltk.download('wordnet')

df = pd.read_csv('date/Reviews.csv')

df['ReadableTime'] = pd.to_datetime(df['Time'], unit='s')

df_cleaned = df.dropna()

df_cleaned.drop(columns=['Time'], inplace=True)

df['Score'] = pd.to_numeric(df['Score'], errors='coerce')

rows_over_5 = df[df['Score'] > 5]

print(rows_over_5[['Id', 'Score', 'Summary']].head(10))

print(f"Total rows with Score > 5: {len(rows_over_5)}")

df_cleaned['Score'] = pd.to_numeric(df_cleaned['Score'], errors='coerce')

df_cleaned = df_cleaned[df_cleaned['Score'].between(1, 5)]

df_cleaned['Score'] = df_cleaned['Score'].astype(int)

lemmatizer = WordNetLemmatizer()

stop_words = set(stopwords.words('english'))

def preprocess_text(text):

    text = str(text).lower()           # lowercase

    text = re.sub(r"http\S+|www\S+", "", text) # eliminare link-uri

    text = re.sub(r"^[^\w\s]", "", text)      # eliminare punctuație

    tokens = nltk.word_tokenize(text)        # tokenizare

    tokens = [t for t in tokens if t not in stop_words] # eliminare stopwords

    tokens = [lemmatizer.lemmatize(t) for t in tokens] # lematizare

    return " ".join(tokens)

df_cleaned['Summary'] = df_cleaned['Summary'].apply(preprocess_text)

df_cleaned['Text'] = df_cleaned['Text'].apply(preprocess_text)

```

```
print(df_cleaned['Score'].value_counts().sort_index())
```

```
Score
1      52264
2      29743
3      42638
4      80654
5     363102
Name: count, dtype: int64
```

```
df_cleaned.to_csv('date/newReviews.csv', index=False)
```

```
print("PANDAS is saving to:", os.path.abspath("date/ReviewsCleaned.csv"))
```

#### 4. Grupări și agregări de date – DataFrames și SparkSQL

- **Inițializarea Spark și citirea datelor:** Se creează o sesiune Spark și se definește o schemă explicită pentru citirea fișierului **ReviewsCleaned.csv**. Această schemă include coloane precum **Id**, **UserId**, **Score**, **Summary**, **Text** și **ReadableTime**, iar datele sunt citite cu **spark.read.csv()**.
- **Inspectarea structurii și a conținutului:** Cu **printSchema()** se verifică tipurile de date ale coloanelor. Se afișează primele 5 rânduri din DataFrame pentru a confirma citirea corectă.
- **Analiză: Top utilizatori activi:** Se grupează recenziile după **UserId** și se numără **câte recenzii a lăsat fiecare utilizator**. Rezultatul este sortat **descrescător** și se afișează **primii 10 utilizatori cei mai activi**.
- **Analiză: Top 5 profiluri după ProfileName (cu Spark SQL):** Se creează o view temporară denumită **reviews** pentru a permite interogări SQL directe. Se rulează o **interogare SQL** care calculează **câte recenzii a scris fiecare ProfileName**, sortate **descrescător**.
- **Conversie și filtrare temporală:** Coloana **ReadableTime (string)** este convertită într-un tip de dată real (**DateType**) și salvată într-o coloană nouă numită **ReadableDate**. Se filtrează toate recenziile care au fost scrise **după data de 01-01-2011**. Rezultatele sunt afișate folosind **show(truncate=False)** pentru a nu tăia conținutul lung.

- **Interogare SQL echivalentă pentru filtrare temporală:** Se recrează view-ul reviews pentru a include noua coloană (ReadableDate). Se rulează o interogare SQL echivalentă care selectează toate recenziile postate după 1 ianuarie 2011.

```
from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, StringType, IntegerType

from pyspark.sql.functions import col, avg

from pyspark.sql.functions import to_date, date_format

import os

print(os.path.abspath("date/ReviewsCleaned.csv"))

spark = SparkSession.builder \
    .appName("Review analysis") \
    .getOrCreate()

schema = StructType([
    StructField("Id", IntegerType(), True),
    StructField("ProductId", StringType(), True),
    StructField("UserId", StringType(), True),
    StructField("ProfileName", StringType(), True),
    StructField("HelpfulnessNumerator", IntegerType(), True),
    StructField("HelpfulnessDenominator", IntegerType(), True),
    StructField("Score", IntegerType(), True), # this is key
    StructField("Summary", StringType(), True),
    StructField("Text", StringType(), True),
    StructField("ReadableTime", StringType(), True),
])

df = spark.read.csv("date/ReviewsCleaned.csv", header=True, schema=schema)

df.printSchema()

df.show(5)
```

```

root
|-- Id: integer (nullable = true)
|-- ProductId: string (nullable = true)
|-- UserId: string (nullable = true)
|-- ProfileName: string (nullable = true)
|-- HelpfulnessNumerator: integer (nullable = true)
|-- HelpfulnessDenominator: integer (nullable = true)
|-- Score: integer (nullable = true)
|-- Summary: string (nullable = true)
|-- Text: string (nullable = true)
|-- ReadableTime: string (nullable = true)

