Data Intensive Computing (CSE – 587)

Assignment 3 Report

Kaggle Team name : Sai Soumyya Anuj

Group Members :

1) Soumyya Kanti Datta- soumyyak – 50336866

2) Anuj Narayanaswamy – anujnara - 50341631

3) Sai Abhishek Koppu – saiabhis - 50317410

Part 1: Basic Model

Steps followed:

- Created spark Dataframes to store Train.csv, Test.csv, Mapping.csv
- Used RegexTokenizer that takes text in 'plot' and breaks it into individual terms.
- Used StopWordsRemover to remove all stopwords from output of RegexTokenizer. Takes input as a sequence of strings and drops all the stop words from the input sequences.
- Then we used CountVectorizer to convert a collection of strings to vectors of token counts. Here we set VocabSize = 1500. During the fitting process, CountVectorier selects the top 1500 words ordered by term frequency across the corpus.
- Next, we generate a new column ***'index'*** that converts labels in ***'Genre'*** to indexes. We have done this by mapping the *mapping.csv* to 'Genre' column using a method we wrote '***function1***'.
- We have made another function ***'function2'***, that takes ***'index'*** column and for each instance creates an array of size 20, and sets elements to 1 for corresponding 'Genre' labels. In other words, we are doing One-Hot Encoding of the multi-label Genre column. With this we generate a new column ***'label'.***
- Now we train the model using ***LogisticRegressionWithLBFGS*** with pre-processed train dataset, and save the model.
- We make a prediction using each model for each genre. And then we append these predictions to an array.
- Finally, we construct a dictionary with key as movie_id and value as the prediction array which is exported into a csv file.
- The model scored an **F1-score of .97557**

Part 2: Using TF-IDF to improve model

- Now to improve the performance of the model, we implemented a TF-IDF based feature engineering technique, that gives a better understanding of how much importance a term provides.
- **Term frequency** is the number of times that term *t* appears in document *d* while **document frequency** is the number of documents that contains the term *t*.
- In pySpark, we use a function called **HashingTF** which takes sets of terms from the and converts those sets into fixed-length feature vectors, utilizing hashing. A raw feature is mapped into an index (term) by applying a hash function. Then *term frequencies (tf)* are calculated based on the mapped indices. We also set numFeatures = 15000.
- Then we apply **IDF** which takes feature vectors created from **HashingTF** and scales each feature. It down-weights features which appear frequently in a corpus.
- These two functions are applied successively after removing stopwords.
- After making this chance, the model gave an improved F1 score of .
- The model achieved an **F1 score of 0.98631.**

Part 3: Custom Feature Engineering

- To further improve the model, we used **word2vec** function to map each word to a unique fixed-size vector. We set vectorsize = 150.
- The model achieved an **F1 score of 0.99990.**

Part 4: BONUS

- Kaggle@Submission 1: Improved the **F1-score of the model to 1** on setting the vectorsize to 500 for **word2vec**.
- Kaggle@Submission 2: Improved the **F1-score of the model to 1** on setting the vectorsize to 1500 for **word2vec**.

Note: We have trained our models, and saved the models for better efficiency. The training code is commented out but still present.

Resources: https://spark.apache.org/docs/latest/ml-features.html