Trending Topic Discovery via Yearly Headline Count (16 marks)

Background: Identifying popular and emerging topics from news headlines is a key task in understanding trends in public discourse. In this project, you will analyze a dataset of news headlines published by the Australian Broadcasting Corporation (ABC) using Spark. Your goal is to detect and track high-impact topic terms across years by computing how frequently they appear in news headlines. This analysis will help highlight the most discussed topics in different years and provide insights into how public attention shifts over time.

Input file: The dataset you will use contains data from news headlines published over several years. In this text file, each line is a **headline of a news article**, in the format of "*date*,term1 term2 ... ... ". The date and text are separated by a comma, and the terms are separated by a space character. A sample file is like the one below:

```
20030219,council chief executive fails to secure position 20030219,council
welcomes ambulance levy decision 20030219,council welcomes insurance
breakthrough 20030219,fed opp to re introduce national insurance
20040501,cowboys survive eels comeback 20040501,cowboys withstand eels
fightback 20040502,castro vows cuban socialism to survive bush
20200401,coronanomics things learnt about how coronavirus economy
20200401,coronavirus at home test kits selling in the chinese community
20200401,coronavirus campbell remess streams bear making classes
20201016,china builds pig apartment blocks to guard against swine flu
```

When you click the panel on the right, you'll get a connection to a server that has, in your home directory, a text file called "abcnews.txt", containing some sample text (feel free to open the file and explore its contents). The entire dataset can be downloaded from https://www.kaggle.com/therohk/million-headlines.

Stopword Filtering and Topic Term Discovery

To remove uninformative terms, you must first identify and filter out stopwords using raw term frequency (the global frequency of each term in the whole dataset and break ties alphabetically).

Then, remove stopwords, compute document frequency (number of headlines they appear in), and extract the top-*m* terms (by number of headlines they appear in and break ties alphabetically) as topic terms. Note that a term will only be counted once even it appears more than once in one headline.

Problem: Your task is to output the top-k topic terms per year, based on the number of headlines they appear in. In every year, each topic-count pair should be ranked by the headline count (descending), then the topic term (alphabetically) in case of a tie.

Your program should output the results in the format below. For each year, print the year and the number of topic terms covered in this year headlines, then the top-k topic terms each per line, sorted by headline count (descending), then alphabetically. For example, with n=1 (top-1 stopword filtered), m=5 (top-5 topic terms), and k=3 (top-3 output per year), your output may look like:

```
2003:2 council 3 insurance 2 2004:2 cowboys 2 eels 2 2020:1 coronavirus 3
```

Code Format: The code template has been provided. You need to submit two solutions, one using only RDD APIs and the other using only DataFrame APIs. Your code should take six parameters: the input file, the output folder, the value of n, m, and k, e.g.,: `spark-submit project2_rdd.py "file:///home/abcnews.txt" "file:///home/output" 1 5 3`

Some notes

1. You can read the files from either HDFS or the local file system. Using local files is more convenient, but you must use the prefix "file:///...". Spark uses HDFS by default if the path does not have a prefix.
2. You are not allowed to use numpy or pandas, since we aim to assess your understanding of the RDD/DataFrame APIs.
3. You can use coalesce(1) to merge the data into a single partition and then save the data to disk.
4. In the DataFrame solution, it is not allowed to use the spark.sql() function to pass the SQL statement to Spark directly.
5. It does not matter if you have a new line at the end of the output file or not. It will not affect the correctness of your solution.

Marking Criteria (8 marks for each solution)

- You must complete this assignment using Spark RDD/DataFrame APIs. Submissions only contain regular Python techniques will be marked as 0.
- Submission can be compiled and run on Spark => +3
- Submission can obtain correct top-k results (including correct topics, counts, format and order) => +3
- Submissions correctly using Spark APIs (RDD/DataFrame solution only RDD/DataFrame APIs allowed) => +0.5
- Submissions with excellent code format and structure, readability, and documentation => 0.5
- Submissions efficiently removing highly frequent terms and counting topic terms => 1