+-----+-----+-----+-----+-----+-----+-----+-----+
| Id| ProductId|      UserId|      ProfileName|HelpfulnessNumerator|HelpfulnessDenominator|Score|      Summary|      Text|ReadableTime|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1|B001E4KF00|A3SGXH7AUHU8GW|delmartian|1|5|good quality dog ...|bought several vi...| 2011-04-27| |
| 2|B00813GRG4|A1D87F6ZCVE5NK|dl1 pa|0|1|advertised|product arrived l...| 2012-09-07|
| 3|B000LQ0CH0|ABXLMWJDXAIN|"Natalia Corres "...|1|1|4|delight say|confection around...| 2008-08-18|
| 4|B000UA00TQ|A395BORC6FGVXV|Karl|3|2|cough medicine|looking secret in...| 2011-06-13|
| 5|B006K2ZZ7K|A1UQRSCLF8GW1T|"Michael D. Bigha...|0|5|great taffy|great taffy great...| 2012-10-21|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```
df.groupBy("UserId").count().orderBy("count", ascending=False).show(10)
```

```

+-----+-----+
|      UserId|count|
+-----+-----+
|A30XHLG6DIBRW8| 448|
|A1YUL9PCJR3JTY| 421|
|AY12DBB0U420B| 389|
|A281NPSIMI1C2R| 365|
|A1Z54EM24Y40LL| 256|
|A1TMAVN4CEM8U8| 204|
|A2MUGFV2TDQ47K| 201|
|A3TVZM3ZIXG8YW| 199|
|A3PJZ8TU8FDQ1K| 178|
|AQQLWCMRNDFGI| 176|
+-----+-----+
only showing top 10 rows

```

```
df.createOrReplaceTempView("reviews")
```

```
spark.sql("""
```

```
    SELECT ProfileName, COUNT(*) as num_reviews
```

```
    FROM reviews
```

```
    GROUP BY ProfileName
```

```
    ORDER BY num_reviews DESC
```

```
    LIMIT 5
```

```
""").show()
```

ProfileName	num_reviews
"C. F. Hill ""CFH""	451
"O. Brown ""Ms. O..."	421
Gary Peterson	389
"Rebecca of Amazo..."	365
Chris	363

```
df = df.withColumn("ReadableDate", to_date(col("ReadableTime")))
```

```
# Filtrare după 01-01-2011
```

```
df_filtered = df.filter(col("ReadableDate") > "2011-01-01")
```

```
# Afișare rezultate
```

```
df_filtered.show(truncate=False)
```

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary	Text
1	B001E4KFG0	A3SGH7AUH8GM	delmartian	1	1	5	good quality dog food	bought several vitality canned dog food product
2	B00813GRG4	A1D87F6ZCVE5NK	d11 pa	0	0	1	advertised	product arrived labeled jumbo salted peanuts
4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	cough medicine	looking secret ingredient robittussin believe
5	B006K2Z77K	A1UQRSCLF8GW1T	"Michael D. Bigham ""M. Wassir""	0	0	5	great taffy	great taffy great price wide assortment yummy
6	B006K2Z77K	ADT0SRK1MG0EU	Twoapennything	0	0	4	nice taffy	got wild hair taffy ordered five pound bag
7	B006K2Z77K	A1SP2KVKFXXRUI	David C. Sullivan	0	0	5	great good expensive brand	saltwater taffy great flavor soft chewy candy
8	B006K2Z77K	A3JRGQVEQN31IQ	Pamela G. Williams	0	0	5	wonderful tasty taffy	taffy good soft chewy flavor amazing would
9	B000E7L2R4	A1MZY09TZK0BBI	R. James	1	1	5	lay barley	right im mostly sprouting cat eat grass love
10	B00171APVA	A21BT40VZCCYT4	Carol A. Reed	0	0	5	healthy dog food	healthy dog food good digestion also good
13	B0009XLVG0	A327PCT23YH90	LT	1	1	1	cat fan new food	cat happily eating felidae platinum two year
17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	0	2	poor taste	love eating good watching tv looking movie
18	B001GVISJM	AFKW14U97Z6Q0	Becca	0	0	5	love	satisfied twizzler purchase shared others
19	B001GVISJM	A2A9X58G2GTBLP	Wolfee1	0	0	5	great sweet candy	twizzlers strawberry childhood favorite
20	B001GVISJM	A3IV7CL2C13K2U	Greg	0	0	5	home delivered twizzlers	candy delivered fast purchased reasonable
21	B001GVISJM	A1M0KGLPR5PV6	mom2emma	0	0	5	always fresh	husband twizzlers addict weve bought many
22	B001GVISJM	AZOF9E17RGZH8	Tammy Anderson	0	0	5	twizzlers	bought husband currently overseas love
23	B001GVISJM	ARYVQL4N737A1	Charles Brown	0	0	5	delicious product	remember buying candy kid quality hasnt
24	B001GVISJM	AJ6130LZZUG7V	Mare's	0	0	5	twizzlers	love candy weight watcher cut back still
25	B001GVISJM	A22P2J09NJ9HKE	"S. Cabanaugh ""jilly pepper""	0	0	5	please sell mexico	lived u 7 yr miss twizzlers go back visit
27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	1	1	nasty flavor	candy red flavor plan chewy would never

