Title: Report

Name: Umm-e-Hani , Laiba Batool , Dawood Tanvir

Roll No.: 21I-1715 , 21I-1781, 21I-1665

Section: M

MODEL TRAINING:

We tried training the model on the whole dataset but that wasn't possible. I kept getting the error given below. Tried to train the dataset on 50% data still the same error. I tried multiple solutions but it was giving a memory error. After many tries I decided to pick up on 1 million rows and still the same error so I had to train the model on an even smaller dataset.

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[8], line 1
----> 1 transformed = pipeline.fit(sampled_df).transform(sampled_df)

File C:\spark\python\pyspark\ml\base.py:205, in Estimator.fit(self, dataset, params)
    203         return self.copy(params)._fit(dataset)
    204     else:
--> 205         return self._fit(dataset)
    206 else:
    207     raise TypeError(
    208         "Params must be either a param map or a list/tuple of param maps, "
    209         "but got %s." % type(params)
    210     )

File C:\spark\python\pyspark\ml\pipeline.py:134, in Pipeline._fit(self, dataset)
    132     dataset = stage.transform(dataset)
    133 else:  # must be an Estimator
--> 134     model = stage.fit(dataset)
    135     transformers.append(model)
    136     if i < indexOfLastEstimator:

File C:\spark\python\pyspark\ml\base.py:205, in Estimator.fit(self, dataset, params)
    203         return self.copy(params)._fit(dataset)
    204     else:
--> 205         return self._fit(dataset)
    206 else:
    207     raise TypeError(
    208         "Params must be either a param map or a list/tuple of param maps, "
    209         "but got %s." % type(params)
    210     )

File C:\spark\python\pyspark\ml\wrapper.py:383, in JavaEstimator._fit(self, dataset)
    382 def _fit(self, dataset: DataFrame) -> JM:
--> 383     java_model = self._fit_java(dataset)
    384     model = self._create_model(java_model)
```

KAFKA CONNECTION AND MONGODB:

This code is a Flask-based web application that uses Apache Spark and Kafka to provide product recommendations to users based on their reviews. The application provides a web form where users can input their review, and then uses Spark to recommend products to the user. The recommended products are stored in a MongoDB database, and a Kafka producer sends the recommendations to the Kafka consumer, which displays the recommended products to the user in real-time.The code uses a pre-trained machine learning model built using the ALS algorithm for collaborative filtering to recommend products to users. The application has several dependencies, including the Flask, Kafka, and PySpark libraries. It also requires a MongoDB database and a pre-trained ALS model.The code has two routes - the first route displays the web form, while the second route displays the recommended products. The first route sends a post request with the user's review, and the application uses Spark to recommend products to the user. The recommended products are stored in the MongoDB database, and the Kafka producer sends the recommendations to the Kafka consumer. The Kafka consumer then displays the recommended products to the user on the second route.Overall, this Flask-based application provides an effective way to recommend Amazon products to users based on their reviews, using powerful technologies like Spark and Kafka for large-scale data processing and real-time communication

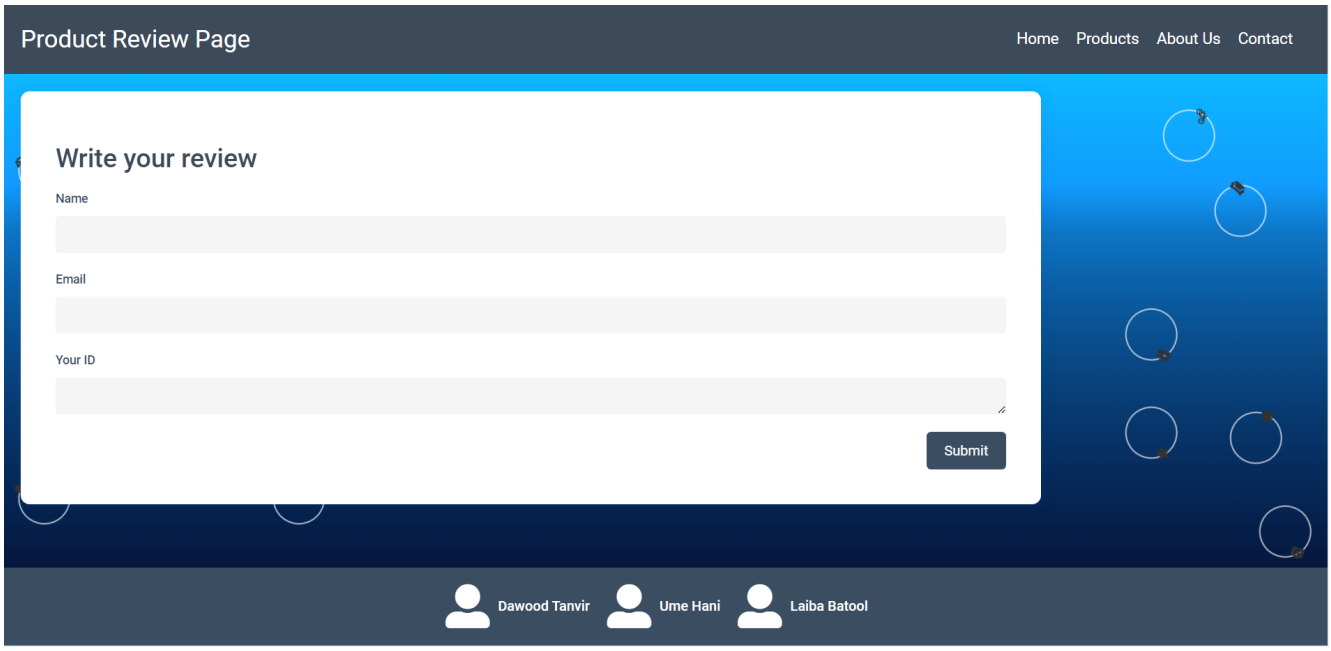
```python
from flask import Flask, render_template, request
import warnings
from flask import redirect
from pymongo import MongoClient, collection

