CS 6350
ASSIGNMENT _2_

Names of students in your group:

Kuei-Yu Tsai (kxt230002)

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.

1. JohnSnowLabs Spark-NLP: https://github.com/JohnSnowLabs/spark-nlp

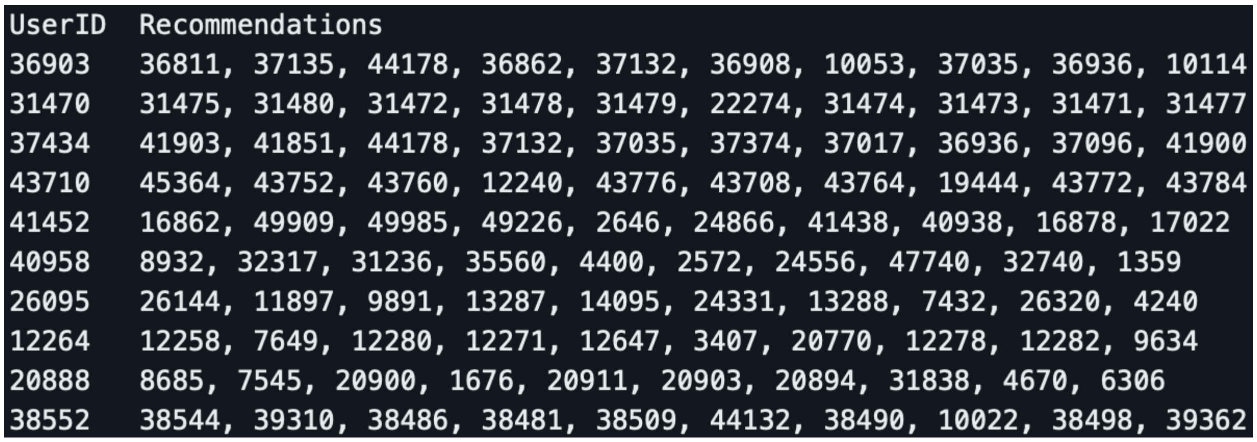
2. The 20 newsgroups text dataset: https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

1. Friend Recommendation using Mutual Friends:
   https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3469934735814656/3340785753073842/520102209841210/latest.html

   Detailed algorithm and pseudo-code:

```
# 1. Read the dataset
#     - Input: File "soc-LiveJournal1Adj.txt"
#     - Output: An RDD in the format (UserID, [Friend1, Friend2, ...])
# 2. Define function parse_line(line)
#     - Split each line into UserID and friends list
#     - Return (UserID, [Friend1, Friend2, ...])
# 3. Apply parse_line function to data to get each user's friends list
#     - user_friends = RDD.map(parse_line)
# 4. Define function generate_potential_recommendations(user, friends)
#     - For each friend Friend1
#         - For each friend of Friend1, Friend2
#             - If Friend2 != User and Friend2 != Friend1
#                 - Generate a recommendation pair (User, Friend2) -> 1
# 5. Apply flatMap to user_friends to generate all potential recommendation pairs
#     - potential_recommendations =
user_friends.flatMap(generate_potential_recommendations)
# 6. Calculate the number of common friends
#     Aggregate the number of common friends and record the number of common friends
for each pair of potential recommendations (user, friends).
# 7. Define function filter_existing_friends(record)
#     - Get the recommendation pair (User, SuggestedFriend) and count
#     - Filtered if SuggestedFriend is already a friend of User
# 8. Sort recommendations by the number of mutual friends (count), and select top 10
for each user
# 9. Randomly select 10 users and display their recommendations
#     - Random 10 sample_users
#     - For each user and recommendations:
#         - Print "UserID recommended friends list" in the format
"UserID\tFriend1,Friend2,..."
```

Result:



```
UserID  Recommendations
36903   36811, 37135, 44178, 36862, 37132, 36908, 10053, 37035, 36936, 10114
31470   31475, 31480, 31472, 31478, 31479, 22274, 31474, 31473, 31471, 31477
37434   41903, 41851, 44178, 37132, 37035, 37374, 37017, 36936, 37096, 41900
43710   45364, 43752, 43760, 12240, 43776, 43708, 43764, 19444, 43772, 43784
41452   16862, 49909, 49985, 49226, 2646, 24866, 41438, 40938, 16878, 17022
40958   8932, 32317, 31236, 35560, 4400, 2572, 24556, 47740, 32740, 1359
26095   26144, 11897, 9891, 13287, 14095, 24331, 13288, 7432, 26320, 4240
12264   12258, 7649, 12280, 12271, 12647, 3407, 20770, 12278, 12282, 9634
20888   8685, 7545, 20900, 1676, 20911, 20903, 20894, 31838, 4670, 6306
38552   38544, 39310, 38486, 38481, 38509, 44132, 38490, 10022, 38498, 39362
```