```
df.createOrReplaceTempView("reviews")
```

```
spark.sql("""
```

```
SELECT *
```

```
FROM reviews
```

```
WHERE to_date(ReadableTime) > '2011-01-01'
```

```
""").show(truncate=False)
```

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Summary	Text
1	B001E4KFG0	A3SGXH7AUHU8GM	delmartian	1	1	5	good quality dog food	bought several vitality canned dog food product
2	B00813GRG4	A1D87F6ZCVESNK	dll pa	0	0	1	advertised	product arrived labeled jumbo salted peanutsthe
4	B000UA0Q1Q	A395B0RC6FGVXV	Kar1	3	3	2	cough medicine	looking secret ingredient robitussin believe fou
5	B006K2Z27K	A1UQRSCLF8GW1T	"Michael D. Bigham ""M. Wassir""	0	0	5	great taffy	great taffy great price wide assortment yummy ta
6	B006K2Z27K	ADT0SRK1MGOEU	Twoapennything	0	0	4	nice taffy	got wild hair taffy ordered five pound bag taffy
7	B006K2Z27K	A1SP2KVKFXRU1	David C. Sullivan	0	0	5	great good expensive brand	saltwater taffy great flavor soft chewy candy in
8	B006K2Z27K	A3JRGQVEQN31IQ	Pamela G. Williams	0	0	5	wonderful tasty taffy	taffy good soft chewy flavor amazing would defin
9	B000E7L2R4	A1MZY09TZK0BBI	R. James	1	1	5	yay barley	right im mostly sprouting cat eat grass love rot
10	B00171APVA	A21BT40VZCCY14	Carol A. Reed	0	0	5	healthy dog food	healthy dog food good digestion also good small
13	B0009XLVG0	A327PCT23YH90	LT	1	1	1	cat fan new food	cat happily eating felidae platinum two year got
17	B001GVISJM	A3KLNFMQ5BHY0	Erica Neathery	0	0	2	poor taste	love eating good watching tv looking movie sweet
18	B001GVISJM	AFK0L14U97Z6Q0	Becca	0	0	5	love	satisfied twizzler purchase shared others enjoye
19	B001GVISJM	A2A9X58G2GTBLP	Wolfee1	0	0	5	great sweet candy	twizzlers strawberry childhood favorite candy ma
20	B001GVISJM	A3IV7CL2C13K2U	Greg	0	0	5	home delivered twizlers	candy delivered fast purchased reasonable price
21	B001GVISJM	A1W00KGLPR5PV6	mom2emma	0	0	5	always fresh	husband twizzlers addict weve bought many time a
22	B001GVISJM	AZOF9E17RGZH8	Tammy Anderson	0	0	5	twizzlers	bought husband currently overseas love apparentl
23	B001GVISJM	ARYVQL4N737A1	Charles Brown	0	0	5	delicious product	remember buying candy kid quality hasnt dropped
24	B001GVISJM	AJ6130LZZUG7V	Mare's	0	0	5	twizzlers	love candy weight watcher cut back still craving
25	B001GVISJM	A22P2J09ND9HKE	"S. Cabanaugh ""jilly pepper""	0	0	5	please sell mexico	lived u 7 yr miss twizzlers go back visit someon
27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	1	1	nasty flavor	candy red flavor plan chewy would never buy

only showing top 20 rows

## 5. Dataset split și enunțul problemei

Va urma să discut despre metodele ML in cadrul Spark. Înainte de a vorbi de primul model, am împărțit dataset-ul în 80% train și 20% test.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Încărcare dataset
df = pd.read_csv('date/ReviewsCleaned.csv')

# Split 80% train, 20% test
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Salvare în fișiere noi
train_df.to_csv('date/train.csv', index=False)
test_df.to_csv('date/test.csv', index=False)

print("Split realizat și fișiere salvate ca 'date/train.csv' și 'date/test.csv'")
```

Se dorește construirea unui model de învățare automată care să clasifice recenziile de produse Amazon pe baza conținutului textual. Mai exact, modelul trebuie să învețe să prezică scorul (de la 1 la 5 stele) acordat de un utilizator unui produs, folosind textul recenziei (rezumatul și descrierea completă). Clasificarea scorurilor ajută la automatizarea evaluării sentimentului și satisfacției utilizatorilor pe platforme de e-commerce.

