EECS E6893
Homework 2
Jun Hu
jh3846

October 9, 2017

Contents

Chapter 1

Question 1 Part 1: Building an Explicit Movie Recommendation System with Spark MLlib

1.1 Import required libraries

```
In [1]: import subprocess
        import findspark
        findspark.init()
        from pyspark.sql import SparkSession
        from pyspark.ml.evaluation import RegressionEvaluator
        from pyspark.ml.recommendation import ALS
        from pyspark.sql import Row

        spark = SparkSession \
            .builder \
            .appName("recommendation_explicit") \
            .config("spark.som.config.option", "some-value") \
            .getOrCreate()
```

1.2 Download/Unzip the MovieLens 1M dataset from http://grouplens.org/datasets/movielens

```
In [2]: subprocess.call(["wget", "http://files.grouplens.org/datasets/movielens/ml-1m.zip"])
        subprocess.call(["unzip", "ml-1m.zip"])

Out[2]: 1
```

1.3 Read and Convert ratings data to a DataFrame

```
In [3]: lines = spark.read.text("./ml-1m/ratings.dat").rdd
        parts = lines.map(lambda row: row.value.split("::"))
        ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                             rating=float(p[2]), timestamp=int(p[3])))
        ratings = spark.createDataFrame(ratingsRDD)
```

1.4 Show the number of ratings in the dataset

```
In [4]: print("Number of ratings = " + str(ratings.count()))
```

```
Number of ratings = 1000209
```

1.5 Show a sample of the Ratings DataFrame

```
In [5]: ratings.sample(False, 0.0001, seed=0).show(10)
```

```
+-------+------+---------+------+
|movieId|rating|timestamp|userId|
+-------+------+---------+------+
|   2908|   5.0|977895809|    68|
|   3730|   5.0|978554445|   173|
|   2917|   2.0|976301830|   456|
|    589|   4.0|976161565|   526|
|   2348|   3.0|976207524|   533|
|   1285|   4.0|979154572|   588|
|   1206|   4.0|980628867|   711|
|   3361|   4.0|975510209|   730|
|   3203|   5.0|975435824|   779|
|   1196|   4.0|975356701|   843|
+-------+------+---------+------+
only showing top 10 rows
```

1.6 Show sample number of ratings per user

```
In [6]: grouped_ratings = ratings.groupBy("userId").count().withColumnRenamed("count", "No. of ratings"
        grouped_ratings.show(10)
```

```
+------+--------------+
|userId|No. of ratings|
+------+--------------+
|    26|           400|
|    29|           108|
|   474|           318|
|   964|            78|
|  1677|            43|
|  1697|           354|
|  1806|           214|
|  1950|           137|
|  2040|            46|
|  2214|            81|
+------+--------------+
only showing top 10 rows
```

1.7 Show the number of users in the dataset

```
In [7]: print("Number of users = " + str(grouped_ratings.count()))
```

```
Number of users = 6040
```

1.8 Split Ratings data into Training (80%) and Test (20%) datasets

```
In [8]: (training, test) = ratings.randomSplit([0.8, 0.2])
```

1.9 Show resulting Ratings dataset counts

```
In [9]: trainingRatio = float(training.count())/float(ratings.count())*100
        testRatio = float(test.count())/float(ratings.count())*100

        print("Total number of ratings = " + str(ratings.count()))
        print("Training dataset count = " + str(training.count()) + ", " + str(trainingRatio) + "%")
        print("Test dataset count = " + str(test.count()) + ", " + str(testRatio) + "%")
```

```
Total number of ratings = 1000209
Training dataset count = 800880, 80.07126510559293%
Test dataset count = 199329, 19.928734894407068%
```

1.10 Build the recommendation model on the training data using ALS-explicit

```
In [10]: als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
                   coldStartStrategy="drop")
         model = als.fit(training)
```

1.11 Run the model against the Test data and show a sample of the predictions

```
In [11]: predictions = model.transform(test).na.drop()
         predictions.show(10)
```

```
+-------+------+---------+------+----------+
|movieId|rating|timestamp|userId|prediction|
+-------+------+---------+------+----------+
|    148|   1.0|976295338|   840| 2.9349167|
|    148|   2.0|974875106|  1150| 2.9894443|
|    148|   2.0|974178993|  2456| 3.9975448|
|    463|   5.0|968916009|  3151|  3.967182|
|    463|   3.0|963746396|  4858| 2.0730953|
|    463|   4.0|973625620|  2629| 3.1774714|
|    463|   1.0|966523740|  3683| 1.1212827|
|    463|   2.0|966790403|  3562|  2.780132|
|    463|   4.0|975775726|   721| 3.3978982|
|    463|   3.0|965308300|  4252| 0.9944763|
```

```
+-------+------+---------+------+----------+
only showing top 10 rows
```

1.12 Evaluate the model by computing the RMSE on the test data

```
In [12]: predictions = model.transform(test)
         evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                         predictionCol="prediction")
         rmse = evaluator.evaluate(predictions)
         print("Root-mean-square error = " + str(rmse))
```

```
Root-mean-square error = 0.8908929362860674
```

1.13 Show that a smaller value of rmse is better

This is obviously the case since RMSE is an aggregation of all the error. Thus evaluator.isLargerBetter should be 'false'.

```
In [13]: evaluator.isLargerBetter()
```

```
Out[13]: False
```

1.14 Make movie recommendations

```
In [14]: # Generate top 10 movie recommendations for each user
         userRecs = model.recommendForAllUsers(10)
         # Generate top 10 user recommendations for each movie
         movieRecs = model.recommendForAllItems(10)
```

1.15 Show sample recommendations per user

```
In [15]: userRecs.sample(False, 0.01).show(10, False)
```

```
+------+--------------------------------------------------------------------------------------------
|userId|recommendations
+------+--------------------------------------------------------------------------------------------
|148   |[[1780,7.2854385], [1369,6.99533], [666,6.6703053], [2892,6.5549903], [1741,6.528875], [3523,6.0
|5173  |[[3245,7.7563887], [1038,7.52281], [3867,7.2047706], [632,7.0838833], [37,7.0073814], [751,6.936
|5695  |[[1458,9.663776], [3855,9.074218], [3106,9.053921], [2837,9.043263], [2192,8.797422], [2397,8.78
|1863  |[[962,6.392259], [2175,6.2921085], [2984,6.027778], [759,5.9641767], [3737,5.929455], [2284,5.91
|1924  |[[1038,8.618518], [219,7.9083204], [131,7.871811], [632,7.788521], [1458,7.681244], [1574,7.4738
|4610  |[[3670,6.8609476], [1117,6.645418], [2994,6.6018786], [2830,6.596518], [2934,6.505612], [3851,6.
|4104  |[[649,7.115762], [1421,6.597936], [3885,6.493393], [1585,6.441885], [1741,6.0131593], [503,5.839
|1249  |[[3636,8.443559], [1420,7.907082], [1664,7.8959613], [3456,7.7776465], [2697,7.7743106], [702,7.
|855   |[[3670,5.6403356], [557,5.452341], [503,4.9971642], [3338,4.9897413], [3012,4.9187536], [2830,4.
|5361  |[[3523,8.854711], [1662,7.23445], [2388,7.046192], [1780,6.9375844], [2892,6.929945], [2164,6.62
+------+--------------------------------------------------------------------------------------------
only showing top 10 rows
```

