CS 6350
ASSIGNMENT 2

Names of students in your group:

Yoon Kyung Lee (yxl240011)
Changhui Sun (cxs240024)

Number of free late days used: ___0___

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

- Course lecture notes and assignment description
- dataset
  - soc-LiveJournal1Adj.txt from the Carnegie Mellon Movie Summary Corpus
  - SMS Spam Collection Dataset from UCI Machine Learning Repository
- Official PySpark documentation (https://spark.apache.org/docs/latest/api/python/) for reference on RDD and DataFrame transformations such as map, reduceByKey, join, groupBy, explode, Window, etc.
- NLTK documentation for stopword removal and text preprocessing background.
- Suggested links in the assignment for cosine similarity background

Assignment 2-1: Friend Recommendation using Mutual Friends

Course: CS6350.002 Big Data Management & Analytics
Team Members:

- Yoonkyung Lee (NetID: yxl240011)
- Changhui Sun (NetID: cxs240024)

1. Algorithm

The first step is to load and clean the data, keeping only the lines that contain a tab character. Then, we parse and explode each line to obtain a DataFrame with two columns: user and friend.
Next, we aim to find all pairs of mutual friends for a given user A. To do this, we transform the data so that each user A produces pairs such as [(B, C), (B, D), ...], meaning that B and C are mutual friends of A.
After generating these pairs, we remove those pairs that are already directly connected in the original friendship list. For example, if B and C are both friends of A, but B and C are already direct friends with each other, that pair will be excluded from the recommendation set.
The next step is to count the number of mutual friends for each remaining pair. If two users share more mutual friends, the score of recommendation will high. Once the mutual-friend counts are computed, we need to turned the relationship into two directional recommendations. Ensures that both users receive each other as potential friend suggestions. Then, for each user, we rank all recommended friends by descending mutual-friend count and keep the top 10 recommendations per user.
(In my real code, I randomly select 10 users as a sample because using the entire dataset would cause memory issues during the aggregation step.)

2. Pseudo-code

Load & Clean

1
Input: text file soc-LiveJournal1Adj.txt
Read all lines into dataframe df Keep only rows that contain a tab character

Parse & Explode

2
For each row in df:
Split the line by tab into (user, friends_string)
Split friends_string by commas to get friend_list
Create dataframe (user, friend) by exploding friend_list

Friend-of-friend pairs

3
For each user:
For every pair of friends (f1, f2):
If f1 < f2:
Record pair (f1, f2) with mutualFriend = user

Remove direct friends from above

4
Create set of all direct friendships (user, friend)
Remove any (f1, f2) pairs that already exist in the direct friend set

Count mutual friends

5
Group all remaining pairs (f1, f2)
Count the number of distinct mutualFriend values for each pair

Create directional recommendations

6
For each pair (f1, f2, mutualCount):
Output two directed records:
(src = f1, dst = f2, mutualCount)
(src = f2, dst = f1, mutualCount)

Rank and keep top 10 recommendations

7
For each user src:
Sort recommended users dst by:
1. mutualCount descending
2. dst ascending (tie-breaking)
Keep the first 10 entries