## 6. Metode ML – LogisticRegression, pipeline, optimizarea hiperparametrilor

- **Inițializare Spark și citirea datelor:** Se pornește o sesiune Spark. Se citesc două fișiere CSV: unul pentru **antrenare** (train.csv) și unul pentru **testare** (test.csv), care conțin recenzii de produse Amazon.



- **Preprocesare text:** Coloanele **Summary** și **Text** sunt combinate într-o coloană nouă `combined_text`, care reprezintă textul complet al recenziei. Se aplică preprocesare NLP: `Tokenizer`: împarte textul în cuvinte. `StopWordsRemover`: elimină cuvintele comune (stopwords) care nu adaugă valoare semantică. `HashingTF`: transformă cuvintele într-un vector numeric folosind hashing (TF = term frequency). `IDF`: ajustează frecvențele cu Inverse Document Frequency, pentru a reduce importanța cuvintelor frecvente.
- **Etichetare și model ML:** `StringIndexer` transformă scorurile (ex: 1–5) într-un format numeric (label) necesar pentru modelul ML. Se folosește un **model de regresie logistică** (`LogisticRegression`), configurat cu: `maxIter=25`: numărul maxim de iterații, `regParam=0.001`: regularizare L2, `elasticNetParam=0.7`: combină L1 și L2. (una dintre rulări)
- **Crearea pipeline-ului complet:** Se construiește un **Pipeline** care leagă toți pașii într-un flux unitar: preprocesare text → extragere caracteristici → clasificare.
- **Antrenare și testare:** Pipeline-ul este antrenat pe datele de antrenament (`fit(train_df)`). Modelul este aplicat pe setul de test pentru a obține predicții (`transform(test_df)`).
- **Evaluare:** Se evaluează performanța modelului folosind metrice standard: **accuracy** – acuratețea generală, **precision** – precizia medie ponderată, **recall** – capacitatea de a regăsi clasele corecte, **f1-score** – media armonică între precizie și recall. Valorile metrice sunt afișate la final pentru interpretare.
- **Oprire Spark:** Sesiunea Spark este închisă la final cu `spark.stop()` pentru eliberarea resurselor.

```
from pyspark.sql import SparkSession
```

```
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF, StringIndexer
```

```
from pyspark.ml.classification import LogisticRegression
```

```
from pyspark.ml import Pipeline
```

```
from pyspark.sql.functions import col
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
from pyspark.sql.functions import concat_ws
```

```

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# 1. Inițializare sesiune Spark

spark = SparkSession.builder.appName("ReviewScoreClassification").getOrCreate()

# 2. Citire date CSV

train_df = spark.read.csv("date/train.csv", header=True, inferSchema=True)

test_df = spark.read.csv("date/test.csv", header=True, inferSchema=True)

# 3. Combinare 'Text' și 'Summary' într-o singură coloană

train_df = train_df.withColumn("combined_text", concat_ws(" ", "Summary", "Text"))

test_df = test_df.withColumn("combined_text", concat_ws(" ", "Summary", "Text"))

# 4. Preprocesare NLP

tokenizer = Tokenizer(inputCol="combined_text", outputCol="words")

remover = StopWordsRemover(inputCol="words", outputCol="filtered")

hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures",
numFeatures=10000)

idf = IDF(inputCol="rawFeatures", outputCol="features")

# 5. Pregătire label și model

label_indexer = StringIndexer(inputCol="Score", outputCol="label", handleInvalid="skip")

lr = LogisticRegression(maxIter=25, regParam=0.001, elasticNetParam=0.7)

# 6. Pipeline complet

pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, idf, label_indexer, lr])

# 7. Antrenare model

model = pipeline.fit(train_df)

# 8. Predicții pe test

predictions = model.transform(test_df)

# 9. Evaluare

```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",  
metricName="accuracy")
```

```
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction", metricName="accuracy")
```

```
evaluator_precision = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction", metricName="weightedPrecision")
```

```
evaluator_recall = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction", metricName="weightedRecall")
```

```
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction", metricName="f1")
```