1.16 Show sample recommendations per user

```
In [16]: movieRecs.sample(False, 0.01).show(10, False)
```

```
+-------+---------------------------------------------------------------------------------------------------------------
|movieId|recommendations
+-------+---------------------------------------------------------------------------------------------------------------
|3844   |[[1213,7.3201046], [2441,6.9640417], [5297,6.8789372], [2549,6.8698826], [2816,6.507644], [197
|1031   |[[1070,5.9382234], [4143,5.8492775], [3897,5.841146], [2755,5.6947303], [4282,5.6827908], [527,
|26     |[[1213,7.0531287], [2640,6.3756685], [879,6.1351347], [2502,6.0931673], [5298,5.9518814], [642,
|626    |[[4504,9.705521], [3222,8.426963], [1713,8.153491], [5863,7.892766], [4583,7.852765], [3113,7.6
|3752   |[[5670,6.538592], [21,5.9881763], [5258,5.949679], [4393,5.7138], [4028,5.6019115], [1025,5.45
|2256   |[[745,7.8676734], [2469,7.4058766], [906,7.213084], [2431,7.1617584], [1754,7.1158795], [5030,7
|3793   |[[640,5.7342196], [5218,5.440282], [1673,5.2526026], [947,5.2225814], [2694,5.2105126], [2879,5
|2867   |[[745,5.992924], [2534,5.8074617], [527,5.6805005], [2755,5.653826], [283,5.3882546], [3587,5.3
|846    |[[4008,10.775237], [4504,10.658872], [3222,9.88133], [399,9.678963], [5240,9.402692], [144,9.30
|729    |[[665,11.115968], [1459,9.497441], [5803,7.76634], [1384,7.726793], [4317,7.657247], [640,7.614
+-------+---------------------------------------------------------------------------------------------------------------
only showing top 10 rows
```

Chapter 2

Question 1 Part 2: Building an Implicit Music Recommendation System with Spark MLlib

2.1 Import required libraries

```
In [2]: import subprocess
        import findspark
        findspark.init()
        from pyspark.sql import SparkSession
        from pyspark.ml.evaluation import RegressionEvaluator
        from pyspark.ml.recommendation import ALS
        from pyspark.ml.recommendation import ALSModel
        from pyspark.sql import Row

        spark = SparkSession \
            .builder \
            .appName("recommendation_implicit") \
            .config("spark.som.config.option", "some-value") \
            .getOrCreate()
```

2.2 Download/Unzip Audioscrobbler dataset from http://www.iro.umontreal.ca/~lisa/datasets/profiledata_06-May-2005.tar.gz

```
In [2]: # subprocess.call(["wget", "http://www.iro.umontreal.ca/~lisa/datasets/profiledata_06-May-2005.t
```

2.3 Read and Convert ratings data to a DataFrame

```
In [3]: lines = spark.read.text("./profiledata_06-May-2005/user_artist_data.txt").rdd
        parts = lines.map(lambda row: row.value.split(" ")).filter(lambda line: line!=None)
        userArtistRDD = parts.map(lambda p: Row(userId=int(p[0]), artistId=int(p[1]),
                                              count=int(p[2])))
        userArtist = spark.createDataFrame(userArtistRDD)
```

2.4 Show the number of userArtist in the dataset

```
In [6]: print("Number of userArtist = " + str(userArtist.count()))
```

```
Number of userArtist = 24296858
```

2.5 Show a sample of the userArtist DataFrame

```
In [4]: userArtist.sample(False, 0.0001, seed=23).show(10)
```

```
+--------+-----+-------+
|artistId|count| userId|
+--------+-----+-------+
| 1054292|    1|1000127|
| 1033246|    1|1000215|
|    1269|   13|1000357|
|     630|    1|1000410|
| 1000428|    2|1000657|
| 1234327|    1|1000911|
|      45|   34|1000923|
|     969|    4|1000927|
|    1235|    1|1000928|
|    3950|    2|1001009|
+--------+-----+-------+
only showing top 10 rows
```

2.6 Split userArtist data into Training (80%) and Test (20%) datasets

```
In [3]: (training, test) = userArtist.randomSplit([0.8, 0.2])
```

2.7 Show resulting userArtist dataset counts

```
In [9]: trainingRatio = float(training.count())/float(userArtist.count())*100
        testRatio = float(test.count())/float(userArtist.count())*100

        print("Total number of userArtist = " + str(userArtist.count()))
        print("Training dataset count = " + str(training.count()) + ", " + str(trainingRatio) + "%")
        print("Test dataset count = " + str(test.count()) + ", " + str(testRatio) + "%")
```

2.8 Build the recommendation model on the training data using ALS-implicit

```
In [7]: als = ALS(maxIter=5, regParam=0.01, implicitPrefs=True, userCol="userId", itemCol="artistId", ra
        model = als.fit(training)
```

2.9 Save and load model

```
In [4]: # from pyspark.mllib.recommendation import MatrixFactorizationModel
        # model.save("hdfs://localhost:9000/user/vibrioh/implicit_model")
        sameModel = ALSModel.load("hdfs://localhost:9000/user/vibrioh/implicit_model")
```

2.10 Run the model against the Test data and show a sample of the predictions

```
In [5]: predictions = sameModel.transform(test).na.drop()
        predictions.show(10)
```

```
+--------+-----+-------+----------+
|artistId|count| userId|prediction|
+--------+-----+-------+----------+
|     463|    1|1000117| 1.0815843|
|     463|    1|1000221| 0.3066332|
|     463|    1|1000401|0.41309264|
|     463|    1|1000463|0.92721707|
|     463|    1|1000614|0.66710955|
|     463|    1|1000745| 0.5759305|
|     463|    1|1000771|0.10025656|
|     463|    1|1000920|0.14627631|
|     463|    1|1001192| 0.4186052|
|     463|    1|1001239|0.11049299|
+--------+-----+-------+----------+
only showing top 10 rows
```