Solution

```
<=== dataframe ===>
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
import sys

class Project2:
    def run(self, inputPath, outputPath, stopwords,topic, k):
        spark =
SparkSession.builder.master("local").appName("project2_df").getOrCreate()

        # Fill in your code here
        n = int(stopwords)
        m = int(topic)
        k = int(k)

        df = spark.read.text(inputPath).select(split(col("value"), ",",
2).getItem(0).alias("date"), split(col("value"), ",",
2).getItem(1).alias("text"))
        df_terms = df.withColumn("terms", split(col("text"), "
")).withColumn("unique_terms", array_distinct(col("terms")))

        exploded = df_terms.select(explode(col("terms")).alias("term"))
        terms_freq = exploded.groupBy("term").count()
        stopwords = [row["term"] for row in
terms_freq.orderBy(desc("count"), asc("term")).limit(n).collect()]

        sw_array = array(*[lit(w) for w in stopwords])
        df_filtered = df_terms.withColumn("filtered_terms",
array_except(col("unique_terms"), sw_array))
        exploded_ft =
df_filtered.select(explode(col("filtered_terms")).alias("term"))
        doc_freq = exploded_ft.groupBy("term").count()
        topics = [row["term"] for row in doc_freq.orderBy(desc("count"),
asc("term")).limit(m).collect()]

        df_year = df_filtered.withColumn("year", substring(col("date"), 1,
4))
        df_yt = df_year.select("year",
explode(col("filtered_terms")).alias("term")).filter(col("term").isin(topics
))
        df_yr_terms = df_yt.groupBy("year", "term").count()

        data = df_yr_terms.collect()
        year_map = {}
        for row in data:
            year_map.setdefault(row["year"],[]).append((row["term"],
row["count"]))
```

```
        output = []
        for year in sorted(year_map):
            items = sorted(year_map[year], key= lambda x: (-x[1],x[0]))
            covered = len(items)
            topk = items[:k]
            line = f"{year}:{covered}"
            for term, c in topk:
                line += f"\n{term}      {c}"
            output.append(line)
        spark.sparkContext.parallelize(output, 1).saveAsTextFile(outputPath)
        spark.stop()

if __name__ == "__main__":
    if len(sys.argv) != 6:
        print("Wrong arguments")
        sys.exit(-1)
    Project2().run(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4],
sys.argv[5])
```

```
<=== RDD ===>
from pyspark import SparkContext, SparkConf
import sys
import math

class Project2:
    def run(self, inputPath, outputPath, stopwords, topic, k):
        conf = SparkConf().setAppName("project2_rdd").setMaster("local")
        sc = SparkContext(conf=conf)

        # Fill in your code here
        n = int(stopwords)
        m = int(topic)
        k = int(k)
        lines = sc.textFile(inputPath)

        terms = lines.map(lambda line: line.split(',',1)[1].split())
        terms_freq = terms.flatMap(lambda x:x).map(lambda t:
(t,1)).reduceByKey(lambda a,b: a+b)
        stopwords_lt = terms_freq.sortBy(lambda y: (-y[1],y[0])).map(lambda
y: y[0]).take(n)
        bc_stop = sc.broadcast(set(stopwords_lt))

        doc_terms = terms.map(lambda x: set(x)).map(lambda x: [t for t in x
if t not in bc_stop.value])
        doc_terms_freq = doc_terms.flatMap(lambda x: [(t,1) for t in
x]).reduceByKey(lambda a,b: a+b)
        topic_lt = doc_terms_freq.sortBy(lambda y: (-y[1],y[0])).map(lambda
y: y[0]).take(m)
```

```
        bc_topic = sc.broadcast(set(topic_lt))

        year = lines.map(lambda line: (line.split(",",1)[0]
[:4],set(line.split(",",1)[1].split())))
        yr_terms = year.map(lambda yr: (yr[0], [t for t in yr[1] if t in
bc_topic.value])).flatMap(lambda yr: [((yr[0],t),1) for t in yr[1]])
        yr_terms_sum = yr_terms.reduceByKey(lambda a,b : a+b)

        per_yr = yr_terms_sum.map(lambda y: (y[0][0],(y[0]
[1],y[1]))).groupByKey().mapValues(list)

        def format_line(year, terms_sum):
            items = [(t,c) for t, c in terms_sum if c >0]
            covered = len(items)
            topk = sorted(items, key=lambda y: (-y[1], y[0]))[:k]
            parts = "\n".join(f"{t}     {c}" for t,c in topk)
            return f"{year}:{covered}\n{parts}"

        output = per_yr.map(lambda yr: format_line(yr[0],
yr[1])).sortBy(lambda line: line.split(":")[0])
        output.coalesce(1).saveAsTextFile(outputPath)



        sc.stop()

if __name__ == "__main__":
    if len(sys.argv) != 6:
        print("Wrong arguments")
        sys.exit(-1)
    Project2().run(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4],
sys.argv[5])
```