```
accuracy = evaluator_accuracy.evaluate(predictions)
```

```
precision = evaluator_precision.evaluate(predictions)
```

```
recall = evaluator_recall.evaluate(predictions)
```

```
f1 = evaluator_f1.evaluate(predictions)
```

```
print("=== Evaluation Metrics ===")
```

```
print(f"Accuracy : {accuracy:.4f}")
```

```
print(f"Precision : {precision:.4f}")
```

```
print(f"Recall : {recall:.4f}")
```

```
print(f"F1 Score : {f1:.4f}")
```

```
spark.stop()
```

```

1  lr = LogisticRegression(maxIter=20, regParam=0.001)
2  === Evaluation Metrics ===
3  Accuracy      : 0.7497
4  Precision     : 0.7193
5  Recall        : 0.7497
6  F1 Score      : 0.7249
7
8  lr = LogisticRegression(maxIter=20, regParam=0.001, elasticNetParam=0.8)
9  === Evaluation Metrics ===
10 Accuracy      : 0.7348
11 Precision     : 0.6914
12 Recall        : 0.7348
13 F1 Score      : 0.6919
14
15 lr = LogisticRegression(maxIter=10, regParam=0.001, elasticNetParam=0.8)
16 === Evaluation Metrics ===
17 Accuracy      : 0.7312
18 Precision     : 0.6874
19 Recall        : 0.7312
20 F1 Score      : 0.6854

```

```

22 lr = LogisticRegression(maxIter=10, regParam=0.01, elasticNetParam=0.8)
23 === Evaluation Metrics ===
24 Accuracy      : 0.6777
25 Precision     : 0.6199
26 Recall        : 0.6777
27 F1 Score      : 0.5935
28
29 lr = LogisticRegression(maxIter=50, regParam=0.01, elasticNetParam=0.8)
30 === Evaluation Metrics ===
31 Accuracy      : 0.6795
32 Precision     : 0.6198
33 Recall        : 0.6795
34 F1 Score      : 0.5975
35
36 lr = LogisticRegression(maxIter=25, regParam=0.001, elasticNetParam=0.8)
37 === Evaluation Metrics ===
38 Accuracy      : 0.7359
39 Precision     : 0.6929
40 Recall        : 0.7359
41 F1 Score      : 0.6933
42
43 lr = LogisticRegression(maxIter=25, regParam=0.001, elasticNetParam=0.7)
44 === Evaluation Metrics ===
45 Accuracy      : 0.7377
46 Precision     : 0.6957
47 Recall        : 0.7377
48 F1 Score      : 0.6964

```

## 7. Metode ML – Naïve Bayes, pipeline, optimizarea hiperparametrilor

- **Etichetare și modelare:** Coloana Score, care conține evaluările sub formă de valori de la 1 la 5, este transformată într-o coloană label numerică, folosind StringIndexer, pentru a fi compatibilă cu algoritmul de clasificare. Se utilizează algoritmul **Naïve Bayes**, configurat în varianta multinomial, potrivită pentru text, cu un parametru de netezire (smoothing=0.9) care ajută la gestionarea cuvintelor rare.
- **Construirea pipeline-ului:** Toate etapele — de la tokenizare, eliminarea stopwords, transformarea în vectori numerici și până la clasificare — sunt grupate într-un **pipeline Spark ML**. Acesta permite rularea întregului proces ca un flux continuu, atât pentru antrenare, cât și pentru testare.
- **Antrenarea modelului:** Modelul Naïve Bayes este antrenat folosind datele din fișierul train.csv, prin aplicarea pipeline-ului pe setul de date de antrenament. Astfel, modelul învață relațiile dintre cuvintele din recenzie și scorul acordat.
- **Predicție și evaluare:** Modelul antrenat este aplicat pe setul test.csv pentru a genera predicții asupra datelor noi. Performanța este evaluată utilizând patru metrici clasice: **accuracy** (procentul total de clasificări corecte), **precision** (procentul de predicții corecte dintre cele pozitive), **recall** (procentul de exemple corecte identificate) și **F1-score** (media armonică dintre precision și recall). Aceste rezultate oferă o imagine clară asupra eficienței modelului Naïve Bayes în clasificarea recenziilor Amazon pe baza textului.

*# 1. Spark session*

```
spark = SparkSession.builder.appName("ReviewScoreClassification_NB").getOrCreate()
```

*# 2. Load data*

```
train_df = spark.read.csv("data/train.csv", header=True, inferSchema=True)
```

```
test_df = spark.read.csv("data/test.csv", header=True, inferSchema=True)
```

*# 3. Combine 'Text' and 'Summary'*

```
train_df = train_df.withColumn("combined_text", concat_ws(" ", "Summary", "Text"))
```

```
test_df = test_df.withColumn("combined_text", concat_ws(" ", "Summary", "Text"))
```

*# 4. NLP preprocessing*

```
tokenizer = Tokenizer(inputCol="combined_text", outputCol="words")
```

```
remover = StopWordsRemover(inputCol="words", outputCol="filtered")
```

```
hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures",  
numFeatures=10000)
```

*# Note: Naïve Bayes expects raw term frequencies (TF), not TF-IDF.*