2.11 Show recommendations high and low

```
In [6]: predictions.registerTempTable("predictions")
        spark.sql("SELECT * FROM predictions ORDER BY prediction DESC").show()
        spark.sql("SELECT * FROM predictions ORDER BY prediction").show()
```

```
+--------+-----+-------+----------+
|artistId|count| userId|prediction|
+--------+-----+-------+----------+
|     393|   44|1044648|  2.048578|
| 1002862|    3|1000764| 2.0438032|
| 1245208|   16|1077252| 1.9122567|
|    1457|    3|1052461| 1.8336266|
|    4538|    1|1044648| 1.8334882|
| 1003361|    8|1052461| 1.7632964|
| 1105069|   12|1045876| 1.7556672|
|     670|   22|2058707| 1.7466245|
| 1034635|  208|1038380| 1.7438686|
| 1043348|   56|1021501| 1.7395341|
| 1003133|    3|1044648| 1.7320846|
| 1299851|   55|1007308| 1.7286341|
| 1296257|   27|1007308| 1.7270842|
| 1034635|   54|1047668| 1.7216785|
```

```
| 1031984|    3|1038826|  1.719785|
| 1001169|    9|1072865| 1.7158957|
|    2842|    3|1077252| 1.7109416|
|    2017|  128|2089146|  1.698533|
| 1012494|    5|1070844| 1.6947658|
| 1034635|    4|1001562| 1.6939092|
+--------+-----+-------+----------+
only showing top 20 rows

+--------+-----+-------+-----------+
|artistId|count| userId| prediction|
+--------+-----+-------+-----------+
|     606|    1|1072273| -1.1034482|
| 1062984|    5|1052461| -1.0605614|
| 1002751|    1|2005325| -0.9165062|
| 1000660|    1|2114152|-0.82818604|
| 1000270|    1|1070177| -0.7904455|
| 1001428|    1|2019216|-0.78354007|
|    4569|    4|1072273| -0.7598609|
| 1010728|   22|1054893| -0.7471355|
| 1006594|    3|2231283| -0.7382127|
| 1020783|    6|2200013| -0.6836228|
|    1400|    4|2287446| -0.6611168|
| 1003458|    1|1003897| -0.6341311|
| 1007027|    8|1000647|  -0.631236|
| 1003084|    1|1063655|-0.62958777|
|    2138|    1|2147892| -0.6157164|
|    1179|    2|1055807| -0.6106846|
|    1223|    1|2269169| -0.6081734|
| 1000418|    1|1020855| -0.5983083|
| 1002451|    2|2216293| -0.5928024|
| 1027267|    1|2200013|-0.58894813|
+--------+-----+-------+-----------+
only showing top 20 rows
```

Chapter 3

Question 2 Part 1: Clustering on News Articles

3.1 Download news articles from: https://www.kaggle.com/asad1m9a9h6mood/nev articles

3.1.1 This Dataset is scraped from https://www.thenews.com.pk website. It has news articles from 2015 till date related to business and sports. It Contains the Heading of the particular Article, Its content and its date. The content also contains the place from where the statement or Article was published.

3.2 Import required libraries

```
In [6]: from pyspark.ml import Pipeline
        from pyspark.ml.clustering import BisectingKMeans, KMeans, GaussianMixture
        from pyspark.ml.feature import HashingTF, Tokenizer, NGram, IDF, StopWordsRemover
        from pyspark.sql import Row
        import os
        import pandas as pd
        from pyspark.sql import SparkSession


        spark = SparkSession \
            .builder \
            .appName("news_clustering") \
            .config("spark.som.config.option", "some-value") \
            .getOrCreate()
```

3.3 Explore the dataset in Pandas

```
In [7]: data_df = pd.read_csv("./Articles.csv",sep=",", encoding = "ISO-8859-1")
        data_df.head()
```

```
Out[7]: <div>
        <style>
            .dataframe thead tr:only-child th {
                text-align: right;
            }
```

```
    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>Article</th>
      <th>Date</th>
      <th>Heading</th>
      <th>NewsType</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>0</th>
      <td>KARACHI: The Sindh government has decided to b...</td>
      <td>1/1/2015</td>
      <td>sindh govt decides to cut public transport far...</td>
      <td>business</td>
    </tr>
    <tr>
      <th>1</th>
      <td>HONG KONG: Asian markets started 2015 on an up...</td>
      <td>1/2/2015</td>
      <td>asia stocks up in new year trad</td>
      <td>business</td>
    </tr>
    <tr>
      <th>2</th>
      <td>HONG KONG:  Hong Kong shares opened 0.66 perce...</td>
      <td>1/5/2015</td>
      <td>hong kong stocks open 0.66 percent lower</td>
      <td>business</td>
    </tr>
    <tr>
      <th>3</th>
      <td>HONG KONG: Asian markets tumbled Tuesday follo...</td>
      <td>1/6/2015</td>
      <td>asian stocks sink euro near nine year</td>
      <td>business</td>
    </tr>
    <tr>
      <th>4</th>
      <td>NEW YORK: US oil prices Monday slipped below $...</td>
      <td>1/6/2015</td>
      <td>us oil prices slip below 50 a barr</td>
      <td>business</td>
```

```
    </tr>
  </tbody>
</table>
</div>
```

In [8]:
```
data_df.describe()
```

Out[8]:
```
<div>
<style>
    .dataframe thead tr:only-child th {
        text-align: right;
    }

    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>Article</th>
      <th>Date</th>
      <th>Heading</th>
      <th>NewsType</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>count</th>
      <td>2692</td>
      <td>2692</td>
      <td>2692</td>
      <td>2692</td>
    </tr>
    <tr>
      <th>unique</th>
      <td>2584</td>
      <td>666</td>
      <td>2581</td>
      <td>2</td>
    </tr>
    <tr>
      <th>top</th>
      <td>strong&gt;TOKYO: Tokyo stocks climbed in early tr...</td>
      <td>8/1/2016</td>
      <td>Tokyo stocks open lower after BoJ under</td>
      <td>sports</td>
    </tr>
    <tr>
      <th>freq</th>
```

```
            <td>5</td>
            <td>27</td>
            <td>5</td>
            <td>1408</td>
          </tr>
        </tbody>
      </table>
      </div>
```

```
In [9]: data = spark.createDataFrame(data_df)
        data.printSchema()
        data.sample(False, 0.05).show(5)
```

```
root
 |-- Article: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- Heading: string (nullable = true)
 |-- NewsType: string (nullable = true)

+--------------------+---------+--------------------+--------+
|             Article|     Date|             Heading|NewsType|
+--------------------+---------+--------------------+--------+
|HONG KONG: Asian ...| 1/6/2015|asian stocks sink...|business|
|ISLAMABAD: The Na...|1/23/2015|nepra prevents k ...|business|
|ISLAMABAD: Pakist...|1/26/2015|pakistan fuel cri...|business|
|ISLAMABAD: Federa...| 3/4/2015|pact with k elect...|business|
|London: Oil price...| 3/6/2015|oil prices rise b...|business|
+--------------------+---------+--------------------+--------+
only showing top 5 rows
```

3.4 Split data into Training (80%) and Test (20%) datasets

```
In [10]: (training, test) = data.randomSplit([0.8, 0.2])
```