import numpy as np
from kafka import KafkaProducer
from kafka import KafkaConsumer

warnings.simplefilter("ignore", UserWarning)
warnings.filterwarnings('ignore')
import pyspark
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, BooleanType, LongType, IntegerType
from pyspark.sql.functions import col, sum
from pyspark.sql.functions import mean, min, max
from pyspark.sql.functions import from_unixtime
import numpy
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import pyspark.sql.functions as F
from pyspark.ml.feature import StringIndexer
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

# spark = SparkSession.builder.appName("Amazon Reviews").getOrCreate()
# https://img.freepik.com/free-vector/musical-pentagram-sound-waves-notes-background_1017-33911.jpg?w=2000
```

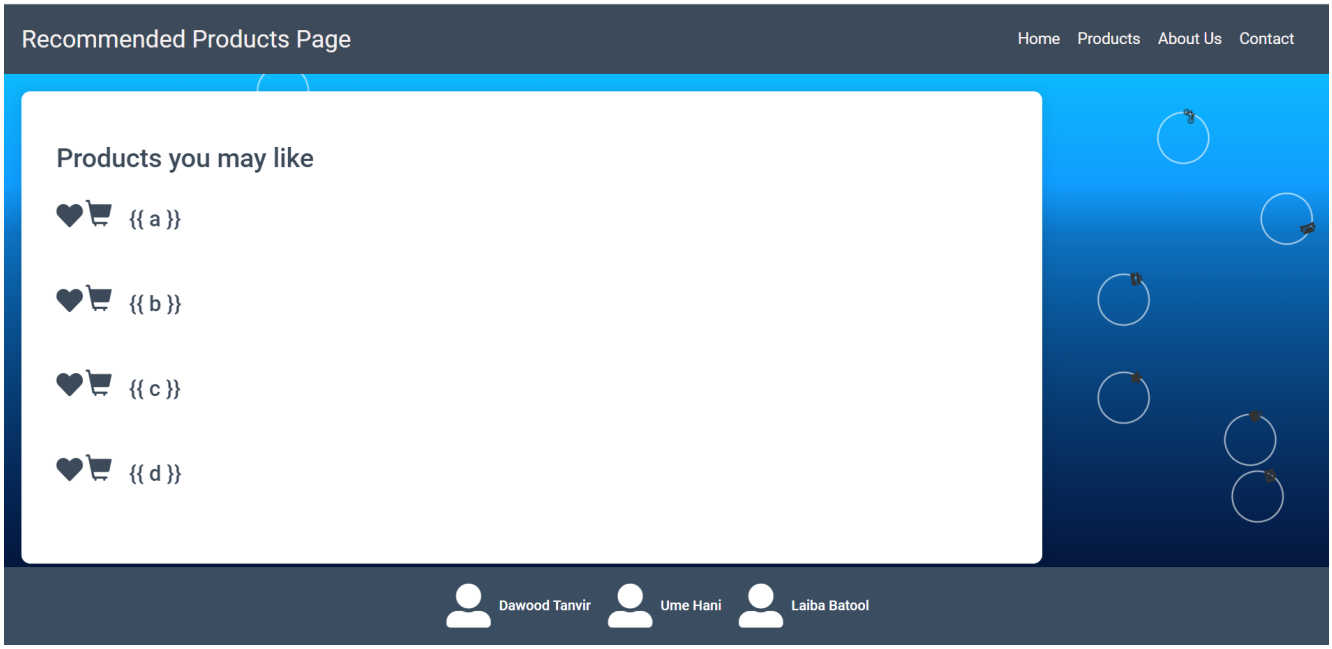
```python
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