```

# So we skip IDF and feed 'rawFeatures' directly to the classifier.

# 5. Label encoding
label_indexer = StringIndexer(inputCol="Score", outputCol="label", handleInvalid="skip")

# 6. Classifier: Naive Bayes
nb = NaiveBayes(labelCol="label", featuresCol="rawFeatures", modelType="multinomial",
smoothing=0.9)

# 7. Pipeline
pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, label_indexer, nb])

# 8. Train model
model = pipeline.fit(train_df)

# 9. Predictions
predictions = model.transform(test_df)

# 10. Evaluation
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
evaluator_precision = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedPrecision")
evaluator_recall = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedRecall")
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="f1")

accuracy = evaluator_accuracy.evaluate(predictions)
precision = evaluator_precision.evaluate(predictions)
recall = evaluator_recall.evaluate(predictions)
f1 = evaluator_f1.evaluate(predictions)

print("=== Evaluation Metrics (Naive Bayes) ===")
print(f"Accuracy      : {accuracy:.4f}")
print(f"Precision     : {precision:.4f}")
print(f"Recall        : {recall:.4f}")
print(f"F1 Score      : {f1:.4f}")

```

```

nb = NaiveBayes(labelCol="label", featuresCol="rawFeatures", modelType="multinomial")
=== Evaluation Metrics (Naive Bayes) ===
Accuracy      : 0.6906
Precision     : 0.6845
Recall        : 0.6906
F1 Score      : 0.6870

nb = NaiveBayes(labelCol="label", featuresCol="rawFeatures", modelType="multinomial", smoothing=0.7)
=== Evaluation Metrics (Naive Bayes) ===
Accuracy      : 0.6907
Precision     : 0.6848
Recall        : 0.6907
F1 Score      : 0.6872

nb = NaiveBayes(labelCol="label", featuresCol="rawFeatures", modelType="multinomial", smoothing=0.9)
=== Evaluation Metrics (Naive Bayes) ===
Accuracy      : 0.6908
Precision     : 0.6847
Recall        : 0.6908
F1 Score      : 0.6872

```

## 8. Metodă DL – RNN (LSTM)

- **Citirea fișierelor train.csv și test.csv:** Codul începe prin citirea celor două fișiere CSV care conțin recenziile Amazon, deja împărțite în seturi de antrenare și testare. Este utilizat motorul de citire "python" pentru a gestiona în mod sigur eventuale anomalii în structura CSV-ului (precum virgule în interiorul textului).
- **Combinarea textului (Summary și Text):** Pentru fiecare recenzie, câmpurile Summary (titlul recenziei) și Text (conținutul complet) sunt combinate într-o coloană nouă denumită combined\_text. Această coloană este folosită ca input pentru modelul de procesare a limbajului.
- **Curățarea scorurilor (Score):** Se transformă coloana Score în valori numerice, eliminând eventualele valori lipsă sau invalide. Ulterior, se filtrează doar recenziile care au scoruri între 1 și 5, întrucât acestea reprezintă clasificarea relevantă pentru problemă.
- **Reactualizarea dataframe-urilor:** Deoarece filtrarea anterioară asupra scorurilor s-a făcut pe o copie temporară (`df = df[...]`), se reaplică corect filtrarea pe `train_df` și `test_df` pentru a actualiza datele efectiv utilizate.

- **Codificarea etichetelor:** Scorurile (1–5) sunt transformate în etichete numerice (0–4) prin LabelEncoder. Această conversie este necesară pentru ca modelul de clasificare să poată lucra cu etichete numerice.
- **Tokenizarea textului:** Se creează un Tokenizer Keras care transformă textul în secvențe de indici, pe baza celor mai frecvente 10.000 de cuvinte. Acesta este antrenat pe textul din setul de antrenare, pentru a construi un vocabular comun.
- **Transformarea textului în secvențe numerice + padding:** Atât datele de antrenament, cât și cele de testare sunt convertite în secvențe de indici (cuvinte -> numere) și apoi completate (padded) la aceeași lungime (200), pentru a putea fi introduse în rețea.
- **Definirea modelului LSTM:** Se construiește o rețea neuronală secvențială care conține: un strat de embedding (care transformă indici în vectori semantici), un strat LSTM (care învață dependențele din secvență), un strat dropout (pentru prevenirea overfitting-ului), un strat dens intermediar și unul final (cu funcție softmax pentru clasificare multi-clasă).
- **Compilarea și antrenarea modelului:** Modelul este compilat cu funcția de pierdere sparse\_categorical\_crossentropy, optimizatorul adam și este antrenat timp de 5 epoci cu un batch size de 128. O porțiune de 10% din datele de antrenament este rezervată pentru validare.
- **Evaluarea performanței:** După antrenare, modelul este testat pe datele de test, iar predicțiile sunt comparate cu valorile reale folosind classification\_report() din scikit-learn. Sunt afișate metricele pentru fiecare clasă (precizie, recall, F1-score), precum și scorurile agregate.