3.5 Configure an ML pipeline

```
In [12]: tokenizer = Tokenizer(inputCol="Article", outputCol="words")
         remover = StopWordsRemover(inputCol=tokenizer.getOutputCol(),outputCol="filtered")
         hashingTF = HashingTF(inputCol=remover.getOutputCol(), outputCol="features",numFeatures=104857
```

3.6 Clustering by K-MEANS

```
In [13]: kmeans = KMeans().setK(2).setSeed(1)
         pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, kmeans])
         model = pipeline.fit(training)
```

3.7 Make predictions on test and print interested columns of different clusters

In [20]:
```
predictions = model.transform(test)
predictions.registerTempTable("predictions")
spark.sql("SELECT Article, NewsType, prediction FROM predictions WHERE NewsType = 'business'")
spark.sql("SELECT Article, NewsType, prediction FROM predictions WHERE NewsType = 'sports'").s
```

```
+--------------------+--------+----------+
|             Article|NewsType|prediction|
+--------------------+--------+----------+
|A major rally in ...|business|         1|
|ATLANTA: Twelve P...|business|         1|
|BEIJING: Pakistan...|business|         0|
|Brussels: The EU ...|business|         0|
|DUBAI: Talks betw...|business|         0|
|HONG KONG: Hong K...|business|         0|
|Hong Kong: Asian ...|business|         1|
|Hong Kong: Asian ...|business|         1|
|Hong Kong: Asian ...|business|         1|
|Hong Kong: Asian ...|business|         1|
+--------------------+--------+----------+
only showing top 10 rows

+--------------------+--------+----------+
|             Article|NewsType|prediction|
+--------------------+--------+----------+
|AUCKLAND: Martin ...|  sports|         0|
|Australia win run...|  sports|         0|
|CAPE TOWN: Alex H...|  sports|         0|
|CAPE TOWN: Captai...|  sports|         0|
|CAPE TOWN: Poor w...|  sports|         0|
|CAPE TOWN: Tiny T...|  sports|         0|
|DHAKA: Bangladesh...|  sports|         0|
|DHAKA: Bangladesh...|  sports|         0|
|DHAKA: Captain of...|  sports|         0|
|DHAKA: Hasan Mohs...|  sports|         0|
+--------------------+--------+----------+
only showing top 10 rows
```

Chapter 4

Question 2 Part 2: Clustering on Wikipedia articles

4.1 Import required libraries

```
In [51]: from pyspark.ml import Pipeline
         from pyspark.ml.clustering import BisectingKMeans, KMeans, GaussianMixture
         from pyspark.ml.feature import HashingTF, Tokenizer, NGram, IDF, StopWordsRemover
         from pyspark.sql import Row
         import os
         import pandas as pd
         from pyspark.sql import SparkSession
         import subprocess
         import json


         spark = SparkSession \
             .builder \
             .appName("wiki_clustering") \
             .config("spark.som.config.option", "some-value") \
             .getOrCreate()
```

4.2 Download/Unzip https://dumps.wikimedia.org/enwiki/20170920/enwiki-20170920-pages-articles14.xml-p7697599p7744799.bz2

```
In [52]: #subprocess.call(["wget", "https://dumps.wikimedia.org/enwiki/20170920/enwiki-20170920-pages-a
```

4.3 Parse xml to json using WikiExtractorw (https://github.com/attardi/wikiextract

```
In [53]: # subprocess.call(["python3", "WikiExtractor.py", "-o wiki_extracted", "--json", "-b 230M", "/
```

4.4 Explore the dataset in Pandas

```
In [54]: with open('/Users/vibrioh/local_projects/spark/ wiki_extracted/AA/wiki_00', encoding='utf-8') a
             data_json = []
             for line in f:
                 data_json.append(json.loads(line))
```

```
pd_df = pd.DataFrame(data_json)
pd_df.head()
```

Out[54]:
```
<div>
<style>
    .dataframe thead tr:only-child th {
        text-align: right;
    }

    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>id</th>
      <th>text</th>
      <th>title</th>
      <th>url</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>0</th>
      <td>7697605</td>
      <td>Konica Minolta Cup\n\nKonica Minolta Cup may r...</td>
      <td>Konica Minolta Cup</td>
      <td>https://en.wikipedia.org/wiki?curid=7697605</td>
    </tr>
    <tr>
      <th>1</th>
      <td>7697611</td>
      <td>Archer (typeface)\n\nArcher is a slab serif ty...</td>
      <td>Archer (typeface)</td>
      <td>https://en.wikipedia.org/wiki?curid=7697611</td>
    </tr>
    <tr>
      <th>2</th>
      <td>7697612</td>
      <td>Stockton Airport\n\nStockton Airport may refer...</td>
      <td>Stockton Airport</td>
      <td>https://en.wikipedia.org/wiki?curid=7697612</td>
    </tr>
    <tr>
      <th>3</th>
      <td>7697626</td>
      <td>Ricky Minard\n\nRicky Donell Minard Jr. (born ...</td>
      <td>Ricky Minard</td>
```

```
      <td>https://en.wikipedia.org/wiki?curid=7697626</td>
    </tr>
    <tr>
      <th>4</th>
      <td>7697641</td>
      <td>Alexander Peya\n\nAlexander Peya (born 27 June...</td>
      <td>Alexander Peya</td>
      <td>https://en.wikipedia.org/wiki?curid=7697641</td>
    </tr>
  </tbody>
</table>
</div>
```

In [55]:
```
pd_df.describe()
```

Out[55]:
```
<div>
<style>
    .dataframe thead tr:only-child th {
        text-align: right;
    }

    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>id</th>
      <th>text</th>
      <th>title</th>
      <th>url</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>count</th>
      <td>4577</td>
      <td>4577</td>
      <td>4577</td>
      <td>4577</td>
    </tr>
    <tr>
      <th>unique</th>
      <td>4577</td>
      <td>4577</td>
      <td>4577</td>
      <td>4577</td>
    </tr>
    <tr>
```

```
      <th>top</th>
      <td>7736826</td>
      <td>Lebe lauter\n\nLebe lauter () is the third stu...</td>
      <td>Cyprus at the 1988 Winter Olympics</td>
      <td>https://en.wikipedia.org/wiki?curid=7716931</td>
    </tr>
    <tr>
      <th>freq</th>
      <td>1</td>
      <td>1</td>
      <td>1</td>
      <td>1</td>
    </tr>
  </tbody>
</table>
</div>
```

```
In [56]: data = spark.createDataFrame(pd_df)
         data.printSchema()
         data.sample(False, 0.27).show(3)
```

```
root
 |-- id: string (nullable = true)
 |-- text: string (nullable = true)
 |-- title: string (nullable = true)
 |-- url: string (nullable = true)

+-------+--------------------+--------------------+--------------------+
|     id|                text|               title|                 url|
+-------+--------------------+--------------------+--------------------+
|7697612|Stockton Airport
...|    Stockton Airport|https://en.wikipe...|
|7697675|Lobo (wrestler)

...|     Lobo (wrestler)|https://en.wikipe...|
|7697715|Anti-submarine mi...|Anti-submarine mi...|https://en.wikipe...|
+-------+--------------------+--------------------+--------------------+
only showing top 3 rows
```