# spark = SparkSession.builder.appName("Amazon Reviews").getOrCreate()
# https://img.freepik.com/free-vector/musical-pentagram-sound-waves-notes-background_1017-33911.jpg?w=2000
spark = SparkSession.builder.appName("Amazon Reviews") \
    .config("spark.driver.memory", "4g") \
    .config("spark.executor.memory", "4g") \
    .config("spark.executor.heartbeatInterval", "5000s") \
    .config("spark.network.timeout", "10000s") \
    .getOrCreate()

mongo_client = MongoClient('mongodb://localhost:27017')
db = mongo_client['RECOMMENDATIONS']
collection = db['User_Recommendation']
app = Flask(__name__)

recommendations = []


@app.route('/', methods=['GET'])
def home():
    return render_template('review.html')


@app.route('/', methods=['POST'])
def index():
    from pyspark.ml.recommendation import ALS, ALSModel
    models = ALSModel.load("E:\\AmazonResc")
```

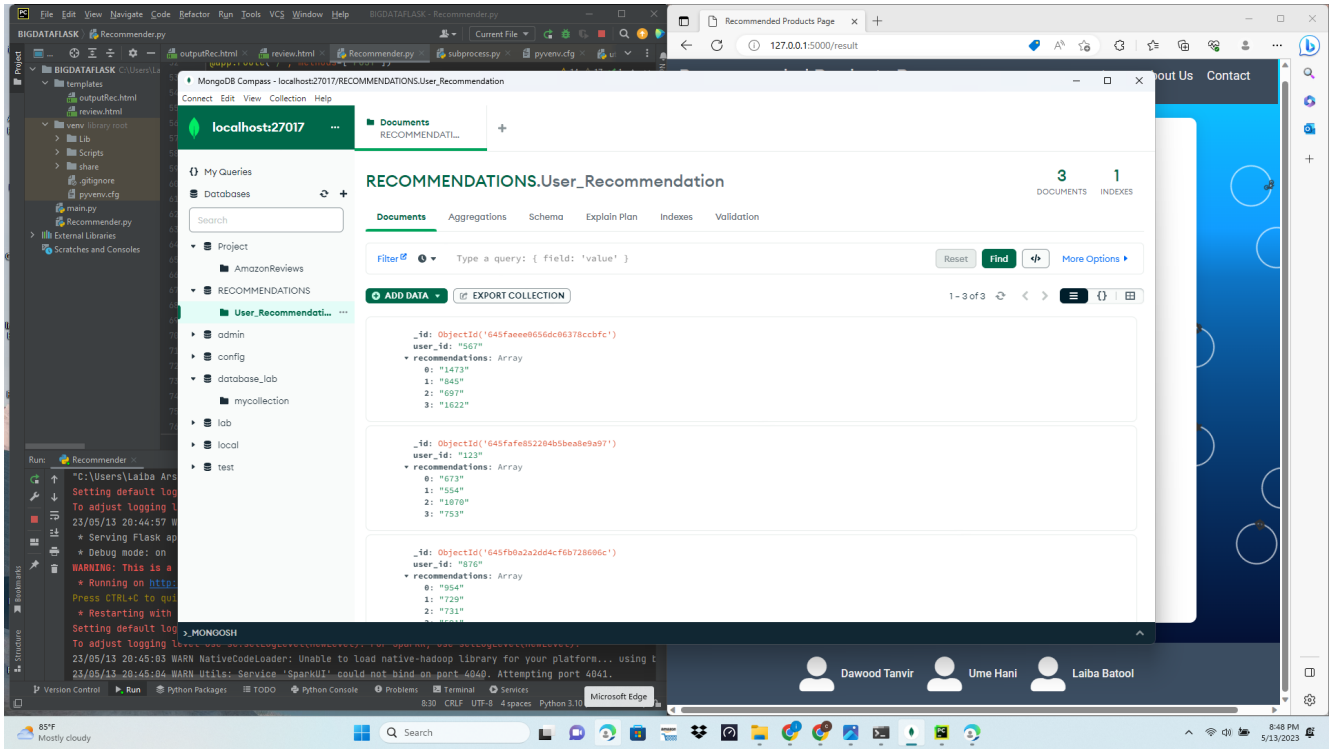
```python
    return render_template('review.html')


