Data Intensive Compu0ng – Assignment-3
Spring Semester -2020

Team Members
Name: Akhil Koppera
UBIT Name: akhilkop
UBID: 50318499

This report gives an overview of the data pre-processing, concepts and the outputs obtained from the codes of Spark used for basic text processing (Big Data processing). Machine learning algorithms from MLlib were used to build the models to predict the movie Genre based on the given data and to get and to provide an overview of saving the models created and predicUons made.

Python programming plaWorm was used for coding and data pre-processing was done using Spark libraries.

PART 1: Term Document Matrix:

In this the given data was loaded using pandas and then converted into spark data frame, then the feature “plot” was tokenized using RegexTokenizer library in Spark. The stop-words were removed from each row vector obtained a\er tokenizing. The term document matrix was created based using CountVectorizer of pyspark with a minDF of 20 and the vector length was limited to 5000 features based on the filtered words obtained a\er removing the stop words, stopwords were removed using spark in built library and few addiUonal stop words were added manually. The given data was divided into train and test so that, tuning of hyperparameters can be done to fit a model, based on the opUmal values obtained “logisUc regression model was fit to the enUre data”, separate models were built for each genre given in the data and the models were saved to use for further predicUons of the test data provided in Kaggle compeUUon. These models we were saving in the disk and then loading into the jupyter notebook to make predicUons. Then, we saved our predicUons in the csv file.

PART 2: TF IDF:

Used the feature tokenized and stop-words removed from data and then converted to TF-IDF, TF-IDF is a feature vectorizaUon method widely used in text mining to reflect the importance of a term to a document in the corpus. Denote a term by t, a document by d, and the corpus by D. Term frequency $TF(t,d)$ is the number of Umes that term t appears in document d, while document frequency $DF(t,D)$ is the number of documents that contains term t. If we only use term frequency to measure the importance, it is very easy to over-emphasize terms that appear very o\en but carry licle informaUon about the document, e.g., “a”, “the”, and “of”. If a term appears very o\en across the corpus, it means it doesn’t carry special informaUon about a parUcular document. Inverse document frequency is a numerical measure of how much informaUon a term provides. TF_IDF was done by using an inbuilt library of Spark and similar to part 1 hyperparameter tuning was done and individual logisUc regression models were fit for each genre given in the data on the enUre data and saved, used to make predicUon on the test data provided. We used numFeatures 15000 for the hashingTf and minDocFreq 3 for converUng to TF-IDF.

We saved the models in the disk and loaded into jupyter notebook to make predicUons. The predicUons were saved in the csv file.

PART 3: Custom Feature Engineering:

Here in this part, we used the same tokenized feature after removing the stopwords at the start. And, we used word2Vec. It computes distributed vector representaUon of words. The main advantage of the distributed representaUons is that similar words are close in the vector space, which makes generalizaUon to novel pacerns easier and model esUmaUon more robust. The reason for choosing this was, Distributed vector representaUon has showed to be useful in many natural language processing applicaUons such as named enUty recogniUon, disambiguaUon, parsing, tagging and machine translaUon. Then based on the features generated by word2vec library of the spark with vector size 300 and minCount 10, individual logisUc regression models were built for each genre and stored so that they can be used for future predicUons. We were able achieve higher F score in Kaggle a\er performing word2vec feature engineering in comparison to the previous feature engineering.

Making PredicUons for given Test data:
Test data was pre-processed by using the pre-processing parameters used for train data and the obtained features were fed to the models created in each part and predicUons were made for different genres and then combined the predicUons to give all the genres to which the movie belongs to based and the combined results were uploaded to Kaggle to obtain the F-score. The F-scores obtained for:

Part A: Term Document Matrix = 0.97993
Part B: TF-IDF = 0.98602
Part C: Custom Feature Engineering = 0.96295

We saved our training model for part1 in pert1 folder. And, for part 2 in pert2 folder. We were loading the model from these folders to make predicUon. For part 3, we saved for part 3 in pert3 folder.

The predicUon file for each part.

Part 1: a3part1.csv
Part 2: a3part2.csv
Part 3: a3part3.csv