Assignment 3

Anders Nylund

Problem 4.1

```
import findspark
findspark.init()
import pyspark
import pyspark.sql.functions as func

spark = pyspark.sql.SparkSession.builder \
    .master("local") \
    .appName("movies") \
    .getOrCreate()

df =
spark.read.csv(path="./data/movielens/ratings.csv",
header=True)

max_ratings = df.groupBy("movie_id") \
    .agg(func.count(func.lit(1)).alias("ratings")) \
    .agg({"ratings": "max"}) \
    .collect()[0]["max(ratings)"]

twenty5th_percentile = df.groupBy("movie_id") \
    .agg(func.count(func.lit(1)).alias("ratings")) \
    .filter("ratings/"+str(max_ratings)+" > 0.25") \
    .orderBy("movie_id") \
    .collect()
```

Problem 4.2

```
import findspark

findspark.init()
import pyspark
from pyspark.sql.types import IntegerType
from pyspark import SparkContext
import numpy as np
import csv
import math

spark = pyspark.sql.SparkSession.builder \
    .master("local") \
    .appName("movies") \
    .getOrCreate()

df =
spark.read.csv(path="./data/movielens/ratings.csv",
header=True)
df = df.withColumn("rating",
df["rating"].cast(IntegerType()))

averages = df \
    .groupBy("user_id") \
    .avg("rating") \
    .select("*")

joined = averages \
    .join(df, df["user_id"] == averages["user_id"]) \
    .select(df["user_id"], "avg(rating)", "rating",
"movie_id") \
    .collect()

sc = SparkContext.getOrCreate()
rdd = sc.parallelize(joined)


def seq_op(acc, obj):
    user_id = obj["user_id"]
```

```
    movie_id = obj["movie_id"]
    average = obj["average"]
    rating = obj["rating"]
    diff = obj["diff"]

    if user_id not in acc:
        acc[user_id] = {
            "average": average,
            "ratings": {
                movie_id: {
                    "rating": rating,
                    "diff": diff
                }
            }
        }
    else:
        acc[user_id]["ratings"][movie_id] = {
            "rating": rating,
            "diff": diff
        }
    return acc


combOp = (lambda x, y: {**x, **y})

mapped = rdd.map(lambda row: {"user_id":
row["user_id"],
                              "rating":
row["rating"],
                              "movie_id":
row["movie_id"],
                              "average":
row["avg(rating)"],
                              "diff": row["rating"] -
row["avg(rating)"]
                              }) \
    .aggregate({}, seq_op, combOp)
```

```
with open("averages.csv", "w") as the_file:
    writer = csv.writer(the_file)
    writer.writerow(("user", "average"))
    for user in mapped:
        writer.writerow((user, mapped[user]
["average"]))


def cosine_similarity(first_list, second_list):
    a = np.array(first_list)
    b = np.array(second_list)

    dot_product = np.dot(a, b)

    a_length = np.linalg.norm(a)
    b_length = np.linalg.norm(b)

    similarity = dot_product / (a_length * b_length)

    if math.isnan(similarity):
        similarity = 0

    similarity = "{0:.10f}".format(similarity)

    return similarity


def compare(user_id):
    comparison = set()
    user_movies = mapped[user_id]['ratings']

    for other_user_id in mapped:

        if int(user_id) != int(other_user_id):
            user = []
            other = []
            other_user_movies = mapped[other_user_id]
['ratings']
            for movie_id in other_user_movies:
```

```
                if movie_id in user_movies:  # movie
rated by both
                    user.append(user_movies[movie_id]
['diff'])

other.append(other_user_movies[movie_id]['diff'])

            # convert ids to int for correct
comparison
            user_id = int(user_id)
            other_user_id = int(other_user_id)

            smaller_id = user_id if user_id <
other_user_id else other_user_id
            bigger_id = user_id if user_id >
other_user_id else other_user_id

            similarity = cosine_similarity(user,
other)
            comparison.add((smaller_id, bigger_id,
similarity))

    return comparison


def combine_sets(set1, set2):
    set1.update(set2)
    return set1


print("Starting mapping")

user_ids = []

for key in mapped:
    user_ids.append(key)

result = sc.parallelize(user_ids) \
    .map(lambda user_id: compare(user_id)) \
```

```
    .aggregate(set(), combine_sets, combine_sets)


with open("comparison.csv", "w") as the_file:
    writer = csv.writer(the_file)
    writer.writerow(("first", "second",
"similarity"))
    for tup in result:
        writer.writerow(tup)
```

Problem 4.3

```
import pyspark
from pyspark import SparkContext
from pyspark.sql.types import FloatType


spark = pyspark.sql.SparkSession.builder \
    .master("local") \
    .appName("movies") \
    .getOrCreate()

df = spark.read.csv(path="./comparison.csv",
header=True)

df = df.withColumn("similarity",
df["similarity"].cast(FloatType()))\
    .orderBy("similarity", ascending=[0]) \
    .collect()

sc = SparkContext.getOrCreate()
rdd = sc.parallelize(df)


def seq_op(acc, row):
    similarity = row["similarity"]
```

```
    if row["first"] in acc:
        acc[row["first"]].append((row["second"],
similarity))
    else:
        acc[row["first"]] = [(row["second"],
similarity)]

    if row["second"] in acc:
        acc[row["second"]].append((row["first"],
similarity))
    else:
        acc[row["second"]] = [(row["first"],
similarity)]

    return acc


similarities = rdd.aggregate({}, seq_op, lambda x, y:
{**x, **y})


def map_averages(acc, row):
    acc[row["user"]] = row["average"]
    return acc


averages = spark.read.csv(path="./averages.csv",
header=True).rdd.aggregate({}, map_averages, lambda
x, y: {**x, **y})


movies =
spark.read.csv(path='./data/movielens/movies.csv',
header=True).collect()


def mapping(user, nearest):
```

```
    predicted_ratings = []
    user_average = averages[user]

    for movie in movies:
        for other in nearest:

sc.parallelize(similarities).map(lambda user:
mapping(user, similarities[user][:10])).collect()
```