4.5 Split data into Training (80%) and Test (20%) datasets

```
In [57]: (training, test) = data.randomSplit([0.8, 0.2])
```

4.6 Configure an ML pipeline

```
In [58]: tokenizer = Tokenizer(inputCol="text", outputCol="words")
         remover = StopWordsRemover(inputCol=tokenizer.getOutputCol(),outputCol="filtered")
         hashingTF = HashingTF(inputCol=remover.getOutputCol(), outputCol="features",numFeatures=350)
```

4.7 Clustering by K-MEANS

In [59]:
```
kmeans = KMeans().setK(2).setSeed(1)
pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, kmeans])
model = pipeline.fit(data)
```

4.8 Make predictions on test and print interested columns of different clusters

In [61]:
```
predictions = model.transform(data)
predictions.registerTempTable("predictions")
spark.sql("SELECT id, title, prediction FROM predictions WHERE prediction = '0'").show(10)
spark.sql("SELECT id, title, prediction FROM predictions WHERE prediction = '1'").show(10)
spark.sql("SELECT count(*) FROM predictions GROUP BY prediction").show()
```

```
+-------+--------------------+----------+
|     id|               title|prediction|
+-------+--------------------+----------+
|7697605|  Konica Minolta Cup|         0|
|7697611|    Archer (typeface)|         0|
|7697612|     Stockton Airport|         0|
|7697626|         Ricky Minard|         0|
|7697641|       Alexander Peya|         0|
|7697655|   Swiss chalet style|         0|
|7697664|European Federati...|         0|
|7697671|The Best Is Yet t...|         0|
|7697675|       Lobo (wrestler)|         0|
|7697715|Anti-submarine mi...|         0|
+-------+--------------------+----------+
only showing top 10 rows

+-------+--------------------+----------+
|     id|               title|prediction|
+-------+--------------------+----------+
|7698038|       Radamel Falcao|         1|
|7698053| Panjshir offensives|         1|
|7698941|Istanbul High School|         1|
|7699151|The Market for Li...|         1|
|7699200|      Parchís (group)|         1|
|7700918|             Manikata|         1|
|7701000|2007 Major League...|         1|
|7701470|World War II pers...|         1|
|7701711|       Luck by Chance|         1|
|7702313|        Yoga Vasistha|         1|
+-------+--------------------+----------+
only showing top 10 rows

+--------+
|count(1)|
+--------+
|     154|
|    4423|
+--------+
```

Chapter 5

Question 3 Part 1: Fist Dataset with Logistic Regression and NaiveBayes classification

5.1 Import required labraries

```
In [100]: from pyspark.ml import Pipeline, PipelineModel
          from pyspark.ml.clustering import BisectingKMeans, KMeans, GaussianMixture
          from pyspark.ml.feature import HashingTF, Tokenizer, NGram, IDF, StopWordsRemover
          import os
          import pandas as pd
          from pyspark.sql import SparkSession
          from pyspark.sql.types import *
          from pyspark.sql import Row
          from pyspark.ml.linalg import Vectors
          from pyspark.ml.classification import LogisticRegression, NaiveBayes
          from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
          from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvalu

          spark = SparkSession \
              .builder \
              .appName("dataset1_classification") \
              .config("spark.som.config.option", "some-value") \
              .getOrCreate()
```

5.2 Read and Vectorize the raw data

```
In [101]: with open('/Users/vibrioh/local_projects/spark/datasets/ta/set1/pima-indians-diabetes.data', 
              col1 = []
              col2 = []
              for line in f:
                  parts = line.split(',')
                  label = float(parts[len(parts)-1])
                  features = Vectors.dense([float(parts[x]) for x in range(0,len(parts)-1)])
                  col1.append(label)
                  col2.append(features)
              dict = {"label": col1, "features": col2}
```

```
pd_d1 = pd.DataFrame(dict)
pd_d1.head()
```

Out[101]:

```
<div>
<style>
    .dataframe thead tr:only-child th {
        text-align: right;
    }

    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>features</th>
      <th>label</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>0</th>
      <td>[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0]</td>
      <td>1.0</td>
    </tr>
    <tr>
      <th>1</th>
      <td>[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0]</td>
      <td>0.0</td>
    </tr>
    <tr>
      <th>2</th>
      <td>[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0]</td>
      <td>1.0</td>
    </tr>
    <tr>
      <th>3</th>
      <td>[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0]</td>
      <td>0.0</td>
    </tr>
    <tr>
      <th>4</th>
      <td>[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 3...</td>
      <td>1.0</td>
    </tr>
  </tbody>
</table>
</div>
```

```
In [102]: data1 = spark.createDataFrame(pd_d1)
          data1.printSchema()
          data1.sample(False, 0.27).show(3)
```

```
root
 |-- features: vector (nullable = true)
 |-- label: double (nullable = true)

+--------------------+-----+
|            features|label|
+--------------------+-----+
|[8.0,183.0,64.0,0...|  1.0|
|[1.0,89.0,66.0,23...|  0.0|
|[10.0,115.0,0.0,0...|  0.0|
+--------------------+-----+
only showing top 3 rows
```

5.3 Split data into Training (80%) and Test (20%) datasets

```
In [103]: (training, test) = data1.randomSplit([0.8, 0.2], seed=23)
```

5.4 Train Logistic Regression and Naive Bayes models

```
In [104]: # create the trainer and set its parameters
          lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

          # Fit the model
          lrModel = lr.fit(training)

          # create the trainer and set its parameters
          nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

          # train the model
          nbModel = nb.fit(training)
```