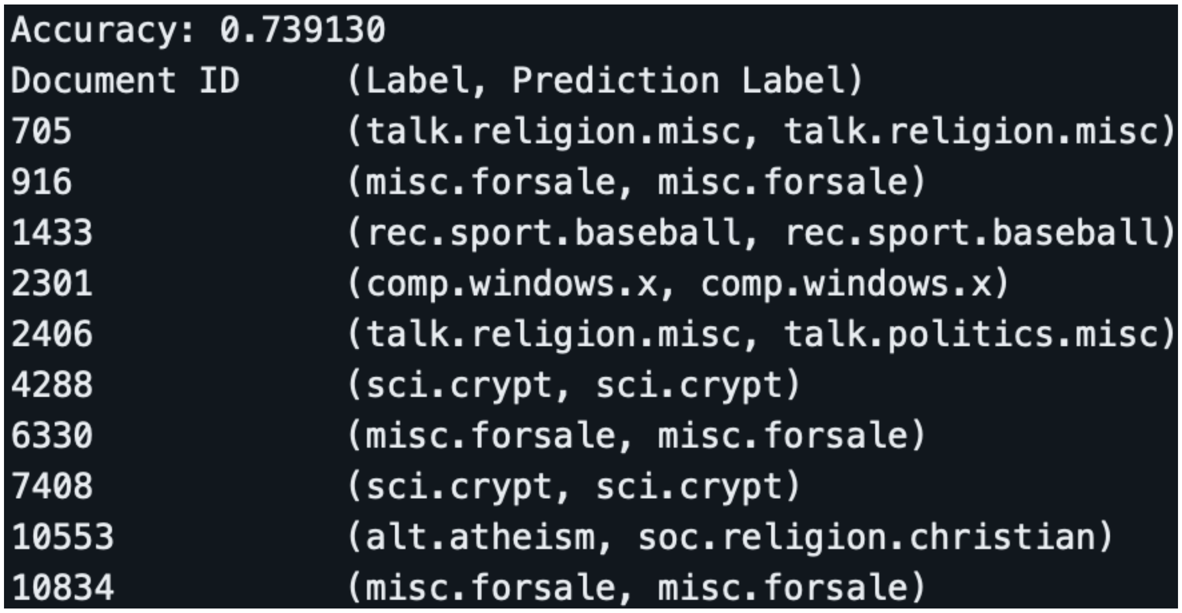
2. Implementing Naive Bayes Classifier using Spark MapReduce:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3469934735814656/3104755756097190/520102209841210/latest.html

Detailed algorithm and pseudo-code:

```
# 1. Load dataset
#     - Input: File "20newsgroups"
#     - Output: Accuracy and an RDD in the format (DocID, Label, Prediction Label)
# 2. Preprocess the text
#     - Tokenization
#     - Stopwords removal
#     - Convert DataFrame to RDD (id, label, words) for MapReduce operations
#     - Seperate training set and testing set
# 3. Calculate Prior Probability P(Class)
#     - Compute the prior probability:
                         $P(C) = \frac{Count(C)}{Total\ Documents}$
#     - Use log probabilities to avoid numerical underflow:
                         $\log(P(C))$
# 4. Calculate Conditional Probability P(Word | Class) with Laplace Smoothing
#     - Count occurrences of each word per class.
#     - Compute total words in each class.
#     - Apply Laplace Smoothing to avoid zero probability:
                         $P(W|C) = \frac{(Count(W,C) + 1)}{Total\ Words\ in\ Class\ C + Vocabulary\ Size}$
# 5. Define Naive Bayes classifier function:
#     - Start with the log prior probability of each class:
                         $\log(P(C))$
#     - For each document:
#         - Calculate class scores based on P(C) and P(W|C):
                         $log(P(C|W_1, \dots, W_n)) = log(P(C)) + \sum_{i=1}^{n} log(P(W_i|C))$
#         - Assign the class with the highest score
# 6. Test the model on the test set
#     - Run Classify with RDD
# 7. Calculate accuracy:
                         $Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$
# 8. Display accuracy and select 10 documents display their label and prediction label
#     - For each document:
#         - Print "DocID Prediction" in the format "DocID\tLabel\tPrediction Label"
```

Result:



```
Accuracy: 0.739130
Document ID     (Label, Prediction Label)
705             (talk.religion.misc, talk.religion.misc)
916             (misc.forsale, misc.forsale)
1433            (rec.sport.baseball, rec.sport.baseball)
2301            (comp.windows.x, comp.windows.x)
2406            (talk.religion.misc, talk.politics.misc)
4288            (sci.crypt, sci.crypt)
6330            (misc.forsale, misc.forsale)
7408            (sci.crypt, sci.crypt)
10553           (alt.atheism, soc.religion.christian)
10834           (misc.forsale, misc.forsale)
```