@app.route('/', methods=['POST'])
def index():
    from pyspark.ml.recommendation import ALS, ALSModel
    models = ALSModel.load("E:\\AmazonResc")
    user_id = request.form['review']

    user_data = spark.createDataFrame([(user_id,)], ["Revindexed"])

    userRecs = models.recommendForUserSubset(user_data, numItems=5)  # Specify the number of recommendations
    rec = userRecs.collect()
    producer = KafkaProducer(bootstrap_servers=['localhost:9092'])


    recommendations.append(str(rec[0][1][1][0]))
    recommendations.append(str(rec[0][1][2][0]))
    recommendations.append(str(rec[0][1][3][0]))
    recommendations.append(str(rec[0][1][4][0]))

    document = {
        'user_id': user_id,
        'recommendations': recommendations
    }
#insert document in mongodb
    collection.insert_one(document)
    recommendations.append(user_id)


    for recommendation in recommendations:
```

```
    return render_template('review.html')


@app.route('/', methods=['POST'])
def index():
    from pyspark.ml.recommendation import ALS, ALSModel
    models = ALSModel.load("E:\\AmazonResc")
    user_id = request.form['review']

    user_data = spark.createDataFrame([(user_id,)], ["Revindexed"])

    userRecs = models.recommendForUserSubset(user_data, numItems=5)  # Specify the number of recommendations
    rec = userRecs.collect()
    producer = KafkaProducer(bootstrap_servers=['localhost:9092'])


    recommendations.append(str(rec[0][1][1][0]))
    recommendations.append(str(rec[0][1][2][0]))
    recommendations.append(str(rec[0][1][3][0]))
    recommendations.append(str(rec[0][1][4][0]))

    document = {
        'user_id': user_id,
        'recommendations': recommendations
    }
#insert document in mongodb
    collection.insert_one(document)
    recommendations.append(user_id)


    for recommendation in recommendations:
```

```
    recommendations.append(user_id)


    for recommendation in recommendations:
        producer.send('Rec', value=recommendation.encode())

    return redirect('/result')


@app.route('/result')
def display_result():
    consumer = KafkaConsumer(
        'Rec',
        bootstrap_servers='localhost:9092',
        auto_offset_reset='earliest',
        group_id='i23'
    )
    i = 0
    rec = []
    # Poll for new messages
    for message in consumer:
        value = message.value.decode()
        rec.append(value)
        consumer.commit_async()
        i += 1
        if i == 4:
            break

    return render_template("outputRec.html", a=rec[0], b=rec[1], c=rec[2], d=rec[3])
```


Product Review Page
Home
Products
About Us
Contact
Write your review
Name
Email
Your ID
Submit
Dawood Tanvir
Ume Hani
Laiba Batool





CONTRIBUTIONS:

UMM E HANI : MODEL TRAINING
DAWOOD & LAIBA : MONGO AND KAFKA SETUP