5.5 Display the predictions

```
In [106]: lrPredictions = lrModel.transform(test)
          print("Logistic Regression classifier predictions")
          lrPredictions.show()
          nbPredictions = nbModel.transform(test)
          print("Naive Bayes classifier predictions")
          nbPredictions.show()
```

```
Logistic Regression classifier predictions
+--------------------+-----+--------------------+-----------+----------+
|            features|label|       rawPrediction|probability|prediction|
+--------------------+-----+--------------------+-----------+----------+
|[0.0,118.0,84.0,4...|  1.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[0.0,119.0,64.0,1...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
```

```
|[0.0,125.0,96.0,0...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[0.0,146.0,82.0,0...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,0.0,74.0,20...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,89.0,66.0,23...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,95.0,66.0,13...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,101.0,50.0,1...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,109.0,56.0,2...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,115.0,70.0,3...|  1.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,126.0,56.0,2...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[1.0,163.0,72.0,0...|  1.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[2.0,85.0,65.0,0...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[2.0,100.0,66.0,2...|  1.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[2.0,110.0,74.0,2...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[3.0,83.0,58.0,31...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[3.0,88.0,58.0,11...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[3.0,128.0,78.0,0...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[3.0,158.0,76.0,3...|  1.0|[0.61903920840622...|[0.65,0.35]|       0.0|
|[4.0,103.0,60.0,3...|  0.0|[0.61903920840622...|[0.65,0.35]|       0.0|
+--------------------+-----+--------------------+----------+----------+
only showing top 20 rows

Naive Bayes classifier predictions
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[0.0,118.0,84.0,4...|  1.0|[-940.61590258077...|[0.00222035512724...|       1.0|
|[0.0,119.0,64.0,1...|  0.0|[-570.93042316226...|[0.34024409829781...|       1.0|
|[0.0,125.0,96.0,0...|  0.0|[-398.51412035553...|[0.99984493731386...|       0.0|
|[0.0,146.0,82.0,0...|  0.0|[-507.15808445073...|[0.99944541772140...|       0.0|
|[1.0,0.0,74.0,20...|  0.0|[-333.42573140269...|[0.99999659118712...|       0.0|
|[1.0,89.0,66.0,23...|  0.0|[-537.77736461550...|[0.72794260819999...|       0.0|
|[1.0,95.0,66.0,13...|  0.0|[-419.94009000027...|[0.98632919581248...|       0.0|
|[1.0,101.0,50.0,1...|  0.0|[-418.12769782653...|[0.92458101473041...|       0.0|
|[1.0,109.0,56.0,2...|  0.0|[-604.05216987817...|[0.00475046122727...|       1.0|
|[1.0,115.0,70.0,3...|  1.0|[-639.76679384547...|[0.82243219784355...|       0.0|
|[1.0,126.0,56.0,2...|  0.0|[-675.05572884516...|[0.00121490185273...|       1.0|
|[1.0,163.0,72.0,0...|  1.0|[-481.24197102653...|[0.99041485445669...|       0.0|
|[2.0,85.0,65.0,0...|  0.0|[-373.38413562769...|[0.99933714490051...|       0.0|
|[2.0,100.0,66.0,2...|  1.0|[-572.78674440437...|[0.66969877398212...|       0.0|
|[2.0,110.0,74.0,2...|  0.0|[-671.37589458991...|[0.29372603707675...|       1.0|
|[3.0,83.0,58.0,31...|  0.0|[-457.68905156759...|[0.99967549740400...|       0.0|
|[3.0,88.0,58.0,11...|  0.0|[-432.71723920739...|[0.83926761944996...|       0.0|
|[3.0,128.0,78.0,0...|  0.0|[-464.07127210943...|[0.99897195376000...|       0.0|
|[3.0,158.0,76.0,3...|  1.0|[-939.34332498455...|[1.87311685420356...|       1.0|
|[4.0,103.0,60.0,3...|  0.0|[-761.19283980910...|[2.13987700098319...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

5.6 Evalue the modles with AUC and overall Accuracy

```
In [107]: evaluator1 = BinaryClassificationEvaluator().setMetricName("areaUnderROC")
          evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", m
          print ("Logistic Regression: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(lrPred
          print ("Naive Bayes: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(nbPredictions)
```

```
Logistic Regression:
          Area under ROC curve (AUC): 0.5
         Accuracy:  0.6554054054054054
Naive Bayes:
          Area under ROC curve (AUC): 0.29694764503739646
         Accuracy:  0.7027027027027027
```

5.7 A simple comparison on the two modles's performance

5.7.1 Overall Accuracy:

- Naive Bayes classifier carried out a higher overall accuracy (0.70 vs 0.66)
- Both modle can yield predictions accuracy than 0.5 of random
- So Naive Bayes made better classification over Logistic Regression on this training/test set

5.7.2 Area under ROC curve (AUC):

- Logistic Regression classifier carried out a higher AUC (0.5 vs 0.3)
- Logistic Regression classifier has higher discriminative power over class distribution

5.7.3 In summary, on the selected training/test set, Naive Bayes classifier has the better result. However, the Logistic Regression classifier may be more stable on other datasets.

Chapter 6

Question 3 Part 2: Second Dataset with Logistic Regression and NaiveBayes classification

6.1 Import required labraries

```
In [108]: from pyspark.ml import Pipeline, PipelineModel
          from pyspark.ml.clustering import BisectingKMeans, KMeans, GaussianMixture
          from pyspark.ml.feature import HashingTF, Tokenizer, NGram, IDF, StopWordsRemover
          import os
          import pandas as pd
          from pyspark.sql import SparkSession
          from pyspark.sql.types import *
          from pyspark.sql import Row
          from pyspark.ml.linalg import Vectors
          from pyspark.ml.classification import LogisticRegression, NaiveBayes
          from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
          from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvalu

          spark = SparkSession \
              .builder \
              .appName("dataset2_classification") \
              .config("spark.som.config.option", "some-value") \
              .getOrCreate()
```

6.2 Read and Vectorize the raw data

```
In [109]: with open('/Users/vibrioh/local_projects/spark/datasets/ta/set2/australian.dat', encoding='ut
              col1 = []
              col2 = []
              for line in f:
                  parts = line.split(' ')
                  label = float(parts[len(parts)-1])
                  features = Vectors.dense([float(parts[x]) for x in range(0,len(parts)-1)])
                  col1.append(label)
                  col2.append(features)
              dict = {"label": col1, "features": col2}
```

```
pd_d2 = pd.DataFrame(dict)
pd_d2.head()
```

Out[109]:
```
<div>
<style>
    .dataframe thead tr:only-child th {
        text-align: right;
    }

    .dataframe thead th {
        text-align: left;
    }

    .dataframe tbody tr th {
        vertical-align: top;
    }
</style>
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>features</th>
      <th>label</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>0</th>
      <td>[1.0, 22.08, 11.46, 2.0, 4.0, 4.0, 1.585, 0.0,...</td>
      <td>0.0</td>
    </tr>
    <tr>
      <th>1</th>
      <td>[0.0, 22.67, 7.0, 2.0, 8.0, 4.0, 0.165, 0.0, 0...</td>
      <td>0.0</td>
    </tr>
    <tr>
      <th>2</th>
      <td>[0.0, 29.58, 1.75, 1.0, 4.0, 4.0, 1.25, 0.0, 0...</td>
      <td>0.0</td>
    </tr>
    <tr>
      <th>3</th>
      <td>[0.0, 21.67, 11.5, 1.0, 5.0, 3.0, 0.0, 1.0, 1...</td>
      <td>1.0</td>
    </tr>
    <tr>
      <th>4</th>
      <td>[1.0, 20.17, 8.17, 2.0, 6.0, 4.0, 1.96, 1.0, 1...</td>
      <td>1.0</td>
    </tr>
  </tbody>
</table>
</div>
```

```
In [110]: data2 = spark.createDataFrame(pd_d2)
          data2.printSchema()
          data2.sample(False, 0.27).show(3)
```

```
root
 |-- features: vector (nullable = true)
 |-- label: double (nullable = true)