Format and output

8
For each user src:

Combine their top 10 dst IDs into one comma-separated list

2. Code

In [ ]:

```
from pyspark.sql import SparkSession
import subprocess
from pyspark.sql import functions as F
from pyspark.sql.functions import split, col, explode, least, greatest
from pyspark.sql.window import Window
spark = (
    SparkSession.builder
    .appName("MutualFriends")
    .master("local[*]")
    .config("spark.driver.memory", "6g")
    .getOrCreate()
)
subprocess.run(
    ["wget", "https://an-ml.s3.us-west-1.amazonaws.com/soc-LiveJournal1Adj.txt"], c
)
df = spark.read.text("soc-LiveJournal1Adj.txt")
df.show()
dfClean = df.filter(col("value").contains("\t"))
df = (dfClean.withColumn("user", split(col("value"), "\\t").getItem(0)).withColumn(
df.show()

dfExploded = df.withColumn("friendSigle", explode(col("friend")))

targets = (
    dfExploded.select(col("user").cast("int").alias("user")).distinct().orderBy(F.r

dfTargets = dfExploded.join(targets, on="user", how="inner")

dfPair = dfTargets.alias("a").join(dfTargets.alias("b"), on="user") \
    .where(col("a.friendSigle") < col("b.friendSigle")) \
    .select(col("user").alias("mutualFriend"),
            col("a.friendSigle").alias("f1"),
            col("b.friendSigle").alias("f2"))



dfDirect = dfTargets.select(col("user").alias("df1"), col("friendSigle").alias("df2


dfDirectNorm = dfDirect.select(least(col("df1").cast("int"), col("df2").cast("int")

dfDirectNorm.filter(col("f1") > col("f2")).count()
dfPair.filter(col("f1") > col("f2")).count()

dfNoDirect = dfPair.join(dfDirectNorm, on=["f1", "f2"], how="left_anti")

dfMutualCount = (dfNoDirect.select("f1","f2","mutualFriend").distinct().groupBy("f1

# directed recommendation
a = dfMutualCount.select(F.col("f1").alias("src"), F.col("f2").alias("dst"), "mutua
b = dfMutualCount.select(F.col("f2").alias("src"), F.col("f1").alias("dst"), "mutua
```

```python
directed = a.unionByName(b)

w = Window.partitionBy("src").orderBy(F.col("mutualCount").desc(), F.col("dst").asc
top10 = directed.withColumn("rk", F.row_number().over(w)).where(F.col("rk") <= 10)

# format
out = (
    top10.groupBy("src")
    .agg(F.collect_list(F.struct("rk", "dst")).alias("lst"))
    .select(
        "src",
        F.expr("concat_ws(',', transform(array_sort(lst), x -> cast(x.dst as string
    )
)

sample10 = out.orderBy(F.rand()).limit(10)
sample10.show(truncate=False)
lines = sample10.selectExpr("concat(cast(src as string), '\t', top10) as line")
with open("q1_output.txt", "w", encoding="utf-8") as f:
    for row in lines.toLocalIterator():
        f.write(row["line"] + "\n")

spark.stop()
```

3. Output

6958 32743,32744,32749,32826,32838,32839,33443,34573

42575 42573,42577,42584,42585,42587,42588,42608

10350 10906,14432,19109,25327,33517,40937,7427,771

771 10350,10906,14432,19109,25327,33517,40937,7427

32749 32743,32744,32826,32838,32839,33443,34573,6958

23786 23717,23753,32557,49577

42585 42573,42575,42577,42584,42587,42588,42608

42584 42573,42575,42577,42585,42587,42588,42608

10906 10350,14432,19109,25327,33517,40937,7427,771

18862 24140,24240,24246,29592,29612,7293,7444

Assignment 2-2: Implementing Naive Bayes Classifier using Spark MapReduce

Course: CS6350.002 Big Data Management & Analytics
Team Members:

- Yoonkyung Lee (NetID: yxl240011)
- Changhui Sun (NetID: cxs240024)

---

1. Dataset Description

We used the **SMS Spam Collection Dataset** from the UCI Machine Learning Repository:
https://archive.ics.uci.edu/dataset/228/sms+spam+collection

- Total number of samples: **4503**
- Label distribution:
  - **Ham**: 3903
  - **Spam**: 600

---

2. Pseudo-code of Naive Bayes using MapReduce in PySpark

Training Phase

Map

```
(trainingData: RDD[(label, SparseVector)])
.map(lambda x: (label, DenseVector(x[1].toArray())))
```

Reduce

```
.reduceByKey(lambda x, y: x + y)
```

Conditional Probabilities

```
P(w_i | C) = (count(w_i in class C) + 1) / (totalWordsInClass + VocabSize)
```

Testing Phase

For each test message:

1. Initialize:
   ```
   logProbHam = log(P(ham))
   logProbSpam = log(P(spam))
   ```
2. For each word `w_i` with frequency `f_i` in message:
   ```
   logProbHam += f_i * log(P(w_i | ham))
   logProbSpam += f_i * log(P(w_i | spam))
   ```
3. Choose label:
   ```
   predicted = argmax(logProbHam, logProbSpam)
   ```

---

3. Summary of Results

- **Vocabulary Size**: 13423
- **P(spam)**: 0.1332

- **P(ham)**: 0.8668
- **Accuracy**: **97.48%** (1044 correct / 1071 total)

---

4. Notes on Implementation

- We implemented the Naive Bayes classifier **entirely from scratch**, using **RDD transformations** (`map` , `reduceByKey`) to calculate class priors and conditional probabilities.
- We used `CountVectorizer` from `pyspark.ml` to tokenize and vectorize the text.
- `DenseVector` was used internally to simplify summation of word counts per class.

---

5. File List

- `q2.py` : Full implementation code.
- `naive_bayes_report.md` : This report.
