

CODE DOCUMENTATION

Group 8 : 122ad0045, 122ad0043, 122ad0018

1. Install findspark Package

Purpose: Installs the findspark library to locate and initialize the Spark installation in Python.

Code:

```
pip install findspark
```

2. Initialize Spark Session

Purpose: Sets up the environment for PySpark and starts a Spark session for distributed data processing.

Code:

```
import os
```

```
import findspark
```

```
# Set Spark environment variables
```

```
os.environ["SPARK_HOME"] = "/home/iiitdmk-sic40/spark-3.5.5" # Update  
if different
```

```
os.environ["PYSPARK_PYTHON"] = "/usr/bin/python3"
```

```
os.environ["PYSPARK_DRIVER_PYTHON"] = "jupyter"
```

```
os.environ["PYSPARK_DRIVER_PYTHON_OPTS"] = "notebook"
```

```
# Initialize findspark
```

```
findspark.init()
```

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

```
# Start Spark session
```

```
spark = SparkSession.builder \
    .master("spark://172.16.72.48:7077") \
    .appName("Amazon_Reviews") \
    .config("spark.executor.memory", "4g") \
    .config("spark.executor.cores", "2") \
    .getOrCreate()
```

```
print("Amazon_Reviews Spark Session Started 🚀")
```

Details:

- **Input:** Environment variables for Spark home, Python executable, and Jupyter settings.
- **Output:** A Spark session object (spark) and a confirmation message.
- **Key Components:**
 - Sets SPARK_HOME to the Spark installation path.
 - Configures Python and Jupyter for PySpark execution.
 - Initializes findspark to locate Spark.
 - Creates a Spark session with:
 - Master URL: spark://172.16.72.48:7077 (indicates a Spark cluster).

- Application name: Amazon_Reviews.
- Executor resources: 4GB memory, 2 cores.
- **Issues:** A warning (WARN SparkSession: Using an existing Spark session) suggests a pre-existing Spark session, which may cause configuration conflict.
- **3. Import Required Libraries**
- **Purpose:** Imports libraries for data processing, text preprocessing, visualization, and machine learning.
- fails if not properly managed.

3. Import Required Libraries

Purpose: Imports libraries for data processing, text preprocessing, visualization, and machine learning.

Code:

#Required Libraries

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from bs4 import BeautifulSoup

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer, WordNetLemmatizer

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

```
from sklearn.svm import LinearSVC
```

```
from sklearn.metrics import classification_report, confusion_matrix,  
accuracy_score
```

Details:

- **Input:** None.
- **Output:** Imported libraries ready for use.
- **Key Libraries:**
 - os, numpy, pandas: General data handling.
 - matplotlib, seaborn: Visualization (unused in the provided code).
 - BeautifulSoup, re: HTML parsing and text cleaning.
 - nltk: Natural language processing (stopwords, stemming, lemmatization).
 - sklearn: Machine learning (splitting data, TF-IDF, SVC, evaluation metrics).
- **Notes:** Comprehensive set of libraries for end-to-end sentiment analysis.

4. Load Data from HDFS

- **Purpose:** Loads the Amazon reviews dataset from HDFS into a PySpark DataFrame.

Code:

```
df = spark.read.csv("hdfs://localhost:9000/user/iiitdmk/dataset/Amazon_reviews.csv",  
header=True, inferSchema=True)
```

```
df.show(5) # Display the first 5 rows
```

Details:

- **Input:** CSV file path on HDFS
(hdfs://localhost:9000/user/iiitdmk/dataset/Amazon_reviews.csv).

- **Output:** A PySpark DataFrame (df) and a preview of the first 5 rows.
- **Key Features:**
 - header=True: Uses the first row as column names.
 - inferSchema=True: Automatically detects column data types.
 - Columns include: Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text.
- **Notes:** Successfully loads the dataset, displaying a structured view of review data.

5. Download NLTK Resources

- **Purpose:** Downloads necessary NLTK data for text preprocessing.

Code:

```
import nltk

nltk.download('wordnet')

nltk.download('stopwords')

nltk.download('punkt')
```

Details:

- **Input:** None.
- **Output:** Confirmation of downloaded resources (wordnet, stopwords, punkt) or a message indicating they are up-to-date.
- **Key Resources:**
 - wordnet: For lemmatization.
 - stopwords: For removing common words (e.g., "the", "is").
 - punkt: For tokenization.
- **Notes:** Essential for the text preprocessing pipeline.

6. Reconfigure Spark Environment

- **Purpose:** Updates Spark environment variables (appears redundant).

Code:

```
# ✅ Start Spark Session
spark = SparkSession.builder \
    .master("local[*]") \
    .appName("Amazon_Reviews") \
    .getOrCreate()
```

7. Convert PySpark DataFrame to Pandas

- **Purpose:** Converts the PySpark DataFrame to a Pandas DataFrame for easier manipulation.