+--------------------+-----+
|            features|label|
+--------------------+-----+
|[0.0,21.67,11.5,1...|  1.0|
|[1.0,20.17,8.17,2...|  1.0|
|[0.0,15.83,0.585,...|  1.0|
+--------------------+-----+
only showing top 3 rows
```

6.3 Split data into Training (80%) and Test (20%) datasets

```
In [111]: (training, test) = data2.randomSplit([0.8, 0.2], seed=23)
```

6.4 Train Logistic Regression and Naive Bayes models

```
In [112]: # create the trainer and set its parameters
          lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

          # Fit the model
          lrModel = lr.fit(training)

          # create the trainer and set its parameters
          nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

          # train the model
          nbModel = nb.fit(training)
```

6.5 Display the predictions

```
In [114]: lrPredictions = lrModel.transform(test)
          print("Logistic Regression classifier predictions")
          lrPredictions.show()
          nbPredictions = nbModel.transform(test)
          print("Naive Bayes classifier predictions")
          nbPredictions.show()
```

```
Logistic Regression classifier predictions
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[0.0,20.42,10.5,1...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[0.0,20.75,10.25,...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
```

```
|[0.0,20.75,10.335...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,22.67,7.0,2...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[0.0,24.75,12.5,2...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,30.67,12.0,2...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,32.17,1.46,2...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,35.75,0.915,...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,38.92,1.665,...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[0.0,39.08,4.0,2...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[0.0,47.0,13.0,2...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[0.0,55.75,7.08,2...|  0.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[1.0,16.17,0.04,2...|  1.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[1.0,19.0,0.0,1.0...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[1.0,20.0,1.25,1...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[1.0,21.5,11.5,2...|  0.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[1.0,22.0,0.79,2...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[1.0,22.67,1.585,...|  1.0|[-0.1767811960708...|[0.45591944020243...|       1.0|
|[1.0,23.08,0.0,2...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
|[1.0,23.75,0.415,...|  0.0|[0.53467906597270...|[0.63057376704516...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows

Naive Bayes classifier predictions
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[0.0,20.42,10.5,1...|  0.0|[-374.71894703985...|[1.0,7.1691492545...|       0.0|
|[0.0,20.75,10.25,...|  1.0|[-272.61645936976...|[1.0,7.9512154983...|       0.0|
|[0.0,20.75,10.335...|  1.0|[-358.18373217373...|[1.0,2.7828549813...|       0.0|
|[0.0,22.67,7.0,2...|  0.0|[-296.52550073657...|[1.0,6.1034404981...|       0.0|
|[0.0,24.75,12.5,2...|  1.0|[-882.80205063407...|[9.16938606433626...|       1.0|
|[0.0,30.67,12.0,2...|  1.0|[-437.66967570185...|[1.0,2.3001494432...|       0.0|
|[0.0,32.17,1.46,2...|  1.0|[-2172.3166551623...|           [0.0,1.0]|       1.0|
|[0.0,35.75,0.915,...|  1.0|[-1569.0382419969...|           [0.0,1.0]|       1.0|
|[0.0,38.92,1.665,...|  0.0|[-516.68712024168...|[2.85619336654972...|       1.0|
|[0.0,39.08,4.0,2...|  0.0|[-590.51320509544...|           [1.0,0.0]|       0.0|
|[0.0,47.0,13.0,2...|  1.0|[-361.69906371960...|[1.0,4.2718077713...|       0.0|
|[0.0,55.75,7.08,2...|  0.0|[-465.09686661564...|[1.0,3.1852541001...|       0.0|
|[1.0,16.17,0.04,2...|  1.0|[-127.34207448934...|[1.0,1.2049030087...|       0.0|
|[1.0,19.0,0.0,1.0...|  0.0|[-154.55466029514...|[1.0,2.9832451582...|       0.0|
|[1.0,20.0,1.25,1...|  0.0|[-232.95891654432...|[1.0,1.7522842178...|       0.0|
|[1.0,21.5,11.5,2...|  0.0|[-328.39292616749...|[1.0,1.4690448845...|       0.0|
|[1.0,22.0,0.79,2...|  0.0|[-733.32230070910...|[1.0,1.8294086996...|       0.0|
|[1.0,22.67,1.585,...|  1.0|[-285.51305623302...|[1.0,4.5883044328...|       0.0|
|[1.0,23.08,0.0,2...|  0.0|[-208.73577776562...|[1.0,6.0553106679...|       0.0|
|[1.0,23.75,0.415,...|  0.0|[-268.99747875558...|[1.0,3.0571078699...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

6.6 Evalue the modles with AUC and overall Accuracy

```
In [116]: evaluator1 = BinaryClassificationEvaluator().setMetricName("areaUnderROC")
          evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", m
          print ("Logistic Regression: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(lrPred
          print ("Naive Bayes: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(nbPredictions)
```

```
Logistic Regression:
          Area under ROC curve (AUC): 0.8758116883116883
         Accuracy:  0.8646616541353384
Naive Bayes:
          Area under ROC curve (AUC): 0.3773191094619667
         Accuracy:  0.631578947368421
```

6.7 A simple comparison on the two modles's performance

6.7.1 Overall Accuracy:

- Logistic Regression classifier carried out a higher overall accuracy (0.86 vs 0.63)
- Both modle can yield predictions accuracy than 0.5 of random
- So Logistic Regression made better classification over Naive Bayes on this training/test set

6.7.2 Area under ROC curve (AUC):

- Logistic Regression classifier carried out a higher AUC (0.88 vs 0.38)
- Logistic Regression classifier has higher discriminative power over class distribution

6.7.3 In summary, the Logistic Regression classifier has much better performance over the Naive Bayes, whatever on this perticular training/test set or potential future test datasets.

Chapter 7

Question 3 Part 3: Wikipedia Dataset with Logistic Regression and NaiveBayes classification

7.1 Import required labraries

```
In [117]: from pyspark.ml import Pipeline, PipelineModel
          from pyspark.ml.clustering import BisectingKMeans, KMeans, GaussianMixture
          from pyspark.ml.feature import HashingTF, Tokenizer, NGram, IDF, StopWordsRemover
          import os
          import pandas as pd
          from pyspark.sql import SparkSession
          from pyspark.sql.types import *
          from pyspark.sql import Row
          from pyspark.ml.linalg import Vectors
          from pyspark.ml.classification import LogisticRegression, NaiveBayes
          from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
          from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvalu

          spark = SparkSession \
              .builder \
              .appName("wikipedia_classification") \
              .config("spark.som.config.option", "some-value") \
              .getOrCreate()
```