```
import pandas as pd
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense
```

```
from sklearn.metrics import classification_report
```



```

import numpy as np

# 1. Citire fişierele

train_df = pd.read_csv("date/train.csv", engine="python")

test_df = pd.read_csv("date/test.csv", engine="python")

# 2. Combinare 'Summary' şi 'Text' în 'combined_text'

for df in [train_df, test_df]:

    df["Summary"] = df["Summary"].fillna("")

    df["Text"] = df["Text"].fillna("")

    df["combined_text"] = df["Summary"].astype(str) + " " + df["Text"].astype(str)

# 3. Curăţare scoruri (păstrăm doar 1-5)

for df in [train_df, test_df]:

    df["Score"] = pd.to_numeric(df["Score"], errors="coerce")

    df.dropna(subset=["Score"], inplace=True)

    df["Score"] = df["Score"].astype(int)

    df = df[df["Score"].isin([1, 2, 3, 4, 5])]

# 4. Reaplicare modificările pe dataframes (deoarece `df = df[...]` nu modifică în loc)

train_df = train_df[train_df["Score"].isin([1, 2, 3, 4, 5])]

test_df = test_df[test_df["Score"].isin([1, 2, 3, 4, 5])]

# 5. Codificare etichete

le = LabelEncoder()

train_df["label"] = le.fit_transform(train_df["Score"])

test_df["label"] = le.transform(test_df["Score"]) # trebuie să folosească acelaşi encoder

# 6. Tokenizare pe text

tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")

tokenizer.fit_on_texts(train_df["combined_text"])

```

*# 7. Transformare în secvențe + padding*

*X\_train = tokenizer.texts\_to\_sequences(train\_df["combined\_text"])*

*X\_test = tokenizer.texts\_to\_sequences(test\_df["combined\_text"])*

*X\_train = pad\_sequences(X\_train, maxlen=200)*

*X\_test = pad\_sequences(X\_test, maxlen=200)*

*y\_train = train\_df["label"].values*

*y\_test = test\_df["label"].values*

*# 8. Model LSTM*

*model = Sequential([*

*Embedding(input\_dim=10000, output\_dim=64, input\_length=200),*

*LSTM(64),*

*Dropout(0.5),*

*Dense(32, activation='relu'),*

*Dense(len(le.classes\_), activation='softmax')*

*])*

*model.compile(*

*loss="sparse\_categorical\_crossentropy",*

*optimizer="adam",*

*metrics=["accuracy"]*

*)*

*model.summary()*

*# 9. Antrenare*

*model.fit(*

```

X_train, y_train,

epochs=5,

batch_size=128,

validation_split=0.1

)

# 10. Evaluaire

y_pred = model.predict(X_test).argmax(axis=1)

unique_labels = np.unique(y_test)

print(classification_report(

    y_test,

    y_pred,

    labels=unique_labels,

    target_names=le.inverse_transform(unique_labels).astype(str)