Code:

```
# ✅ Set Up PySpark for Jupyter Notebook

os.environ["SPARK_HOME"] = "/opt/spark"

os.environ["PYSPARK_PYTHON"] = "/usr/bin/python3"
```

Details:

- **Input:** PySpark DataFrame (df).
- **Output:** Pandas DataFrame (df) and a preview of the first 5 rows.
- **Key Features:**
 - Checks if df is a PySpark DataFrame before conversion.
 - Displays columns: Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text.
- **Notes:** Conversion to Pandas is necessary for scikit-learn compatibility but may be memory-intensive for large datasets

8. Data Cleaning (Error-Prone)

- **Purpose:** Attempts to clean the dataset by removing missing values and transforming the Score column.

Details:

- **Input:** Pandas DataFrame (df) from Cell 11.
- **Output:** Error: AttributeError: 'DataFrame' object has no attribute 'na'.
- **Intended Functionality:**
 - Remove rows with missing Text or Score values.
 - Transform Score into binary labels: 0 (negative, scores 1 or 2), 1 (positive, scores 4 or 5).

- Filter out neutral reviews (score 3).
- **Issues:**
 - The code uses PySpark syntax (na.drop, withColumn, filter) on a Pandas DataFrame, causing the error.
 - Should use Pandas equivalents (e.g., dropna, mask, query).

9. Convert to Pandas and Split Data

- **Purpose:** Converts the DataFrame to Pandas (redundant) and splits data into training and test sets.


Details:

- **Input:** PySpark DataFrame (df).
- **Output:** Training and test sets (x_train, x_test, y_train, y_test).
- **Key Features:**
 - Converts df to Pandas (unnecessary since Cell 11 already did this).
 - Splits Text (features) and Score (labels) with 70% training, 30% testing.

10. Text Preprocessing

- **Purpose:** Preprocesses review text by removing HTML, stop words, and applying stemming/lemmatization.

Code:

```
#  Text Preprocessing

nltk.download('wordnet')

nltk.download('stopwords')

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

stemmer = SnowballStemmer('english')

lemmatizer = WordNetLemmatizer()

def preprocess_text(text):

    soup = BeautifulSoup(text, "html.parser")

    review = soup.get_text()

    review = re.sub(r'[^\w-zA-Z]', ' ', review).lower().split()
```

```
review = [stemmer.stem(lemmatizer.lemmatize(word)) for word in review if word not in stop_words]
```

```
return ''.join(review)
```

```
corpus_train = [preprocess_text(text) for text in x_train]
```

```
corpus_test = [preprocess_text(text) for text in x_test]
```

Details:

- **Input:** Training and test text data (x_train, x_test).
- **Output:** Preprocessed text corpora (corpus_train, corpus_test).
- **Key Steps:**
 - Removes HTML tags using BeautifulSoup.
 - Converts text to lowercase, removes non-letters.
 - Applies lemmatization and stemming, removes stop words.
- **Issues:** Warnings indicate BeautifulSoup misinterprets some text as filenames/URLs, but preprocessing completes successfully.

11. TF-IDF Vectorization

- **Purpose:** Converts preprocessed text into TF-IDF features.

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.model_selection import train_test_split
```

```
# ✅ Define Text Column
```

```
text_column = "Text" # Make sure this column exists in your DataFrame
```

```
# ✅ Ensure Data Exists
```

```
if text_column not in df.columns:
```

```
    raise ValueError(f'❌ ERROR: Column '{text_column}' not found in DataFrame!')
```

```
# ✅ Fill missing text values
```

```
df[text_column] = df[text_column].fillna("")
```

```
# ✅ Split Data into Train & Test
```

```
corpus_train, corpus_test = train_test_split(df[text_column], test_size=0.2, random_state=42)
```



```
# ✅ Apply TF-IDF Vectorization

tfidf_vec = TfidfVectorizer(ngram_range=(1, 3), max_features=5000)

tfidf_vec_train = tfidf_vec.fit_transform(corpus_train)

tfidf_vec_test = tfidf_vec.transform(corpus_test)

print("✅ TF-IDF transformation completed successfully!")
```

Details:

- **Input:** Text column (Text) from df.
- **Output:** TF-IDF matrices (tfidf_vec_train, tfidf_vec_test).
- **Key Features:**
 - Fills missing text values with empty strings.
 - Splits data (80% train, 20% test).
 - Uses TfidfVectorizer with unigrams to trigrams, capped at 5000 features.

12. Logistic Regression Model

- **Purpose:** Trains a Logistic Regression model on TF-IDF features and evaluates accuracy.

```
# ✅ Logistic Regression (handles large sparse data better)

log_reg = LogisticRegression(max_iter=5000, solver="lbfgs")

log_reg.fit(tfidf_vec_train, y_train)

# ✅ Predictions

predictions = log_reg.predict(tfidf_vec_test)

# ✅ Accuracy

accuracy = accuracy_score(y_test, predictions)

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

Details:

- **Input:** Text and score columns from df.
- **Output:** Trained Logistic Regression model and accuracy (0.7419).
- **Key Steps:**
 - Fills missing values in Text and Score.
 - Encodes Score labels numerically.
 - Splits data and applies TF-IDF (redundant with Cell 16).
 - Trains Logistic Regression with 5000 iterations.