7.2 Read the raw data and assigne lables with clustering results by K-MEANS

```
In [120]: wikiDF = spark.sql("SELECT id, title, text, prediction as assignedLable FROM predictions")
          #print schema
          wikiDF.printSchema()
          wikiDF.registerTempTable("wikiDF")
          wikiDF.show(2)
          spark.sql("SELECT count(*) FROM wikiDF GROUP BY assignedLable").show()
```

```
root
 |-- id: string (nullable = true)
```

```
 |-- title: string (nullable = true)
 |-- text: string (nullable = true)
 |-- assignedLable: integer (nullable = true)

+-------+------------------+--------------------+-------------+
|     id|             title|                text|assignedLable|
+-------+------------------+--------------------+-------------+
|7697605|Konica Minolta Cup|Konica Minolta Cu...|            0|
|7697611| Archer (typeface)|Archer (typeface)...|            0|
+-------+------------------+--------------------+-------------+
only showing top 2 rows

+--------+
|count(1)|
+--------+
|     154|
|    4423|
+--------+
```

7.3 Cast assigned lables to double and store to DataFrame

```
In [121]: wikiDF = wikiDF.withColumn("label", wikiDF.assignedLable.cast("double"))
          wikiDF.printSchema()
          wikiDF.sample(False, 0.23).show(2)
```

```
root
 |-- id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- text: string (nullable = true)
 |-- assignedLable: integer (nullable = true)
 |-- label: double (nullable = true)

+-------+--------------------+--------------------+-------------+-----+
|     id|               title|                text|assignedLable|label|
+-------+--------------------+--------------------+-------------+-----+
|7697715|Anti-submarine mi...|Anti-submarine mi...|            0|  0.0|
|7697725|Northern river shark|Northern river sh...|            0|  0.0|
+-------+--------------------+--------------------+-------------+-----+
only showing top 2 rows
```

7.4 Split data into Training (80%) and Test (20%) datasets

```
In [123]: training, test = wikiDF.randomSplit([0.8, 0.2], 2388)
          print ("Total document count:",wikiDF.count())
          print ("Training-set count:",training.count())
          print ("Test-set count:",test.count())
```

```
Total document count: 4577
Training-set count: 3645
```

```
Test-set count: 932
```

7.5 Config the pipeines for Logistic Regression and Naive Bayes

```
In [124]: tokenizer = Tokenizer().setInputCol("text").setOutputCol("words")
          remover= StopWordsRemover().setInputCol("words").setOutputCol("filtered").setCaseSensitive(Fal
          hashingTF = HashingTF().setNumFeatures(1000).setInputCol("filtered").setOutputCol("rawFeature
          idf = IDF().setInputCol("rawFeatures").setOutputCol("features").setMinDocFreq(0)
          lr = LogisticRegression().setRegParam(0.01).setThreshold(0.5)
          nb = NaiveBayes().setModelType("multinomial").setSmoothing(1.0)
          lrPipeline=Pipeline(stages=[tokenizer, remover, hashingTF, idf, lr])
          nbPipeline=Pipeline(stages=[tokenizer, remover, hashingTF, idf, nb])
```

7.6 Train Logistic Regression and Naive Bayes models

```
In [125]: lrModel = lrPipeline.fit(training)
          nbModel = nbPipeline.fit(training)
```

7.7 Display predictions on the test data

```
In [127]: lrPredictions = lrModel.transform(test)
          lrPredictions.registerTempTable("lrPredictions")
          print("Logistic Regression classifier predictions")
          spark.sql("SELECT id, label, prediction FROM lrPredictions WHERE prediction = '0'").show(5)
          spark.sql("SELECT id, label, prediction FROM lrPredictions WHERE prediction = '1'").show(5)
          nbPredictions = nbModel.transform(test)
          nbPredictions.registerTempTable("nbPredictions")
          print("Naive Bayes classifier predictions")
          spark.sql("SELECT id, label, prediction FROM nbPredictions WHERE prediction = '0'").show(5)
          spark.sql("SELECT id, label, prediction FROM nbPredictions WHERE prediction = '1'").show(5)
```

```
Logistic Regression classifier predictions
+-------+-----+----------+
|     id|label|prediction|
+-------+-----+----------+
|7697757|  0.0|       0.0|
|7697782|  0.0|       0.0|
|7697786|  0.0|       0.0|
|7697794|  0.0|       0.0|
|7697805|  0.0|       0.0|
+-------+-----+----------+
only showing top 5 rows

+-------+-----+----------+
|     id|label|prediction|
+-------+-----+----------+
|7698941|  1.0|       1.0|
|7701470|  1.0|       1.0|
|7703762|  1.0|       1.0|
|7705039|  1.0|       1.0|
|7705856|  1.0|       1.0|
```

```
+-------+-----+----------+
only showing top 5 rows

Naive Bayes classifier predictions
+-------+-----+----------+
|     id|label|prediction|
+-------+-----+----------+
|7697757|  0.0|       0.0|
|7697782|  0.0|       0.0|
|7697786|  0.0|       0.0|
|7697794|  0.0|       0.0|
|7697808|  0.0|       0.0|
+-------+-----+----------+
only showing top 5 rows

+-------+-----+----------+
|     id|label|prediction|
+-------+-----+----------+
|7697805|  0.0|       1.0|
|7698625|  0.0|       1.0|
|7699045|  0.0|       1.0|
|7699145|  0.0|       1.0|
|7699710|  0.0|       1.0|
+-------+-----+----------+
only showing top 5 rows
```

7.8 Evaluate the models with Area under ROC curve (AUC) and overall Accuracy

In [128]:
```
evaluator1 = BinaryClassificationEvaluator().setMetricName("areaUnderROC")
evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", m
print ("Logistic Regression: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(lrPred
print ("Naive Bayes: \n", "\t Area under ROC curve (AUC):", evaluator1.evaluate(nbPredictions)
```

```
Logistic Regression:
 	 Area under ROC curve (AUC): 0.9990286117979512
 	Accuracy:  0.9924892703862661
Naive Bayes:
 	 Area under ROC curve (AUC): 0.00029436006122689424
 	Accuracy:  0.8594420600858369
```

7.9 A simple comparison on the two modles's performance

7.9.1 Overall Accuracy:

- lables are assigned by K-MEANS clustering, so overall accuracy was very high
- Logistic Regression classifier carried out a higher overall accuracy (0.99 vs 0.86)
- Both modle can yield predictions accuracy than 0.5 of random
- So Logistic Regression made better classification over Naive Bayes on this training/test set

7.9.2 Area under ROC curve (AUC):

- Logistic Regression classifier carried out a much higher AUC (0.85 vs 0.00)
- Logistic Regression classifier has higher discriminative power over class distribution

7.9.3 In summary, the Logistic Regression classifier has much better performance over the Naive Bayes, whatever on this perticular training/test set or potential future test datasets.

7.9.4 In our raw data showed at the beginning, one class had much more data points (4423) than the other class (154). This bias significantly affected Naive Bayes classifier -- it had very hight error costs (false positive and false negative cost), but not the Logistic Regression classifier. So in our experiment, at least we can conclude that under this kind of bias, we should avoid using Naive Bayes classifier.