))

```

```

Epoch 1/5
3198/3198 ————— 369s 114ms/step - accuracy: 0.7008 - loss: 0.8344 - val_accuracy: 0.7551 - val_loss: 0.6541
Epoch 2/5
3198/3198 ————— 432s 135ms/step - accuracy: 0.7648 - loss: 0.6297 - val_accuracy: 0.7698 - val_loss: 0.6253
Epoch 3/5
3198/3198 ————— 371s 116ms/step - accuracy: 0.7854 - loss: 0.5793 - val_accuracy: 0.7785 - val_loss: 0.6054
Epoch 4/5
3198/3198 ————— 397s 124ms/step - accuracy: 0.8049 - loss: 0.5331 - val_accuracy: 0.7871 - val_loss: 0.5929
Epoch 5/5
3198/3198 ————— 403s 126ms/step - accuracy: 0.8207 - loss: 0.4959 - val_accuracy: 0.7901 - val_loss: 0.5890

```

- **Accuracy-ul de validare** crește constant până la ~**79%**, ceea ce indică o bună capacitate de generalizare.
- **Val loss-ul** scade, ceea ce confirmă că modelul nu suferă de overfitting până la epoca 5.

	precision	recall	f1-score	support
1	0.71	0.77	0.74	10515
2	0.54	0.37	0.44	5937
3	0.54	0.54	0.54	8460
4	0.62	0.43	0.51	16026
5	0.87	0.94	0.90	72743
accuracy			0.79	113681
macro avg	0.66	0.61	0.63	113681
weighted avg	0.78	0.79	0.78	113681

- Clasa **5** este recunoscută excelent de model (precision și recall > 0.9), datorită numărului mare de exemple.
- Clasele **2 și 4** sunt cele mai slabe, cu F1-score sub 0.51, ceea ce sugerează **dezechilibru** și **ambiguitate semantică** între ele și alte clase apropiate.
- Clasa **3** are o performanță echilibrată (0.54), dar relativ modestă.

## 9. Spark Streaming

Se dorește procesarea în timp real a unui flux de fișiere CSV care conțin recenzii ale utilizatorilor despre produse. Pentru fiecare nou fișier adăugat într-un director monitorizat, trebuie calculat numărul total de recenzii și scorul mediu acordat produselor.

- **Configurare mediu Spark și creare folder temporar de streaming:** Se setează variabilele de mediu JAVA\_HOME și SPARK\_HOME. Se creează un folder stream\_input și un fișier dummy pentru a porni fluxul.
- **Inițializare SparkSession și definirea schemei:** Se creează un SparkSession pentru aplicația Spark. Se definește schema recenziilor pentru a asigura parsarea corectă a fișierelor CSV.
- **Citirea fluxului de fișiere din folderul stream\_input:** Se citește în mod continuu directorul cu ajutorul readStream, cu opțiuni pentru fișiere CSV cu antet.
- **Procesarea fiecărui lot de date:** Funcția procesare\_lot este apelată pentru fiecare fișier nou, calculând numărul de recenzii și scorul mediu.

- **Adăugarea unui fișier nou într-un thread separat:** Pentru a simula comportamentul real al unui flux, se adaugă manual un fișier nou (recenzii\_mic.csv) în directorul urmărit, după ce fluxul devine activ.
- **Rezultat:** Se observă în consolă cum sunt procesate loturile 0 și 1. Primul lot conține doar recenzia dummy, iar lotul 1 conține cele 20 de recenzii reale cu scorul mediu calculat.

```
import os
import shutil
import time
import threading
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, count

# === Setări de mediu ===
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.4.1-bin-hadoop3"

import findspark
findspark.init()

# === Pregătire folder și fișier dummy ===
folder = "/content/stream_input"
shutil.rmtree(folder, ignore_errors=True)
os.makedirs(folder, exist_ok=True)

with open(os.path.join(folder, "dummy.csv"), "w", encoding="utf-8") as f:

f.write("Id,ProductId,UserId,ProfileName,HelpfulnessNumerator,HelpfulnessDenominator,Score,Summary,Text,ReadableTime\n")
    f.write("0,DUMMY,U0,Test,0,0,5,Dummy summary,Dummy text,2020-01-01\n")

# === Inițializare SparkSession ===
spark = SparkSession.builder \
    .appName("StreamingReviewStats") \
    .master("local[*]") \
    .getOrCreate()

# === Schema ===
schema = "Id INT, ProductId STRING, UserId STRING, ProfileName STRING, HelpfulnessNumerator INT, HelpfulnessDenominator INT, Score INT, Summary STRING, Text STRING, ReadableTime STRING"

# === Streaming cu foreachBatch ===
stream_df = spark.readStream \
    .schema(schema) \
    .option("header", True) \
    .option("multiLine", True) \
```

```

.option("quote", "") \
.option("sep", ",") \
.option("mode", "PERMISSIVE") \
.csv(folder)

def procesare_lot(batch_df, batch_id):
    print(f"□ Procesare lot {batch_id}")
    batch_df.groupBy().agg(
        count("*").alias("nr_recenzii"),
        avg("Score").alias("scor_mediu")
    ).show(truncate=False)

query = stream_df.writeStream \
    .outputMode("append") \
    .foreachBatch(procesare_lot) \
    .trigger(processingTime="5 seconds") \
    .start()

# === Funcție care adaugă un fișier NOU după ce stream-ul devine activ ===
def adauga_fisier():
    import uuid
    while not query.isActive:
        print("□ Aștept ca stream-ul să pornească...")
        time.sleep(0.5)

    time.sleep(2)

    file_name = f"recenzii_{uuid.uuid4().hex}.csv"
    with open("recenzii_mic.csv", "r", encoding="utf-8") as src:
        content = src.read()


    with open(os.path.join(folder, file_name), "w", encoding="utf-8") as dst:
        dst.write(content)
        dst.flush()
        os.fsync(dst.fileno())

    print(f"□ Fișierul {file_name} a fost scris manual și este nou!")

# === Rulează funcția într-un thread separat ===
threading.Thread(target=adauga_fisier).start()


# === Confirmare și așteptare ===
time.sleep(2)
print("□ Stream isActive:", query.isActive)
query.awaitTermination(60)


```

 Procesare lot 0

nr_recenzii	scor_mediu
1	5.0

 Stream isActive: True

 Fișierul recenzii\_85e1b33d34f2448db02309f8a8549a85.csv a fost scris manual și este nou!

 Procesare lot 1

nr_recenzii	scor_mediu
20	4.35